Le livre blanc des Serveurs d'Applications

Mars 99

Auteurs :

Les auteurs de ce document sont des consultants d'OCTO Technology, par ordre alphabétique F.Beaudrée, T.Brethes, L.Cinquin, E.Groise, F.Hisquin, V.Massol, P.Pezziardi, D.Schneider.

Pour tous renseignements complémentaires, veuillez contacter F.Hisquin (fhisquin@octo.fr).

Table des matières

Introduction

Objectif du document

L'objectif de ce document est double : faire le point sur les architectures orientées composants et sur les solutions proposées par les éditeurs pour les mettre en place : les serveurs d'applications.

Ce document souhaite adresser le public le plus large possible.

Il s'adresse, d'une part, aux responsables des Systèmes d'Information, passionnés de technologies ou non, qui trouveront à l'issue de chaque chapitre et du document une vision synthétique mettant en avant notre point de vue d'experts sur le thème traité, et, d'autre part, aux architectes et aux développeurs en leur présentant un concentré de réflexions technologiques.

Présentation du document

Dans un premier temps le document présente les principes d'architecture, en mettant en regard des contraintes et des enjeux métier : réactivité dans la mise à disposition des applications, contrainte d'ouverture des canaux de distribution et interconnexion des Systèmes d'Information.

Ce chapitre traite notamment :

- Des architectures multi-tiers, du transactionnel et des architectures à base de composants,
- Des orientations concernant le Poste de Travail,
- Des contraintes de développement,
- De l'administration et de l'exploitation des applications,
- De la sécurité.

Dans un second temps, le document présente les offres actuelles des éditeurs, couramment appelées les serveurs d'applications[1], soit (par ordre alphabétique) :

- L'offre BEA : M3 et WebLogic
- L'offre IBM : WebSphere
- L'offre Inprise : Inprise Application Server
- L'offre Microsoft : Distributed Network Architecture (DNA)
- L'offre Netscape : Netscape Application Server
- L'offre Oracle : Oracle Application Server
- L'offre Sun : NetDynamics
- Les offres d'éditeurs moins "institutionnels" : PowerTier, Gemstone, Secant.

Le lecteur averti aura noté l'absence de Sybase. Cette offre, peu connue des consultants d'OCTO, n'a pas été écartée a priori, mais a fortiori, par la difficulté à obtenir un interlocuteur chez l'éditeur capable de répondre à nos questions. Nous espérons compléter cette étude dans une version prochaine.

1 Application Server en anglais

Principes d'architecture

Les Architectures multi-tiers

Présentation

Une architecture multi-tiers est un modèle d'architecture d'applications dans lequel on sépare la présentation, les traitements et les données[2]. L'objectif poursuivi est de permettre une évolution de l'un de ces trois tiers de façon "relativement" indépendante des deux autres.

L'implémentation physique de ces architectures est souvent soumise aux contraintes de l'existant. Ainsi, elles sont parfois mises en œuvre au travers de plates-formes et de systèmes d'exploitation différents, ce qui complique évidemment leur conception, leur mise au point et leur exploitation.

Les enjeux liés à cette séparation logique sont considérables :

- Améliorer la réactivité de mise à disposition des applications par la réutilisation de "briques" existantes,
- Autoriser l'accès au système par le biais de canaux variés tels que l'internet, les "device mobiles", le Poste de Travail,...
- Capitaliser sur l'existant.

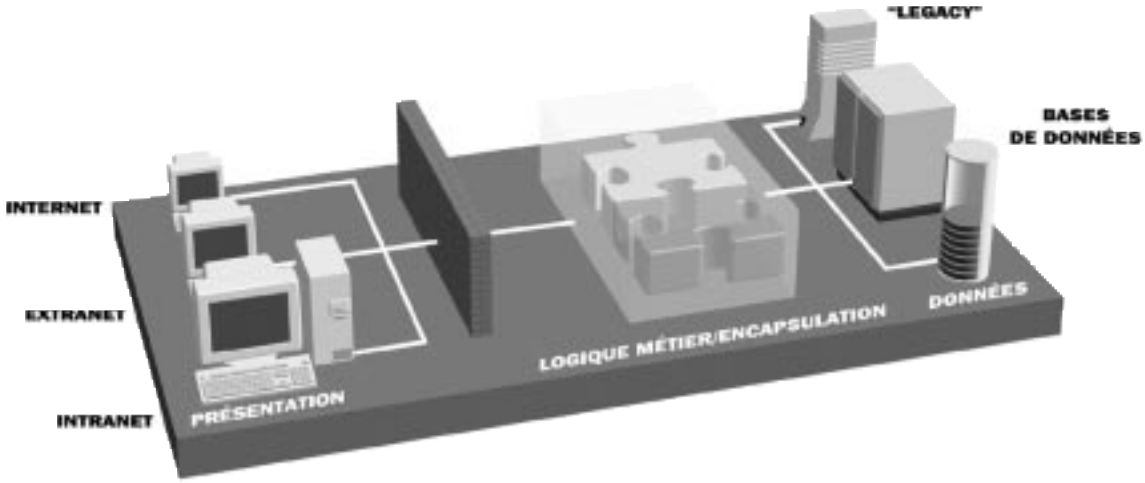


Figure 1 : Architecture 3-tiers

Ce concept ne date pas d'hier ! En effet, par le passé, certains architectes recommandaient déjà d'isoler les traitements métier de la présentation 3270. Heureuses sont les entreprises qui les ont écoutés, elles sont aujourd'hui les plus aptes à profiter des nouvelles technologies de type Web...

A l'inverse, la plupart des applications mode texte (VT, 3270) ou client/serveur de données développées à l'aide d'AGLs, ou, plus récemment les applications Web "dynamiques", n'adhèrent pas à ce modèle.

Il devient alors particulièrement difficile de réutiliser du code métier existant dans de nouvelles applications (modifier l'IHM d'une application, créer un canal de distribution Minitel ou Web, etc.).

La solution adaptée reste alors l'utilisation de produits de "revamping" dont on connaît les qualités médiocres de stabilité et de maintenabilité.

2 Certains modèles plus détaillés insèrent un niveau entre la présentation et les traitements, dont l'objet est de naviguer entre les applications tout en gardant un contexte entre elles.

Le client/serveur de données, avant-dernière grande révolution en date dans l'informatique de gestion, supporte mal les montées en charge, en particulier à cause des protocoles verbeux mis en œuvre entre le client et le serveur, et de l'absence d'un réel multiplexeur d'accès à ces données permettant d'optimiser les consommations de ressources physiques. Ces problèmes, réels en entreprise, ont aujourd'hui des palliatifs auprès des principaux éditeurs (Multithreaded Server, Connection Pooling, etc.). Néanmoins, ces architectures et leurs outils associés ne permettent pas de répondre à l'engouement du client léger.

Enfin, l'informatique centrée sur la donnée pose des problèmes à l'échelle d'un Système d'Information lorsque plusieurs applications doivent communiquer ou utiliser mutuellement des services communs. Leur interopérabilité se fonde sur la donnée : partage de bases, transferts de fichiers, de messages, etc.

Les architectures multi-tiers prennent en compte la montée en charge par la séparation des tiers et la possibilité native de déployer sur plusieurs serveurs physiques.

Le problème de l'interopérabilité n'est cependant pas définitivement réglé par les architectures 3-tiers. Faire coopérer deux applications de manière synchrone, en mode objet, tisse aussi des adhérences fortes au niveau du composant entre ces applications (l'IDL des objets manipulés : l'application A a directement dans son code les appels de méthodes d'un objet B). Si l'on n'y prend pas garde, on risque de déplacer le problème des adhérences de la donnée vers les objets ! La mise en place d'un serveur d'applications doit s'insérer dans un projet d'Architecture global intégrant cette problématique.

Conclusion

Les architectures multi-tiers à base de composants sont aujourd'hui le meilleur choix pour réaliser des applications de qualité car elles obligent à adopter une démarche de conception et séparent dans la réalisation les différents éléments IHM, logique et données.

Néanmoins, attention aux mirages de l'interopérabilité généralisée entre objets : tisser des adhérences entre applications peut s'avérer un réel danger, dont des produits comme les Message Broker tentent d'adresser la problématique.

Les chapitres suivants se proposent de faire le point sur les concepts, les acronymes et les termes consacrés définissant les architectures n-tiers orientées composants.

Les Architectures Transactionnelles

Présentation

Le transactionnel est issu des technologies Mainframe d'IBM des années 70. Il s'agissait alors de mettre massivement à disposition de terminaux des applications en mode texte. Le terme de transaction vient du fait que plusieurs écrans peuvent s'enchaîner avant qu'une réelle modification dans le système ne soit réalisée. Ces transactions s'entendent en revanche mono-moniteur (CICS ou IMS) et mono-base.

Le terme de transactionnel s'est étendu et intègre des notions de distribution de transactions par l'intermédiaire du protocole de Two-Phase Commit. Le Transaction Manager (TM) pilote des sources de données ou de traitements distribuées, les Resource Manager (RM). Le TM garantit l'unité d'œuvre (l'ACIDité pour les puristes) de la transaction : soit l'ensemble des tâches est réalisé, soit aucune ne l'est.

Les TM sont implémentés dans des moniteurs transactionnels, comme Tuxedo (qui utilise le standard XATMI comme interface entre l'application et le TM), Encina, ou plus récemment certains serveurs d'applications (MTS, OAS, etc.).

Les RM sont implémentés dans les principaux SGBD/R (Sybase, Oracle, DB2), et dans certains MOM (MQ*Series[3] par exemple, utilisable via son API MQI). L'interopérabilité des TM et des RM est assurée au travers du support du protocole XA de l'X/Open.

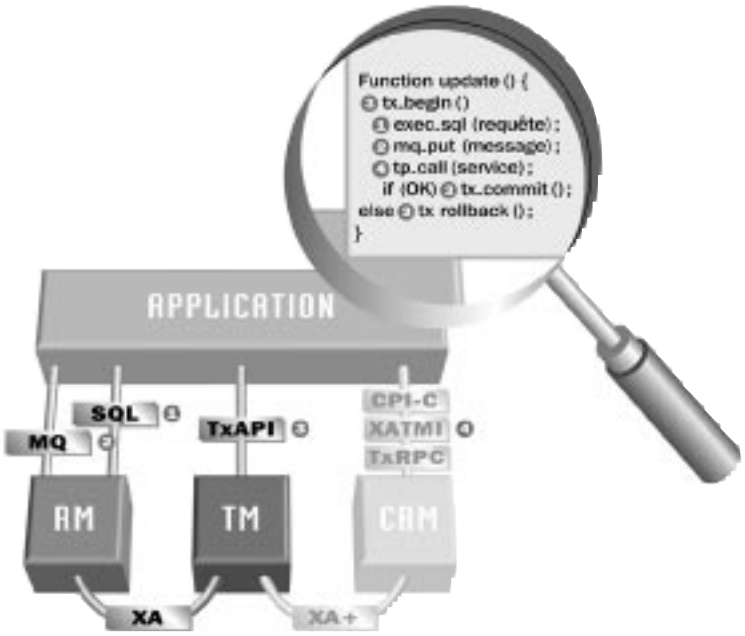


Figure 2 : Architecture transactionnelle orientée traitements.

Le schéma précédent résume ce style d'architecture et effectue un zoom sur le code nécessaire.

Deux constats importants se dégagent :

- La gestion du transactionnel est faite **programmatiquement** : elle nécessite l'écriture de code spécifique utilisant des implémentations de TxAPI dépendantes du moniteur utilisé (aucune norme ne s'est pratiquement imposée),

3 A noter que MQSeries dans sa version 5 peut être utilisé comme un TM.

- Bien que l'X/Open ait défini une norme d'interopérabilité entre les TM (XA+), celle-ci est très rarement implémentée dans les moniteurs transactionnels, et ne répondait pas à un besoin réel. Les communications s'effectuent par l'intermédiaire de passerelles.

De plus, les moniteurs transactionnels assurent une fonction de moniteur, qui représente l'aspect concentrateur de requêtes clients. Cette fonction permet de décharger les SGBD/R (optimisation des connexions ouvertes), et donc d'autoriser une montée en charge lourde.

Ces concepts de transactionnel et de moniteur restent valables dans les serveurs d'applications orientés composants mais leur mise en œuvre en est facilitée.

En effet, les composants ActiveX ou EJB offrent une gestion de la transaction transparente pour le développement. Celle-ci est assurée par le serveur d'applications et peut être externalisée du code. Ceci est l'objet du chapitre suivant.

Conclusion

Une architecture multi-tiers doit anticiper les problématiques liées à la gestion de la performance. L'utilisation d'un moniteur transactionnel est alors incontournable afin de garantir la montée en puissance de l'application et l'intégrité transactionnelle si celle-ci utilise plusieurs sources de données ou de messages.

La transition d'un paradigme traitement vers un paradigme objet n'est pas possible sans rupture. Il faut être conscient qu'il ne s'agit pas d'une évolution mais d'une révolution. La cohabitation technique se fera par l'intermédiaire de passerelles.

Les architectures à base de composants

Présentation

Les moniteurs transactionnels et les serveurs d'applications se différencient par le paradigme qu'ils implémentent, respectivement serveur de traitements et serveur de composants. Dans les deux cas, le concept est orienté service.

On définit le composant comme un objet qui adhère à un modèle, c'est-à-dire qu'il implémente un ensemble de méthodes normées qui lui permettent :

- D'être exécuté dans un serveur d'applications en supportant le transactionnel, la sécurité, le multi-accès, l'indépendance à la localisation, etc. et ce sans ajout de code particulier,
- D'être intégré aisément dans un AGL dans la phase de développement, grâce aux méthodes auto-descriptives d'introspection.

  Cette notion permet un réel assemblage de composants avec des outils de haut niveau, sans se soucier du détail de l'implémentation.

Le modèle de composant apporte également un ensemble d'objets et de méthodes, utilisables dans le code du composant et qui permettent de gérer plus finement les aspects transactionnels et sécurité (voir les chapitres suivants).

Ces deux facteurs permettent de diminuer considérablement la complexité de mise en œuvre d'une architecture 3-tiers, principal reproche fait aux moniteurs TP.

Cette évolution du paradigme traitement, relativement bas niveau, vers le paradigme objet/composant paraît par ailleurs inéluctable, lorsqu'on la met en regard des évolutions du marché :

- Emergence de puissants outils de conception UML (Rational Rose en particulier), maturité des formations aux développeurs,
- AGL de plus en plus orientés objet : PowerBuilder, Delphi, Forté, Visual Studio, etc.
- Offres de moniteurs TP classiques s'orientant vers le composant, notamment BEA avec M3 et IBM avec WebSphere,
- Framework d'objets métier (IBM San Francisco, Microsoft DNAfs, etc.), accès aux ERP via l'objet, etc.

Les modèles

Deux modèles de composants se partagent aujourd'hui le marché :

- Le modèle COM, issu de Microsoft et du monde OLE, désormais renommé ActiveX et dont la première implémentation côté serveur date de 1996 (MTS 1.0),
- Le modèle Enterprise JavaBean (EJB), spécification multi-éditeurs dont Sun et IBM sont à l'origine, et dont les premières implémentations apparaissent actuellement sur le marché.

Le schéma suivant propose une analogie avec les architectures transactionnelles.

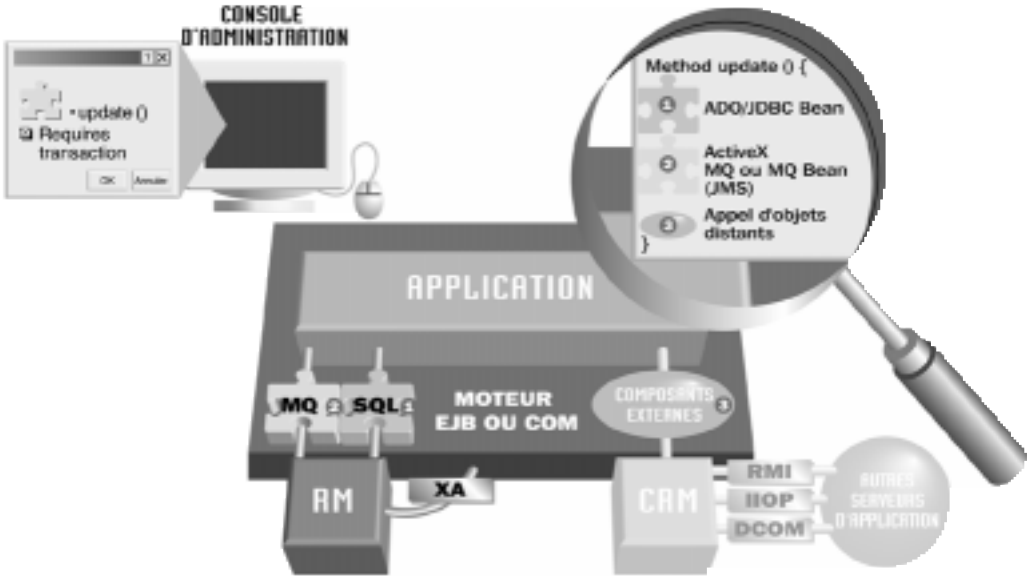


Figure 3 : Architecture transactionnelle orientée composants.

Les éléments importants sont les suivants :

- le développeur est dégagé de la problématique bas niveau de gestion du transactionnel, cette gestion se fait **déclarativement** depuis les outils d'administration,
- l'accès au RM ne se fait plus par API bas niveau mais par utilisation de **composants d'accès** dans l'outil de développement[4],
- l'appel à des composants distants peut se faire au travers d'interfaces RMI, CORBA (via IIOP) ou DCOM suivant la technologie du serveur. Cependant, à l'image des architectures transactionnelles, ce n'est pas une voie à conseiller, notamment à cause des adhérences qu'elle génère et des problèmes d'interopérabilité multi-éditeurs.

Le **modèle COM** permet de développer des composants graphiques (contrôles ActiveX) et des composants non-graphiques (objets métier) tournant sous MTS. Tous deux se retrouvent sous la même appellation composants "ActiveX".

Ces composants pourront être développés en C++, en Visual Basic, voire en Java ou à l'aide d'AGL : Delphi, PowerBuilder, etc.

Le modèle COM n'impose pas de langage d'implémentation. En revanche il est clair que Microsoft étant le seul dépositaire de la "norme", l'utilisateur est tenté d'utiliser les outils de l'éditeur (Visual C++ ou Visual Basic) pour sécuriser son choix, notamment par rapport à ses futures évolutions (COM+).

Malgré des initiatives de portage sous Unix, le choix COM impose de facto la plate-forme Windows NT sur le serveur intermédiaire.

4 Le schéma décrit respectivement les composants du monde COM (ADO, AxtiveX MQ, DCOM) et les composants du monde EJB (JDBC, JMS, RMI ou CORBA).

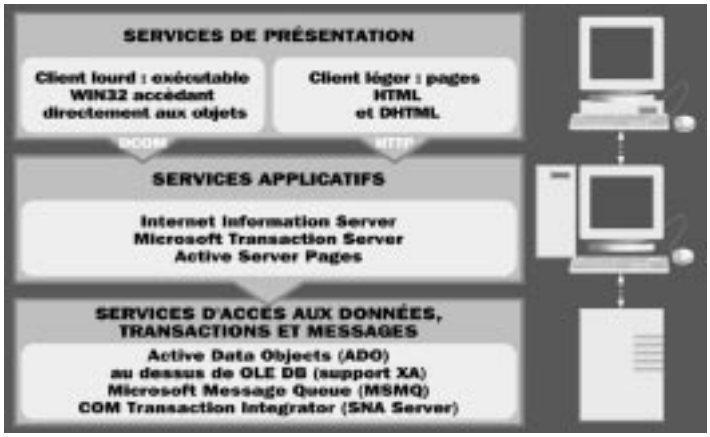


Figure 4 : Architecture dans le monde COM

EJB est un modèle qui assure le développement et le déploiement de composants développés en Java. En ce sens, il se différencie du modèle COM, qui est indépendant du langage.

Comme les objets COM destinés à MTS, les composants EJB ajoutent des méthodes orientées “objet métier” non-graphiques (sécurité, transactionnel, cycle de vie) aux composants JavaBeans classiques.

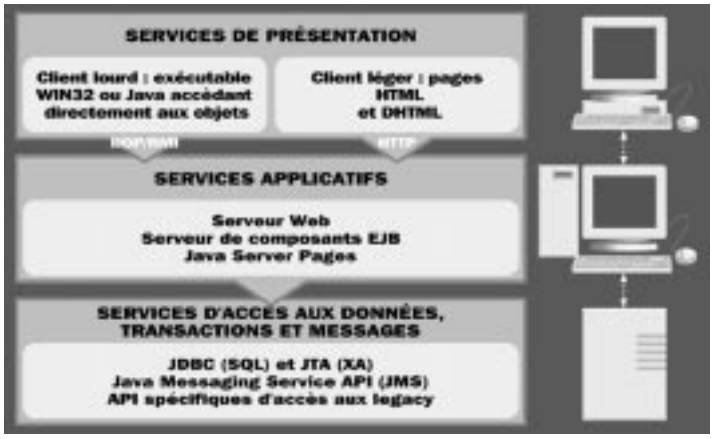


Figure 5 : Architecture dans le monde EJB

Côté **architecture de déploiement**, les composants métier sous MTS peuvent être sollicités par des clients au travers de DCOM (client lourd) ou de HTTP au travers d’IIS et de pages HTML via les ASP (client léger).

Un serveur EJB s’appuie généralement sur un moteur CORBA qui lui fournit les services bas niveau nécessaires à la localisation des objets, à la gestion des événements, à l’appel de méthodes, etc.

En revanche il ne l’impose pas. Comme corollaire, on peut accéder aux composants EJB au travers des protocoles RMI, IIOP, voire COM, au travers d’un proxy RMI ActiveX (client lourd).

A l’instar des technologies ASP de Microsoft, il est possible de piloter ces composants dans des pages HTML par l’utilisation de la spécification JSP de Sun. On retrouve peu ou prou cette technologie dans la plupart des serveurs d’applications.

En ce sens, EJB encapsule totalement les aspects CORBA bas niveau : IDL, BOA, etc. simplifiant d'autant la tâche du développeur. Ceci ajouté aux faibles chances d'interopérabilité directe entre les moteurs EJB[5] place CORBA dans une position délicate… on aurait pu penser que seul COM/DCOM pouvait concurrencer CORBA, on se trompait : c'est EJB qui s'est chargé d'enterrer le modèle !

La spécification EJB définit un modèle de **support transactionnel** déclaratif, relativement identique à celui de MTS : les composants peuvent être déclarativement estampillés comme transactionnels, sans code spécifique. Contrairement à MTS, la granularité peut descendre jusqu'au au niveau de la méthode. Il est aussi possible de gérer encore plus finement ce modèle en utilisant l'API JTA dans les composants. Cette API utilise les services de l'OTM sous-jacent. Elle est semblable à ce que l'on trouve sous MTS (SetComplete, SetAbort,…).

Sur le papier, il est ainsi envisageable de délimiter des transactions mettant en jeu des SGBDR, des serveurs IIOP compatibles OTS1.1, des MOM,… ceci restant théorique. La réalité aujourd'hui est moins brillante : seul Oracle dispose d'un driver JTS pour son SGBDR, mais utilisable uniquement avec son serveur d'applications. La spécification elle-même n'est pas encore stabilisée. Comme toujours au final, l'interopérabilité ne se fondera pas sur les standards mais sur des accords factuels entre éditeurs. Exemple : Microsoft MTS ne supporte pas directement XA, mais est capable de faire du transactionnel avec DB2 et Oracle…

La principale différence des moteurs EJB par rapport au "moteur" COM MTS est la définition d'une **persistance** gérée par le moteur lui-même (container-based persistence). Cet aspect de la spécification est optionnel pour EJB1.0 et sera rendu obligatoire pour le support de EJB2.0. Néanmoins, seuls quelques produits l'implémentent aujourd'hui[6].

En effet, le problème est loin d'être simple puisqu'il relève finalement de la problématique des SGBD/O : comment stocker simplement un objet, sans code supplémentaire (SQL ou autre) et comment le restituer à la demande de manière performante.
Des éditeurs comme O2, Versant ou Object Design, ont démontré la faisabilité du modèle, cependant aucune référence de système de gestion sur des volumes significatifs n'existent en production. On peut cependant espérer que l'essor des serveurs d'applications pourra dynamiser ce marché, soit au travers de SGBD/O natifs, soit au travers de ponts relationnels/objets entre les moteurs et les SGBD/R classiques.

Sans rentrer ici dans les détails techniques complexes, les difficultés d'implémentation sont essentiellement liées à l'héritage et à la gestion de la profondeur du cache.

Dans l'une ou l'autre de ces technologies (EJB/COM), le modèle ne fait pas tout. S'y lancer nécessite au minimum :

- Une évolution dans la manière de **concevoir les applications**

  Idéalement on devrait se placer dans une logique horizontale séparant la réalisation et les test des composants de leur intégration.

- Le développement d'un **framework technique** pour assister le développeur

  Ce type de framework est d'autant plus important aujourd'hui que le niveau de maturité des outils est relativement faible. Un tel framework automatise la gestion des locks applicatifs, de la sécurité, des portions d'IHM standard, etc.

5 La seule intéropérabilité viable se fonde sur une même souche logicielle, et malheureusement pas sur des spécifications générales… c'est pourquoi de nombreux éditeurs intègrent une version de l'ORB Visibroker dans leur serveur.

6 Voir l'état de l'art

Intégration de l'existant

Tous les serveurs d'applications proposent, à un plus ou moins bon niveau de qualité, des interfaces permettant de se connecter aux systèmes existants : systèmes transactionnels, MOM, etc. Cette intégration est réalisée au travers de la fourniture d'API propriétaires, à embarquer dans le code du composant. On citera de manière non-exhaustive et dans le désordre : Microsoft SNA Server, BEA /HOST, Oracle Gateways, IBM e-Business Connectors, Netscape Kiva extensions.

Les principaux ERP (SAP, PeopleSoft) offrent ou annoncent quant à eux des interfaces dans les deux mondes : COM et Java. Il convient d'en limiter l'utilisation aux purs besoins d'interopérabilité synchrone.

Ces passerelles sont un élément capital de l'offre, car le succès du serveur de composants dépend de sa capacité à devenir le **portail du Système d'Information**. En effet, un des grands objectifs de ce style d'architecture est de capitaliser à long terme sur des composants, dont le code peut être dans le composant lui-même bien sûr, mais aussi et surtout dans des systèmes existants.

Conclusion

COM ou EJB ? Quel choix pour les grandes entreprises ?... La question n'est pas là aujourd'hui, et ce pour un certain nombre de raisons :

- Les différences entre les deux technologies sont très faibles,
- Le fait que Microsoft édite des produits, puis ouvre ensuite l'API, n'est pas en soi rédhibitoire face à la stratégie de Sun qui consiste à spécifier l'API, puis encourager les éditeurs à l'implémenter : dans les deux cas, l'interopérabilité inter-éditeur reste un mythe,
- La montée en puissance des serveurs NT sur le "middle-tiers" se fait indifféremment autour des deux technologies,
- Les deux "standards" évoluent de concert (respectivement COM+ et EJB2.0) dans des directions qui sont les mêmes : développement plus aisé, administration facilitée, gestion de la sécurité simplifiée, etc.

Le problème n'est donc pas tant le choix de l'une ou l'autre des technologies - les deux apparaissent aussi pérennes - qu'un virage essentiellement culturel à négocier avec les équipes de conception, de développement et de production.

Le Poste de Travail : thin client ou fat client

Introduction

On l'aura compris, la nature même des serveurs d'applications apporte une grande liberté et modularité pour le Poste de Travail, permettant une évolution progressive des applications.
On distingue quatre grands types d'applications clientes :

- Les applications C/S traditionnelles (Win32),
- Les applications Java,
- Les applications émulées,
- Les applications Web.

Chacun possède ses propres caractéristiques qui sont souvent complémentaires. On peut imaginer une application cliente Windows ou Java pour des postes Intranet, la même application en mode émulé pour des agences distantes à l'étranger et une application Web pour des fournisseurs via Internet. L'intérêt remarquable est que toutes ces applications clientes, de technologies et d'usages très différents, utilisent toutes les mêmes composants applicatifs se trouvant sur un serveur d'applications, ainsi que la même source de données sous-jacente.

Application C/S Windows

Le client traditionnel est constitué d'une application Windows (.EXE) s'exécutant sur le poste client et se trouvant généralement sur le poste client. Cette application est développée avec les outils classiques L4G (Power Builder, NSDK, Visual Basic, Delphi, ...). Elle utilise les composants traitements via le réseau en utilisant le middleware fourni par le serveur d'applications : DCOM (dans le cas de MTS) ou CORBA/IIOP (dans les cas des serveurs d'applications conformes à la norme CORBA 2.0). Ceci nécessite la présence de proxy (représentant les composants distants sur le poste client). Dans le cas de DCOM, ces proxy sont des ActiveX et s'intègrent dans les outils de développement existants.

Dans le cas des serveurs d'applications basés sur CORBA, la situation est plus délicate, il n'y a pas de proxy directement utilisables spécifiés dans la norme pour les L4G. On passe généralement par une passerelle CORBA/ActiveX fournie par l'éditeur du serveur d'applications ou une société tierce.

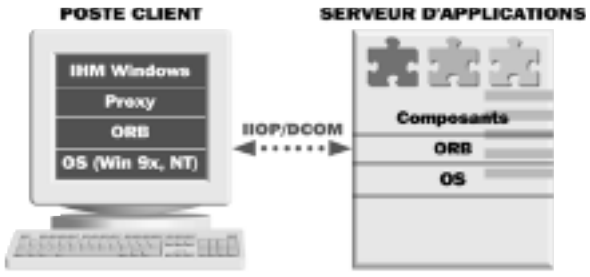


Figure 6 : Architecture C/S Windows

■ Avantages

L'utilisation des L4G, qui sont des produits matures et stabilisés, procure indéniablement des avantages. Ils sont simples d'emploi et ne demandent pas de compétences nouvelles. Ils permettent de développer rapidement des interfaces graphiques riches en terme de fonctionnalités (drag and drop, contrôles

graphiques évolués, menus contextuels, ...) dont les utilisateurs sont devenus maintenant familiers. Ils offrent également des fonctionnalités évoluées d'intégration avec les outils bureautiques via le support de OLE Automation, pour exporter des données vers Excel, générer un document Word, envoyer des messages électroniques, ou encore générer des rapports d'impression complexes.

■ Inconvénients

Les applications Windows n'en présentent pas moins des inconvénients, tant techniques que fonctionnels.
Techniquement parlant, l'instanciation à distance des composants serveurs depuis le poste client nécessite la présence sur le poste client des librairies implémentant le protocole DCOM ou IIOP (DCOM est fourni en standard avec les plates-formes Windows 9.x et NT, sous réserve de quelques Service Packs...).
Les proxy doivent être regénérés et redéployés sur chaque poste client lorsque l'interface des composants serveurs est modifiée.

Enfin la configuration de ces proxy n'est pas toujours d'une simplicité et d'une maintenance à toute épreuve. En effet, le protocole DCOM nécessite d'enregistrer dans la base de registre du poste client le proxy, ainsi que le nom de la machine serveur où se trouve le composant distant correspondant. Si jamais le composant serveur est déplacé vers un autre serveur, la base de registre de chaque poste client doit être mise à jour. L'utilisation du protocole DCOM ou IIOP peut poser un problème s'il y a un firewall entre le poste client et le serveur d'applications. Peu de firewalls supportent en effet ces protocoles. Ce problème peut être contourné en encapsulant DCOM ou IIOP dans des requêtes HTTP (HTTP tunnelling).

Tous ces problèmes, surmontables il est vrai, se rajoutent à la complexité et au coût inhérent au déploiement et à la maintenance des applications Windows elles-mêmes[7]. Elle peut être contournée par un outil d'administration global ayant acquis une maturité suffisante (CA TNG, Tivoli)
Bien évidemment, ce type de clients n'a de sens que dans un Intranet, dans lequel la configuration des postes clients est maîtrisée par l'entreprise.

Applications/Applets Java

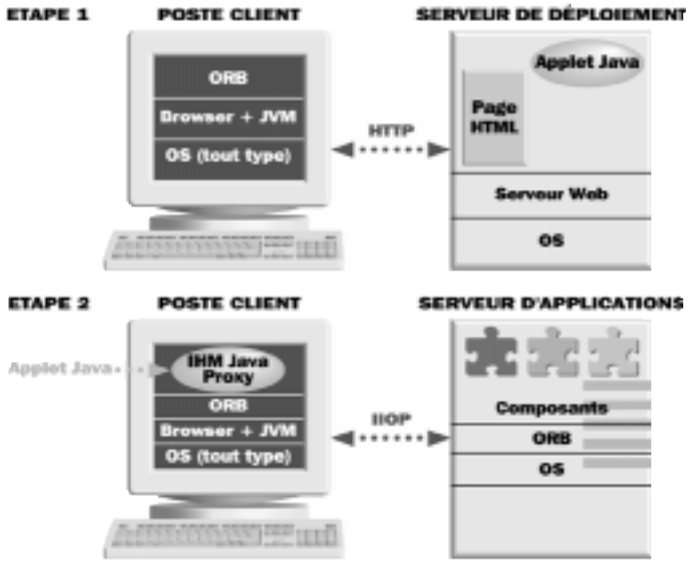


Figure 7 : Applications/Applets Java

7 Voir les rapports concernant le Total Cost of Ownership du Gartner Group.

Les applications ou applets Java ont un fonctionnement similaire à celles des applications Windows traditionnelles. La différence principale concerne le langage et l'outil de développement, basé sur Java plutôt que sur des langages L4G. La communication est basée sur le middleware fourni par le serveur d'applications ou plus simplement sur RMI (Remote Method Invocation) inclus dans la plate-forme Java en standard.

■ Avantages

L'utilisation des proxy représentant les composants applicatifs distants est simplifiée, puisque la norme CORBA a spécifié des proxy (ou stub) en Java. L'utilisation de Java garantit "en théorie" la portabilité de l'application cliente sur toute plate-forme.

Un autre intérêt essentiel est le mécanisme "d'auto-déploiement" de l'application Java à travers un browser. Ce mécanisme est maintenant stable et performant, avec l'utilisation des fichiers compressés JAR ou CAB permettant de transférer sur le poste client l'ensemble des classes Java (bytecode) nécessaires à l'application.
Les outils de développement Java convergent de plus en plus vers les outils L4G traditionnels et facilitent grandement les développements d'applications graphiques à la mode Windows (ex : JBuilder, Visual Café, Visual Age for Java...).

■ Inconvénients

L'utilisation d'applications ou d'applets Java riches en terme d'IHM nécessite d'avoir évidemment la bonne version du runtime Java et de ses librairies associées (en pratique Java 2 incluant les composants graphiques Swing). Ceci implique d'avoir une bonne maîtrise des postes clients sur lesquels s'exécute l'application et donc limite quelque peu la portabilité de l'application sur toute plate-forme dans le cadre d'un Extranet ou d'Internet.

L'intégration bureautique (vers Microsoft Office ou pour générer des rapports d'impression complexes) est relativement problématique, Java et les technologies OLE Automation ne proposant pas d'interfaçage simple et natif.
Les applications et applets Java restent encore malgré tout plus lentes au démarrage et pour l'affichage graphique que les applications Windows natives.

Emulation Windows

Un nouveau type de client prometteur est le client Windows émulé. La société Citrix avec son offre MetaFrame, couplée avec la version Windows Terminal Server pour Windows NT permettent d'exécuter des applications Windows sur un serveur NT et de n'envoyer au poste client, réduit à sa plus simple expression, que le flux vidéo (plus précisément le différentiel des écrans pour plus d'efficacité...). Le client envoie simplement au serveur les entrées clavier et le mouvement de la souris. C'est un mécanisme identique à l'émulation X dans le monde Unix.

Le seul pré-requis est d'installer un client WTS ou un client Citrix sur le poste client. Les clients WTS ne sont disponibles que pour Windows 95, 98, NT et Windows CE, mais des clients Citrix sont disponibles non seulement pour les plates-formes Windows, mais également pour Mac et les principaux UNIX.
Les protocoles utilisés (ICA pour Citrix et RDP pour Microsoft) fonctionnent au dessus de TCP/IP et sont très peu gourmands en terme de bande passante, de l'ordre de 20-30 kb par client. Aujourd'hui, seule la solution Citrix Métaframe sur un socle WTS permet de mettre en place une solution performante et outillée dans un environnement complexe.

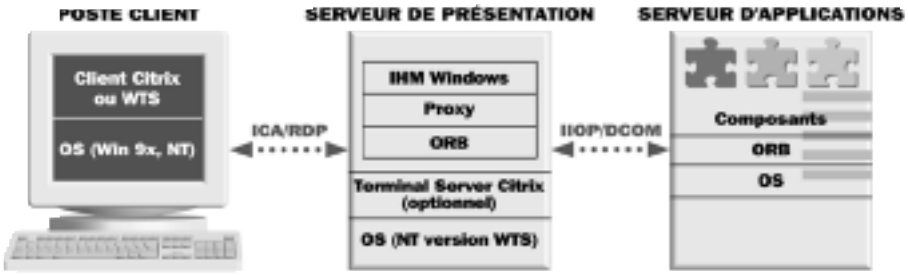


Figure 8 : Emulation Windows

■ Avantages

L'avantage évident est de pouvoir bénéficier de l'intérêt des applications Windows, sans en avoir le principal inconvénient, qui est le coût du déploiement et de maintenance.
La consommation réduite de bande passante permet des accès distants en utilisant des lignes à faibles débits, notamment pour les centres ne possédant pas une équipe informatique sur place.

■ Inconvénients

Les applications nécessitent parfois certaines modifications pour fonctionner en mode multi-utilisateurs (ne plus écrire dans un fichier fixe, faire une copie du normal.dot, estampiller des dll pour qu'elles soient multi-user, gérer les profils utilisateurs, etc.).

Le protocole RDP ne gère pas le support du son, mais cela ne constitue généralement pas un inconvénient majeur pour des applications professionnelles.

Applications Web

Elles se distinguent des applications Windows/Java par l'utilisation du protocole HTTP entre le client (un browser), et le serveur applicatif.
Il convient alors d'installer un serveur Web qui reçoit les requêtes HTTP et d'une "passerelle" vers les composants serveurs.

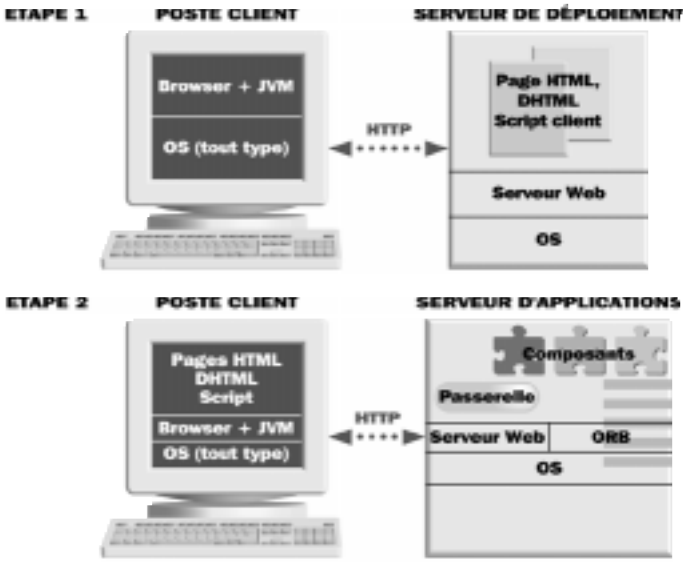


Figure 9 : Applications Web

Cette passerelle peut être une extension classique (DLL ISAPI, NSAPI ou encore Servlet Java) ou bien du script se trouvant dans les pages HTML et s'exécutant côté serveur (ASP ou JSP). Vis à vis du composant du tiers traitement, le client n'est pas le browser ou le poste client, mais l'extension Web ou le script serveur dans la page HTML.
Le client "logique" du composant applicatif se trouve derrière le serveur Web, généralement sur la même machine que l'application serveur. Les problèmes de configuration, de déploiement et de firewall ne s'appliquent donc pas dans ce cas.

■ Avantages

Ces applications sont particulièrement adaptées pour résoudre simplement les problèmes de déploiement et de configuration des postes de travail : L'application est centralisée sur un serveur Web.

De plus, elles véhiculent un nouveau paradigme en terme d'interface utilisateur : la page HTML et le lien hypertexte. L'interface graphique et la navigation en sont grandement simplifiées, voire intuitive par rapport aux interfaces Windows ou Java. Le fait de pouvoir mixer de manière native des applications Web et des documents statiques est également très intéressant pour l'utilisateur.
La généralisation des technologies Web en entreprise en font un standard incontournable et les compétences dans ce domaine commencent à se généraliser. La simplicité des technologies en elles-mêmes (protocole HTTP et langage HTML) représentent un avantage crucial.

Enfin la technologie Web est la seule qui permette de réaliser aujourd'hui des applications réellement universelles (indépendantes du poste client), moyennant quelques contraintes[8]. Ceci est particulièrement important quand il s'agit de proposer des services au grand public, via Internet, que ce soit pour du commerce électronique ou des services personnalisés à valeur ajoutée (banque en ligne, consultation de consommation téléphonique, ...)

■ Inconvénients

La richesse et l'ergonomie de l'IHM est souvent mise en cause. Tout en simplifiant l'interface utilisateur (en apportant de nouveaux paradigmes comme la notion de navigation par liens hypertextes et les frames au lieu du multi-fenêtrage), elles n'offrent pas toute la richesse nécessaire pour bon nombre d'applications, comme les grilles de données, les contrôles graphiques évolués (activation/désactivation, liens entre champs, calendrier, charts, ...).

Cette limitation, inhérente au langage HTML, peut être contournée en partie par des utilisations astucieuses des frames, des images et du scripting côté client. Il n'en reste pas moins que ces "astuces" augmentent de manière non négligeable le temps et le coût du développement des interfaces Web et qu'elles ne sont pas industrialisables.
C'est pourquoi l'utilisation des améliorations et extensions du langage HTML qu'est le Dynamic HTML (DHTML) permet entre autre de définir des composants HTML réutilisables et fournit toute la richesse souhaitée en terme d'IHM. La relative jeunesse de cette technologie explique aujourd'hui le manque d'outils de type L4G pour construire des pages DHTML. Cette lacune devrait être comblée rapidement compte tenu de l'engouement du marché sur ces technologies.

Conclusion

Le schéma suivant est assez instructif ; on y voit tout d'abord des applications Web qui tendent à devenir aussi riches que des applications classiques Windows, notamment au travers des technologies DHTML.

8 Seul le support de HTML 3.2 est réellement assuré à l'identique chez Netscape et Microsoft... moyennant une bonne dose d'intégration pour obtenir le même résultat sur toutes plates-formes et toutes versions. Les dernières extensions des navigateurs, et notamment le DHTML sont aujourd'hui totalement incompatibles et nécessitent une prise en compte spécifique.

On perçoit ensuite la possibilité d'augmenter efficacement la capacité de déploiement d'applications existantes au travers de technologies d'émulation Windows.
Enfin, les applications Java ou ActiveX, déployées sur un Poste de Travail, ne trouveront de débouchés que si elles représentent une réelle alternative aux deux options évoquées plus haut.

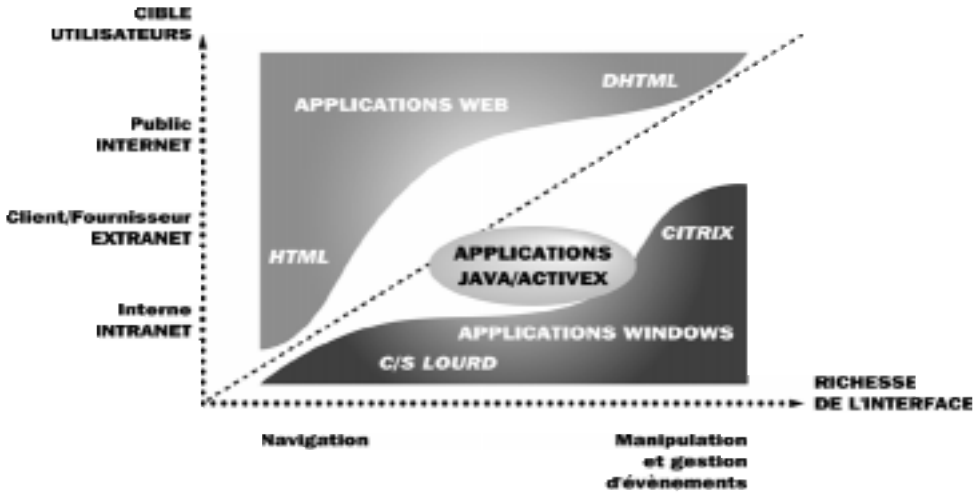


Figure 10 : Choix de technologie

Ce n'est pas le cas aujourd'hui. D'une part, le paradigme Java "write once, run everywhere" est un mythe pour qui a tenté un développement multi-plates-formes[9], et de l'autre, malgré les fonctionnalités avancées du navigateur Internet Explorer, déployer du COM demeure un casse-tête : gestion des versions, des clés de registre, des versions de DCOM,...

Pour Java, une issue possible est l'apparition de postes de travail non-Windows performants de type Network Computer Java, Linux Box, etc. Cette possibilité est le fer de lance d'éditeurs comme Sun, Oracle ou IBM, mais n'a jusqu'à maintenant pas tenu ses promesses... à suivre néanmoins.

En conclusion, miser sur les technologies Web est clairement un bon pari aujourd'hui, au vu des initiatives de tous les éditeurs dans ce domaine. Quant à Java sur le Poste de Travail, il est difficile d'en percevoir l'intérêt sans une adoption massive de terminaux dédiés à cette technologie.

9 Le lecteur pourra pour s'en convaincre essayer de développer une application Java «normale», c'est-à-dire de plusieurs dizaines d'écrans, avec menus, boîtes de dialogues, etc. et se lancer dans une intégration sous Netscape, Internet Explorer, JRE sous Solaris, Linux, Mac ! Certaines surprises sont au rendez-vous : gestion différente de la modalité, des polices, des alignements, des rafraîchissements, des raccourcis, etc.

Les contraintes de développement

Les principes des architectures 3-tiers à base de composants fonctionnant autour d'un serveur d'applications ont été exposés précédemment. Reste maintenant à considérer les outils qui vont permettre de les développer.

Des atouts pour le développement

■ Une complexité masquée

Rappelons-le, les serveurs d'applications assurent bon nombre de services techniques. Au final, les composants métier peuvent être développés comme s'ils étaient mono-utilisateurs. Une fois déployés au sein d'un serveur d'applications, ceux-ci vont pouvoir supporter des appels concurrents et une montée en charge. Car tel est bien l'enjeu : libérer le développeur de tous les aspects techniques (gestion des threads, concurrence d'accès, gestion des connexions SGBDR...) pour que celui-ci soit à même de se concentrer sur la seule logique métier.

■ Des fonctions de sécurité et d'administration déportées

Les serveurs d'applications disposent de consoles qui permettent de gérer de manière centralisée les problématiques d'administration et de sécurité des composants métier. Il n'est donc pas nécessaire de vérifier à l'intérieur du code d'un composant si l'utilisateur est habilité à accéder aux méthodes de ce composant. Là encore, le développeur se voit soulagé d'une part non négligeable et fastidieuse du développement. Ce type de gestion de la sécurité est très souple. Un même utilisateur pourra avoir accès à une base en écriture par l'intermédiaire d'une application donnée, alors que cet accès lui sera refusé dans le cadre d'une autre application.

Les limites actuelles

■ Les enjeux de la conception

Les architectures à base de composants métier rendent à la phase de conception le rôle central qui doit être le sien. Le découpage fonctionnel impose en effet d'isoler les règles métier des éléments de présentation. Par ailleurs, la mise en place de stratégies de réutilisabilité des composants passe par une définition très fine des objets métier sur lesquels les applications vont se bâtir. En conséquence, il devient beaucoup plus délicat de développer "à l'intuition" une application jetable. Mais ce type d'applications s'avérant bien souvent impossible à maintenir, la rigueur qu'impose le modèle à base de composants ne doit pas être considérée comme une contrainte.
Les architecture 3-tiers exigent un minimum de rigueur sous peine d'échec. Les outils de conception deviennent donc des instruments indispensables à tout développement. Ils vont générer automatiquement le squelette de code qui reflète la structure de l'application.

Les processus de construction d'une application ne sont jamais tout à fait linéaires. Quelques considérations techniques ou fonctionnelles qui n'avaient pas été anticipées peuvent faire évoluer des éléments de la structure de l'application. C'est dans ce processus itératif que les fonctionnalités de Reverse Engineering des outils de conception prennent tout leur sens puisqu'elles vont permettre et normaliser d'indispensables allées et venues entre d'une part, le modèle logique et conceptuel de haut niveau issu de la formalisation, et d'autre part, l'application réelle. Soulignons cependant que le Reverse Engineering reste une opération complexe rarement supportée de façon complètement satisfaisante par les produits actuels.

De plus en plus, les environnements de développement intègrent en standard des outils de conception légers capables de générer un squelette dans le langage de l'environnement et d'assurer un Reverse Engineering élémentaire.
Cependant ce type d'outils restent particulièrement limités pour la génération de code IHM et en particulier n'englobe à l'heure actuelle pas les interfaces Web.

■ Multiplicité des outils de développement.

Les architectures 3-tiers à base de composants présentent, nous l'avons vu, l'avantage de coordonner et de faire collaborer des éléments hétérogènes du Système d'Information.

A l'heure actuelle, il faut souvent un environnement de développement pour la réalisation du tiers serveur et un autre pour la réalisation du tiers client. Sans compter les composants techniques pointus pour lesquels on opte pour le C++ !
L'inconvénient immédiat de cette prolifération d'outils est évidemment la mise à jour et la gestion des compétences internes.

A cet égard, les technologies Java bénéficient d'une approche intégrée et cohérente qui leur confère un avantage intéressant. En effet, le langage Java peut être utilisé aussi bien pour le développement des composants serveurs, les EJB, que pour le développement de l'interface utilisateur sur le poste client.

Microsoft, de son côté, s'efforce de fusionner son langage de script DHTML, le VBScript avec le langage de développement Visual Basic, l'un des standards du marché des outils RAD. Pour le développeur, la stratégie est payante, puisqu'il peut, avec un minimum de dépaysement, développer aussi bien un composant serveur qu'une IHM cliente… qui ne fonctionnera qu'avec les browsers Microsoft.

Quoiqu'il en soit, une chose semble sure, l'avenir appartient aux ateliers de développement qui permettront à la fois, et avec un minimum de différences, de développer un composant serveur et une IHM cliente indépendamment de la technologie retenue (application Web ou non).

■ Gestion de la persistance

Nous l'avons vu auparavant, les serveurs d'applications gèrent des notions de pools de connexions aux SGBDR et utilisent pour ce faire un compte générique. D'autre part, la multiplicité des accès qu'ils doivent être à même de supporter, et notamment les accès Internet, dont nul ne peut désormais faire l'économie, impose d'avoir des composants métier susceptibles de supporter des modes d'utilisation déconnectés, composants dits sans état (stateless).

Les conséquences sont importantes. En premier lieu, la traçabilité des accès à la base de données n'est plus aussi automatique. En effet, le SGBDR ne voit plus qu'un seul utilisateur, le serveur d'applications.
Dès lors, il devient essentiel d'assurer de façon programmatique le suivi des accès au SGBDR par des mécanismes de traces placés directement dans le code des composants.

L'autre limite importante liée aux modes d'utilisation non connectés est l'impossibilité d'utiliser les mécanismes de verrouillage des enregistrements que propose le SGBDR en mode SQL. Ces mécanismes doivent être intégrés dans le code des composants. Deux stratégies existent. Le mode optimiste et le mode pessimiste. Les deux méthodes imposent d'ajouter une colonne "estampille" aux tables de la base de données.

La **méthode pessimiste** consiste à signaler dans la colonne estampille un accès pour modification à un enregistrement. Tout autre accès au même enregistrement pendant ce laps de temps sera refusé sur la base de la valeur de l'estampille. Cependant, la logique de l'application ne suffit pas à garantir que le verrou sera supprimé lorsque l'utilisateur n'aura plus besoin de l'enregistrement en question. En effet, le poste de l'utilisateur peut tomber sans avoir eu le temps de notifier qu'il relâche l'enregistrement. La difficulté principale consiste alors à mettre en place un mécanisme capable de supprimer, après un certain temps, les verrous caduques dans les tables.

Ce mécanisme est adapté lorsque deux utilisateurs sont susceptibles très fréquemment d'accéder en modification aux mêmes données en même temps. Cette situation reste assez rare.

La **méthode optimiste** consiste à considérer l'estampille comme un numéro de version qui sera incrémenté à chaque mise à jour. Si un utilisateur demande un enregistrement et souhaite valider ses modifications et si entre-temps, l'estampille de l'enregistrement a changé, les modifications ne seront pas prises en compte. Un utilisateur ne peut modifier un enregistrement que si son numéro de version est le même que celui dans la base de données. La difficulté réside alors dans la gestion des données utilisateurs dont la mise à jour a été refusée.

Ce mécanisme est adapté lorsque les appels concurrents à la base en modification sont rares.Ces fonctionnalités seraient les bienvenues comme composants techniques dans les futures versions des serveurs d'applications.

■ Mapping de l'interface utilisateur sur les composants

A l'heure actuelle, les environnements de développement de type L4G et RAD sont particulièrement bien outillés pour la réalisation d'applications 2-tiers. En effet les éditeurs de ce type de produits ont mis l'accent sur l'accès aux données et offrent désormais des interfaces très conviviales qui permettent de "mapper" en quelques clicks de souris une IHM sur des données persistantes dans un SGBDR.

Ce niveau d'intégration n'est pas encore aussi abouti en ce qui concerne les composants. En effet, le développement de composants bénéficie des outils performants d'accès aux données disponibles dans un SGBDR, mais la construction d'une application cliente faisant appel à des composants serveurs reste délicate. En résumé, les outils de développement assurent des fonctionnalités de "DataBinding" vers les SGBDR mais intègrent peu et mal le "ComponentBinding".

Il y a en effet une rupture de style entre le tiers traitement et le tiers présentation qui ne mettent bien souvent pas en œuvre ni les mêmes langages ni les mêmes technologies.

Dès lors, le développeur doit encore, en règle générale, implémenter lui-même les mécanismes qui vont lui permettre d'associer les valeurs affichées à l'écran aux propriétés des composants métier et les événements clients aux méthodes des composants.

Les outils orientés vers le développement d'architectures multi-tiers à base de composants devront donc proposer des assistants graphiques où il suffira de choisir dans une fenêtre l'un des composants métier disponibles et de le glisser sur un formulaire pour que les propriétés s'affichent sous forme d'Edit Box avec masque de saisie, et les méthodes, sous forme de boutons par exemple.

On est encore loin de ce niveau d'intégration des outils entre le tiers client et le tiers serveur.

■ Débogage

De la même façon que pour les autres aspects que nous venons d'aborder, les outils de développement ont bénéficié de progrès considérables dans les possibilités de débogage… des applications 2 tiers. De plus en plus d'éditeurs d'environnements de développement proposent de fait des fonctionnalités extrêmement évoluées, avec observation des instances d'objet, modification des valeurs à la volée… De plus en plus couramment, ces produits proposent également des outils de type "remote debugger". Utiles et performants, ces modules restent limités. En effet, ils sont incapables d'assurer le débogage sur l'ensemble de la chaîne applicative, notamment lorsque l'interface utilisateur est développée dans une technologie différente (typiquement une interface Web).

En résumé le débogage unitaire des composants ne pose désormais aucune difficulté. A l'inverse, le débogage des composants en situation d'exploitation reste particulièrement délicat et passe d'avantage par des astuces que par l'utilisation d'outils académiques. Pour remédier à cette limitation, il convient de mettre en place dans les entreprises des stratégies d'intégration adaptées et clairement définies sur la base de processus itératifs : tests unitaires, tests d'intégration pour chacun des tiers, tests de l'interfaçage entre les tiers…

L'avenir est donc aux outils de débogage qui rendront transparente la rupture physique entre l'interface cliente et les composants métier.

Conclusion

Les architectures à base de composants sont devenues une réalité mais les outils de développement restent perfectibles. Cette évolution se fera nécessairement et naturellement, à condition que les technologies se stabilisent.

On peut raisonnablement prédire que les éditeurs qui sauront s'imposer sur ce créneau sont ceux qui, les premiers se montreront capables d'intégrer dans un seul atelier de développement des outils distincts qui vont de la conception jusqu'au déploiement en passant par le développement des tiers serveurs et clients, le tout avec une interface cohérente et homogène.

De la même façon, les entreprises qui capitaliseront le plus rapidement sur le développement de composants et donc rentabiliseront le ticket d'entrée sont celles qui sauront intégrer les évolutions en terme d'organisation qu'implique ce modèle d'architecture.

Le paragraphe précédent évoquait les aspects “amont” d’une application, c’est-à-dire sa conception et son développement. Cette partie effectue ici un zoom sur les aspects “aval”, c’est-à-dire la capacité de mettre en production de telles applications.

Comme pour toute nouvelle technologie, “l’administrabilité” et “l’exploitabilité” des produits ne sont pas toujours au rendez-vous… L’objectif de ce paragraphe est de définir les besoins et de décrire globalement les solutions proposées par les éditeurs.

Administration

Listons les services nécessaires à l’administration d’une application à base de composants :

- **Packaging et Distribution** des trois tiers : présentation (pages, classes Java, etc.), composants, troisième tiers de données ou de traitements (legacy).

  L’objectif est de pouvoir, à partir d’une console centrale, mettre en production la version N de l’application, constituée d’un ensemble de code qui va de la présentation jusqu’aux données ou traitements.

  Mais classiquement, les outils ne prennent en charge que leur partie, c’est-à-dire les premiers et deuxième tiers. A charge pour l’exploitation de prévoir la mise à jour des tables relationnelles, des traitements existants, ou de tout autre système lié à la version mise en production.

- **Inventaire et gestion de versions** : la programmation par composants est censée augmenter la réutilisabilité, cependant encore faut-il que ceux-ci soient disponibles dans un référentiel accessible depuis les outils de développement.

  Aujourd’hui pourtant, aucun serveur d’applications ne propose un réelle fonction de ce type. L’utilisateur un tant soit peu avisé est ainsi obligé d’utiliser des produits de gestion de version du marché, tels PVCS, SourceSafe ou Continuus, et de les intégrer aux outils de distribution du serveur d’applications.

- **Configuration et Optimisation** : load balancing, fail over

  Généralement un point fort, les serveurs d’applications proposent tous en standard la gestion de clusters applicatifs, répartissant la charge et accroissant la robustesse.

Exploitation

Côté exploitation, outre les tâches classiques de surveillance, d’ordonnancement, ou de sauvegardes, orientées système, l’ajout d’un serveur de composants ne doit pas perturber la maîtrise de bout en bout du système.

Une application 3-tiers fonctionne si les trois tiers fonctionnent. Dit de manière plus réaliste, cela signifie surtout qu’elle ne marche pas dès qu’un tiers ne fonctionne plus. Le besoin de l’administrateur est donc de disposer d’une **surveillance de bout en bout**.

Cette surveillance devra prendre en compte les aspects relatifs à chacun des tiers : Postes de Travail et serveurs Web, moteurs de composants, serveurs de données et autres canaux : MOM, ERP, Legacy, etc.

Comme pour l'administration, aucun outil ne couvre le spectre total, bien entendu. Ce que l'on est en droit d'attendre en revanche, c'est la capacité de s'interfacer avec les principaux frameworks d'administration du marché que sont CA-TNG et Tivoli, dans une logique Manager/Agent. Ces progiciels se chargeant, grâce à une consolidation des remontées d'événements, de la surveillance globale du système, de l'intégration dans le système de gestion d'incidents, dans le capacity planning, etc.

Comme le détaille le chapitre concernant l'état de l'art, la maturité technologique des solutions actuelles n'offre qu'une intégration faible (au travers de SNMP) voire nulle avec ces outils.

Conclusion

Les besoins sont clairement identifiés chez les éditeurs, qui savent que "l'administrabilité" de leur solution est une des clés du succès.

Cependant force est de constater qu'aujourd'hui certains points restent encore à améliorer :

- Les outils manquent de maturité, ce qui est normal vu la nouveauté des produits, mais des fonctionnalités telles que la présence d'un référentiel de gestion de configuration laissent à penser que le chemin est encore long avant de disposer d'outils de qualité globalement intégrés,
- L'intégration avec les frameworks du marché est faible, voire nulle, même pour la partie exploitation/surveillance.

L'administration et l'exploitation ne sont donc pas des points forts des offres actuelles. En revanche, certains éditeurs mettent un accent particulier sur l'offre des services de qualité dans leur futures versions de produits.

Sécurité

La sécurité apparaît souvent comme le "parent pauvre" des architectures techniques. La complexité inhérente à ses mécanismes ainsi que les difficultés d'administration font qu'elle est souvent mal maîtrisée dans les environnements de production.

Or les serveurs d'applications apparaissent de plus en plus comme le point d'entrée unique au Système d'Information au travers des composants : accès aux données (SGBDR, fichiers, etc.) et aux traitements (legacy, ERP, etc.). La sécurisation de l'architecture est donc un aspect vital dans la mise en œuvre de ces technologies.

Quels mécanismes et outils apportent les serveurs d'applications dans ce domaine ? C'est la question à laquelle tente de répondre ce chapitre.

Authentification

Tout commence bien entendu par la capacité du système à déterminer de manière sûre l'identité de l'utilisateur. Pour cela, tous les mécanismes sont bons, du mot de passe en clair, au certificat X.509, en passant par les calculettes à mot de passe unique ou autres.

Dans un système 3-tiers, c'est généralement le serveur de composants qui va authentifier l'utilisateur, et ce avec ses propres mécanismes. Typiquement pour la plupart des serveurs actuels, il s'agit de mécanismes propriétaires internes. Inconvénient : on ajoute un nouveau référentiel utilisateurs à administrer.
C'est pourquoi certaines offres se dégagent en proposant une intégration avec les deux standards (respectivement de facto et de jure...) que sont l'annuaire des domaines NT et l'annuaire LDAP[10].

On privilégiera donc ces serveurs, capables d'exploiter des données d'authentification externes, qu'elles soient sous forme de username/password, identifiants SecurID, ou certificats X.509.

Contrôle d'accès

Ici, le problème se corse, car actuellement, la gestion des droits d'accès est typiquement quelque chose qui relève du SGBDR dans le cas d'accès aux données (contrôle d'accès sur les tables), ou des systèmes transactionnels (CICS, Tuxedo, ERP, autres) lorsqu'il s'agit de traitements. Or lorsqu'on place un moniteur entre ces sources de données ou traitements, deux cas sont possibles :

- soit le moniteur propage l'information de login, et on a alors la possibilité de gérer la sécurité au niveau du serveur de composants et au niveau du troisième tiers,
- soit le moniteur ne propage pas l'information de login (c'est le cas pour les accès SGBDR pour des questions de performance), et la sécurité doit être gérée au niveau du moniteur.

D'expérience, il est clair que sécurité rime avec simplicité. Il est donc déconseillé de gérer la sécurité à deux endroits différents. Compte tenu des contraintes évoquées plus haut, le seul choix possible consiste donc à gérer la sécurité d'accès au niveau des composants.

Il faudra veiller bien entendu à ce que les données ne soient accédées qu'au travers de ceux-ci, grâce à des mécanismes de firewall ou de gestion avancée des connexions au niveau du SGBDR.

10 NT 2000 devrait permettre une interopérabilité entre les deux solutions.

Les outils du marché dans ce domaine proposent :

- une **gestion déclarative** (hors du code), dont la granularité est généralement le composant (et pas chacune de ses méthodes),

  Cette gestion est généralement insuffisante car elle ne permet pas, par exemple, d'autoriser le groupe A à utiliser toutes les méthodes et le groupe B uniquement les méthodes de type read(). La norme EJB impose une granularité de niveau méthode mais n'est pas toujours implémentée. MTS ne gère le contrôle d'accès qu'au niveau du composant.

- une **gestion programmatique** (par utilisation d'API dans le code), dont la granularité sera plus fine : la méthode, voire sur ses paramètres.

  Exemple : j'autorise le groupe A à invoquer Operation.valider() pour des opérations d'un montant inférieur à XX par exemple.

Ces mécanismes permettent de gérer finement la sécurité d'accès, cependant l'écueil classique consiste à "polluer" le code applicatif de code de sécurité utilisant les API propriétaires ou en cours de standardisation (EJB) des moteurs. Un palliatif consiste à encapsuler ces appels dans un service technique de sécurité spécifique.

Intégrité/Confidentialité

On en rêvait, il est enfin possible de sécuriser totalement et simplement le trafic entre client et serveur. En effet, le marché a globalement adopté le standard de l'Internet SSL, que ce soit dans les dialogues HTTP ou IIOP. Il est donc possible de mettre en œuvre ce mécanisme, soit en utilisant uniquement un certificat pour le serveur, soit en généralisant à chaque utilisateur, lorsque cela est possible.
Le protocole SSL ne nécessite qu'un certificat serveur pour sécuriser (crypter et signer) une communication.

Par décision ministérielle, l'emploi des mécanismes cryptographiques, autrefois sévèrement réglementé, est aujourd'hui libéralisé. C'est-à-dire qu'il est possible d'utiliser des clés DES ou RC4 de 128 bits (40 bits avant la loi, soit 288 fois plus vulnérables), ce qui correspond à une protection extrêmement forte (utilisable sur Internet).

Conclusion

La sécurité est globalement prise en compte dans l'offre du marché. Un certain nombre de points apparaissent différenciateurs entre les produits :

- Capacité de centralisation des informations de sécurité dans un référentiel externe,

  Interfaçage avec l'annuaire NT, l'annuaire LDAP Netscape, etc.

- Capacité d'administration de la sécurité des accès, au travers d'outils visuels.

  Positionnement des droits d'accès, capacité d'audit de ces droits.

- Support des protocoles de sécurité de l'Internet, les seuls pérennes : SSL, X.509.

Il est important de noter que la problématique de sécurité a été intégrée dès le départ dans ces produits, ce qui permet de bénéficier tout de suite des dernières technologies de sécurité, sans nécessiter d'intégration particulière avec des produits tiers, comme c'est le cas dans la plupart des offres moins récentes (SGBDR, MOM, etc.).

L'état de l'art

On présente ici les offres majeures des acteurs importants du marché, soit (par ordre alphabétique) :

- L'offre BEA : M3 et WebLogic
- L'offre IBM : WebSphere
- L'offre Inprise : Inprise Application Server
- L'offre Microsoft : Distributed Network Architecture (DNA)
- L'offre Netscape : Netscape Application Server
- L'offre Oracle : Oracle Application Server
- L'offre Sun : NetDynamics
- Les offres d'éditeurs moins "institutionnels" : PowerTier, Gemstone, Secant.

Elles sont présentées indépendamment des systèmes d'exploitation, la portabilité de l'offre variant considérablement d'un éditeur à l'autre, de Windows NT pour Microsoft, à l'ensemble des plates-formes de la gamme pour IBM. La stratégie de choix d'un serveur d'applications pouvant être liée à la stratégie de l'entreprise en terme de système d'exploitation, le lecteur orientera sa lecture selon son gré…

BEA

Historique

Créée en 1995 par Bill, Edward et Alfred, BEA Systems a toujours eu pour ambition de devenir le leader dans le monde du middleware. Après le rachat du moniteur transactionnel Tuxedo à Novell (produit leader dans le petit monde des moniteurs TP sous UNIX), BEA a saisi l'opportunité d'achat d'Objectbroker de DEC. L'objectif de cette acquisition était clair, combiner ORB et technologies transactionnelles. Ce projet, connu sous le nom de code Iceberg, vit le jour sous le nom de M3 (pour Middleware troisième génération). L'année dernière aura vue le rachat de Weblogic (l'éditeur de Tengah), qui vient compléter l'offre de l'éditeur, à défaut de la clarifier.

Brief technique

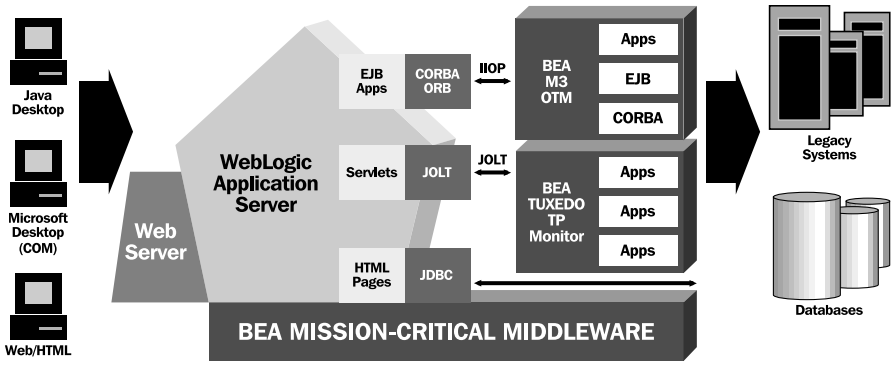


Figure 11 : Offre marketing[11] BEA

Bien que BEA Tuxedo, complété par BEA Jolt, permette de concevoir ce que l'on pourrait appeler un serveur d'applications, ce document se consacre aux serveurs d'applications orientés composants. Nous limiterons donc la présentation aux offres M3 et Weblogic Application Server.

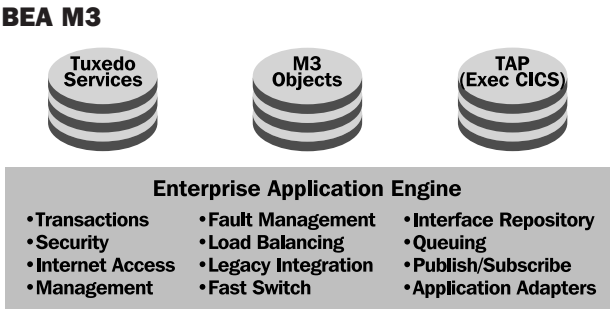


Figure 12 : Architecture des produits BEA

L'architecture proposée par BEA se décompose en deux parties :

- Une partie commune à l'ensemble des offres de l'éditeur, BEA Enterprise Application Engine, remplissant le rôle de socle de services communs à l'offre, intégrant les fonctions transactionnelles, de sécurité, d'administration, d'équilibrage de charge,... et les passerelles. Ce socle commun assure la portabilité et l'interopérabilité de l'offre.

11 L'intégration et le positionnement étant un peu plus complexe.

- La partie M3 (Services Tuxedo, Objets M3 et TAP) à proprement parler qui s'appuie sur ce socle.

Grâce à cette Architecture, les applications M3 peuvent appeler des services Tuxedo. En pratique, la migration d'applications Tuxedo vers M3 sous forme binaire est possible sans modifications de code[12].

De nouvelles applications M3 CORBA ont la possibilité d'inclure des appels natifs Tuxedo pour s'intégrer avec des applications Tuxedo existantes. A l'opposé, les applications Tuxedo existantes peuvent invoquer les services d'objets M3 à travers les interfaces CORBA IDL.

Les applications M3 peuvent également, sous couvert de la présence de Tuxedo, appeler des transactions CICS ou IMS par l'intermédiaire de la passerelle BEA Connect et envoyer des messages par l'intermédiaire de BEA MessageQ.

M3 assure les fonctionnalités attendues d'un OTM, à savoir la gestion de l'instanciation des objets et des ressources consommées, ainsi que l'optimisation de la gestion des connexions aux différents "resource managers" utilisés (via le support de XA).

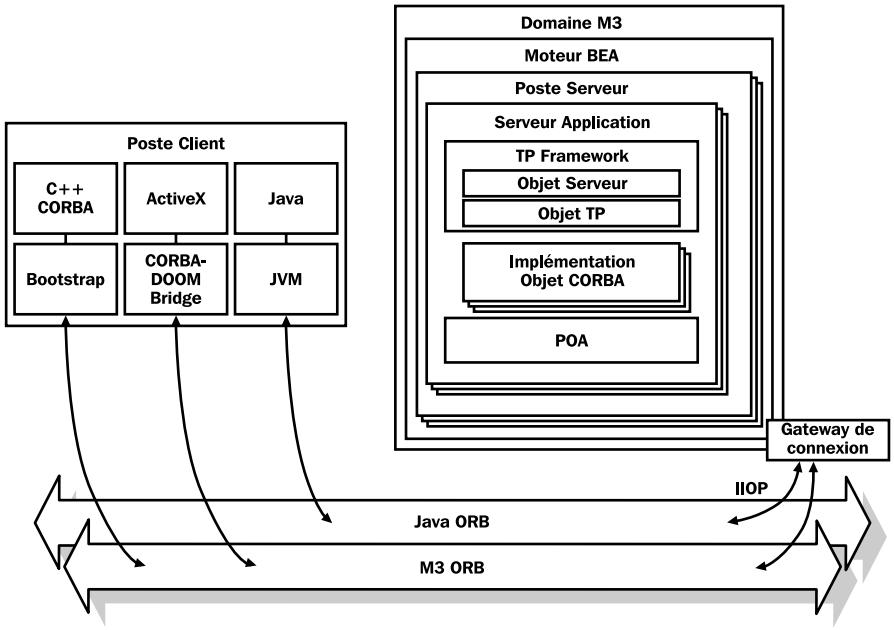


Figure 13 : Architecture M3

Plusieurs types de clients peuvent se connecter à BEA M3 :

- Les clients CORBA Java ou C++,
- Les clients ActiveX via BEA Desktop Connection for ActiveX client.

BEA M3 apporte également un ensemble d'objets permettant de faciliter le développement des clients vers des objets BEA M3.

On trouve ainsi l'objet Bootstrap qui retourne au client des références vers l'objet BEA M3 principal, ainsi que vers les objets de gestion des transactions, de la sécurité, ou vers le "FactoryFinder".

Les objets s'exécutent dans des processus serveur BEA M3.

La gestion du cycle de vie de ces objets est assurée par le TP Framework et de POA (Portable Object Adapter) de BEA M3.

12 On ne change pas de paradigme pour autant.

- Le **TP Framework** de M3 est un acteur primordial dans la vie d'un objet M3. Il permet d'activer et de désactiver les objets, gérer les opérations de validation (commit) lors des transactions, notifier aux objets les événements système et enfin lancer les objets. Pour ce faire le TP Monitor s'appuie sur le POA.
- Le **POA** (Portable Objet Adapter) : il s'agit de l'évolution du BOA (Basic Object Adapter) que l'on trouvait dans les spécifications de CORBA 2. Les objets CORBA utilisent son interface pour créer ou détruire des instances, leur transmettre des requêtes, et récupérer les résultats. Au sein de BEA M3, c'est le TP framework qui utilise ses fonctionnalités pour gérer le cycle de vie des objets M3.

Les serveurs M3 sont disponibles sur Windows NT et la plupart des environnements UNIX. Les composants sont écrits en C++ dans la version actuelle du produit (Version 2 .2), en Java dans la prochaine version (3.0) annoncée pour l'été 99.

Le support des EJB ne devrait pas être disponible avant la fin 99.

■ Développement

Le développement d'une application M3 consiste, d'une part, à développer un ensemble d'objets CORBA en utilisant l'IDL et un outil de développement non fourni par l 'éditeur, et, d'autre part, à développer une interface utilisateur avec l'outil de votre choix. Le seul langage supporté à ce jour pour ces composants est malheureusement le C++ .
Comme avec Tuxedo, l'intégration de M3 et des environnements de développement constitue indéniablement un point faible lié à la stratégie de ce spécialiste du middleware. En particulier au regard des services apportés par les environnements de développement visuels intégrés fournis maintenant par la concurrence.

A partir d'un IDL (qui peut être généré par un outil de conception), M3 peut générer le stub et le squelette du composant serveur. Suite à des accords récents, Rational Rose permet la génération directe de code pour M3.

Il faut faire un choix d'utilisation de M3, à savoir utiliser M3 comme un ORB standard ou intégrer les fonctionnalités du TP Framework.

Administration et exploitation

M3 propose des mécanismes de tolérance de pannes et de recovery permettant de protéger les applications des défaillances éventuelles du client, du réseau et du serveur. M3 dispose d'une MIB mettant à disposition des outils d'administration les informations concernant l'état des objets et des applications.

L'administration des objets se fait via une console d'administration écrite en Java, permettant d'administrer les objets, de monitorer et de tuner les performances des applications dynamiquement.

■ Sécurité

M3 implémente les services de sécurité de CORBA (niveau 2 des services de sécurité CORBA) pour l'authentification et l'encryption des mots de passe. De plus, M3 supporte l'encryption RSA des messages inter-serveurs.
Pour le moment, il n'y a pas d'encryption des messages en provenance des clients (cela sera implémenté quand la spécification OMG IIOP sur la sécurité sera disponible).

■ BEA WebLogic Application Server

WebLogic Application Server (WLAS, ex-Tengah) a acquis ses galons en devenant le premier serveur d'applications 100% Java supportant complétement les EJB. En effet, depuis la version 3.1, il implémente la spécification 1.0 des EJBs et notamment les extensions optionnelles suivantes :

- Le Bean Entité, dont la persistance est gérée soit par le Bean lui-même, soit automatiquement par le container (sauvegarde du Bean dans des fichiers ou des RDBMS),
- Les transactions distribuées (implémentation de JTA et JTS) gérée par le container ou par le Bean.

WLAS supporte pratiquement toutes les API Sun JPE (Java Platform for the Enterprise) : EJB, JNDI, RMI, Java IDL, Servlets et JSP, JTA, JTS, JDBC et bientôt JMS[13] (pour le moment WebLogic utilise son propre système d'événements, qui permet l'abonnement par des clients à des événements qui seront envoyés par le serveur).
En outre, WLAS gère l'intégration de composants Microsoft COM dans son framework en générant automatiquement des classes Java d'interface (Cela peut permettre, par exemple, à des utilisateurs sous Solaris d'exécuter des applications Windows depuis leur station de travail).

WLAS intègre des drivers JDBC de type 2, 3 et 4 :

- Type 2 : drivers (écrits en partie en Java et en C++) utilisant les API natives des DBMS. Utilisés pour des connexions 2-tiers à la base de données. Les drivers sont fournis pour Oracle, Sybase et SQL Server,
- Type 3 : drivers (écrits en Java) qui convertissent les appels JDBC en un format indépendant des DBMS. Ce format est ensuite converti à son tour par le serveur en un format spécifique DBMS. propriétaire WebLogic. Ce type de drivers se trouvent sur la machine cliente, dans une architecture 3-tiers,
- Type 4 : drivers (écrits en Java) qui convertissent les appels JDBC au format des protocoles réseaux natifs des DBMS. Cela permet, comme le type 3, des appels directs aux bases de données de la part des applications clientes. Les drivers fournis sont pour Informix et SQL Server.

WLAS est écrit en Java et à ce titre est un peu moins rapide que ses homologues écrits en C, C++,... Cependant, de nombreuses fonctionnalités ont été mises en œuvre pour accélérer les traitements et la montée en charge :

- Les requêtes en provenance des clients utilisent une seule connexion au serveur. WLAS utilise un mécanisme de queues pour gérer les paquets d'informations (une queue d'entrée et une queue de sortie). Cela lui confère des avantages des MOMs : persistance des messages (un client peut se déconnecter et en se reconnectant, il retrouvera son état antérieur),...
- Des drivers en mode natif (voir ci-dessus),
- Gestion de "pools" de connexions aux bases de données,
- WLAS peut mettre en cache les requêtes SGBD les plus fréquemment utilisées (à condition que la base de données ne soit pas partagée...), ce qui évite un accès à la base de données,
- Des connexions clients/serveur persistantes : chaque client a un "espace" réservé sur le serveur. Un client, en se reconnectant, retrouve donc son état précédent (il retrouve aussi ses connexions en cache au SGBD). De plus, cet "espace" peut être utilisé pour stocker des objets (comme des DataSets) en vue d'utilisations ultérieures.

13 Voir le glossaire en fin de document.

- Répartition automatique de la charge (en utilisant un mécanisme dynamique de déplacement des composants applicatifs) et mise en “cluster” de serveurs (répartition de la charge au niveau des requêtes par réplication des états des sessions sur les différents serveurs)

WLAS est disponible sur les plates-formes NT, Unix et AS/400.

■ Développement

Pour le programmeur, WLAS se présente comme un ensemble de classes Java et de JavaBeans. WLAS n’a pas d’IDE propre mais a été prévu afin de pouvoir fonctionner avec tous les environnements de développement standards du marché : Visual Café, JBuilder, PowerJ, VisualAge for Java, Supercede et VisualJ++.

Dans ces IDEs, le programmeur utilise les Beans WebLogic pour construire une application WLAS.

■ Administration et exploitation

WLAS intègre une console graphique d’administration sous la forme d’une applet ou d’une application Java. Toute l’administration se fait à partir de cette console très riche en fonctionnalités.

Depuis la version 3.1, WLAS supporte la distribution automatique d’applets et d’applications vers les clients au travers de l’outil ZAC (Zero Administration Client).

■ Sécurité

WLAS gère l’authentification à travers des certificats X.509 et l’encryption à travers SSL, HTTPS et des listes de contrôle d’accès (ACL).

De plus, l’utilisation de pools de connexions DBMS masque, pour les applications clientes, le login et mot de passe pour se connecter à ces bases.

Positionnement / Stratégie

Le positionnement de BEA sur le créneau exclusif du middleware, fait de ce spécialiste un des acteurs majeurs sur ce créneau. Les produits qu’il a acquis ont été choisis parmi les meilleurs disponibles.
La réputation concernant les performances et la stabilité de ses produits n’est plus à faire.

Il reste à BEA à clarifier son offre et les recouvrements des différents produits.

BEA profite par le biais de ses options de rachat[14] et de sa stratégie commerciale d’une base client considérable lui garantissant une source de revenus qui devrait lui permettre d’améliorer son offre ou de procéder à d’autres acquisitions.

Néanmoins, le marché tire vers des solutions intégrant middleware et environnement de développement permettant la mise en place rapide de nouvelles applications.

On se prend alors à rêver d’une acquisition d’un environnement de développement permettant à BEA de construire et de proposer une solution globale, sans dépendre de la volonté des autres éditeurs d’intégrer ses solutions.

14 N’oublions pas d’inclure le rachat de TOP END à NCR en 1998

Références

■ M3

- En suisse : Banque Sarazin
- En Nouvelle Zélande : Dept of Wellfare avec 7000 clients

D'autre part, beaucoup de grands comptes évaluent M3, certains l'ont déjà choisi et ne communiqueront qu'en phase de déploiement.

■ WebLogic

WebLogic compte plus de 800 clients dans le monde entier. Cependant, sur l'ensemble de ces clients, seule une minorité utilise WLAS (la majorité utilise les produits jdbcKona). On peut citer, entre autres : AT&T Labs, Cloudscape, Xerox, Extensity, Moai Technologies, Net Explorer, ZenaComp, American President Lines, Corporate Express,...

Conclusion

■ Avantages (M3)

- Performance et robustesse,
- Intégration avec l'offre BEA (Passerelles).

■ Inconvénients (M3)

- Développement non intégré,
- Pas de support de JAVA et retard sur la mise en place des EJB,
- Absence d'ouverture vers LDAP.

■ Avantages (WLAS)

- Support 100% des EJBs 1.0 et des API JPE,
- Drivers JDBC natifs,
- Déjà utilisé en production.

■ Inconvénients (WLAS)

- WLAS étant écrit en Java, ses performances dépendent du JVM.

Historique

En mars 1997, IBM renforce ses outils de création de sites Web en prenant une part majoritaire dans la société NetObjects.

Au début de l'année 1998, IBM, NetObjects et Apache Group s'associent autour du projet WebSphere d'IBM : Apache Group fournit son serveur Web, IBM l'intègre dans son offre WebSphere et NetObjects adapte ses outils de création de sites Web à WebSphere.

La concrétisation de ce projet verra le jour en juin 1998 avec la sortie de la version "Standard" 1.0 de WebSphere Application Server (WAS). La version "Advanced" (à partir de la version 2.0), supportant le modèle de composants EJB est sortie en décembre 1998.

Brief technique

WAS existe en 3 versions (Standard, Advanced et Enterprise) qui correspondent à différents niveaux de besoins, de la publication de pages Web à la gestion de sites de commerce électronique :

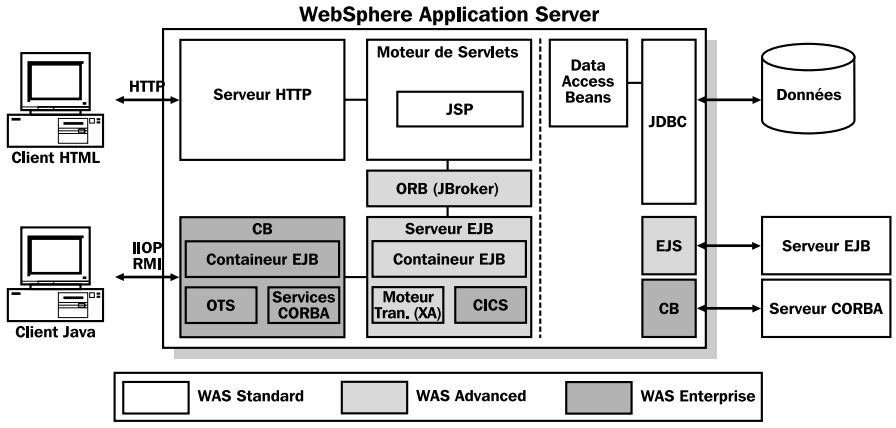


Figure 14 : Gamme WebSphere

Les différents composants de l'architecture WebSphere sont :

- Serveur HTTP : WAS est livré en standard avec le serveur Web Apache mais il supporte aussi les serveurs Web suivants : Domino Go Webserver, Microsoft IIS et Netscape Webserver.
- Moteur de Servlets : WAS implémente la version 2.0 de Sun JDSK et supporte la spécification 0.91 des Java Server Pages (JSP),
- Data Access Beans : ce sont des classes Java qui facilitent l'accès aux bases de données et en améliorent les performances (gestion de pools de connexions). Ces Beans utilisent JDBC.
- Serveur EJB : WAS est un serveur d'applications java qui supporte les spécifications EJB 1.0 (dans la version "Advanced") y compris les caractéristiques optionnelles suivantes :
  - Gestion de la persistence des Beans Entités par le Bean lui-même ou par le container,
  - Gestion du mode transactionnel (implémentation des API JTS/JTA). La

version “Advanced” de WAS comprend un moteur transactionnel supportant le protocole XA. Ce moteur supporte le protocole XA. A partir de la version “Enterprise”, l’EJS utilisera TxSeries comme moteur transactionnel.

- Intégration de Component Broker (CB) dans la version “Enterprise”. Un container EJB sera aussi intégré à CB, ce qui permettra aux EJBs déployés dans ce container d’utiliser les services CORBA de CB ainsi que l’OTS de CB. En pratique, l’intérêt de CB dans WAS est faible : les utilisateurs actuels (il n’y en a pas beaucoup) de CB peuvent continuer à utiliser leurs applications CORBA sous WAS.
- Bases de données : WAS utilise les drivers JDBC fournis par les éditeurs des bases de données (aucun driver n’est fourni avec WAS).
- WAS existe sous NT, AIX, Solaris, OS/400, OS/2 et OS/390.

■ Connecteurs et modules externes

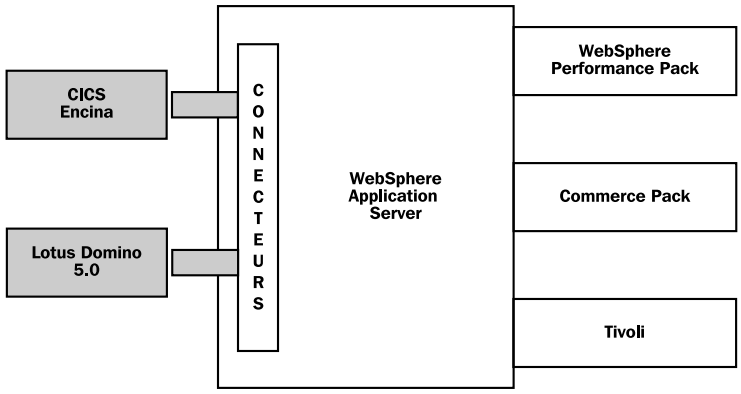


Figure 15 : Connecteurs et modules WebSphere

WAS offre des connecteurs qui permettent aux applications WebSphere de communiquer avec les applications existantes IBM (“legacy applications”) : CICS, Encina et Lotus Domino 5.0.

De plus, 3 modules viennent compléter l’offre WebSphere :

- “WebSphere Performance Pack” : gère de manière hardware (duplication de serveurs physiques) une répartition dynamique de la charge et la mise en clusters.
- “Commerce Pack” : Gestion d’un catalogue de produits, paiement sécurisé, “suggestion” d’un produit et gestion des promotions.
- Intégration avec Tivoli : IBM dispose de modules Tivoli pour WebSphere qui permettent de surveiller des serveurs WAS (surveillance de la charge, des objets EJBs, des threads, ...), d’effectuer des opérations simples (arrêt, redémarrage). Cependant, il manque encore la gestion du déploiement d’applications WebSphere (seul le déploiement d’EJB est prévu).

■ Développement

WebSphere fournit une suite complète d’outils de développement (à l’exception d’un outil de conception), de la création de pages Web à la création d’EJBs :

- NetObjects Fusion, NetObjects BeanBuilder et NetObjects ScriptBuilder : Création et gestion de sites Web
- WebSphere Studio : gestion de projets WebSphere et assistants de création de pages JSP/Beans d'accès aux données
- VisualAge for Java : création de Servlet et de (Enterprise) JavaBeans. Assistants pour la création et déploiement des EJBs,
- WAS : classes java pour gérer les profils utilisateurs, l'accès aux données (pools de connexions, ...)

IBM préconise d'utiliser le modèle conceptuel de programmation MVC (Model/View/Controller). Dans le modèle WebSphere cela se traduit de la manière suivante (voir schéma ci-dessous) :

- Le Modèle ("Model") qui correspond aux objets métier est implémenté en utilisant les EJBs,
- La Vue ("View") qui correspond aux pages HTML retournées aux clients est implémentée en utilisant les pages JSP. Ces pages JSP utilisent les EJBs pour récupérer les données métier.
- Le Contrôleur ("Controller") qui correspond à la logique applicative nécessaire pour naviguer de page en page sur un site Web par exemple est implémentée en utilisant les Servlets. Ainsi le Servlet choisira, en fonction de critères applicatifs, la page JSP à retourner à l'utilisateur.

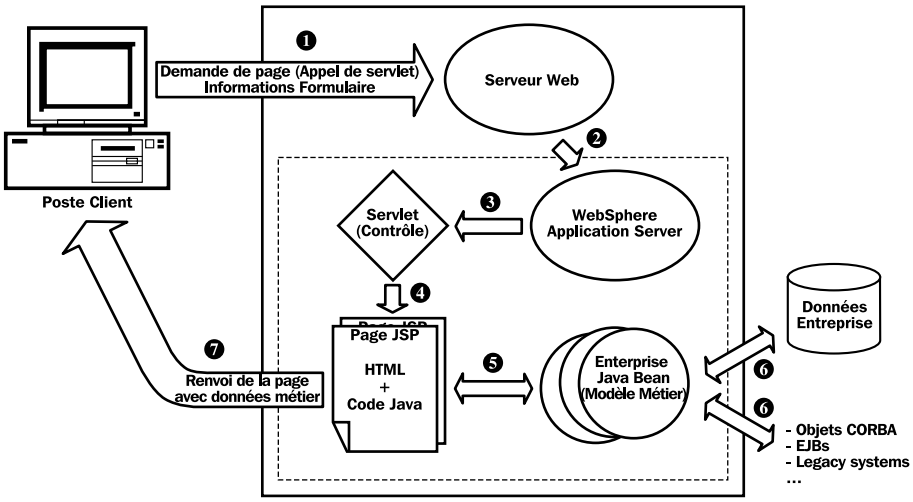


Figure 16 : Cinématique d'une transaction WebSphere

La cinématique est la suivante :

❶ L'utilisateur saisit des champs dans un formulaire et clique sur le bouton de soumission. Cette requête est envoyée au serveur Web (Apache),

❷ Le serveur Web transmet la requête au serveur d'applications WebSphere,

❸ WebSphere appelle la Servlet correspondant à la requête (une Servlet est un programme java qui joue le rôle d'aiguilleur : c'est lui qui va choisir la page qu'il faudra renvoyer à l'utilisateur),

❹ La Servlet choisit la page qu'il faut créer en fonction des données entrées par l'utilisateur et de la page sur laquelle se trouvait l'utilisateur ; il joue le rôle de contrôleur. Les pages créées sont des pages JSP (elles contiennent à la fois du code HTML, du code java et des tags spéciaux pour appeler des JavaBeans),

❺ Les pages JSP appellent des JavaBeans lorsqu'elles ont besoin de récupérer des informations métier. Ces informations métier proviennent de la base de données,

❻ Les JavaBeans sont des composants java. Leur rôle est de récupérer des informations de la base de données et de les traiter. Ces informations formatées sont alors disponibles par les pages JSP pour être incluses dans les pages finales au format HTML renvoyées à l'utilisateur,

❼ Les pages au format HTML sont renvoyées à l'utilisateur.

■ Administration et exploitation

WAS dispose d'une console d'administration sous forme d'applet Java. Elle permet de fixer les paramètres du serveur d'applications, de surveiller/paramétrer les objets EJBs fonctionnant sur le serveur, de surveiller la charge du système et notamment d'établir les paramètres de sécurité.

WAS n'intègre par des fonctions de répartition de charge et de mise en cluster. En revanche, IBM propose en option le produit "WebSphere Performance Pack", qui gère de manière hardware (duplication de serveurs physiques) une répartition dynamique de la charge et la mise en cluster.
L'administration d'un parc de serveurs WAS s'effectue avec les modules Tivoli pour WebSphere.

■ Sécurité

WebSphere gère plusieurs niveaux de sécurité :

- Sécurité au niveau des utilisateurs pour l'accès aux pages Web et aux Servlets,
- Rassemblement des utilisateurs en groupes d'utilisateurs partageant les mêmes droits d'accès,
- Définition des droits d'accès pour les utilisateurs et les groupes en utilisant des listes de contrôle d'accès (ACL),
- Restriction de l'accès aux ressources en spécifiant par exemple des droits d'accès sur des fichiers et répertoires.

Positionnement / Stratégie

WebSphere Application Server se démarque des autres serveurs d'applications par deux points principaux :

- En fournissant une offre importante d'outils de développement dont l'IDE "VisualAge for Java" qui a été intégré.
- En proposant une offre multi-plates-formes, et notamment disponible sur mainframe (OS/390, AS/400). La scalabilité de l'offre se résume donc en deux points : scalabilité matérielle de NT à OS/390 et scalabilité en répartissant la charge sur plusieurs serveurs (WebSphere Performance Pack).

IBM poursuit deux stratégies :

- Offrir une suite d'outils de développements intégrés. Cette stratégie s'est concrétisée par la prise d'une participation dans la société NetObjects, ce qui a permis à IBM d'offrir une panoplie presque complète d'outils. Malheureusement, ces outils ne sont pas ou mal intégrés entre eux.

Cependant, il semblerait qu'IBM veuille combler cette lacune en annonçant l'intégration du produit Build-IT de Wallop Software dans WebSphere.

- Une implication active pour le langage Java et les technologies associées : participation à l'élaboration des spécifications et implémentations. Ainsi, IBM revoit sa gamme de produits afin que ceux-ci supportent les API Java. Pour l'utilisateur, cela signifie une pérennité de la solution qu'il met en œuvre.

■ Perspectives de l'offre

La version "Enterprise" (à partir de la version 3.0) de WAS est annoncée pour juin 1999 et renforcera le support du mode transactionnel en incluant le produit TXSeries dans WebSphere. Enfin, la version 4.0 complète de WebSphere, est prévue pour la fin de l'année 1999. Elle n'offrira pas de nouvelles fonctionnalités essentielles mais apportera un meilleur "packaging" des outils fournis, une meilleure intégration des technologies utilisées et l'apparition de CICS comme moteur transactionnel pour les EJBs (sous mainframe uniquement).
Afin de compléter son offre d'outils de développement autour de WebSphere, IBM achète en octobre 1998, le produit Build-IT de Wallop Software. Ce produit d'intégration d'outils de développement tiers sera intégré en 1999 aux outils de développement de WebSphere. En parallèle, IBM a annoncé en septembre 1998, le support par WebSphere de "San Francisco", un ensemble de composants métier génériques, basés sur les EJBs.

Références

Sur WebSphere proprement dit (associé à VisualAge for Java), on trouve les références américaines CARREER.COM, COMDATA, CHARLES SCHWAB, AMERICAN CENTURY. Actuellement en France 15 projets sont ouverts sur WebSphere.

Conclusion

■ Avantages

- Support des EJB 1.0 dans la version 2.0 "advanced"
- Intégration avec l'offre d'IBM (connecteurs)
- Nombreux outils de développement (création de sites, gestion des ressources d'un projet, environnement de programmation Java/EJB/Servlet)
- Multi-plate-forme et notamment disponible sur mainframe (AS/400 , AS/390)
- Administration centralisée avec les modules Tivoli pour WebSphere.

■ Inconvénients

- Pas d'intégration des outils de développement entre eux
- Produit jeune : pas ou très peu de sites en production
- La gestion robuste et complète du transactionnel ne sera disponible que dans la version "Enterprise" (TxSeries)
- La répartition de charge doit être gérée avec un autre serveur, non intégré à WebSphere.
- Les outils d'administration intégrés ne gèrent qu'un serveur à la fois (pas d'administration de cluster)

Inprise

Historique

Fin février 1998, Borland International et Visigenic Software Inc. fusionnaient pour former Inprise Corporation. Avec Visigenic, Borland venait d'acquérir un des produits leader sur le (petit) marché des ORB (Object Request Broker) CORBA.

La stratégie d'Inprise s'est alors clairement orientée vers la création d'un environnement de développement, de déploiement et d'administration d'applications réparties pour les Systèmes d'Informations d'entreprise.

Le 2 septembre, Inprise annonce la sortie de Inprise Application Server (IAS) et présente l'ensemble des composants de sa suite logicielle.

Brief technique

Le serveur d'applications Inprise Application Server (IAS) est le reflet de la fusion réussie entre Borland et Visigenic.

On y retrouve d'une part le savoir-faire de Borland en tant qu'éditeur d'environnements de développement, à travers les interfaces graphiques des trois consoles principales du produit :

- **JBuilder for Application Server** pour le développement
- **Application Server Console** pour le déploiement
- **AppCenter** pour l'administration

D'autre part, l'ancienne offre de Visigenic devient le socle de ce serveur d'applications puisque celui-ci repose sur VisiBroker, implémentant le standard de l'OMG CORBA.

Notons que VisiBroker est aussi présent dans l'offre d'un certain nombre d'autres éditeurs : Oracle, Netscape, Sun.

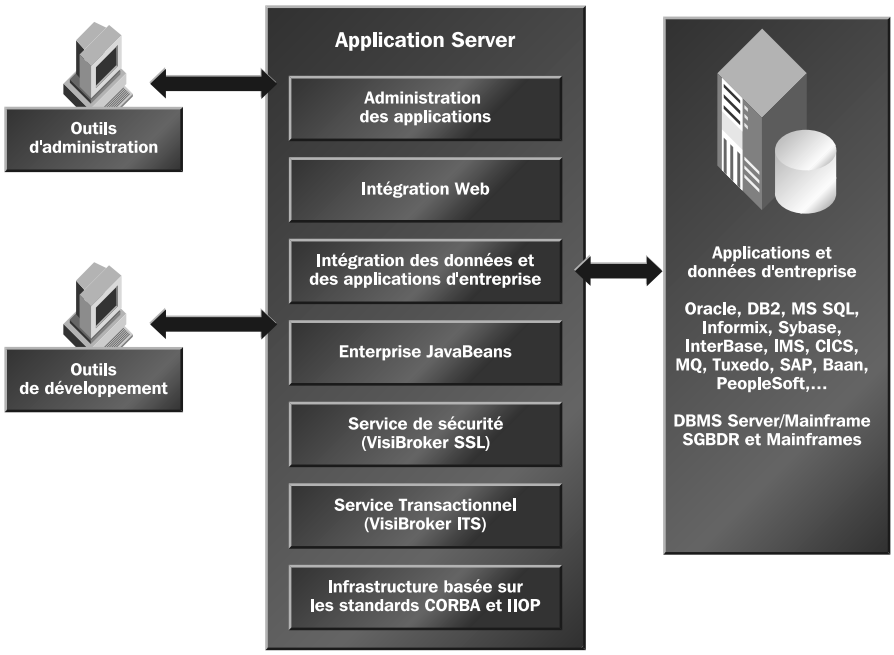


Figure 17 : Architecture technique d'Inprise Application Server

A la base de l'architecture d'Application Server, on retrouve plusieurs standards :

- **CORBA** : l'ORB VisiBroker figure aujourd'hui parmi les leaders du marché aussi bien dans sa version C++ que Java. Il possède un niveau de performances élevé, gère la tolérance aux pannes et possède une gestion relativement fine des connexions et des threads. VisiBroker s'accompagne de 2 passerelles : GateKeeper utilisée pour les communications entre objets via un serveur Web et Object Bridge pour relier les mondes CORBA et COM.
- **IIOP** : l'implémentation d'IIOP de Application Server lui permet de ne pas se restreindre au monde encore jeune des applications Java/RMI. On retrouve d'ailleurs l'implémentation IIOP d'Inprise dans les offres d'autres éditeurs du marché tels que Oracle ou Netscape.
- **EJB** : le modèle de composant EJB est supporté par Inprise Application Server. JBuilder permet, via un assistant, de créer des Enterprise JavaBeans.

Au dessus de cette base robuste, on retrouve d'autres technologies indépendamment éprouvées avant leur intégration dans IAS :

- Les services d'**annuaire et d'événement** de VisiBroker, respectivement Naming Service et Event Service.
- **La gestion des transactions : VisiBroker Integrated Transaction Service** (ITS) est conforme aux spécifications du service transactionnel de l'OMG : OTS. C'est également une implémentation du JTS (Java Transaction Service).

Le schéma suivant illustre l'architecture de ce composant essentiel d'Application Server :

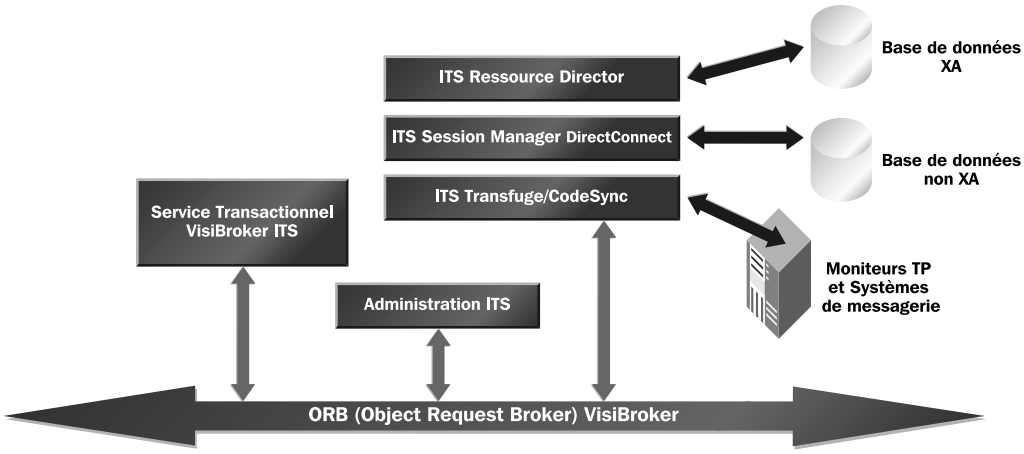


Figure 18 : Gestion des transactions

- **Le service transactionnel d'ITS** est souple d'utilisation. Il peut être déployé comme une instance partagée par plusieurs applications ou bien s'intégrer dans une application. Il est ainsi possible de trouver plusieurs instances du service transactionnel sur le réseau. ITS possède une console d'administration qui permet de piloter les transactions entre différentes sources de données dans l'entreprise.
- **L'ITS Session Manager** permet à une application d'obtenir une connexion ITS vers des sources de données XA ou non XA. Il offre la possibilité de multiplexer les connexions de manière à optimiser la consommation en ressources système.

- Pour les bases de données en environnement XA, ITS Session Manager s'appuie sur **ITS Resource Director**.
- Le module **DirectConnect** d'ITS Session Manager permet d'obtenir des connexions à des bases de données en environnement non XA.

Enfin l'accès aux moniteurs TP et systèmes de messagerie est assuré par deux modules :

- **ITS Transfuse** est une interface qui permet d'établir une connexion vers Tuxedo, CICS , IMS, et MQ Series.
- **ITS CodeSync** permet de générer le code IDL d'objets CORBA à partir de données extraites de sources COBOL ou Tuxedo.

Pour conclure sur l'aspect transactionnel de IAS, ce service permet de développer de nouvelles applications transactionnelles robustes avec le système de validation (commit) sécurisée en deux phases pour les sources hétérogènes, mais aussi d'intégrer ces applications au sein de l'existant (données et applications).

- **La gestion de la sécurité** : Inprise utilise un modèle de sécurité basé sur SSL (Secure Socket Layer) permettant d'avoir une approche sécurisée pour les transactions entre composants d'une application répartie.

Toutes ces briques techniques sont entièrement intégrées dans l'outil de développement JBuilder for Inprise Application Server.

Développement

Les nombreux assistants de **JBuiler for Application Server** accélèrent la création de clients variés :

- Applications java autonomes,
- Applets java pour navigateurs,
- Clients HTML et DHTML.

Côté serveur, JBuilder for Application Server offre également des assistants pour la création des **EJB** ou de **composants CORBA**. Même si Java est le seul langage utilisé à l'heure actuelle dans Application Server, d'autres le rejoindront prochainement : DELPHI et C++. Cependant, les serveurs développés avec JBuilder peuvent dialoguer avec des applications C++, Delphi ou Visual Basic, notamment grâce à la passerelle Object Bridge.

Inprise Application Server est livré avec un **serveur Web** propriétaire, et naturellement, JBuilder permet de créer des extensions offrant une portabilité accrue par rapport aux scripts CGI : les **Servlets**. Il est cependant important de noter qu'Application Server peut fonctionner avec d'autres serveurs Web : Apache, Netscape Enterprise Server et IIS.

Enfin, en réponse au problème de mise au point des applications distribuées qui représente un des problèmes majeurs pour les développeurs, Inprise a doté JBuilder d'un debugger graphique relativement sophistiqué.

Déploiement

Même si le déploiement et la configuration des applications distribuées s'effectuent dans un environnement graphique très visuel, le temps d'adaptation à la console de IAS n'est pas négligeable.

En revanche, une fois l'environnement maîtrisé, il s'avère efficace.

Trois vues principales sont disponibles :

- **Models** : ils permettent de définir dans un premier temps (Catalogues) les composants d'une application indépendamment les uns des autres, puis dans une seconde étape (Applications distribuées) d'assembler ces composants, et de leur attribuer des machines cibles pour le déploiement. Enfin, il est possible de packager l'application et de lancer le déploiement.
- **Applications server** : ce volet apporte une visualisation globale de l'ensemble des instances d'Application Server sur le réseau, et les applications qui s'y exécutent en temps réel, les services standards CORBA de bas niveau (Objet Activation Daemon, Location Service, ITS, agent AppCenter).
- **Services** : ce volet permet de gérer les services de haut niveau de l'environnement distribué, service de localisation, l'annuaire, le répertoire d'interfaces et d'implémentation des composants disponibles sur le réseau.

Le déploiement de certains composants avec des fonctionnalités spécifiques (transactionnel, sécurité) n'est pas trivial et certaines recommandations sont à suivre rigoureusement (présence d'une instance des services nécessaires sur les machines cibles ou le réseau,...).
A noter que la présence d'un runtime IAS sur chaque machine administrée est nécessaire.
Le déploiement en environnement de test ou d'exploitation suit les mêmes règles et se fait avec les mêmes outils, ce qui permet une collaboration efficace entre développeurs et administrateurs.

Administration

L'administration des applications distribuées se fait avec le **Viewer d'AppCenter**. En utilisant cette console, l'administrateur suit les étapes suivantes :

- Modélisation des composants de l'application,
- Définition des règles de dépendance entre ces composants,
- Définition des règles de fonctionnement de l'application,
- Définition des comportements à suivre lors d'un problème (reprise sur erreur, etc.).

La modélisation terminée, l'administrateur peut piloter l'application, à savoir :

- Lancer et arrêter chaque composant,
- Modifier la configuration des composants à distance,
- Créer des "cockpits" sur l'application afin d'observer certains critères comme la montée en charge, la communication entre composants, le nombre d'instances d'un composant,...

Le **Viewer** n'est que l'interface graphique d'AppCenter qui contient également :

- Un **Agent** qui doit être présent sur chaque machine hôte d'un composant de l'application distribuée,
- Un **Moniteur** pour chaque application administrée avec AppCenter,
- Un **Repository** central (format propriétaire) contenant les informations de configuration de chaque application,
- Un **Broker** qui correspond au service d'annuaire propre à AppCenter et qui lui permet de localiser chacun de ses composants.

Positionnement/Stratégie

Comme certains de ses concurrents, Inprise Application Server s'appuie sur les standards de l'industrie (CORBA, IIOP, EJB, JTS, SSL) et intègre des technologies déjà éprouvées indépendamment, ce qui lui confère robustesse et flexibilité.

Le seul outil de développement intégré pour les composants serveur est JBuilder, ce qui limite les développements au langage Java. Cependant Delphi et C++ for Application Server devraient rapidement venir combler cette limitation.

La jeune expérience d'Inprise sur le marché des serveurs d'applications semble aujourd'hui être le frein le plus important pour Inprise Application Server, qui est cependant l'un des produits le plus complet couvrant dans des environnements graphiques homogènes le développement, le déploiement et l'administration.

Références

IAS possède quelques références aux Etats Unis :

- **Bank of America** : Gestion des fusions acquisitions - intégration de systèmes disparates (COM / CORBA)
- **Département d'Etat des transports** : gestion d'applications distribuées orientées transport

D'autre part, quelques grandes entreprises françaises sont en phase d'évaluation du produit, notamment EDF et la Société Générale.

Conclusion

■ **Avantages :**

- L'offre est complète et l'homogénéité des environnements de développement, de déploiement et d'administration est appréciable,
- L'intégration d'un grand nombre de sources de données via le composant transactionnel VisiBroker ITS et ses modules Transfuse et CodeSync,
- La puissance de l'outil de développement avec de nombreux assistants, et un debugger graphique évolué.

■ **Inconvénients :**

- Capacité d'Inprise à s'imposer face à des éditeurs déjà bien en place sur le marché,
- L'intégration d'autres ateliers de développement tels que Delphi et surtout de C++ pour les composants serveurs se fait attendre.

Historique

L'offre actuelle de Microsoft repose historiquement sur deux piliers technologiques : le modèle de composants COM (Component Object Model), introduit il y a un peu plus de quatre ans comme support de la technologie OLE, et MTS (Microsoft Transaction Server), qui a offert il y deux ans le premier support du transactionnel pour des composants COM DNA (Distributed iNternet Architecture) est la nouvelle offre marketing de Microsoft rassemblant l'ensemble des technologies COM, MTS, Web, ainsi que la suite d'outils de développement et d'administration permettant de réaliser des architectures 3-tiers.

Brief technique

L'offre Microsoft est fondée sur Microsoft Transaction Server, qui est un serveur d'applications intégré à Windows NT. MTS offre des services techniques à valeur ajoutée à des composants applicatifs conformes au modèle COM/ActiveX :

- Le support du **transactionnel**, via MS DTC, le Transaction Manager de MTS, (2-phase commit avec le protocole XA ou OLETX),
- Une **optimisation de la gestion des ressources systèmes** (connexions réseaux, connexions SGBD, threads,...),
- Une **gestion fine de la sécurité** et des contrôles d'accès à chaque composant, s'appuyant sur la sécurité NT,
- Une interface universelle d'accès aux données (OLEDB) et aux systèmes transactionnels existants (COMTI),
- Une **configuration déclarative des composants**, qui permet de découpler les problématiques techniques du code applicatif (en ce qui concerne leur modèle de threading, leurs caractéristiques transactionnelles, leur sécurité, ...),
- Une intégration poussée avec le serveur Web IIS 4.0, via ASP.

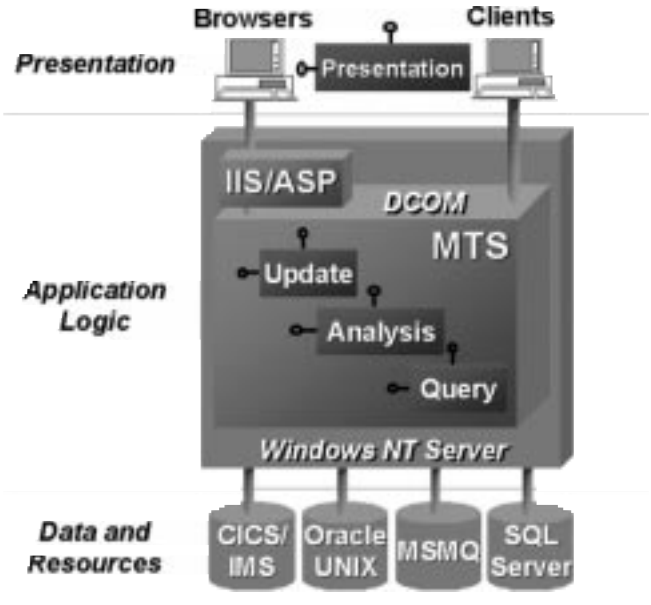


Figure 19 : Le modèle de composants COM/MTS

Le modèle de composants utilisé par MTS est COM/ActiveX. Plus précisément, les composants MTS sont des ActiveX Automation Server. MTS apporte quelques restrictions et spécificités à ce modèle :

- Les composants MTS doivent être des ActiveX IN-PROCESS (se trouvant dans des DLL). En effet, la gestion des process dans lesquels s'exécutent les composants est prise en charge par MTS lui-même. Les composants OUT-PROCESS (.EXE) ne sont donc pas utilisés.
- MTS associe automatiquement à chaque instance de composant applicatif un contexte, IObjectContext, qui contient toutes ses caractéristiques techniques.... Il contient également des méthodes permettant au composant applicatif d'interagir avec MTS. En particulier, les méthodes SetAbort et SetComplete permettent d'indiquer à MTS si le travail du composant s'est correctement effectué ou s'il y a eu des erreurs. MTS utilise ces informations pour valider ou abandonner la transaction en cours[15].
- Les composants participant à une transaction sont recyclés lorsque la transaction se termine : leur état doit donc être sauvé dans le SGBD ou bien grâce à un composant fourni avec MTS, le SPM (Shared Properties Manager).

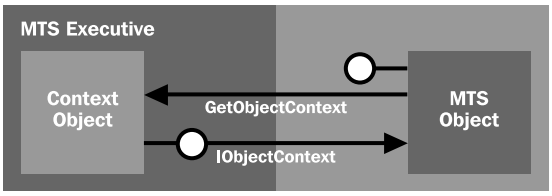


Figure 20 : Gestion des transactions

MTS introduit un nouveau concept par rapport aux systèmes transactionnels classiques : la notion de transaction automatique. Le serveur d'applications gère automatiquement la transaction en s'appuyant sur les propriétés transactionnelles du composant applicatif.

Ces propriétés sont spécifiées de manière déclarative, lors de la configuration de l'application. Les différentes valeurs pour la propriété transactionnelle d'un composant MTS sont :

- *Requires New Transaction* : une nouvelle transaction est déclenchée, lorsqu'une méthode du composant est appelée, même si une transaction est déjà en cours,
- *Requires Transaction* : une nouvelle transaction est déclenchée, lorsqu'une méthode du composant est appelée, sauf si une transaction est déjà en cours, dans ce cas la méthode appelée participe à la transaction,
- *Supports Transaction* : la méthode appelée du composant participe à la transaction en cours s'il en y a une, mais ne déclenche pas de nouvelle transaction si aucune transaction est en cours,
- *Does not Support Transaction* : la méthode appelée du composant ne participe jamais à aucune transaction,

■ Gestion de la sécurité

Toujours dans l'objectif de découpler au maximum les caractéristiques techniques du code applicatif, MTS introduit la notion de rôle pour gérer la sécurité.

15 dans le cas ou le composant s'exécute dans une transaction.

Un rôle est un groupe logique d'utilisateurs, qui est défini de manière déclarative. Les rôles sont affectés dans la console d'administration de MTS aux composants. Le runtime MTS vérifie à l'exécution lors de l'appel des composants que le client a des droits suffisants.

Lors du déploiement des composants sur un serveur de production, on mappe les rôles (notion logique indépendante du déploiement) à des utilisateurs ou des groupes d'utilisateurs NT (dépendant du serveur de déploiement). La gestion des utilisateurs est donc celle de NT.

Si l'on souhaite avoir un comportement plus fin en terme de gestion de la sécurité, on peut également utiliser l'objet Contexte pour implémenter des règles de sécurité.

■ Connexion aux sources de données

La connectivité aux bases de données est assurée via des drivers ODBC supportant le transactionnel : à ce jour, Microsoft en fournit essentiellement pour SQLServer, Oracle et DB2. Ces bases de données sont vues par MTS comme des RM (Resource Manager). Les connexions ODBC sont recyclées par MTS, qui garde un pool de connexions pour les composants MTS. Ce pooling de connexions est totalement transparent pour le développeur de composants applicatifs.

La seule contrainte est qu'il ne doit pas essayer de maintenir lui-même des connexions SGBD : il doit ouvrir, puis refermer une connexion ODBC à chaque appel de requête SQL ou de procédures stockées. En fait, ces connexions ne sont pas vraiment ouvertes ni refermées, mais recyclées par MTS.

Le développeur de composants MTS peut soit utiliser l'interface ODBC directement pour avoir des performances optimales, soit utiliser un composant de plus haut niveau, masquant la complexité de l'interface ODBC. Il s'agit du composant technique **ADO** (Advanced Data Object), qui est généralement utilisé avec des composants MTS écrits en Visual Basic.

Il est recommandé d'utiliser ADO, car ADO encapsule non seulement l'API ODBC, mais également l'interface OLEDB, qui est la nouvelle interface universelle selon Microsoft d'accès aux données. Elle devrait à terme remplacer ODBC.

Microsoft fournit de nouveaux drivers OLEDB pour les principaux SGBD du marché.

A ce jour, les drivers OLEDB ne supportent pas le transactionnel, ni le pooling des connexions. Il faut donc passer par un driver générique OLEDB/ODBC et utiliser les drivers ODBC transactionnels fournis par Microsoft. Cette lacune devrait être comblée rapidement.

■ Connexion à l'existant

MTS fournit une passerelle d'accès aux moniteurs transactionnels IBM CICS et IMS, via SNA Server et COMTI (COM Transaction Integrator), permettant de propager le contexte transactionnel. COMTI comprend également un outil qui génère automatiquement des composants MTS représentant les transactions CICS ou IMS, afin de faciliter leur intégration des les outils de développement.

D'autre part, MTS permet de se connecter à un MOM : avec MSMQ (Microsoft Message Queueing) de manière native et avec MQSeries via une passerelle livré avec SNA Server (produit Level8) . Ces MOM sont considérés comme des RM et peuvent donc participer à une transaction.

Développement

Tout outil permettant de construire des composants ActiveX est susceptible d'être utilisé pour fabriquer des composants MTS. En effet un composant MTS n'est qu'un composant ActiveX devant suivre quelques restrictions en terme de programmation et de gestion du contexte. Deux outils sont particulièrement adapté pour MTS :

- **Visual Basic**, qui permet de manière très simple de créer des composants ActiveX/MTS, orienté métier. Ces composants métier utilisent généralement un composant d'accès aux données ADODB, fourni par Microsoft et optimisé pour MTS/IIS4.0,
- **Visual C++**, qui permet d'écrire des composants ActiveX/MTS très performants et très légers en utilisant le framework ATL. Cet outil est généralement utilisé pour des composants ActiveX techniques supplémentaires. Cet outil requiert une bonne maîtrise du framework ATL, qui est très technique.

Visual Basic ainsi que Visual C++ offrent tous les deux des fonctions de déboggage puissantes, permettant et de débogguer des composants s'exécutant sous MTS en local et même en distant.

L'outil Visual Database Tools livré avec la suite de développement Visual Studio fournit un environnement graphique permettant de créer des requêtes SQL et des procédures stockées, spécialement adapté pour les bases SQLServer et Oracle. Il offre une excellente intégration avec les composants ADO et facilite donc la phase fastidieuse de développement de la persistance des composants MTS. Il s'intègre parfaitement avec les outils Visual Basic et Visual C++.

L'outil Visual Modeler, également fourni avec Visual Studio est un outil de conception UML (une version simplifiée de Rational Rose développé par Rational pour Microsoft). Cet outil offre d'intéressantes fonctions de génération de squelette de code pour Visual Basic et Visual C++. Une fonction de re-engineering est disponible pour Visual Basic, ce qui permet de mettre à jour automatiquement dans l'outil de modélisation les modifications effectuées dans Visual Basic.

Un nouvel outil d'analyse, Visual Analyser, permet d'intercepter de manière transparente tous les évènements importants lors de l'exécution d'une application MTS (appel de méthodes, connexion base de données,...). Il donne une vue globale d'une application. Il permet d'analyser, de filtrer et de fournir des statistiques et des indicateurs de performances, afin de comprendre et de résoudre des éventuels problèmes de performances. Un intérêt majeur de ce produit et de n'entraîner aucune adhérence avec les applications : l'outil n'interagit directement qu'avec MTS et le système d'exploitation.

Administration

MTS offre également une **console d'administration** permettant de :

- Définir des packages ou lots, pour découper de manière logique l'application. Ces packages contiennent les composants MTS, ainsi que l'ensemble de leurs caractéristiques,
- Modifier de manière graphique la configuration de chaque package et de chaque lot,
- Déployer à distance les packages sur des serveurs de production,
- Visualiser l'état des packages et des composants (transactions commitées, composants activés,...). Les indicateurs fournis sont cependant relativement limités.

Une interface programmatique COM permet d'automatiser le déploiement sur des serveurs multiples. Enfin, la console d'administration de MTS est intégrée à MMC (Microsoft Management Console).

Stratégie de l'éditeur

L'offre DNA est aujourd'hui disponible avec NT4 Option Pack (fournie gratuitement) qui est une sur-couche au-dessus de NT. La stratégie de Microsoft est d'intégrer complètement son offre de serveur d'applications dans son futur système d'exploitation Windows 2000 :

- Le modèle COM/ActiveX ainsi que les extensions transactionnelles et de sécurité de MTS seront fusionnées dans le modèle **COM+**, partie intégrante de Windows 2000,
- MSMQ, le middleware asynchrone de Microsoft fera partie du modèle COM+, dans lequel on pourra appeler de manière asynchrone des composants distants en lieu et place du protocole DCOM,
- Le load balancing dynamique entre composants sera intégré à COM+,
- La sécurité sera gérée dans COM+ en s'appuyant sur l'Active Directory de Windows 2000.

Avec l'intégration de son modèle de composants COM+ et ses middlewares dans le système d'exploitation, les clients de Microsoft peuvent s'interroger sur l'opportunité d'acquérir un serveur d'applications qu'ils possèdent déjà. L'issue du célèbre procès verra peut être poindre une stratégie adaptée au verdict.

Références

De nombreux clients, en particulier dans le monde de la finance, ont choisi de refondre leur Système d'Information sur la base d'une architecture DNA. Citons notamment, Merryll Lynch, qui a bâti l'architecture de son réseau d'agences (35 000 PdT) sur une architecture de trois niveaux physiques, intégrant l'ensemble des technologies Microsoft sur les deux premiers niveaux, y compris la passerelle COMTI pour l'intégration des transactions des mainframes.

D'autre part, de nombreuses référence de commerce électronique de l'éditeur utilise les technologies DNA en back-office :

www.gap.com, www.levi.com,...

Conclusion

■ **Avantages :**

- L'offre est complète et intégrée à Windows NT (sécurité, administration,...),
- L'utilisation du modèle de composant COM/ActiveX, largement répandue dans le monde Windows, permet de **bénéficier d'outils de développement existants, connus, et de haut niveau** (L4G comme Visual Basic ou Delphi), ainsi que d'une gamme de produits de conception et d'analyse très complets,
- L'utilisation du modèle de composant COM/ActiveX, robuste et stable, actuellement le plus utilisé dans le monde, est gage de **pérennité** (ce qui est non négligeable pour une approche orientée composants),
- De nombreuses références US.

■ **Inconvénients**

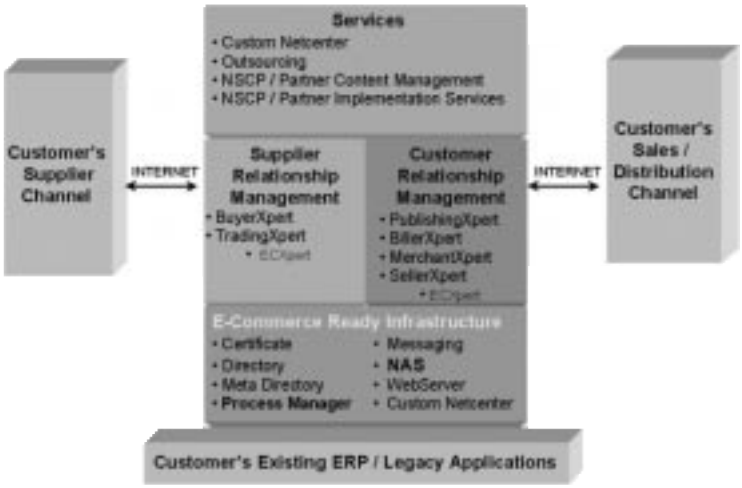
- Il s'agit d'une solution ne tournant que sous Windows NT,
- Des fonctionnalités avancées mais essentielles pour des déploiements à grande échelle (Load Balancing, support intégré de l'asynchrone,...) ne seront disponibles qu'avec COM+ et Windows 2000.

Netscape

Historique

La société Netscape est née en Avril 1994, fondée par Jim Clark (fondateur de Silicon Graphics) et Marc Andreessen (concepteur du navigateur Internet NCSA Mosaic en 92). Dans le cadre du renforcement de son offre serveur, Netscape rachète la Société KIVA en novembre 1997 afin d'acquérir leur serveur d'applications (Kiva Enterprise Server 2.0 reçoit le prix "BEST INTERNET SOFTWARE" au COMDEX 97). KES devient Netscape Enterprise Server 2.0 en février 1998 .

La stratégie



Offre Netscape

Aujourd'hui Netscape axe sa stratégie sur le Net Commerce. Elle comporte deux axes :

- Un axe client : mettre en relation la compagnie avec ses clients finaux. Il consiste à publier des informations, gérer un catalogue de produits, un annuaire de clients et des opérations de paiement sécurisés, et bien sur vendre.
- Un axe fournisseur : externaliser une partie de la gestion directement chez les fournisseurs.

Netscape Application Server en constitue pierre angulaire car il est destiné à supporté toute la logique métier de l'entreprise.

Il permet de construire de nouvelles applications basées sur l'architecture multi-tiers et intégrer les appréciatifs existants grâce aux extensions.

Brief technique

L'offre serveurs d'applications de Netscape se compose de :

- **Netscape Application Server** : Le serveur d'applications et ses outils d'administration
- **Netscape Extensions** : Interfaces (IDL) qui permettent l'accès à quelques systèmes backend.

- **Netscape Application Builder** : Un environnement de développement et de test d'applications
- **Netscape Extension Builder** : Le générateur d'interfaces (IDL) qui permet d'étendre l'accès aux systèmes backend.

C'est un serveur d'applications orienté Web. Il est supporté sur la plupart des plates-formes UNIX et sous NT.

■ Le serveur applicatif

Netscape Application Server et disponible en version 3.0. Une nouvelle version 4.0 est prévue pour le 3e trimestre 1999.

C'est un serveur de modules applicatifs appelés **applogic**. Ils sont répartis soit individuellement soit au travers d'un groupe de modules applicatifs. Ils sont gérés par l'Application Logic Processing System (ALPS) qui sollicite les services d'accès aux données, de gestion de transaction et de présentation.

Il supporte la plupart des bases de données du marché, via JDBC et ODBC.

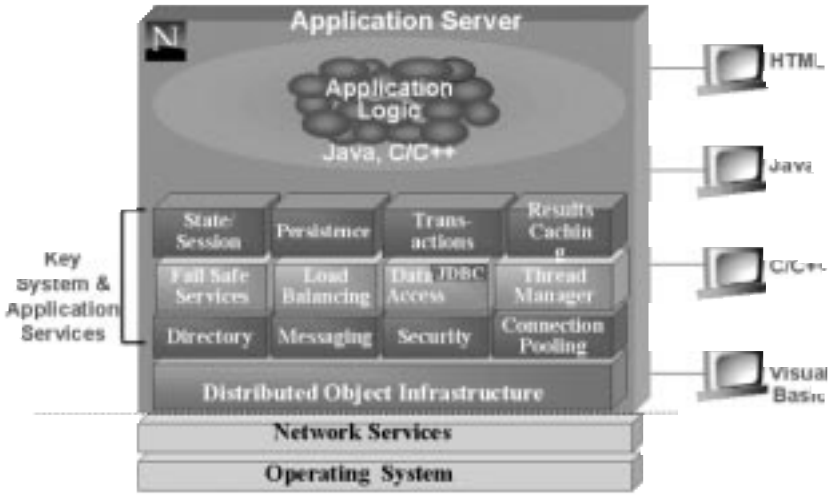


Figure 21 : Architecture Netscape Application Server

■ Multi-serveurs

En environnement multi-serveurs, les requêtes sont d'abord traitées par le serveur de load balancing. Ce dernier les redirige vers le serveur reconnu comme étant le plus "apte" à traiter la requête.

La communication entre les serveurs d'un cluster NAS est assurée par le protocole de communication optimisé (KCP) utilisant des interfaces stream (les données sont transférées sans attendre la fin d'exécution des requêtes).

■ Multithread

Le RMS (Request Management System) se charge de l'attribution des threads en fonction de paramètres modifiables depuis l'interface d'administration.

■ Caches distribués

Le développeur peut demander à NAS de conserver les résultats d'exécution des applogics. Il définit également des critères de conservation : temps de rafraîchissement, valeurs des paramètres fournis à l'applogic…

Ces résultats sont distribués, et donc disponibles pour l'ensemble des serveurs.

Load balancing

Le service de load balancing maintient en permanence une base d'informations sur l'activité et la disponibilité de chaque serveur et des services d'exécution (chaque serveur broadcast régulièrement ses statistiques aux autres serveurs du cluster).
A l'aide de l'outil d'administration graphique, les responsables d'exploitation déterminent le partitionnement applicatif. (Cette répartition est aussi bien transparente aux utilisateurs qu'au développeurs d'applogics). Ces partitions sont ensuite réparties sur l'ensemble des serveurs du cluster NAS.
Certains serveurs peuvent être "spécialisés" afin d'y diriger systématiquement certaines requêtes. Les motivations peuvent être : optimisation de l'effet de cache, serveur implémentant seul (ou mieux que les autres) des interfaces d'accès à d'autres produits ou sources de données.

Disponibilité

Tous les composants peuvent être redondants et substituables. Une requête peut être traitée indifféremment par n'importe quel serveur disponible et disposant de l'applogic concerné. Il n'existe aucun point de passage obligatoire entre le client et le serveur : single point of failure. (serveurs en "hot backup", serveurs secondaires, contextes distribués)

La mise à jour de l'application peut être réalisée sans interruption de service.

- Mise à jour à la volée : le nouveau composant remplace progressivement l'ancien.
- Mise à jour par dégradation de service (en cas d'incompatibilité entre versions) : On désactive sur une partie des serveurs les anciens composants, on installe les nouveaux, puis, simultanément on désactive les anciennes versions sur les serveurs sur lesquels elles sont encore disponibles tout en activant les nouvelles versions sur les serveurs sur lesquels elles viennent d'être déployées.

Outil d'administration

La console d'administration offre les fonctions suivantes :

Configuration

Configuration des serveurs NAS, de leurs process et des serveurs Web frontaux.
Marche/Arrêt des groupes de serveurs ou des groupes de process.
Configuration du/des serveurs "d'état/sessions" qui seront responsables de conserver toutes les informations de sessions applicatives et utilisateurs de manière protégée et distribuée.

Surveillance

Surveillance en temps réel de l'activité du serveur.
Positionnement des alertes et actions sur des événements définis.
La journalisation s'effectue par fichiers en rotation ou directement en base de données.

Sécurité

Définition des contrôles d'accès, des utilisateurs, groupes et profils par application, serveur ou module.

Applications

Déploiement d'application ou de modules
Activation ou désactivation d'un module sur une machine
Passage en mode local/distribué.

Load Balancing

Définition des critères de répartition de charge, des périodes de rafraîchissement des statistiques et de broadcast avec les autres serveurs participant à la répartition.

Développement

Netscape Application Builder (NAB) est l'environnement de développement et de test d'applications destinées à être déployées sur un serveur d'applications NAS. Il supporte le C++ ou Java (JDK 1.1) avec le même niveau de fonctionnalité. Il s'intègre dans Visual Studio, Powersoft tools, Symantec Visual Cafe, Sun Java Workshop, Wallop Build-It.

NAB utilise un modèle de développement (Client Independant Programming Model) qui permet d'isoler les règles de gestion et les objets applicatifs de leur mode de présentation par l'utilisation d'un moteur de génération de page HTML à partir de modèles prédéfinis. Ce modèle permet à une application NAS de s'adresser naturellement à des clients HTML (browser), à des clients légers (application Java, C++,...) ou à des applications externes.

Il comprend :

- Un ensemble de classes Java/C++ pour le développement de programmes serveurs (NAS) et clients (OCL).
- Des interfaces natives vers les bases de données Oracle, Informix, Sybase, MS SQLServer, DB2 et le support d'ODBC et JDBC.
- Une version développeur de NAS et un outil de déploiement pour tester les applications.

On y manipule 3 types d'éléments ;

- Requêtes bases de données
- Modules applicatifs (applogics)
- Templates HTML.

Le **project Manager** permet de grouper et gérer les composants de l'application. Il permet également de tester et déployer localement ou sur un/plusieurs serveurs NAS.

Le **HTML Designer** permet de construire les templates HTML. Il génère les tags utiles au service de fusion de données, et fournit des assistants permettant d'associer les traitements aux pages.

Le **Query Designer** permet de construire graphiquement des requêtes, et de générer le code associé (inséré directement dans le source, ou sous forme de référence).

L'**Applogic Designer** est l'outil de développement des modules applicatifs (applogic). Il comporte de multiples assistants capables de générer les codes nécessaires à l'accès base de données, au maintien de session, aux interfaces template HTML,...

Extensions

Un certain nombre d'extensions NAS sont disponibles.
Elles permettent d'avoir accès aux API externes sous formes de fonctionnalités au standard NAB. Elles jouent donc un rôle d'encapsulation du service externe et utilisent la solution la plus performante possible du produit externe. (par exemple pour SAP, l'interface Web pourtant décrétée comme étant l'interface officielle à été abandonnée au profit d'une autre API pour des soucis de performances).

- Extension pour CICS/IMS
  L'extension pour CICS permet d'intégrer des appels à des transactions CICS ou IMS à partir de l'environnement standard de développement NAS, en Java ou C++. Un outil génère automatiquement les clients Java ou

C++ à partir des programmes COBOL. La connexion avec les systèmes hôtes se fait directement en LU6.2 ou TCP/IP et ne nécessite pas d'installation de logiciel supplémentaire.

- Extension pour Tuxedo
  L'extension pour Tuxedo s'appuie sur l'environnement client de /WS. L'ensemble des appels ATMI sont supportés et permettent l'appel de services Tuxedo locaux ou distants.

- Extension pour MQ/series
  L'extension pour MQSeries encapsule le client MQSeries sans nécessiter l'installation d'un serveur MQSeries sur la machine supportant l'extension. Toutes les fonctions du client sont supportées. L'extension permet aux développeurs NAS d'envoyer des messages applicatifs synchrones ou asynchrones, avec garantie de transmission, en utilisant la technologie de message queuing (MQI).

L'**Extension Builder** permet de développer de nouvelles extensions s'intégrant parfaitement à NAB (encapsulation d'API).
Un outil graphique permet de générer les interfaces (IDL) et les hiérarchies ainsi que le makefile permettant de compiler le code.

Références

E*Trade : Palo Alto, Californie , USA
Probablement la meilleure référence NAS. 500.000 utilisateurs (+1000 nouveaux abonnés par jour), 1.000.000 visiteurs par jour.
SGBD : Sybase
Extension utilisée : TUXEDO
Gestion de portefeuilles boursiers, suivi des cours temps réels…
Les clients E*Trade peuvent accéder à leurs comptes 7j/7 et 24h/24.
Numéro un sur ce marché.
Axa SUNLIFE
Extension utilisée : CICS
Service de transaction d'assurances. Devenu leader en UK en quelques mois.

Conclusion

NAS se distingue de ses concurrents par son aptitude à supporter de très grosses montées en charge. L'argument "think big" est clairement mis en avant par Netscape. Son positionnement par rapport au serveur NetDynamics de Sun repose sur cette caractéristique, ainsi que sur la forte intégration des autres solutions de Netscape (annuaire, serveur Web...)

■ Avantages

- NAS est un serveur d'applications destiné à construire des systèmes WEB, capables de supporter une montée en charge progressive mais considérable,
- Tout a été fait dans un objectif de performance et robustesse maximales,
- Un effort important à été porté aux Extensions afin de pouvoir réutiliser les sytèmes existant tout en conservant l'axe performance et robustesse,

- Les développements peuvent être indifféremment réalisés en Java ou C++,
- Sécurise l'offre (ou la future offre) bâtie au-dessus, notamment la gamme Xpert de commerce électronique.

■ Inconvénients

- Les composants applogics reposent sur une architecture propriétaire ce qui peut gêner les utilisateurs familiarisés aux composants Corba,
- Développement peu intégré et virage important dans le développement car les outils vont changer entre applogic et JSP,
- Le support EJB n'est pas encore disponible, (sera intégré dans NAS 4.0 - Q3 1999) compensée par le support réel de services haut niveau associés : gestion déclarative du transactionnel et sécurité,
- Taille de l'éditeur par rapport à ses principaux concurrents ?

Oracle Application Server

Historique

En juin 97, Oracle édite la première version de production de son Web Application Server. Cette version 3.0, dont le développement est indépendant des équipes Oracle8, se positionne sur le marché du Web dynamique. Son point fort est la possibilité de choisir le langage pour créer des pages HTML dynamiques liées à une base Oracle, et ce grâce à la technologie des **cartouches** (PL/SQL, Java, C, Perl).

Le succès n'est cependant pas au rendez-vous, et notamment comparé aux offres de Netscape et Microsoft (voire même Apache) sur le même créneau.

Le successeur de WAS3.0 sera donc plus ambitieux, il se positionnera sur le marché émergeant des serveurs d'applications et sera renommé Oracle Application Server 4.0.

Brief technique

OAS est un serveur d'applications, c'est-à-dire qu'il supporte une architecture multi-tiers à base de composants. La technologie des cartouches de son prédécesseur a été reconduite ici pour offrir une souplesse dans le choix des langages.

Comme le montre le schéma suivant, le moteur supporte donc :

- Les composants CORBA écrits en Java (cartouche JCORBA) et EJB 1.0 dans la prochaine version (cartouche EJB)[16],
- Les composants Perl (cartouche Perl),
- Les composants PL/SQL (cartouche PL/SQL),
- Les composants natifs écrits en C.

Figure 22 : Oracle Application Server

Toutes ces technologies bénéficient d'une connectivité optimisée vers le SGBDR Oracle, et du transactionnel au travers du support de XA (OAS est un Transaction Manager, mais seul Oracle est actuellement supporté comme Resource Manager).

Pour le développement d'applications utilisant ces composants, deux possibilités s'offrent, selon la complexité de l'IHM à réaliser et le public visé (respectivement Internet et Intranet) :

- Client léger HTML : pages HTML dynamiques (Server Side Scripting) au travers soit :
  - De la **cartouche JWeb**[17], qui permet d'invoquer des composants Java à partir de tags interprétés par le serveur,
  - De la **cartouche LiveHTML** et l'utilisation de scripts Perl, C classiques,

16 La cartouche JCORBA et la cartouche EJB sont en fait très proches, JCORBA implémente déjà un certain nombre de services EJB. La première version des EJB arrivera avec la version 4.0.8 de l'Application Server (version de production en Juin). A noter que les premières sont indépendantes du langage (on peut invoquer un composant JCorba depuis une application C++), mais les seconds sont uniquement destinés au monde Java. Les deux technologies pourraient donc cohabiter à terme.

- De la cartouche PL/SQL : un ensemble de fonctions et procédures PL permettent de manipuler et de générer du HTML.

On constate que le choix du serveur Web utilisé dans ce type d'architecture est neutre (Oracle, Netscape ou Microsoft, mais aussi Apache).

- Client léger Java : application Java dialoguant en IIOP avec le serveur (la technologie ORB utilisée est celle de VisiBroker),

Oracle Application Server est actuellement livré en version 4.0.7. Le développement du serveur est réalisé sur plates-formes NT et Solaris, puis un portage vers les autres Unix est annoncé.

■ Développement

Le développement d'applications OAS est facilité par l'utilisation de l'AGL Oracle JDeveloper2 (à ne pas confondre avec Developer tout court). Stratégie, quelque peu surprenante, JDeveloper est issu du rachat du source de JBuilder 2 (Inprise) qui a été adapté. Les deux produits divergent définitivement.

Le produit permet néanmoins de simplifier le développement de composants JCorba ou de pages JWeb. Par exemple : pour créer un composant CORBA dans JDeveloper, il suffit d'étendre les classes oas.jco ; aucun code supplémentaire visible ou à écrire n'est nécessaire pour préparer un package directement utilisable par OAS.

En revanche, à l'instar de la concurrence, il est toujours difficile de lier la présentation Java/IIOP aux composants de manière simple, tel que c'est le cas dans les outils client/serveur classiques : liste de valeurs associés à un champ, appel de méthode sur déclenchement d'un trigger IHM, etc. Des évolutions dans ce sens apparaîtront dans JDeveloper3, qui apportera également le support des EJB.

Côté application Web, le développement sera également amélioré par le support des JSP et la programmation par modèle XSL. XSL permet de décrire de manière indépendante le rendu et la dynamique HTML d'une donnée sous XML. Ce concept permet de séparer plus nettement la présentation des traitements, et également de faire du binding XML côté client.

Enfin, OAS gère le transactionnel en XA vers une base Oracle dans les environnements Java et PL/SQL, et ce de manière programmatique (dans le code) ou déclarative (dans l'interface d'administration ou dans la description externe des composants). Comme dans les autres serveurs de composants, la granularité est le composant. Donc si l'on ne sépare pas les accès en lecture des accès en écriture lors de la conception, les appels de type monComposant.rechercherParNom() déclencheront systématiquement une transaction… d'où un impact sur les performances.

Concernant le support d'autres SGBDR, il est aujourd'hui nécessaire de passer par le SGBDR Oracle pour atteindre un autre système : on utilise les passerelles existantes (Oracle Open Gateways) vers Sybase, DB2, Informix, etc. Cette contrainte pourra disparaître lorsque ces éditeurs livreront leurs propres drivers JTS.
De même pour s'interfacer avec le reste du Système d'Information, la règle est également d'utiliser ces mêmes passerelles : MQ*Series, CICS, ERP, etc.

■ Administration & Exploitation

Depuis les versions initiales, OAS supporte une gestion applicative du clustering pour assurer à la fois la robustesse et l'équilibrage de charge. Ces fonc-

17 JWeb évoluera vers le standard Java Server Page (JSP) de Sun dont la spécification finale est en cours.

tionnalités sont présentes en standard dans l'offre et ne nécessitent aucun codage particulier, les composants sont simplement déployés à plusieurs endroits.

Un outil d'administration Web permet la gestion des composants (déploiement, activation, suppression, etc.), et leur surveillance. Une MIB SNMP est également disponible pour s'interfacer avec les outils d'administration du marché.

A terme, cet outil disparaîtra au profit d'une intégration dans Enterprise Manager qui est la console d'administration et d'exploitation d'Oracle7&8 (très complète). Cette intégration devrait être réalisée en 4.0.8.

A cette date, l'architecture suivra le principe Manager/Agent, à l'image des principaux frameworks d'administration du marché (Tivoli, TNG, etc.). Cette application utilise elle-même l'architecture OAS :

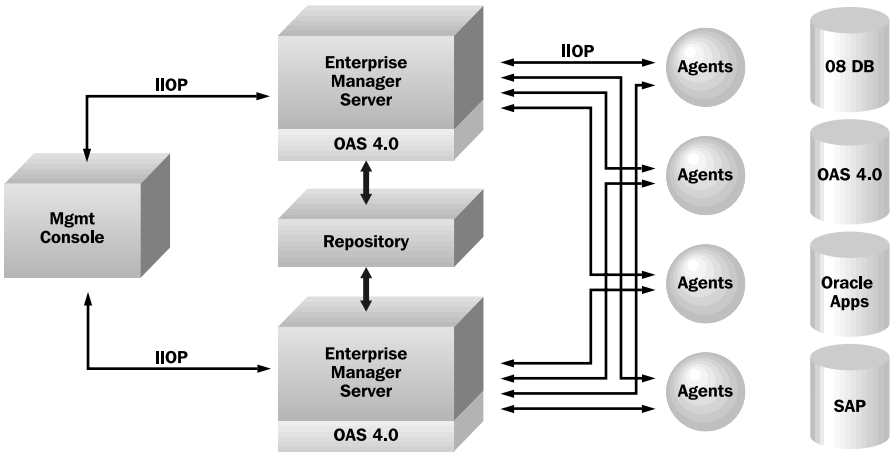


Figure 23 : Administration d'Oracle Application Server

L'idée est ambitieuse, car outre la gamme d'agents Oracle "plugables" à terme dans cette architecture (DB, OAS, Forms, Express, etc.), Oracle souhaite que d'autres éditeurs développent des agents EM pour leurs produits (ici SAP).

■ Sécurité

OAS s'interface avec un annuaire LDAP (Netscape). On peut donc utiliser l'annuaire pour faire de l'authentification et du contrôle d'accès, de manière déclarative ou programmatique.

Par ailleurs, notons le support de SSL sous HTTP et IIOP, garantissant un niveau de sécurité supplémentaire entre client et serveur : authentification forte et confidentialité/intégrité. Le serveur est capable d'utiliser des certificats X.509 pour les utilisateurs (stockés dans l'annuaire). Un outil de gestion des certificats X.509 (Wallet Manager) est fourni.

Positionnement/Stratégie

Malgré les qualités de son moteur, Oracle souffre d'un positionnement flou entre OAS et Oracle8i (sortie en mars). En effet, les deux assureront le support des EJB et d'IIOP… Aujourd'hui, OAS est la plate-forme de choix pour les applications **Internet** (serveur en DMZ et données à l'intérieur du SI, support des JSP), et 8i la plate-forme **Intranet**… cependant on peut douter d'une telle segmentation à moyen terme.

JDeveloper est un axe stratégique pour les développements objet, il ne s'appuiera pas sur ObjectType[18] d'Oracle8 mais générera lui-même le code SQL de persistance[19]. Developper/6 évoluera quant à lui (avec le support d'ObjectType par exemple), mais demeurera un outil dans la continuité de Forms4.5, c'est-à-dire orienté données. Designer supportera l'UML.

Enfin, Oracle se positionne également sur la technologie XML en offrant un parser côté serveur. XML est vu comme une brique d'interopérabilité pour les échanges inter-applicatifs, c'est-à-dire qu'un objet doit être capable d'être représenté en XML pour pouvoir être utilisé par d'autres applications. XML sera également utilisé comme outil de description, notamment dans le Repository des composants (MetaData).

Références

Quelques références françaises en production :

- **Lafarge** : Extranet avec 200 points de vente pour la réalisation des peintures en temps réel.

  Technologies employées : HTML, cartouche PL/SQL

- **Hoescht Marrion Roussel** : consultation de catalogue et achats.

  Technologies employées : HTML, cartouche PL/SQL, Java et C

- **Debitel** : extranet avec 200 points de vente, souscription d'abonnements GSM.

  Technologies employées : HTML, cartouche PL/SQL, interfaces avec d'autres systèmes existants.

Conclusion

■ **Avantages :**

- Capacité à offrir une **solution intégrée** entre les deuxième et troisième tiers y compris en termes d'outils de développement, d'administration et de sécurité.
- Java et les technologies thin-client, sont clairement un **axe stratégique** pour Oracle : OAS, 8i, Forms Server, Express, Internet Commerce Server, etc. toute la gamme des produits migre ou est déjà migrée dans cette architecture.

■ **Inconvénients :**

- Un certain **flou sur la stratégie** : deux technologies de serveur d'applications évoluent parallèlement : OAS et 8i.

Les perspectives de l'offre Oracle nous semblent néanmoins très bonnes, à la fois sur le terrain de l'Internet et de l'Intranet. D'ici un an et les premières adoptions d'Oracle 8i, le positionnement des offres devrait être clarifié, sécurisant d'autant le choix Oracle pour les clients.

18 ObjectType est une option d'Oracle8 permettant de stocker simplement des objets non-hérités dans la base relationnelle.

19 Ce point est important, car il conditionne aussi le futur support de la persistance des Entity Bean par le moteur lui-même (container-based persistence), c'est-à-dire un mécanisme de mapping Objet/Relationnel performant permettant de faire persister de manière transparente des composants. Ce support ne sera assuré que lorsque la technologie aura suffisamment mûri (ObjectType est finalement dans sa version 1.0 ...).

Historique

Avec le rachat de la société NetDynamics et de sa plate-forme de serveurs d'applications (NetDynamics4.0) en juillet 1998, Sun détient un produit destiné à devenir l'élément clé de sa nouvelle offre globale. En effet, le produit constitue un élément complémentaire de la plate-forme JDK JAVA, qui ne contient pas de serveur d'applications.

Brief technique

L'offre App Server de NetDynamincs est consitutée de :

- **NetDynamics Application Server**, le runtime contrôlant l'exécution des composants applicatifs et fournissant des services techniques,
- **NetDynamics Platform Adapter Components (PACs)**, qui sont des connecteurs extensibles permettant d'intégrer des applications existantes de l'entreprise,
- **NetDynamics Command Center** l'outil d'administration associé,
- **NetDynamics Studio**, l'outil de développement intégré pour développer et débugger des applications tournant sous NetDynamics Application Server,
- **NetDynamics Java Object Framework**, un ensemble de classes Java fournissant une interface Java homogène et de haut niveau pour le développement des composants applicatifs.

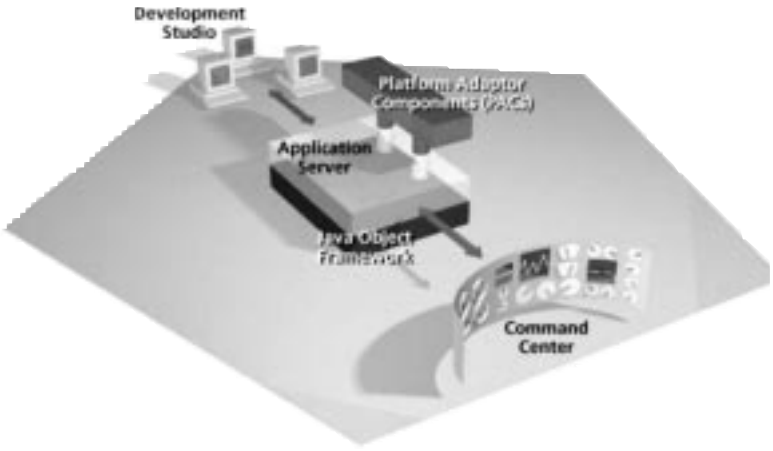


Figure 24 : Console d'Administration de Netdynamics

NetDynamics Application Server est un serveur d'applications "multi-threads" écrit en Java et basé sur CORBA (il utilise l'ORB d'Inprise Visibroker for Java), disponibles sous NT et les principaux UNIX. Il fournit en standard un modèle de composants applicatifs (propriétaire), ainsi que des services techniques de haut niveau. Les composants applications peuvent être écrits en Java ou en C++.

■ Le modèle de composants

Tous les composants NetDynamics sont des serveurs CORBA qui héritent d'une même classe de base (Admin) et d'une même interface IDL IndAdmin. Les composants sont de deux types :

- Des services techniques de base fournis avec l'Application Server (qui sont des exécutables),
- Des composants applicatifs, appelés "worker", conformes à un modèle de composants définis par NetDynamics. Ils doivent implémenter une interface commune permettant aux services techniques de gérer leur cycle de vie et de les administrer via une console d'administration. Les workers sont regroupés logiquement dans un service.
- La figure suivante montre le fonctionnement interne d'un service applicatif. Le composant **Service Monitor** gère un pool d'objets "worker". Au démarrage du service, le composant Service Monitor instancie l'ensemble des objets "workers" du service, selon un certain nombre de paramètres :
- Nombre maximum de "workers" instanciés,
- Nombre maximum de connexions clientes par "worker" (déterminant le nombre de threads exécutés),
- Nombre de processus créés par un "worker".

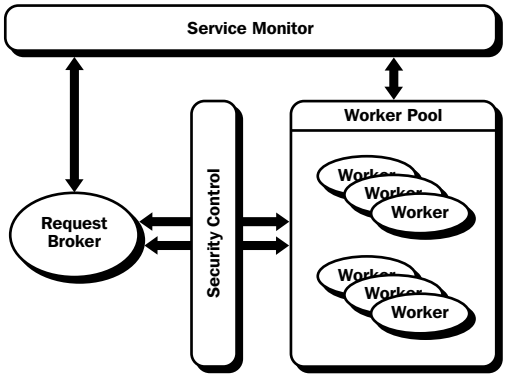


Figure 25 : Le service Monitor

Lors d'une demande d'instanciation d'un nouvel objet worker par un client, le Service Monitor interagit avec l'ORB pour retourner au client un worker du pool, en vérifiant au passage les droit d'accès du client. Lorsque le client a fini d'utiliser l'objet worker, le Service Monitor se charge de le remettre dans le pool.

■ Répartition de charges

La répartition de charges est prise en charge par le **Service Locator**. Le rôle de ce composant est de déterminer le service le moins chargé, puis le composant worker le moins chargé. Si aucun composant n'est disponible, le requête est stockée dans le file d'attente du composant Service Locator.

■ Sécurité

La sécurité est implémentée au niveau des services applicatifs : leur accès est sécurisé par la gestion d'utilisateurs ou de groupes d'utilisateurs à l'aide de trois fonctions distinctes : l'authentification (vérification de l'identité d'un utilisateur), l'autorisation (déterminer le niveau d'accès par la gestion d'ACL) et le cryptage de la connexion (codage des paquets HTTP par SSL). Chaque utilisateur est validé au cours d'une session par une clé (Client Ticket).

■ Connexion aux bases de données

Le service **Relationnal Database Service** constitue le point d'entrée indispensable à la connexion aux bases de données. Ce service offre des connectivités natives pour l'ensemble des SGBD du marché. Il n'utilise actuellement pas JDBC.

■ Gestion des états et du contexte

Le **Persistence Engine Service** permet de stocker en mémoire les états des objets Java et surtout les sessions de ces objets. Les informations pertinentes sont stockées en mémoire indexée par l'ID de la session. Chaque projet NetDynamics définit un temps limite de session déterminant la durée maximum de stockage en mémoire des paramètres de session.

■ Connexion à l'existant

Afin de faciliter l'intégration de son serveur d'applications au sein d'un Système d'Information, NetDynamics fournit un modèle de connecteurs appelée PACs pour (Platform Adapter Components), destinés à se connecter à des applications existantes. Un SDK composé d'un framework et d'outils de développement est fourni pour créer d'autres PACs vers des systèmes non offerts en standard. Les connecteurs ainsi développés bénéficient automatiquement des fonctions de load-balancing, failover, et de sécurité fournies par le serveur d'applications.

De nombreux PACs, développés par des partenaires, existent pour les ERPs : SAP R/3, PeopleSoft et des middlewares : MQ*Series, IMS, CICS, Tuxedo, etc.

Développement

NetDynamics soutient une approche dans laquelle il fournit les outils de développement sur sa plate-forme, dans le but de faciliter au maximum l'intégration avec son offre.
NetDynamics Studio est un atelier de développement Java complet. Cet outil gère à la fois le développement des serveurs CORBA en Java et des applications clientes en Java et en HTML.

Un debugger utilisant l'interface de debuggage propre au JDK 1.1, est intégré à la plate-forme de développement pouvant debugger les applications Java locales et distantes. Le debuggage des applications distantes autorise le développeur à attacher une machine virtuelle Java située aussi bien côté client que côté serveur ou même externe au système. Cet outil est compatible avec la plupart des outils de développement et de gestion de projets en groupe comme Clearcase, et Visual Source Safe.

Administration

NetDynamics fournit une console d'administration de composants constituée d'une applet Java, ce qui autorise une administration en temps réel, locale ou distante du ou des serveurs d'applications.

NetDynamics Command Center permet de gérer plusieurs configurations systèmes sur des serveurs différents, de dupliquer ou de déplacer des composants d'une machine vers une autre. Il permet de dupliquer facilement un composant critique, d'un serveur vers un autre ou au sein d'un même serveur, les clients accédant alors au composant critique sont automatiquement dirigés vers sa copie ou vers lui-même si celui-ci a été relancé.

Stratégie de l'éditeur

La stratégie de Sun est d'offrir les services actuels fournis par NetDynamics de manière propriétaire en les remplaçant par les technologies de la plate-forme Java. La prochaine version de NetDynamics devrait utiliser JDBC pour la connexion aux SGBD, et fournir un premier support (incomplet) des EJB pour le modèle de composants. Le support des JSP (Java Server Pages) et des Servlets est également prévu.

NetDynamics Application Server, aujourd'hui positionné comme un serveur d'intégration de systèmes existants, veut devenir une réelle plate-forme pour les nouvelles applications critiques de l'entreprise.

Cependant, les accords récents entre Netscape et Sun laissent à penser qu'il y a une plate-forme de trop dans l'offre des deux éditeurs, Netscape Application Server et NetDynamics étant totalement concurrents.

Conclusion

Le point fort de l'offre NetDynamics est indéniablement la robustesse de son offre : le grand nombre de ses clients, signe de la maturité de son produit, le confirme très bien.

Un autre point fort est le nombre important de connecteurs PACs pour facilement s'intégrer avec des systèmes existants.

Cependant, outre les aspects recouvrement de l'offre avec celle de Netscape, on peut noter quelques faiblesses du point de vue technique :

- Le support du transactionnel n'existe pas : il n'y a pas d'implémentation d'OTM pour garantir un accès transactionnel à des Resources Manager,
- Le support du modèle de composant EJB et des conteneurs EJB : le framework Java de NetDynamics, ainsi que son modèle de composant, à base de workers n'est pas conforme aux EJB. Ceci s'explique bien évidemment par le simple fait que la spécification EJB est plus récente que le produit de NetDynamics. La conformité EJB étant un point essentiel pour les serveurs d'applications Java/CORBA, on peut espérer que le rachat par Sun va accélérer la mise en conformité EJB du framework Java de NetDynamics.
- Enfin, l'outil de développement NetDynamics, bien que tout à fait suffisant, n'est pas aussi répandu que les outils Java leaders du marché que sont JBuilder, Visual Café ou Visual Age for Java. Il serait souhaitable que l'on puisse utiliser un de ces outils de développement avec l'App Server de NetDynamics. On peut espérer que la conformité EJB des serveurs d'applications et des outils de développement donnent un plus grand choix concernant la plate-forme de développement.

Autres acteurs . . .

Outre les principaux éditeurs évoqués dans ce livre blanc, nous avons voulu souligner la présence d'offres de grande qualité en provenance d'éditeurs moins connus.

Signalons donc les produits PowerTier for Enterprise JavaBean v4.2 de Persistance Software, Secant Extreme Server et Gemstone/J 2.0 de Gemstone.

L'intérêt de ces produits ? Ils sont généralement en avance de phase par rapport aux grands éditeurs. Prenons l'exemple de PowerTier, c'est un serveur EJB complet, supportant ce qu'aucun autre éditeur ne propose encore aujourd'hui :

- Support EJB 1.0,
- Support de la persistance "container managed", c'est-à-dire automatique, sans code de persistance SQL à écrire,
- Support du cache d'objets,

  Ces deux points se fondent sur une technologie Object/Relational Mapping & Caching qui évolue depuis 1993.

- Gestion déclarative du transactionnel jusqu'au niveau de la méthode.

Ces produits, à l'image de PowerTier, disposent donc d'une avance technologique sur la spécification EJB par rapport aux principaux éditeurs.

Leurs faiblesses ? C'est un peu le lièvre et la tortue : ces éditeurs sont partis en avance et disposent d'une dynamique leur permettant de proposer dès aujourd'hui une offre très performante pour de nouvelles applications (orientées SGBDR). En revanche, il va être délicat de tenir sur la durée sans se faire rattraper par les sauriens Oracle, BEA, MS... notamment sur les aspects partenariats et intégration de l'offre : intégration des legacy (ponts MOM, CICS, etc.), outils de développement , outils d'administration, etc.

Il n'est donc pas impossible que l'on retrouve à l'avenir une partie de ces technologies... mais pourquoi pas intégrées dans certaines des offres éditeurs.

Conclusion

Les technologies Internet vont bouleverser la hiérarchie des entreprises en offrant une quantité incommensurable d'opportunités. En trois ans, la capitalisation boursière d'E*Trade, a dépassé celle bâtie par Meryll Lynch en 150 ans. Les entreprises qui vont concourir pour les premières places sont celles qui seront capables de relever les enjeux de cette nouvelle donne, notamment en mettant en place rapidement les outils capables d'offrir de nouveaux services, tant en Intranet qu'en Internet.

Dans ce contexte de compétition accrue, le choix des technologies mises en œuvre pour développer ces nouvelles applications est à la fois stratégique car il s'agit de ne pas se tromper sur l'avenir de la solution retenue, et peu importante, tant l'important est de se jeter à l'eau.

Les applications de demain reposeront sur deux standards, COM ou EJB. Alors, Coca ou Pepsi ? Le parallèle est tentant. En effet, pour les consommateurs occasionnels, il s'agit de deux colas parfaitement identiques. Pour les aficionados, pas question d'en prendre un autre que celui qu'ils ont choisi, le plus souvent sur des critères irrationnels (mode, marketing,...).

Aujourd'hui, force est de constater que la disponibilité des technologies COM est en avance sur celle des EJB. Ce retard va se combler rapidement, tant les concurrents de Microsoft mettent les bouchées doubles pour implémenter la norme EJB. L'interrogation porte sur la réelle interopérabilité ou compatibilité qu'auront ces EJB au sein de serveurs d'applications différents, voir de systèmes d'exploitation différents. L'histoire de l'informatique nous a déjà apporté son lot de leurres concernant les standards. Que ce soit pour COM ou pour EJB, cette tuyauterie sera bientôt partie intégrante de l'OS. La normalisation de demain devra prendre de la hauteur et porter sur un framework d'objets métier qui sera, du moins l'espère-t-on, indépendant de ces standards.

Microsoft, dont la suprématie fait peur au marché, n'a pas d'autre choix que d'être le meilleur pour imposer ses technologies. Il a fixé les règles du jeu : sortir des produits perfectibles mais à un rythme insensé et qui sont rapidement suivis de Service Packs corrigeant les défauts. Même IBM se plie à cette loi, sortant des versions successives de WebSphere n'ayant toujours pas le niveau de stabilité que l'on connaît habituellement chez Big Blue.

Parallèlement, certains éditeurs vont devoir clarifier le recouvrement de leur offre (BEA avec M3 et Weblogic, Oracle avec OAS et 8i, SUN et Netscape).

Les serveurs d'applications qui vont s'imposer sont ceux qui permettront aussi bien de développer des petites applications répondant à un besoin urgent que de refondre le Système d'Information, en rendant la gestion de la performance, de la scalabilité et de la complexité transparente aux développeurs et aux exploitants. Ils seront intégrés avec un environnement de développement et d'administration performant.

Au final, les architectures 3 tiers permettent (et permettront de mieux en mieux) une **encapsulation des accès aux systèmes hétérogènes** constitutifs du SI, et **imposent un modèle de développement** séparant les interfaces des traitements et des données. Ces deux facteurs sont une des clés offrant une réelle **flexibilité** dans l'évolution des Systèmes d'Information.

En conclusion, même si le ticket d'entrée pour l'acquisition de compétences est important et le retour en terme de gains de productivité non immédiat, les architectures à base de composants vont accroître la productivité et la qualité de service sur le moyen terme.

Glossaire

ACID Atomicity, Consistency, Isolation, and Durability.

EJB Enterprise JavaBean, modèle de composants défini par Sun et d'autres éditeurs (IBM, BEA, etc.). EJB 1.0 définit notamment deux types de composants :

- Session Bean : A chaque session cliente correspond une instance d'objet de type Session Bean. Ce type de Beans est utile pour sauvegarder des informations liées à l'utilisateur (informations de contexte).
- Entity Bean : Ce type de Beans est utilisé pour représenter une donnée qui se trouve dans un SGBD. Chaque instance d'un Entity Bean correspond à un ensemble de lignes de tables SGBD. La persistance d'un entity bean peut être codée dans celui-ci (bean-managed persistence), par exemple en JDBC, ou gérée automatiquement par le containeur (container-managed persistence).

Un containeur EJB héberge ces deux types de composants et fournit des services de persistance, de transactions, de sécrutié, que l'on définit de manière declarative dans un fichier de configuration.

IDL Interface Description Language, syntaxe CORBA, indépendante du langage, définissant les interfaces des objets.

IIOP Internet Inter-ORB Protocol, protocole de communication défini par CORBA permettant la communication entre objets distribués. IIOP est le fondement de "l'intéropérabilité" entre serveurs CORBA d'éditeurs différents.

JNDI Java Naming and Directory Interface, standard Sun pour l'accès aux annuaires (LDAP, Netware NDS, DNS, nommage CORBA, etc.)

JSP Java Server Page, standard Sun équivalent des pages ASP de Microsoft, définissant des tags spéciaux permettant d'une part d'ajouter du code Java aux pages HTML et d'autre part d'appeler et d'utiliser des JavaBeans et EJB à partir de ces pages.

JTA Java Transaction API, API pour démarquer programmatiquement des transactions dans du code Java. L'API JTA définit notamment un "mapping" en Java de l'API XA.

JTS Java Transaction Service, "mapping" Java de l'OTS défini par CORBA.

Legacy Littéralement "héritage". Dans ce document, on parlera de "legacy" pour tout ce qui concerne l'existant dans un Système d'Information : applications IMS, CICS, Tuxedo, MQ*Series, base de données, etc., et ERP par extension (abus de langage)

Message Broker Moniteur de messages se plaçant au-dessus d'un MOM pour assurer des fonctions de routage et de transformation de messages.

MOM Message Oriented Middleware, middleware permettant la communication asynchrone entre applications. MQ*Series (IBM) est le standard de facto sur ce marché.

OTM Object Transaction Manager, version objet d'un moniteur transactionnel.

OTS Object Transaction Service, version objet de l'API XA (tx_begin, tx_commit, etc.) normé par CORBA.

RMI Remote Method Invocation, standard Sun définissant la communication entre objets distribués (notamment les EJB). RMI utilise un protocole spécifique (RMCP), la plate-forme Java 2 utilise IIOP pour les communications RMI.

XSL EXtensible Style Language, standard du W3C permettant de formater du XML dans un langage de présentation, HTML par exemple. A l'image de HTML, XML est un langage déclaratif, à base de tags.

XA Modèle de gestion des transactions distribuées de l'X/Open.