
# Maîtriser la persistance objet métier au sein d'une architecture J2EE

*Préparé pour Oracle*

# SOMMAIRE

# Préambule

Il y a tout juste un an, SQLI publiait une étude intitulée « Stratégies de persistance d'objets métiers en architecture J2EE ».
Un mois plus tard, un magazine de référence reprenait l'étude et titrait « J2EE fait vaciller l'édifice e-business ». Il s'avère que nous avions simplement pris nos responsabilités et alerté nos clients sur le manque de maturité d'un pan important de la technologie EJB 2 .0 à savoir les Entity Beans CMP. Nous avons passé beaucoup de temps en conférences et séminaires pour expliquer le contenu précis et les résultats des tests rigoureux menés par nos équipes de R&D.

Nous avons dû aussi rappeler très souvent que malgré les résultats de notre étude, le Groupe SQLI considère que Java en tant que langage et J2EE en tant qu'architecture technique sont parfaitement viables pour sous-tendre des applications transactionnelles lourdes et critiques dans le cadre de projets stratégiques au sein de grandes entreprises.

Nous allons continuer dans cette voie et agir en vue d'assumer pleinement notre rôle de réducteur de risques techniques dans l'adoption des nouvelles technologies au service de nos clients et de la communauté.

Dans ce même objectif un an plus tard, nous avons décidé de faire le point sur cette même problématique et apporter des réponses claires à nos clients pour les aider à faire leur choix d'architectures transactionnelles à base de Java.

La norme J2EE a véritablement standardisé le transactionnel distribué et sécurisé. Il s'agit d'une avancée importante dans la stratégie de couverture de toute la chaîne de liaison de l'architecture d'une application transactionnelle.

Cependant, la couverture en matière de persistance objet du standard reste incomplète, et à cet égard, EJB 2.1 n'apporte pas d'améliorations majeures.

L'apparition toute récente de la norme JDO a suscité beaucoup de controverse. Rompant avec les orientations adoptées jusqu'alors par le modèle EJB, cette norme propose une approche de persistance orthogonale au modèle métier, et neutre vis-à-vis du support de persistance et du modèle d'architecture employés. Si JDO apporte de nouveaux éléments de réponse à la problématique de la persistance, la norme doit encore se positionner par rapport aux technologies de persistance établies telles que EJB Entité et les frameworks de mapping objet-relationnel comme Oracle9iAS TopLink. Par ailleurs, la norme JDO souffre actuellement d'un manque de maturité, et d'un manque de reconnaissance des acteurs de l'industrie, notamment de la part des éditeurs du monde relationnel.

Aujourd'hui, le débat sur la problématique de gestion de la persistance des objets métier en architecture J2EE reste ouvert, et donne l'opportunité à des éditeurs d'apporter leur contribution sous la forme de produits parfois performants et innovants à l'instar de Oracle avec 9iAS TopLink.

Habib GUERGAC²HI
Directeur Technique – Membre du Comité Exécutif
**hguergachi@sqli.com**

# Résumé pour décideurs

Les systèmes de gestion de bases de données relationnels (SGBDR) sont devenus un pilier incontournable dans le développement d'applications de gestion, et seront présents pour longtemps dans les systèmes d'information : le modèle relationnel, simple et puissant est rapidement assimilé par les concepteurs d'applications ; le niveau de fiabilité et de performances des SGBDR ont conquis les responsables d'exploitation et la pérennité des éditeurs rassure les directions informatiques.

Un SGBDR doit cependant être utilisé avec un serveur d'application, afin de satisfaire les contraintes de sécurité, de modularité, de performances et de diffusion multi-canal qui sont imposées aux applications d'aujourd'hui. On dit alors que ces applications sont basées sur des architectures multi-tiers.

Les standards de serveurs d'applications modernes, notamment **Java 2 Enterprise Edition (J2EE)**, se basent sur des technologies orientées objet. Celles-ci permettent aux applications d'atteindre un niveau de qualité et de flexibilité nettement supérieur à ceux d'une modélisation relationnelle. Cependant, les objets modélisés dans les applications, par exemple un objet client, sont associés à des données stockées dans les SGBDR. On parle ainsi de persistance d'objets métiers. Les architectes sont alors confrontés au problème réputé complexe de l'« ***impedance mismatch*** », caractérisant l'écart séparant les mondes relationnel et objet. Plusieurs approches sont alors possibles :

- **Contourner le problème de l'*impedance mismatch*,** en conservant une modélisation relationnelle pure, et en renonçant aux avantages de la technologie objet. Avec cette approche, des efforts importants sont nécessaires pour développer le code d'accès au SGBDR, et l'évolutivité de l'application est faible. Cette solution convient aux applications fonctionnellement simples et aux durées de vies courtes, mais peut se révéler désastreuse pour des applications d'envergure.

- **Développer manuellement le code de projection des objets sur le support relationnel.** Il s'agit d'une opération particulièrement risquée, qui présente des risques de dérapage sur les délais de livraison, d'effondrement des performances et de faible évolutivité. Il s'agit d'un piège classique dans ce type de développement.

- **Utiliser une couche logicielle spécialisée de projection objet-relationnel** : Certains logiciels dédiés, comme Oracle9iAS TopLink, permettent de résoudre simplement le problème d'*impedance mismatch*. Ce type de logiciels permet de réduire jusqu'à 80% le temps de développement du code d'accès à la base de données, tout en bénéficiant des avantages des technologies objet et en conservant la compatibilité avec les SGBDR existants.

La compatibilité aux standards est un autre aspect fondamental du développement d'application à base d'architecture multi-tiers. Elle permet de s'affranchir d'un éditeur en particulier et de pérenniser ses développements. Le standard J2EE a récemment permis de faire d'énormes progrès dans ce domaine. Malgré son acceptation de l'industrie, Le standard J2EE relatif à la persistance objet, **EJB Entité 2.0**, n'a pas atteint la richesse fonctionnelle nécessaire pour résoudre de manière satisfaisante le problème de l'*impedance mismatch*. Un autre standard de Sun, **JDO** (Java Data Objects), est prometteur mais n'a pas encore suscité une adhésion suffisante de l'industrie du logiciel.

Malgré l'absence d'un véritable consensus sur un standard de persistance objet aujourd'hui, Oracle adopte une démarche pragmatique tout en restant attentif et réactif aux évolutions

en ce domaine : dans le monde Java, c'est la maturité des standards et leur acceptation par l'ensemble des acteurs de l'industrie et des clients qui doit guider l'évolution des produits.

Aujourd'hui, Oracle9iAS TopLink offre une réelle compatibilité et interopérabilité avec l'ensemble des spécifications de J2EE tout en comblant les lacunes de ces standards à l'aide d'extensions. Outre la compatibilité avec la norme EJB 2.0, Oracle9iAS TopLink apporte également une compatibilité partielle avec la norme JDO. Oracle9iAS TopLink prend également en charge la majorité des bases de données relationnelles et les principaux serveurs d'application du marché.

La persistance d'objets métier est un aspect délicat des architectures multi-tiers, maîtrisé par une poignée d'éditeurs dans le monde. Avec l'acquisition de la technologie TopLink, Oracle devient le seul *major* du middleware doté d'un framework de persistance objet relationnel éprouvé.

# Introduction

A travers cette étude, nous nous proposons d'étudier la problématique de la persistance objet, et d'évaluer les solutions existantes pour le développement d'applications au sein d'architectures multi-tiers en environnement J2EE.

Les différentes alternatives que nous avons identifiées pour la persistance objet sont :

- le modèle de composants métier persistants EJB Entité de la spécification J2EE
- un nouveau standard de persistance émergent représenté par la norme JDO, apparu fin 1999, et dont la première version a été finalisée au milieu de l'année 2002.
- les frameworks de mapping objet-relationnel, représentés par l'une des principales offres en ce domaine : le produit Oracle9iAS TopLink.

En parallèle à l'évaluation fonctionnelle que nous mènerons sur ces différentes solutions, nous nous attacherons également à expliquer comment tirer parti de la complémentarité de ces technologies pour la mise en œuvre de la persistance objet.

**La première partie** de cette étude sera consacrée à l'explication de la problématique de la persistance objet, de l'intégration des modèles objet et relationnel, et des différents mécanismes mis en œuvre au sein d'un moteur de persistance objet.

**Dans la deuxième partie**, nous évoquerons les différents modèles d'architectures multi-tiers, les services de haut-niveau apportés par les plates-formes J2EE, et l'intégration de l'infrastructure de persistance au sein de ces architectures.

**La troisième partie** sera consacrée à la présentation et à l' évaluation fonctionnelle des différentes solutions de persistance. Les résultats détaillés de l'évaluation fonctionnelle seront présentés en annexe. *En conclusion de cette troisième partie (section 3.5), nous exposons les perspectives de normalisation en matière de persistance objet.*

**Enfin, la quatrième partie** s'attachera a décrire plus en détail les atouts spécifiques qu'offre la solution Oracle9iAS TopLink dans le domaine de la persistance objet au sein d'architectures multi-tiers.

# 1. Les modèles relationnel et objet

Le modèle relationnel, mis au point dans les années 1970 à la suite notamment des travaux théoriques d' Edward F. Codd[†], a constitué une véritable révolution pour la conception des systèmes d'information. Durant les 30 dernières années, les bases de données relationnelles se sont largement répandues et imposées au sein des organisations et entreprises de toutes tailles. La simplicité et la puissance intrinsèques au modèle relationnel, et le support massif des géants du logiciel comme Oracle, IBM ou Microsoft ont contribué à ce mouvement. En gérant avec succès de nombreuses applications critiques, les systèmes de gestion de bases de données relationnelles ont prouvé leur fiabilité et gagné la confiance des directions informatiques.

Parallèlement, la technologie objet voit le jour au début des années 80 et laisse entrevoir de nombreux espoirs : ses capacités de modélisation très évoluées et son haut degré d'abstraction permettent une conception plus riche, plus flexible, plus modulaire, et plus évolutive. Non seulement la conception d'applications se trouve facilitée, mais il est désormais possible d'envisager la création de composants logiciels réutilisables. Dès lors, la technologie objet est en mesure de garantir un meilleur retour sur investissement par rapport au relationnel, lorsqu'elle est correctement utilisée. Néanmoins, eu égard notamment à sa complexité, à la mauvaise utilisation qui en a souvent été faite, et au manque de maturité des outils et des bases de données orientées objet, ce modèle n'a pas remporté le succès escompté. L'arrivée de nouveaux standards comme OQL (Object Query Language) au début des années 90 et de bases de données objet dignes de ce nom sur le marché n'a pas inversé la tendance : les gestionnaires de bases de données objet n'occupent que des marchés de niche (domaine scientifique, domaine des télécoms, etc.).

Dans la plupart des cas, les entreprises ont préféré renoncer aux avantages de l'objet et continuer à utiliser le modèle relationnel. Les raisons invoquées le plus fréquemment sont les suivantes :

- **l'importance de l'existant relationnel :** les investissements massifs réalisés dans les bases relationnelles (compétences, outils de développement, administration, etc.) ne sont parfois pas encore amortis, et l'hésitation des entreprises à faire le pas vers l'objet est légitime ;

- **le risque technologique :** les technologies objet sont relativement complexes à mettre en oeuvre, et le saut culturel engendré par son adoption présente un risque important. Les *middleware* d'utilisation des technologies objet (ORB, bases de données objet, etc.) remettent parfois en cause une partie de l'existant. Les entreprises restent également dubitatives quant à la maturité et à la fiabilité de la technologie objet, notamment en ce qui concerne les bases de données objet.

Cependant, le développement du Web a donné un second souffle aux technologies objet. Les entreprises sont en effet contraintes d'intégrer le Web à leurs systèmes d'information, et les standards *de facto* permettant de réaliser ce type d'architecture, J2EE et Microsoft DNA (maintenant .Net) fonctionnent à base de technologies objet. L'utilisation du modèle objet pour la conception d'entités métier (client, contrat, etc.) pose des problématiques épineuses, notamment celles de la persistance des attributs des objets et des transactions objet. En effet, à la différence des objets techniques (interface utilisateur, requêtes, etc.),

[†] Edward F. CODD, *A Relational Model of Data for Large Shared Data Banks* , in *Communications of the ACM,* vol. 13, issue 6, 1970.

les objets métier doivent être dotés de deux propriétés étroitement liées : persistance et transactionnel.

En parallèle, l'arrivée de nouvelles méthodes de notation comme UML et d'outils de conception associés fait la part belle à l'objet. Les entreprises se trouvent donc confrontées à un choix délicat : comment intégrer les technologies objet dans leurs architectures ?

Dans le monde Java/J2EE, un certain nombre de technologies et de produits sont apparus pour répondre à ces besoins et proposer de véritables outils de modélisation par objets métier. Citons parmi eux les normes de Sun Microsystems : EJB Entité et, plus récemment JDO, ainsi que des produits spécialisés comme Oracle9iAS TopLink. Ces différentes normes et outils apportent des éléments de réponse au problème de la persistance, qui est une nécessité pour l'implémentation des modèles métier.

# 1.1 Présentation du modèle relationnel

## 1.1.1 Caractéristiques principales

Le modèle relationnel se distingue par ses capacités élaborées de gestion des données, parmi lesquelles les plus remarquables sont :

- sa capacité à stocker de manière durable et fiable de très gros volumes de données
- la possibilité d'organiser les données de façon structurée et de définir des relations simples entre ces données
- la gestion de l'intégrité des données et des accès et mises à jour concurrentes
- la possibilité de manipuler les données de façon très rapide et sélective, à travers des requêtes exprimées à l'aide d'un langage spécifique (*DML – Data Manipulation Language*)

### Modélisation des entités

Dans le modèle relationnel, les données sont structurées selon une organisation bi-dimensionnelle. Les informations sont modélisées sous forme de *tables* (ou *relations*), qui se présentent sous la forme d'ensembles de *tuples*. Un tuple est lui-même constitué d'un ensemble d'*attributs* nommés, typés, et valués. On parle également de lignes ou d'enregistrements pour désigner les tuples, et de colonnes ou de champs pour désigner les attributs.

Prenons l'exemple d'un client d'une entreprise. Dans le modèle relationnel, l'entité client sera représentée par une table qui contiendra les différentes occurrences (ou tuples) correspondant à chacun des clients. Chaque tuple est constitué d'un ensemble d'attributs typés et valués. La table **Customers** ci-dessous illustre la structure bi-dimensionnelle du modèle relationnel :

| **id** | lastname | firstname | birthdate | address | zip | city |
|---|---|---|---|---|---|---|
| **1** | Durant | Jean | 01-MAR-70 | 22 rue Rouge | 75004 | PARIS |
| **2** | Dupont | Bill | 22-NOV-75 | 23 rue du Fond | 13008 | MARSEILLE |
| **….** | …. | …. | …. | …. | …. | …. |

Les seules opérations possibles sur un tuple donné d'une table sont :

- la création d'un tuple
- la lecture de la valeur d'un attribut de ce tuple
- la modification de la valeur d'un attribut de ce tuple
- la suppression de ce tuple

### Gestion de l'identité des données

Les tuples stockés dans les tables ne peuvent être identifiés qu'à partir des valeurs des différents attributs qui les composent. Afin de permettre d'identifier de manière unique chaque tuple, il est possible de désigner un ou plusieurs attributs comme *clé primaire*. Le système exige (et garantit) alors que les valeurs de clé primaire de chaque tuple soient distinctes.

De ce concept de clé primaire dérive celui de *clé étrangère*, qui permet de gérer les relations entre plusieurs tables : un champ de clé étrangère est un attribut qui contient la clé primaire d'un enregistrement d'une autre table. De cette manière, les tuples d'une table peuvent référencer des tuples d'autres tables.

Par exemple, on peut modéliser une table **Orders** qui contient les informations relatives aux commandes des clients. Outre sa clé primaire propre (**id**), chaque commande comporte également un champ de clé étrangère (*customerid*) qui fait référence à la clé primaire du client qui a passé cette commande :

| **Id** | *customerid* | orderdate | status |
|---|---|---|---|
| **1** | *1* | 10-DEC-02 | shipped |
| **2** | *1* | 15-DEC-02 | processing |
| **3** | *2* | 17-DEC-02 | processing |
| …. | …. | …. | …. |

### Un langage puissant de manipulation des données

Le modèle relationnel offre un outil concis et puissant pour la manipulation des données structurées sous formes de tables: il s'agit de l'algèbre relationnelle. Les opérations de l'algèbre relationnelle (projection, sélection, jointure, etc.) sont très synthétiques. En pratique, l'algèbre relationnelle s'utilise via divers langages, le plus répandu étant le langage SQL (Structured Query Langage).

Voici quelques exemples de requêtes SQL, qui illustrent la concision et la puissance de ce langage :

| Opération à réaliser | Requête SQL |
|---|---|
| Sélection des clients (nom, prénom, adresse, ville) qui habitent à Marseille | SELECT firstname, lastname, address, zip<br>FROM Customers<br>WHERE city = 'Marseille' |
| Sélection des commandes faites par le client de numéro 1 (à l'aide d'une jointure entre les tables *Customers* et *Orders*) | SELECT ordername<br>FROM Customers INNER JOIN Orders<br>ON Customers.id = Orders.customerid<br>WHERE Customers.id = 1 |
| Mise à jour du champ 'status' à la valeur 'shipped' pour l'enregistrement de la table Commande dont le champ id (clé primaire) égale 1. | UPDATE Orders SET status = 'shipped'<br>WHERE id = 1 |

**Des capacités d'abstraction limitées**

En revanche, le niveau d'abstraction offert par le modèle relationnel reste relativement bas. Certains systèmes offrent des fonctionnalités rudimentaires d'encapsulation, par le biais des *triggers*, qui sont des mécanismes permettant d'associer à chacune des opérations élémentaires (insertion, mise à jour, suppression d'un tuple) le déclenchement de traitements arbitraires exprimés en langage DML. Cependant, ces fonctionnalités restent limitées, et le modèle relationnel ne fournit aucun mécanisme permettant d'associer des *traitements* ou *comportements* dynamiques et arbitraires aux entités. La modélisation des traitements doit donc être réalisée séparément de la modélisation des données. Bien qu'il soit également possible d'implémenter certains traitements sous formes de *procédures stockées* intégrées à la base, les fonctionnalités offertes restent là aussi limitées, et on ne peut pas non plus parler ici réellement d'encapsulation. Le plus souvent, les traitements sont donc implémentés selon un modèle procédural, sous la forme d'applications externes accédant à la base pour manipuler les données. Cette séparation limite la réutilisabilité et la qualité des programmes. Il est a noter qu'au fil des versions les bases de données relationnelles se sont enrichies de fonctionnalités de représentation de types complexes héritées des concepts objets ; toutefois, la structure même des bases de données relationnelles ne permet qu'un support très partiel du modèle objet, et par conséquent ces fonctionnalités ne sont pas largement utilisées dans les applications d'entreprise.

### 1.1.2 La gestion transactionnelle

**Généralités sur les transactions**

Dans un système de base de données, une transaction est une séquence d'opérations élémentaires (lecture ou mise à jour de données sur une ou plusieurs tables) exécutée comme une seule opération indivisible :

- si l'ensemble des opérations est mené à terme avec succès, les modifications effectuées deviennent permanentes — la transaction est validée.
- si l'une des opérations échoue, les modifications partielles déjà effectuées sont annulées et l'état initial de la base de données est restauré — la transaction est annulée.

Par exemple, dans une application bancaire, un virement de compte à compte peut être modélisé comme une transaction constituée de deux opérations complémentaires et indissociables:

- débit du compte source d'une somme *X*
- crédit du compte cible de la même somme *X*

Afin de maintenir l'intégrité des données bancaires, il est primordial de garantir que soit ces deux opérations seront exécutées, soit qu'aucune ne le sera. En aucun cas ne doit exister de situation dans laquelle seule l'une des deux opérations est exécutée. On peut également enrichir cette transaction en y ajoutant une 3ème opération, préliminaire : la vérification de la disponibilité des fonds pour le virement sur le compte source. Si le solde du compte source est insuffisant pour permettre le virement, alors on doit forcer l'annulation de la transaction.

Plusieurs transactions peuvent être exécutées de façon concurrente. Tant qu'une transaction n'est pas menée à son terme, les modifications qu'elle entreprend restent invisibles aux autres transactions qui s'exécutent. Par exemple, plusieurs transactions peuvent potentiellement mettre à jour de façon concurrente un compte bancaire : virement, encaissement ou paiement de chèque, etc.

En fait, le système de gestion de base de données garantit aux transactions quatre propriétés fondamentales, désignées par le sigle ACID :

- **A**tomicité : les opérations constituant la transaction forment une unité indivisible : soit l'ensemble des opérations est mené à terme, soit aucune opération n'est effectuée.

- **C**ohérence : les transactions font passer le système de base de données d'un état initial cohérent à un état final cohérent, dans le respect des règles d'intégrité référentielle définies sur la base de données (cependant, au cours de l'exécution de la transaction, ces règles peuvent être temporairement enfreintes).

- **I**solation : les mises à jour effectuées au cours de l'exécution d'une transaction restent invisibles aux autres transactions. Les modifications ne deviennent visibles qu'une fois la transaction validée. En pratique cependant, cette propriété d'isolation peut être relaxée : les bases de données gèrent en effet plusieurs niveaux d'isolation des transactions, qui permettent de moduler le niveau de visibilité de données non validées entre transactions concurrentes.

- **D**urabilité : après l'aboutissement d'une transaction, les mises à jour effectuées deviennent définitives et ne peuvent plus être annulées.

Au sein des serveurs d'application, il existe un service particulier dédié à la gestion et à la coordination des transactions effectuées par les applications sur une ou plusieurs bases de données : le gestionnaire de transactions.

**Gestion de la concurrence d'accès :**

Pour assurer la gestion de la concurrence entre transactions simultanées, le système de gestion de bases de données a recours à des mécanismes de verrouillage sur les données en cours de modification. La pose d'un verrou par une transaction sur un ou plusieurs enregistrements empêche d'autres transactions de modifier ces données de façon concurrente. Typiquement, les bases de données offrent deux stratégies de verrouillage pour les transactions:

- **mode exclusif ou « pessimiste »** : un verrou est posé sur les données mises à jour pendant toute la durée de la transaction pour empêcher la modification et/ou la lecture concurrente des données; les données en cours de modification restent inaccessibles aux autres transactions en écriture (et potentiellement en lecture également) tant que la transaction n'est pas terminée. Plusieurs types de verrous peuvent être employés : verrou simple contre l'écriture concurrente, verrou contre l'écriture et la lecture concurrente, verrou personnalisé. L'avantage de ce mode est qu'il élimine le risque de perte de modifications tentées par d'autres transactions concurrentes: si les données sont déjà en cours de modification, toute transaction essayant de les modifier est annulée immédiatement (comportement de type « échec rapide », ou *fail-fast behavior* en anglais*)*. L'inconvénient majeur de ce mode est qu'il réduit considérablement le niveau de concurrence d'accès aux données, et peut donc entraîner un ralentissement important des performances.

- **mode partagé ou « optimiste »** : durant l'exécution de la transaction, aucun verrou n'est posé sur les données mises à jour, qui restent accessibles aux autres transactions. A la validation de la transaction, juste avant la mise à jour, le système doit par conséquent vérifier que les données manipulées n'ont pas été modifiées entre-temps par d'autres transactions. Si les données sont intactes, la validation de la transaction s'effectue, avec pose d'un verrou en écriture juste pendant la phase de validation. Si les données ont été modifiées entre-temps par une transaction

concurrente, la transaction est annulée et les modifications sont perdues. L'avantage de ce mode est que les données ne sont verrouillées que pendant des laps de temps très courts (phase de validation uniquement), ce qui favorise une haute concurrence d'accès. L'inconvénient principal est le risque d'échec des transactions et la perte des modifications qui peuvent se produire lorsque plusieurs transactions effectuent des mises à jour sur un même ensemble de données.

**Les différents modèles transactionnels :**

Le modèle de transactions que nous avons décrit correspond au modèle le plus simple : le modèle de transactions linéaires. Dans ce modèle, une transaction est une séquence linéaire d'opérations qui, soit sont toutes exécutées, soit ne le sont pas.

Il existe également un autre modèle qui offre un contrôle plus fin sur le déroulement des opérations: le modèle de transactions imbriquées. Dans ce modèle, il est possible d'imbriquer plusieurs transactions de manière hiérarchique. Ainsi, une transaction peut être constituée elle-même de « sous-transactions » qui peuvent être validées ou annulées indépendamment les unes des autres, sans affecter la validation ou l'annulation de la transaction englobante. Cependant, la validation ou l'annulation effective des sous-transactions dépend précisément de la validation ou de l'annulation de la transaction de niveau supérieur : si la transaction « parente » est validée, alors les transactions « filles » seront annulées ou validées en fonction de leur résultat individuel. En revanche, si la transaction « parente » est annulée, toutes ses sous-transactions le seront également.
Ce modèle permet une plus grande modularité et une plus grande complexité dans la conception des applications transactionnelles. Cependant, ce modèle n'est pas encore aussi largement supporté par les gestionnaires de bases de données ou par les serveurs d'application que le modèle de transactions linéaires.

**Les différents types de transactions :**

On distingue également deux types de transactions, les transactions locales d'une part, et les transactions globales, ou transactions distribuées, d'autre part. La distinction entre les deux s'opère sur le nombre de participants à la transaction :

Dans une transaction locale, seuls deux participants sont impliqués: l'application effectuant la transaction, et la base de données sur laquelle les mises à jour sont effectuées.

Une transaction globale implique plus de deux participants : par exemple, une application effectuant des mises à jour sur plusieurs bases de données, ou bien plusieurs applications coordonnées entre elles pour une mise à jour sur une ou plusieurs bases de données. La mise en œuvre de ce type de transactions nécessite l'utilisation d'un protocole spécifique pour la coordination des opérations entre participants: le protocole de validation à deux phases. Au cours de la première phase, le gestionnaire de transactions demande à chaque participant s'il est en mesure de valider la transaction. Si tous les participants sont prêts, la seconde phase est déclenchée, et le gestionnaire de transactions intime l'ordre à chaque participant de valider la transaction. Dans le cas contraire, si l'un des participants indique qu'il n'est pas en mesure de valider la transaction, le gestionnaire de transactions donne l'ordre à chaque participant d'annuler la transaction au cours de la seconde phase.

**Conclusion :**

Les transactions constituent donc un moyen robuste et fiable d'effectuer des opérations de mise à jour complexes sur une ou plusieurs bases de données. D'autre part, la gestion de la

concurrence entre transactions (propriété d'isolation) permet d'élaborer des scénarios complexes multi-utilisateurs.

De ce fait, les transactions se révèlent indispensables pour la réalisation de systèmes d'information et d'applications de gestion complexes.

# 1.2 Présentation du modèle objet

## 1.2.1 Caractéristiques principales

Alors que le modèle relationnel est centré sur les données, le modèle objet lui repose sur l'intégration étroite entre les données et les traitements.

Ce modèle se distingue par le niveau élevé d'abstraction qu'il offre, ainsi que par ses capacités avancées de modélisation. Parmi les caractéristiques les plus remarquables de ce modèle, on notera :

- l'intégration étroite réalisée entre les données et les traitements appliqués à ces données, appelée *encapsulation* : un objet combine un *état* (ensemble des valeurs courantes des *attributs* de l'objet, qui sont des données typées, et de ses *références*, ou *associations*, vers d'autres objets) et un *comportement* (les opérations possibles sur ces attributs et associations) en une entité de haut niveau.
- la souplesse de modélisation des données: le modèle objet offre le support d'un très large spectre de types de données, et permet d'en définir facilement de nouveaux qui peuvent être combinés avec des types existants. Les deux principes sous-jacents sont l'*héritage* (qui permet la *spécialisation* des objets), et la *composition* ou *agrégation* (qui permet l' *association* de différents objets).
- cette flexibilité au niveau de la modélisation des données se retrouve au niveau de la modélisation des traitements : le *polymorphisme* permet la *spécialisation* des traitements en fonction de la *spécialisation* des objets. De plus, il n'existe aucune restriction quant à la nature ou au nombre des traitements qui peuvent être définis sur les objets.

**Modélisation des entités :**

Dans le modèle objet, les données et les traitements s'appliquant à celles-ci sont intimement liés : un objet combine un état (les données) et un comportement (les opérations pouvant être effectuées sur ces données, appelées également *méthodes*) :


*fig. 1 : Schéma d'un objet*

Alors que le modèle relationnel limite les opérations pouvant être appliquées aux données à quatre primitives de base (création, lecture, mise à jour, destruction), le modèle objet permet de définir un nombre arbitraire d'opérations sur les données.

**Identité vs . Égalité des objets**

Contrairement aux tuples d'une relation, les objets ne disposent pas d'un concept de clé primaire permettant de les identifier de manière unique. L'identification d'un objet est simplement fonction de son *adresse* en mémoire. Ainsi, deux objets peuvent être égaux (les valeurs de leurs attributs sont égales) mais pas identiques (leurs adresses mémoire sont différentes). Ainsi, si l'identité objet est assurée par l'unicité des emplacements mémoire affectés aux objets, il n'existe aucun mécanisme permettant d'interdire l'égalité des attributs entre deux objets de même type comme cela est possible dans le modèle relationnel.

**Gestion des relations d'association entre objets**

Dans le modèle relationnel, des champs particuliers (clés étrangères), et parfois même des tables dédiées (« relations ») sont utilisés pour représenter les relations d'association entre plusieurs tuples.

Contrairement au modèle relationnel qui stocke des *valeurs* d'attributs de clés primaires pour établir les associations entre tuples, le modèle objet fait usage des *références* (l'identité des objets) pour maintenir les associations entre objets liés. Dans le cas de relations de type 1-n, une *collection de références* est maintenue dans un attribut de type vecteur, liste, ou tableau. Afin de récupérer les différents objets associés à un objet particulier, on emploie des algorithmes d'itération sur les collections et de parcours de graphe (*traversée* ou *indirection* de références).

**Des fonctionnalités de manipulation très limitées**

La faiblesse du modèle objet réside dans la manipulation des données. Celle-ci s'effectue par envoi de messages (appels de méthodes), et ce mécanisme s'avère nettement moins puissant que l'algèbre relationnelle.

Historiquement, la technologie objet a manqué de fondements théoriques et de standards au niveau des techniques de manipulation. Les développeurs ont alors adopté des approches très analytiques pour manipuler les ensembles d'objets (traitements par itérations) assez peu efficaces. Des standards pour les bases de données objet, comme le langage OQL (Object Query Language), sont nés beaucoup plus tard et n'ont pas été aussi massivement adoptés que le SQL dans le monde des bases de données relationnelles. D'autre part, les langages de requêtage objet sont beaucoup plus difficiles à maîtriser que leurs homologues relationnels.

### 1.2.2 La problématique de la persistance objet

Si le modèle objet offre des capacités de modélisation et d'abstraction avancées, il est en revanche handicapé par une grave limitation : il se focalise exclusivement sur les aspects dynamiques de l'exécution des programmes, sans couvrir réellement la problématique de la persistance des objets :

Les objets représentent des entités, des informations qui peuvent être partagées par plusieurs applications, et qui peuvent avoir une durée de vie très longue, dépassant la durée d'une session applicative. Cependant, les objets n'existent qu'en mémoire ; par conséquent, une fois la session applicative terminée, les objets sont détruits, et les informations qu'ils contiennent disparaissent avec eux.

Par contraste, dans le modèle relationnel, les données sont persistantes : une fois que la structure du schéma relationnel a été définie, les données qui y sont ajoutées restent accessibles durablement, tant qu'elles ne sont pas explicitement supprimées. La durée de vie des données est donc totalement indépendante du cycle de vie des sessions applicatives qui les manipulent.

Pour apporter ce caractère de durabilité des données au modèle objet, il convient donc de trouver un moyen de prolonger l'existence des objets au-delà de l'existence d'une session applicative. Pour désigner cette capacité des objets à survivre en dehors du contexte d'exécution applicatif, on parle de **persistance objet**. La problématique de la persistance objet consiste donc à élaborer et à mettre en œuvre les mécanismes permettant de sauvegarder de façon fiable et durable l'état des objets manipulés par une application.

L'état d'un objet étant constitué de l'ensemble des valeurs de ses attributs, il suffit donc de sauvegarder l'état de ces différents attributs sur un support de stockage permanent pour être en mesure de reconstituer l'état de l'objet ultérieurement. Cependant, plusieurs considérations viennent compliquer la simplicité apparente de cette tâche :

- la question du type des objets : chaque objet appartient à une classe donnée ; il est donc nécessaire de sauvegarder les informations relatives à l'appartenance de l'objet à une classe particulière
- la question des références vers d'autres objets : au-delà des types scalaires simples, les objets peuvent avoir comme attributs des références vers d'autres objets ; la représentation d'un objet ne s'apparente donc pas à un tableau de valeurs, mais à un graphe de valeurs. Sauvegarder un objet implique donc la sauvegarde complète de ce graphe, c'est à dire la sauvegarde de tous les objets référencés par cet objet.
- la question de l'identité objet : comme nous l'avons indiqué, contrairement aux enregistrements de bases de données, les objets ne disposent pas d'une identité propre, en dehors de leur adresse mémoire à l'exécution. Pour la recherche et la restauration ultérieure des objets, il est nécessaire de leur associer des informations permettant leur identification.

Techniquement, il existe plusieurs approches pour implémenter la persistance objet:

- sérialisation : comme nous l'avons exposé, la représentation d'un objet s'apparente à un graphe. La sérialisation consiste à définir une méthode permettant de « mettre à plat » cette représentation afin de la sauvegarder comme un flux d'information uni-dimensionnel (sériel) dans un fichier. On peut définir et utiliser des flux binaires (fichiers binaires), ou textuels (fichiers XML par exemple).
- mapping objet/relationnel : cette technique vise à définir et utiliser un schéma en base de données relationnelle pour assurer la persistance de l'état des objets. Elle consiste à définir précisément la structure d'une ou plusieurs tables qui seront utilisées pour sauvegarder les différents attributs d'un objet.
- persistance en base de données objet : le problème de la persistance objet a donné lieu au développement d'un nouveau type de bases de données : les bases de données orientées objet. Contrairement aux bases de données relationnelles qui emploient des structures bi-dimensionnelles (les tables), les bases de données objet utilisent les mêmes représentations que les objets : des structures hiérarchiques (graphes).

Au-delà du mécanisme de persistance utilisé, il est également nécessaire d'implémenter la **logique de persistance** qui va permettre aux applications d'effectuer la sauvegarde et la restauration de l'état des objets manipulés. C'est le rôle de la **couche de persistance** (ou **moteur de persistance**). Le moteur de persistance expose une API aux applications pour la création, la mise à jour, et la recherche d'objets sur le support de persistance. A ce moteur de persistance, il est également nécessaire d'associer un **outil de projection** des objets sur le support de persistance : le rôle de cet outil est de fournir le moyen de définir

les règles de transformation entre la représentation des objets en mémoire et la représentation de ces objets sur le support de persistance, avec la création des structures de données nécessaires.

La persistance objet est donc une problématique très complexe, qui nécessite la mise en œuvre de mécanismes spécifiques et sophistiqués.

## 1.3 Différences entre les modèles objet et relationnel

L'exposé des caractéristiques fondamentales des modèles relationnel et objet permet d'apprécier leurs grandes dissimilitudes, ainsi que les difficultés qui se posent lorsqu'on cherche à établir un pont entre ces deux mondes.

Pour caractériser les incompatibilités entre ces modèles, on a coutume de faire une analogie à l'électricité et de les regrouper sous le terme générique de *défaut d'adaptation d'impédance* (« impedance mismatch »). Les principaux obstacles à l'appariement entre les deux modèles qui se cachent derrière cette appellation sont les suivants :

- dans le modèle relationnel, les données sont organisées de manière tabulaire, sous forme matricielle (lignes et colonnes). Dans le modèle objet, les données sont organisées sous la forme d'un graphe d'objets (ensembles d'attributs). Toute la difficulté du mapping objet/relationnel consiste à effectuer la traduction entre ces deux représentations.

- le problème des types de données : le passage d'une représentation objet à une représentation relationnelle implique souvent une adaptation des types de données, et même des conversions de types. Les types de données disponibles et les restrictions sur les valeurs acceptables diffèrent entre les modèles relationnel et objet.

- le problème de l'héritage : le concept d'arbre d'héritage est étranger au modèle relationnel. Il est cependant possible d'accommoder une hiérarchie de classes de plusieurs manières : par exemple, en créant une table pour chaque classe dans la hiérarchie, avec les attributs correspondants, ou bien en créant une seule table contenant l'ensemble des attributs de toutes les classes de la hiérarchie.

- le problème de l'identité objet : dans le modèle relationnel, l'unicité des tuples est garantie par l'utilisation de clés primaires. Par contraste, dans le modèle objet, l'unicité des objets est uniquement fonction de l'unicité des références mémoire. Cette gestion d'identité et de références peut très vite devenir complexe et source d'erreur, notamment dans le cas de relations circulaires entre objets. D'autre part, lorsqu'un objet est sérialisé vers un support de stockage, l'identité de l'objet est perdue. Pour remédier à ce problème, on doit munir chaque objet d'un attribut de clé primaire, et mettre en œuvre un mécanisme logiciel garantissant l'unicité des objets vis à vis des valeurs de ces attributs de clé primaire.

- le problème des associations entre objets : de la même façon, dans le modèle relationnel, les associations entre entités sont gérées à l'aide de champs de clés étrangères, et de tables d'association dans le cas de relations 1-n bi-directionnelles, ou de relations n-n. Dans le modèle objet, les relations entre objets sont maintenues au moyen de références mémoire. Pour maintenir les associations entre objets lors de la persistance de ceux-ci en base relationnelle, il faut traduire ces références en valeurs

d'attributs de clés étrangères. Dans le cas d'associations bi-directionnelles de type n-n, il est nécessaire de créer une table supplémentaire pour représenter les relations d'association.

Ces différents obstacles expliquent en partie la très grande complexité de la problématique d'intégration entre les modèles relationnel et objet. L'expérience prouve que les considérations de maintien de l'intégrité des données entre les deux représentations, de gestion des transactions et d'optimisation de la gestion mémoire du graphe d'objets sont extrêmement difficiles à résoudre.

# 1.4 Intégration des modèles objet et relationnel

## 1.4.1 Différentes approches pour la persistance objet

Le risque technologique présenté par le modèle objet a souvent poussé les entreprises à opter pour une approche 100% relationnelle. Cependant, de plus en plus, de nombreux facteurs obligent les entreprises à intégrer les technologies objet dans leurs nouvelles applications.

En premier lieu, l'utilisation des langages objet (Java, C#) s'est généralisée dans les standards de développement et de déploiement d'architectures n-tiers (J2EE et .Net). Les technologies objet prennent désormais entièrement en compte la multiplicité des interfaces utilisateurs (Windows, HTML et autres) et la diversité des ressources de l'entreprise (SGBD, bases documentaires, messageries, annuaires, ERP et autres). Cette diversité et cette richesse fonctionnelle décuplent les possibilités de ces architectures, et pour peu qu'elles soient bien maîtrisées, apportent des gains substantifs de productivité, de maintenance, et de performances.

Cependant, la problématique de la persistance des objets métier, et de l'intégration du modèle objet avec les systèmes d'information relationnels reste la difficulté la plus sensible, la plus complexe, et sans doute la plus largement sous-estimée dans le développement d'applications d'entreprise. En réalité, plusieurs approches sont envisageables pour mettre en œuvre une solution de persistance d'entités métier :

- **conserver l'approche relationnelle**, en renonçant purement et simplement aux avantages apportés par la modélisation objet. C'est conserver un modèle certes éprouvé, mais un modèle dissociant traitements et données, ce qui ne facilite pas la réalisation d'applications complexes dans une architecture multi-tiers. Cette approche peut cependant être justifiée pour des applications patrimoniales d'informatique de gestion réalisant principalement des traitements procéduraux, ou effectuant des requêtes relationnelles très complexes et qui sont inadaptées à une migration vers un modèle objet.

- **utiliser une base de données objet pure** : bien qu'elle permette de tirer parti des atouts de la modélisation par objets métier, cette approche remet fortement en cause les compétences préalablement acquises et la compatibilité avec les bases de données relationnelles existantes. D'autre part, l'adhérence relative aux standards pour les bases de données objet induit une dépendance face aux produits proposés par les éditeurs ; ces produits étant basés sur des architectures propriétaires et n'étant généralement pas inter-opérables. Cette approche, potentiellement risquée, est cependant envisageable dans des scénarios où l'approche objet s'impose en raison de

la complexité du modèle de données et de besoin de performances extrêmes. C'est le cas par exemple de certaines applications dans des domaines scientifiques ou financiers manipulant des données complexes à caractère dynamique.

- **utiliser une architecture hybride objet-relationnelle** : avec cette approche, on peut tirer parti des avantages de la modélisation par objets métier tout en bénéficiant de la fiabilité et des performances des bases de données relationnelles et en conservant, le cas échéant, la compatibilité avec les bases de données relationnelles existantes. Cette solution marie avec élégance applications anciennes et nouvelles, non seulement en termes d'architecture, mais aussi en termes de compétences investies dans les deux modèles (développement, administration, exploitation… etc.). Cette approche est d'autant plus intéressante que la majorité des entreprises possède un héritage relationnel fort (culture, compétences, produits utilisés, relations commerciales, etc.) et que ce modèle hybride est parfaitement adapté à la majorité des applications développées par les entreprises.

Cette dernière approche est le principal sujet de cette étude.

### 1.4.2 Les différents scénarios pour l'intégration objet-relationnel

Dès lors que l'on décide d'opter pour l'intégration relationnel-objet, on peut envisager deux types de scénarios :

- **intégration avec un existant relationnel**: dans ce scénario, on souhaite interfacer (et éventuellement enrichir) un existant relationnel avec de nouvelles applications objet. Dans ce cas de figure, on a souvent besoin de conserver le modèle relationnel initial intact, ou de limiter l'impact éventuel du mapping sur le schéma de base de données afin d'en préserver la structure, dont dépendent potentiellement nombre d'applications (reporting, infocentre, applications de gestion, traitements batch, etc.). Dans ce cas, une difficulté supplémentaire durant la phase de mapping objet-relationnel consistera à essayer de calquer plus ou moins le modèle objet sur le modèle existant, et éventuellement à ajouter de nouvelles tables pour faciliter la projection du modèle objet (notamment, pour la gestion de l'héritage et des relations d'association entre objets)

- **intégration sans existant relationnel préalable** : deuxième cas de figure, il n'y a aucun existant relationnel. Typiquement, ce scénario correspond à la mise en place d'applications et/ou de systèmes d'information entièrement nouveaux. Dans ce scénario, il n'existe aucun schéma de base de données existant ou prédéfini, et on a donc entière latitude pour définir un modèle relationnel totalement adapté à la projection du modèle objet.

# 1.5 Caractéristiques d'un mécanisme de persistance objet-relationnel transparent

## 1.5.1 Principe de la persistance transparente

A la problématique de l'intégration entre les mondes relationnel et objet s'ajoutent des considérations qui vont caractériser directement la valeur d'un moteur de persistance objet-relationnel: la richesse fonctionnelle de la couche de mapping et du moteur de persistance, et la transparence des mécanismes de persistance vis à vis du modèle objet et des applications, qui doit garantir le découplage entre applications, moteur de persistance, et base de données.

Les bases d'une solution de « persistance transparente », que l'on appelle également « persistance orthogonale » ont été établies de manière théorique :

- **Persistance orthogonale :**

  Le principe fondateur de la persistance orthogonale est le *découplage total* entre l'application et l'infrastructure de persistance. En d'autres termes, l'application et le moteur de persistance doivent être entièrement indépendants.

  L'intérêt de ce concept est de masquer au développeur les mécanismes de transformation entre le graphe d'objets en mémoire et sa représentation sur le support de stockage (base de données relationnelle ou autre). Ainsi, le développeur n'a pas à se préoccuper de la logique de persistance des objets métier au sein de ses applications. Les mécanismes de persistance étant transparents, et indépendants du support de persistance sous-jacent, aucun code technique ne vient polluer le code métier des applications.

  Afin de démontrer ce caractère de persistance orthogonale, une application doit posséder les caractéristiques suivantes :

  - **principe d'orthogonalité** vis-à-vis des types de données : tout objet, quel que soit son type, doit pouvoir être persisté. De façon inverse, tout objet doit pouvoir être déclaré comme étant « temporaire » ou « transitoire » (*transient*), c'est à dire non-persistant.

  - **principe de transparence** vis-à-vis du mécanisme de persistance: du point de vue de l'application, il ne doit y avoir aucune différence entre la manipulation d'objets persistants et celle d'objets non persistants.

  - **principe de persistance transitive**, ou de persistance par accessibilité (*persistence by reachability*) : ce principe stipule qu'un objet non-persistant que l'on référence à travers un objet persistant doit automatiquement devenir persistant. En effet, il est nécessaire de sauvegarder l'ensemble du graphe objet référencé par un objet persistant « racine » afin de pouvoir recréer cet objet ultérieurement. De la même façon, si un objet persistant est supprimé, alors les objets référencés par cet objet doivent également l'être s'ils ne sont pas référencés par d'autres objets existants.

- **principe de persistance par héritage** : si les objets d'une classe particulière sont persistants, alors tout objet de l'une des sous-classes de cette classe doit pouvoir être persisté.

Ces définitions théoriques très strictes doivent être cependant nuancées. Concernant l'orthogonalité des types de données en particulier, il existe souvent en pratique des objets qui ne peuvent pas être rendus persistants: des objets « système », tels que ceux représentant un canal d'entrée-sortie, un processus ou un thread, une exception, un proxy vers un objet distant, etc.

A ces considérations théoriques sur le fondement de la persistance transparente, il faut ajouter les exigences pratiques auxquelles doit satisfaire toute couche de persistance.

# 1.6 Rôle et obligations de la couche de persistance

Comme nous l'avons exposé précédemment, le rôle principal du moteur de persistance est d'implémenter, de façon transparente, la logique de persistance des objets des applications.

La couche de persistance doit être la moins intrusive possible au niveau du modèle objet et du schéma de base de données : en particulier, elle ne doit pas nécessiter l'ajout de code dans les objets métier, ou de modifications particulières dans les tables de bases de données existantes.

## 1.6.1 L'outil de projection du modèle objet

A la couche de persistance doit être associé un outil de projection qui va permettre de définir les structures utilisées pour la persistance des objets, ainsi que les règles de transformation entre la représentation des objets en mémoire et leur représentation sur le support de persistance. La description de ces mappings doit être aisée pour les développeurs. Il est par conséquent nécessaire de fournir un outil graphique d'assistance offrant flexibilité et productivité.

En particulier, l'outil de projection doit proposer une grande variété d'options pour pouvoir s'adapter aux divers scénarios de modèles objet :

- la flexibilité quant au choix des structures de données pour le mapping : utilisé avec une base de données relationnelle, l'outil doit offrir une variété d'options pour définir les structures utilisées pour la persistance des objets :
  - la possibilité de répartir les attributs d'un objet dans plusieurs tables (mapping multi-table)
  - la possibilité d'utiliser des champs binaires (BLOB) pour sauvegarder l'état d'objets dépendants
  - des mécanismes de conversion de types entre types objet et types de bases de données

- la gestion de l'identité objet : l'outil de projection doit proposer plusieurs alternatives à la gestion de l'identité objet, notamment en ce qui concerne la définition d'identifiants objet :
  - le support des clés primaires et étrangères composites (clés composées de plusieurs attributs – ex : pour un objet « article d'une commande », la clé primaire peut être composée du numéro de la commande et d'un numéro de produit)

  - la gestion automatisée de clés primaires « techniques » (surrogate keys). On distingue en effet les clés primaires naturelles, qui possèdent une signification métier (ex : un numéro de sécurité sociale), des clés primaires techniques qui ne revêtent aucune signification. Les clés primaires techniques sont employées pour apporter une identité unique aux objets (ou tuples relationnels) qui ne possèdent aucun attribut susceptible de jouer ce rôle (ex : un objet « Adresse postale »), ou lorsqu'on souhaite éviter d'utiliser en tant que clé primaire des attributs métier susceptibles d'évolution (ex : un numéro de client dont le format sera sans doute amené à être modifié).
  - la gestion automatique de séquences pour l'attribution des valeurs de clés primaires

- la gestion des relations d'association et d'héritage : l'outil de projection doit permettre de traiter tous les types de relations entre objets :
  - le support de l'héritage : possibilité d'utiliser une ou plusieurs tables pour la persistance d'un arbre d'héritage
  - le support des associations 1-1, 1-n, et n-n entre objets, qu'elles soient uni-directionnelles ou bi-directionnelles

### 1.6.2 Accès aux objets et gestion de leur cycle de vie :

La couche de persistance doit prendre en charge l'instanciation (création, recherche) des objets requis par l'application depuis le support de persistance. L'interface entre l'application et le mécanisme de persistance doit être réalisée au niveau objet, et non au niveau base de données ou requête SQL. la couche de persistance doit gérer les connexions entre les sessions applicatives, la base de données et l'infrastructure du serveur d'application, de manière transparente. Enfin, la couche de persistance doit assurer l'unicité de l'identité objet.

L'instanciation, la recherche, et la mise à jour des objets, ainsi que le transit de ces objets entre application et support de persistance, sont des opérations très coûteuses en temps d'accès. Le mécanisme fondamental d'optimisation des performances est l'utilisation d'un cache objet, qui va permettre de réduire les accès au support de persistance en conservant en mémoire les objets les plus fréquemment accédés.

**Cache objet transactionnel :**

Le cache objet, constitué d'un espace mémoire pour le stockage d'objets, et d'un mécanisme de gestion de cet espace, permet de conserver en mémoire une partie des objets stockés sur le support de persistance. Lorsque l'application demande le chargement d'un objet, cet objet est chargé depuis le support de persistance, et stocké dans la mémoire cache. Si l'application accède ultérieurement à cet objet, la couche de persistance peut utiliser la copie de l'objet en cache pour satisfaire la demande de l'application, ce qui évite un accès à la base de données. Au fur et à mesure de l'exécution, les objets chargés par l'application remplissent le cache. Plusieurs techniques existent pour gérer efficacement le cache : on peut choisir de n'y conserver que les objets les plus récemment utilisés, ou bien ne conserver les objets que tant qu'ils sont référencés par l'application.

A travers la couche de persistance, l'accès aux objets doit être réalisé de façon transactionnelle, afin de permettre la concurrence d'accès aux objets et d'assurer les propriétés transactionnelles de base lors des opérations de mise à jour: atomicité, cohérence, isolation, et durabilité. Par conséquent, le cache objet doit lui-même être transactionnel, et assurer la gestion des accès concurrents aux objets à travers l'utilisation de stratégies de verrouillage paramétrables.

### 1.6.3 Techniques d'optimisation

En complément du mécanisme de cache, la couche de persistance doit assurer l'optimisation des interactions avec la base relationnelle. A cet effet, plusieurs techniques peuvent être mises en œuvre pour réduire le nombre et la complexité des accès au support de persistance.

**Techniques d'optimisation pour la lecture et le requêtage des objets :**

La lecture d'un objet, si elle peut sembler simple à première vue, est en réalité un processus complexe qui nécessite la prise en compte de divers aspects. En effet, un objet peut comporter des attributs simples (primitives du langage), mais aussi des objets dépendants, et des références vers d'autres objets avec lesquels il entretient des relations d'association, et qui eux-mêmes peuvent référencer d'autres objets. En d'autres termes, l'objet de départ n'est que la partie visible de l'iceberg : il constitue le sommet d'un graphe constitué de tous les objets qui sont accessibles directement ou indirectement à travers lui, par navigation de références.

Pour optimiser le processus de chargement du graphe objet en mémoire, il est important de déterminer quels sont les objets qu'il est primordial de charger en mémoire, et quels sont ceux qu'il est superflu de charger. La décision de charger tel ou tel objet, ou de charger les objets jusqu'à une « profondeur » donnée, doit pouvoir être contrôlée en fonction du contexte applicatif et de la performance recherchée. Il existe également plusieurs techniques d'optimisation qui peuvent être mises en œuvre pour optimiser les accès à la base nécessaires au chargement des objets.

Dans le cas le plus général, on peut envisager deux approches simples :

- une première approche naïve consiste à charger tous les objets de façon unitaire, au fur et à mesure des accès à ces objets par l'application par navigation de références. Cette approche se révèle coûteuse en performances : un accès à la base doit être effectué à chaque fois que l'application accède pour la première fois à l'un des objets du graphe.

- une seconde approche toute aussi naïve consiste à charger systématiquement l'ensemble du graphe objet. Les accès à la base sont effectués en bloc à la lecture de l'objet « racine », et la navigation ultérieure à travers le graphe ne nécessite aucun appel supplémentaire. Cependant, le temps de chargement du graphe objet peut être très long (dans le cas de graphes complexes). De plus, l'application n'accède en général qu'à un sous-ensemble du graphe objet complet, et la lecture des objets non utilisés entraîne donc un gaspillage de ressources (en termes de temps d'accès et d'occupation mémoire).

Il est donc important que la couche de persistance fournisse à l'application le moyen d'exercer un contrôle fin sur le chargement des objets. En particulier, il est souhaitable que l'application puisse spécifier des règles particulières pour le chargement des objets qui correspondent à des **scénarios pré-établis** :

- **objets à forte accointance** : certains objets peuvent être liés par une relation d'association très étroite, on les appelle des **objets à forte accointance**. Ce peut être le cas par exemple d'un objet « Commande » et des objets « Lignes de commande » associés. En général, si l'application manipule l'objet « Commande », elle accède la plupart du temps aux objets « Lignes de commande » associés. Il convient donc de spécifier que tous les objets « Ligne de commande » associés à un objet « Commande » doivent être chargés en même temps que l'objet « Commande ».

- **lecture différée** (*lazy loading* ou *just-in-time reading*) ou **indirection** : en revanche, d'autres objets, dans une relation 1-1, 1-n, ou n-n, peuvent se trouver dans une relation plus lâche. Dans une application de réservation aérienne, cela peut être le cas par exemple avec un objet « Vol » et la liste des passagers qui sont enregistrés sur ce vol (collection d'objets « Passager »). La recherche d'horaires et de disponibilités sur les « vols » ne nécessite généralement pas l'accès à la liste des passagers. Dans ce cas, le chargement de l'objet « Vol » ne doit donc pas entraîner le chargement des objets « Passager » associés. On emploiera donc dans ce cas un mécanisme de **lecture différée** (*lazy loading* ou *just-in-time reading* en anglais), également appelé mécanisme d'**indirection** : au chargement de l'objet « Vol », les objets associés « Passager » ne seront pas chargés ; la lecture de ces objets sera retardée jusqu'au moment où l'application accèdera explicitement à ces objets par navigation de référence. Le postulat validant l'utilisation de cette technique étant que cette situation ne se produira qu'en de rares occasions, et que dans le cas général, des lectures superflues d'objets non utilisés seront évitées. L'implémentation de ce mécanisme d'indirection peut dans certains cas être effectuée de façon entièrement transparente pour l'application.

- **lecture jointe** (*joined reading*) : pour des objets à forte accointance dans une relation d'association 1-1, on peut optimiser le chargement en ne réalisant qu'un seul accès (au lieu de deux) à la base de données, en utilisant une clause de jointure entre les différentes tables où sont stockés les états de ces objets.

- **lecture par lot** (*batch reading*) : le mécanisme de lecture par lot s'apparente au mécanisme de lecture jointe, et est utilisé pour optimiser la lecture de *n* objets ayant chacun une stricte relation d'association 1-1 avec un autre objet. Par exemple, lecture de *n* objets « Client » et des *n* objets « Adresse» associés. Ce scénario requiert au minimum n+1 accès à la base : 1 accès pour lire l'ensemble des objets « Client », et *n* accès pour lire les *n* objets « Adresse » associés. Or, il est possible d'optimiser les accès de façon à ne réaliser que deux (1+1) accès : 1 accès pour lire l'ensemble des objets « Client », et 1 seul accès pour lire l'ensemble des objets « Adresse » associés. Le gain en performance réalisé est donc de *n-1* accès.

**Techniques d'optimisation pour la mise à jour des objets:**

A l'instar des opérations de lecture, les opérations de mise à jour sur les objets peuvent être accélérées par l'utilisation de certains procédés.

Lorsqu'un graphe objet en mémoire est modifié, le gestionnaire de persistance doit répercuter les modifications sur la base de données. Là aussi, l'objectif principal est de minimiser le nombre d'accès à la base, ainsi que le volume des informations échangées.
Pour atteindre cet objectif, le gestionnaire de persistance doit non seulement déterminer quels objets ont été modifiés, mais également être capable, au niveau de chaque objet, de différencier les attributs modifiés des attributs inchangés. Pour ce faire, le gestionnaire de persistance peut associer à chaque objet et à chaque attribut un indicateur de modification (*dirty flag*) : lorsqu'un attribut est modifié, les indicateurs correspondant à cet attribut et à l'objet sont positionnés. Lors de la mise à jour effective en base de données, seuls les changements sur les attributs modifiés sont répercutés.
D'autre part, afin de favoriser un haut niveau de concurrence d'accès, les opérations de mise à jour doivent de préférence être réalisées à l'aide de transactions de courte durée fonctionnant selon le mode de verrouillage optimiste.

### 1.6.4 Fonctionnalités de requêtage

Grâce à la couche de persistance, les applications vont avoir accès à de gros volumes de données. Il est donc nécessaire de fournir aux applications des mécanismes de recherche sophistiqués sur les objets, implémentés à travers un langage de requêtage objet.

En premier lieu, ce langage doit permettre le parcours sans restriction du graphe objet, et l'utilisation d'expressions conditionnelles et de clauses de tri.

Enfin, plusieurs options doivent être proposées pour l'expression des requêtes sur les objets :

- la possibilité de définir des requêtes statiques paramétrables, définies lors de la phase de développement
- la possibilité d'exécuter des requêtes dynamiques paramétrables (déterminées à l'exécution)
- le support des requêtes polymorphes : possibilité d'effectuer une requête sur une classe qui renvoie des objets de la classe en question ainsi que des objets de toutes les classes dérivées

### 1.6.5 Flexibilité pour le développement et pour le déploiement

La mise en œuvre de la couche de persistance doit s'intégrer naturellement dans le processus général d'ingénierie de l'application.

Au cours du développement d'applications utilisant un modèle d'objets persistants, on se retrouve souvent confronté à des problèmes de performance. Outre les diverses techniques d'optimisation apportées par le moteur de persistance, il est également souhaitable que le moteur de persistance offre des fonctionnalités de *suivi des performances* (*monitoring)* à l'exécution. Par exemple, le moteur de persistance peut établir, pour chaque opération de lecture, mise à jour, ou requête sur les objets, des statistiques sur le nombre et le volume des accès au SGBD, les temps d'exécution correspondant à chaque opération, et les éventuels problèmes de contention sur les transactions (collisions entre plusieurs transactions accédant aux mêmes données).

Au niveau du déploiement, la couche de persistance doit idéalement être utilisable indépendamment du modèle d'architecture de l'application : architecture 2 tiers, 3 tiers, 4 tiers, et d'intégration avec les serveurs d'application.

# 2. J2EE et les architectures multi-tiers

Le développement d'applications peut s'inscrire au sein d'une variété d'architectures logicielles, et les mécanismes de persistance objet doivent par conséquent pouvoir être intégrés à différents modèles d'architecture.

Dans ce chapitre, nous présenterons les principaux modèles d'architecture, et étudierons en détail les services et fonctionnalités apportés par J2EE dans le cadre des modèles d'architecture multi-niveaux. Nous nous intéresserons ensuite à la localisation et à l'intégration des mécanismes de persistance au sein de chacune de ces architectures, en évoquant les différentes alternatives qui peuvent être considérées.

## 2.1 Typologie des architectures multi-tiers

Le concept d'architecture multi-tiers (ou « n-tiers ») propose de découper une application en plusieurs couches logiques spécialisées chacune dans une fonction précise :

- **logique de présentation** : présentation des informations à l'utilisateur, interface de saisie.

- **logique de navigation** : gestion du parcours de l'utilisateur entre les différentes parties de l'application.

- **logique métier** : implémentation des traitements directement liés au métier ; par exemple, calcul d'une prime d'assurance, opération de virement bancaire, etc.

- **logique de persistance** : implémentation des mécanismes permettant de sauvegarder les objets manipulés par l'application sur un support de persistance (typiquement, dans un SGBD), et gestion des accès à ces données et des mises à jour.

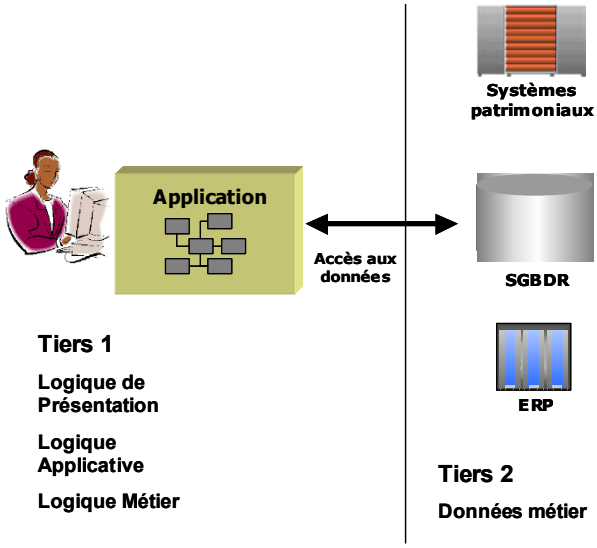
Les couches logiques d'une architecture *n* tiers sont hébergées sur une ou plusieurs couches physiques de l'infrastructure ; une application à architecture logique n-tiers peut être déployée sur toute infrastructure physique constituée de *p* tiers, avec $p \leq n$.

Les architectures multi-tiers ont évolué à partir de l'architecture centralisée des mainframes, dans lesquelles les parties applicatives (traitements), clientes (présentation et navigation), ainsi que les données du système d'information étaient centralisées et situées physiquement sur un même serveur.

### ❑ Les architectures 2 tiers (client-serveur)

L'âge du client-serveur a succédé au règne du mainframe vers le milieu des années 80. L'architecture client-serveur a en quelque sorte inauguré le principe des architectures deux tiers. Dans ce modèle, les traitements sont séparés entre le Client (station de travail utilisateur) et le Serveur (un mainframe ou un serveur puissant). Le Client prend en charge l'ensemble des tâches liées à la présentation (le plus souvent à travers une interface graphique), à la logique applicative (navigation), ainsi que, très souvent, une grande partie

de la logique métier également. Quant au serveur, son rôle est d'héberger les données du système d'information (typiquement, dans un SGBD) et de traiter les requêtes en provenance du Client :



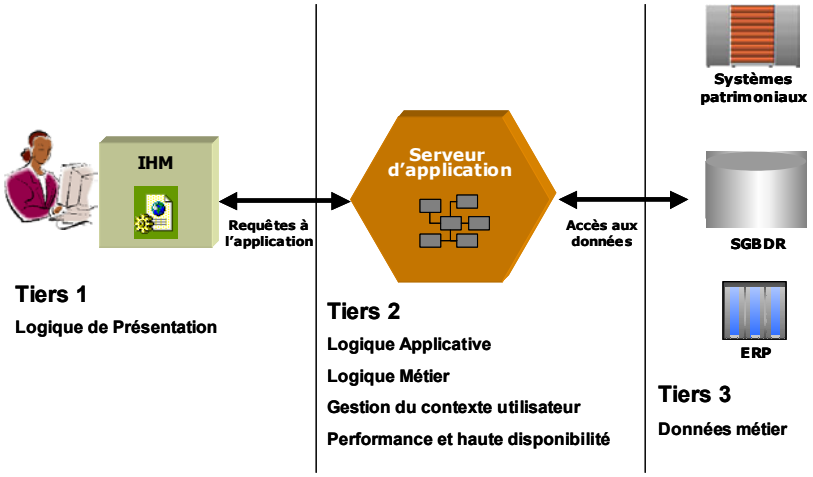

*architecture 2 tiers client-serveur*

Ce modèle permet une approche de développement modulaire, et l'emploi d'interfaces graphiques sophistiquées, mais souffre en revanche d'une lourdeur de déploiement problématique (la mise en production de nouvelles versions des applications nécessite la mise à jour de l'ensemble des postes clients).

## ❑ Les architectures 3 tiers

Les architectures trois tiers, introduite au début des années 90 avec le client serveur de seconde génération, ont ensuite affiné ce modèle en séparant la partie présentation des parties applicative et métier. L'intérêt de cette séparation est que le code applicatif et métier est totalement indépendant des impératifs de la partie présentation (comment l'information doit-elle être présentée, et sur quel type de terminal client). L'architecture trois-tiers classique actuelle, à base des technologies de serveurs Web et d'interfaces CGI pour les traitements, se révèle éminemment plus portable et maintenable que le client-serveur, fonctionne sur plusieurs types de plates-formes, et permet l'équilibrage de charge des requêtes clientes vers des serveurs multiples. L'implémentation de la sécurité est facilitée, du fait que la totalité de la logique métier est désormais déportée hors de la partie client, et les coûts de maintenance sont réduits de façon substantielle . En revanche, l'implémentation des fonctionnalités des services techniques (sécurité, gestion transactionnelle, accès à la couche données) dans le tiers intermédiaire  reste extrêmement coûteuse.

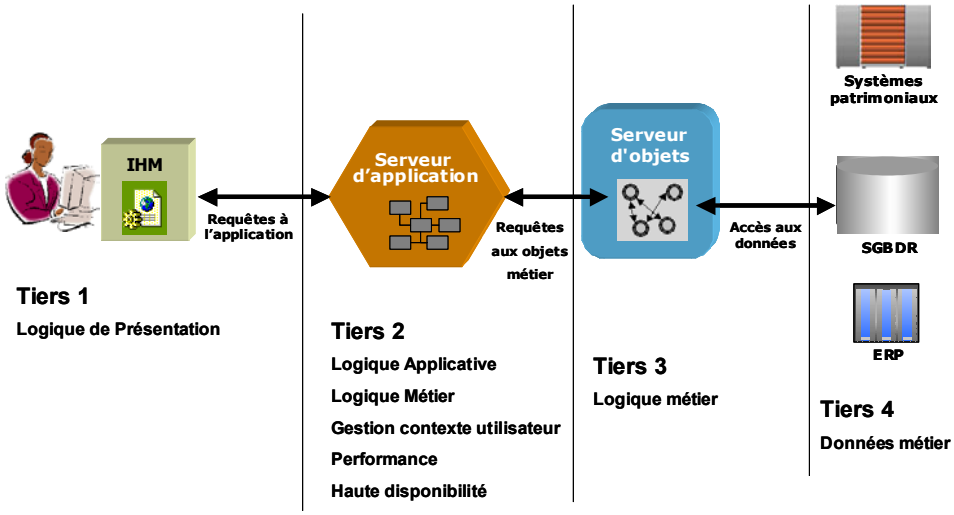
L'émergence des serveurs d'application intégrant un environnement d'exécution pour les applications, et qui prend en charge l'ensemble des problématiques de fourniture de services (sécurité, transaction, accès aux données) a permis de résoudre ces limitations en intégrant des services techniques de bas niveau pour offrir des fonctionnalités de très haut niveau aux applications. Le serveur d'application agit comme *middleware* entre les systèmes d'information de l'entreprise et des clients qui accèdent à ces données. Le code métier est stocké sur le serveur d'application, et est déployé et géré de manière centralisée. L'application peut être rendue accessible à un très grand nombre de clients hétérogènes (clients légers de type navigateur Web, clients lourds de type application graphique Windows, Corba, ou Java, et terminaux mobiles de type téléphone cellulaire ou PDA).




*architecture 3 tiers avec client léger et serveur d'application*

## ❑ Les architectures 4 tiers

Enfin, ultime raffinement de l'architecture n-tiers, l'architecture quatre tiers introduit un tiers spécifique dédié au modèle métier : le serveur d'objets. Le serveur d'objets permet de modéliser la logique métier (processus) et les données métier sous forme d'objets métier.




*architecture 4 tiers avec client léger, serveur d'application et serveur d'objets*

Afin de faciliter le développement de la logique métier et de décharger le développeur d'une partie des aspects techniques de bas-niveau, Le serveur d'objets fournit, de façon transparente, un ensemble de services pour la gestion des objets métier. Les services essentiels sont :

- **la gestion du cycle de vie des objets** : le serveur d'objets intègre des fonctionnalités de base permettant la création, la recherche, la manipulation, et la destruction des objets.

- **la gestion de la persistance** : synchronisation de l'état des objets sur un support de persistance (base de données), afin d'assurer la sauvegarde durable de l'état des

objets. De cette façon, la durée de vie des objets n'est pas limitée à la durée de vie d'une session applicative.

- **la gestion des transactions** : intimement liée à la gestion de la persistance, la gestion transactionnelle des objets permet d'assurer l'intégrité des données et de gérer la concurrence d'accès. Le service de gestion transactionnelle assure aux objets le bénéfice des propriétés transactionnelles de base : atomicité, cohérence, isolation, et durabilité des opérations de mise à jour.

- **la gestion de la sécurité** : le serveur d'objet inclut une infrastructure de sécurité perfectionnée qui permet la mise en place d'un mécanisme d'authentification et de contrôle d'accès aux objets depuis les applications clientes. La définition de permissions sur les opérations de lecture, de mise à jour, et sur les appels aux traitements métier permet de définir des restrictions d'accès très fines basées sur la définition de groupes d'utilisateurs et de rôles logiques.

- **la gestion de la montée en charge** : le serveur d'objets inclut différents mécanismes (pools d'objets, cache transactionnel, etc.…) qui permettent d'améliorer les performances des applications accédant de manière concurrente aux objets.

- **la gestion de la distribution, du clustering, de la reprise sur pannes et de la transparence de localisation des objets**: le serveur d'objet intègre des fonctionnalités de distribution des objets sur plusieurs machines. Ces fonctionnalités permettent non seulement de mieux gérer la montée en charge en autorisant la répartition des requêtes des applications clientes sur plusieurs machines, mais également la mise en place de mécanismes de reprise sur panne (failover). Afin d'assurer la transparence de localisation des objets distribués, le serveur fournit un service de nommage qui permet d'accéder à un objet de manière transparente, indépendamment de sa localisation physique.

**Adéquation des différents types d'architecture aux applications**

Bien que le modèle d'architecture 4 tiers soit le plus abouti et le plus riche en fonctionnalités, il ne constitue pas forcément la solution à chaque situation applicative. La sélection d'un type d'architecture particulier (deux tiers, trois tiers, ou quatre tiers) dépend souvent en priorité des besoins et des spécificités de chaque application : souvent, l'accès à un même système d'information peut être effectué à travers plusieurs applications basées sur des modèles d'architecture différents. En fonction de chaque scénario applicatif, on mettra donc en œuvre un type particulier d'architecture.

Il reste cependant que les architectures 3 ou 4 tiers, de part les services de haut-niveau qu'elles procurent, peuvent contribuer à fortement réduire la complexité et la durée des développements. Il faut également rappeler que certaines fonctionnalités de haut-niveau, dont la prise en charge de la persistance, ne sont parfois, suivant le standard employé, disponibles qu'en architecture 3 ou 4 tiers.

## 2.2 Rappels sur le standard J2EE

J2EE (Java 2 Enterprise Edition) est une norme qui définit et standardise les différentes composantes (environnements d'exécution, services d'architecture, modèle de programmation, et APIs) nécessaires à la réalisation d'architectures Java multi-tiers. Ces spécifications sont implémentées au sein des serveurs d'application J2EE.

J2EE définit un certain nombre de services accessibles par le biais d'APIs normalisées. La plupart de ces services répondent à des problématiques récurrentes rencontrées dans les applications d'entreprise, notamment l'accès aux applications patrimoniales, aux middlewares asynchrones, aux annuaires, ainsi que la gestion transactionnelle et la gestion de la sécurité.

En ce qui concerne la problématique de la persistance objet, J2EE n'apporte pas de réponse spécifique. Cependant, J2EE propose une architecture de composants métier transactionnels et distribués : la norme EJB. Ces composants sont hébergés au sein d'un serveur d'objets spécifique : le conteneur EJB. Parmi les différents types de composants définis par la norme, on retrouve des composants représentant des entités métier, et dotés de la capacité de persistance : les EJB Entité. Ces composants sont eux-mêmes subdivisés en deux catégories, qui correspondent à la stratégie utilisée pour l'implémentation de leur logique de persistance :

- les EJB entité BMP (*Bean-Managed Persistence* – persistance gérée par le composant) pour lesquels la logique de persistance doit être implémentée par programmation.
- les EJB entité CMP (*Container-Managed Persistence* – persistance gérée par le conteneur), pour lesquels la logique de persistance est assurée de façon transparente par le conteneur d'EJB.

| **Services de J2EE 1.3** | **Description et API correspondante** |
|---|---|
| Présentation côté client | Service permettant aux applications de présenter de l'information et de gérer la saisie utilisateur :<br>• Interface graphique pour client lourd Java : AWT et Swing (JFC – Java Foundation Classes)<br>• Interface graphique pour client léger : HTML, XML |
| Présentation côté serveur | Client léger :<br>• Servlets et JSP (Java Server Pages) pour la génération de flux HTML et XML<br>• JAXP, jDOM pour le traitement de fichiers XML |
| Logique applicative | Servlets, frameworks MVC (ex : Jakarta Struts) |
| Accès aux sources de données relationnelles | JDBC (Java DataBase Connectivity) |
| Accès aux systèmes patrimoniaux | Anciennement JCA (Java Connector Architecture), désormais appelée J2C (Java2 Connector). |
| Accès aux systèmes de messagerie Internet (RFC822) | JavaMail et JAF (JavaMail Activation Framework) pour l'envoi d'emails avec support des extensions MIME. |
| Accès aux systèmes de messagerie asynchrone des MOM (middleware orientés objet) | JMS (Java Message Service) permet aux clients Java d'accéder aux systèmes de messagerie asynchrone basés sur les files d'attente. La plupart des serveurs d'application J2EE intègrent également un service de messagerie asynchrone (MOM) compatible JMS. |

| | |
|---|---|
| Accès aux annuaires (LDAP) et aux services de nommage | JNDI (Java Naming and Directory Interface) permet aux applications Java de s'interfacer avec tout serveur d'annuaire proposant une interface LDAP. D'autre part, c'est également à travers les fonctionnalités de JNDI que les applications communiquent avec le service de nommage des serveurs d'application J2EE, pour la publication et l'accès aux ressources du serveur et la localisation des objets. |
| Gestion transactionnelle | JTA (Java Transaction API) – adaptation des APIs applicatives de Corba OTS. Permet l'utilisation des transactions locales (simples) et globales (distribuées) à travers le protocole de validation à 2 phases XA. Supporte uniquement le modèle de transactions linéaires. |
| Service de gestionnaire transactionnel | JTS (Java Transaction Service) – spécification d'un service transactionnel conforme au service transactionnel Corba à bas niveau ((Corba OTS : Object Transaction Service) à bas niveau, et de l'interface JTA à haut niveau. Ce service respecte l'interface XA pour la gestion des transactions distribuées avec les gestionnaires de ressources (SGBD). |
| Gestion de la sécurité applicative | l'API JAAS (Java Authentication and Authorization System) est utilisée pour développer des modules d'authentification interchangeables vers différents référentiels de sécurité comme les annuaires LDAP, les référentiels d'utilisateurs des systèmes d'exploitation, et les bases de données. |
| Modèle de composants distribués transactionnels et persistants | EJB 2.0 définit une architecture d'objets métier distribués, transactionnels, et persistants. Le service de persistance du conteneur EJB doit proposer des fonctionnalités de mapping objet-relationnel afin d'assurer la persistance de l'état des objets dans une base de données relationnelle. |

La dernière version de la norme J2EE (1.3) définit l'ensemble des spécifications (ainsi que les versions respectives de ces spécifications) qui doivent être implémentées par une plate-forme serveur d'application :

| APIs de J2EE 1.3 | Description |
|---|---|
| JDBC 2.0 | API d'accès aux sources de données relationnelles |
| Servlet 2.3 | API d'extension de serveur Web pour la création de scripts côté serveur |
| JSP 1.2 | API d'extension de serveur Web pour le scripting côté serveur au sein de pages compatibles HTML |
| EJB 2.0 | Modèle de composants transactionnels distribués de J2EE, et API pour la mise en œuvre, le déploiement, et l'accès à ces composants depuis les applications d'entreprise. |
| JNDI 1.2 | API d'accès aux services de nommage de J2EE et d'annuaires LDAP |
| JMS 1.0 | API d'interfaçage avec les middleware orientés objet pour la messagerie asynchrone |
| JTA 1.0 | API de gestion des transactions permettant de s'interfacer avec un gestionnaire de ressource ou un moniteur transactionnel afin d'effectuer des transactions. |
| RMI/IIOP 1.0 | API technique encapsulant les appels d'objets distribués RMI à travers le protocole d'échange de CORBA (IIOP), et assurant l'interopérabilité entre objets Java et Corba. |
| JavaMail 1.1 | API pour l'envoi de messages e-mail (RFC 822) |
| JAF 1.0 | API pour la gestion des pièces jointes aux messages e-mail |
| J2C 1.0 | API d'interfaçage avec des applications patrimoniales par le biais de connecteurs spécifiques. |
| JAAS 1.0 | API de gestion de la sécurité (authentification) |

# 2.3 J2EE et les architectures multi-tiers

La norme J2EE se focalise tout particulièrement sur les architectures 3 et 4 tiers, en fournissant des environnements d'exécution et de multiples services de haut niveau pour les applications.

Tout particulièrement en matière de persistance, le modèle EJB proposé par la norme n'est utilisable qu'en architecture 4 tiers avec un conteneur EJB.

Si la norme J2EE ne couvre pas les scénarios d'architecture 2 tiers, il reste néanmoins possible de développer des applications sur ce modèle. En effet, les API standard de Java couvrent en particulier les besoins de la logique de présentation (API Swing pour le développement de clients lourds) et d'accès aux SGBD (API JDBC).

Cependant, il subsiste une lacune importante en matière de persistance objet. Pour combler cette lacune, plusieurs solutions ont vu le jour :

- **les outils de mapping objet/relationnel** : ces produits, spécialisés dans la problématique de la persistance objet en base de données relationnelle, sont apparus assez tôt, et ont proposé une solution adaptée à tout type d'architecture. Ces offres sont constituées d'un atelier de configuration du mapping entre objets métier et bases de données relationnelles, et d'un moteur de persistance autonome. Le moteur de persistance peut être embarqué au sein d'une application pour une utilisation en architecture 2 tiers, ou bien déployé sur un serveur d'application pour une utilisation en architecture 3 ou 4 tiers. On recense aujourd'hui plusieurs solutions sur le marché, dont les principales (parmi lesquelles Oracle9iAS TopLink) bénéficient d'une maturité importante, d'une large base de clients installée, et offrent une grande flexibilité et richesse fonctionnelle.

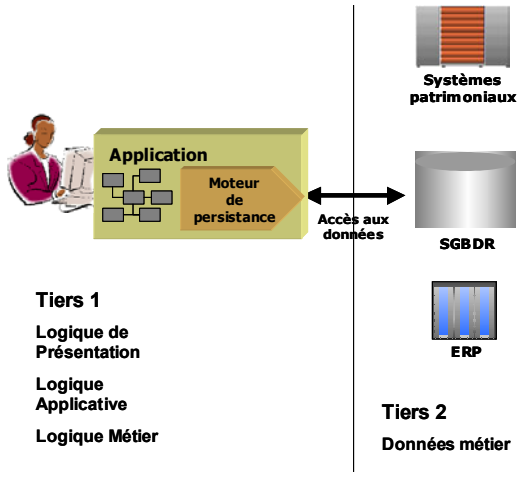
- **la norme JDO** : JDO est une norme toute récente élaborée qui vise à définir un modèle standard pour la persistance d'objets Java. L'architecture définie par JDO pour la persistance est assez proche de celle employée par les outils de mapping objet/relationnel. Les implémentations JDO fournissent un moteur de persistance qui peut être embarqué au sein d'une application autonome en architecture 2 tiers, ou bien déployé dans un serveur d'application en architecture 3 ou 4 tiers. La norme prévoit plusieurs options pour la persistance des objets : utilisation d'un SGBD, d'une base de données objet, ou d'autres supports de persistance à travers des adaptateurs JCA. Si la perspective d'un standard pour la persistance objet semble séduisante, la norme est encore très jeune et critiquée sur certains aspects. De plus, les implémentations disponibles sur le marché n'ont pas encore atteint la maturité et la richesse fonctionnelle des implémentations de la norme EJB ou des outils de mapping objet/relationnel.

Les outils de mapping objet/relationnel et les implémentations JDO offrent donc un service de persistance objet aux applications autonomes déployées en architecture 2 tiers, et qui peut également être utilisé par des applications hébergées au sein de serveurs d'application J2EE en architecture 3 ou 4 tiers.

## Architecture 2-tiers Java pour la persistance objet :

**Tiers 1 – Client :** En architecture 2 tiers Java, la partie cliente est typiquement implémentée sous la forme d'un client lourd avec une interface graphique riche. A cet effet, des APIs spécialisées (l'API Swing, ainsi que la désormais obsolète API AWT) sont fournies, et permettent le développement d'interfaces utilisateur classiques. Le client implémente non seulement la logique de présentation, mais également l'ensemble des couches applicatives et métier.

**Tiers 2 – Serveur de données:** le serveur de base de données héberge les données de l'application (et, optionnellement, certains traitements pouvant être implémentés sous la forme de procédures stockées).



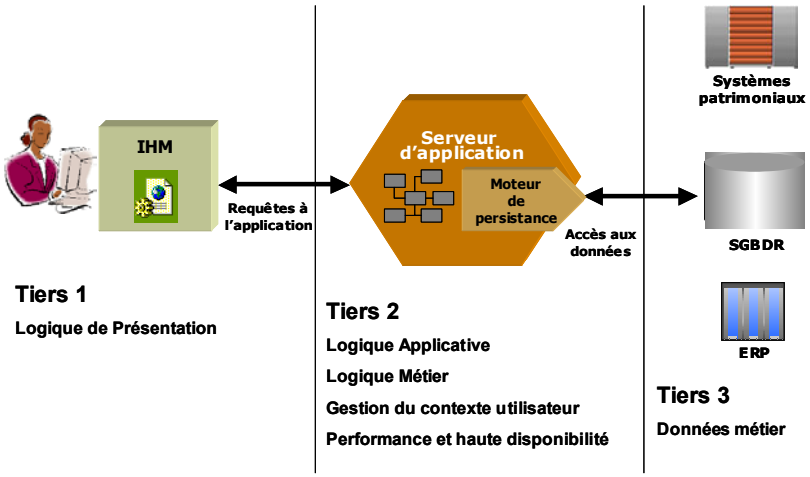

En matière de prise en charge de la persistance d'objets métier, seuls certains modèles de développement sont compatibles avec l'architecture 2 tiers. Ainsi, si JDO et les outils de mapping objet-relationnel comme Oracle9iAS TopLink permettent la réalisation d'applications suivant ce modèle, le standard J2EE/EJB ne le permet pas.

## Architecture 3-tiers J2EE pour la persistance objet

**Tiers 1 – Client :** Dans la majorité des cas, la couche présentation est implémentée sous la forme d'un client léger (interface HTML ou XML). Cependant, il reste possible de développer la couche présentation sous forme de client lourd Java/Swing.

**Tiers 2 – Serveur d'application :** la partie serveur d'application implémente les couches logique métier et logique applicative sous forme de composants applicatifs (Servlets, JavaBeans).

**Tiers 3 – Serveur de données :** Un serveur de bases de données héberge les données de l'application (et optionnellement certains traitements qui peuvent être implémentés sous la forme de procédures stockées).




Comme dans le cas d'une architecture 2 tiers, la prise en charge de la persistance d'objets métier en architecture 3 tiers n'est possible qu'avec JDO et les outils de mapping objet-relationnel comme Oracle9iAS TopLink. Le moteur de persistance est alors embarqué dans le serveur d'application utilisé. Quant au modèle EJB, il nécessite l'utilisation d'un tiers dédié à la persistance, le serveur d'objets.

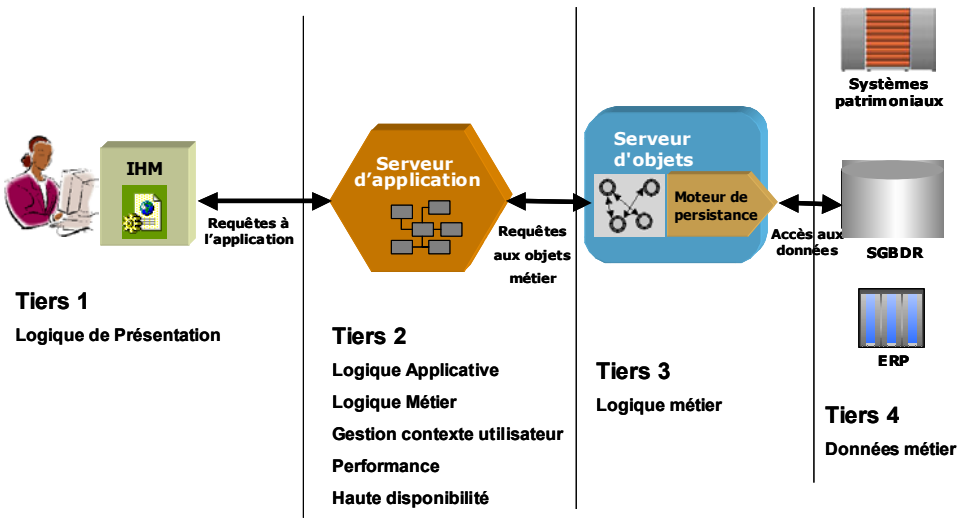
## ❑ Architecture 4-tiers J2EE pour la persistance objet

**Tiers 1 – Client :** comme en architecture 3 tiers, la couche présentation est le plus souvent implémentée sous la forme d'un client léger (interface HTML ou XML), mais peut également être réalisée sous la forme d'un client lourd Java/Swing.

**Tiers 2 – Serveur d'application:** contrairement à l'architecture 3 tiers, la partie serveur d'application n'héberge ici que la couche logique applicative.

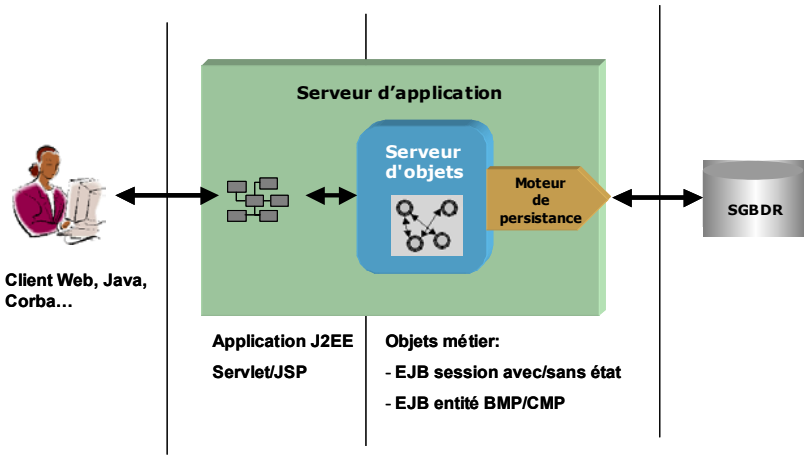
**Tiers 3 –Serveur d'objets:** Le serveur d'objets héberge la couche logique métier des applications. Cette couche métier intègre le modèle d'entités métier de l'application, qui est implémenté sous la forme d'objets ou de composants métier persistants.

**Tiers 4 – Partie serveur de données :** Le serveur de bases de données est utilisé par le serveur d'objets pour assurer la persistance des objets métier.



En architecture 4 tiers, l'ensemble des modèles de développement (JDO, outils de mapping objet-relationnel, et EJB) offre la prise en charge de la persistance d'objets métier. Comme en architecture 3 tiers, les implémentations JDO et les outils de mapping objet-relationnel exigent que le moteur de persistance soit embarqué dans le serveur d'application. Pour ce qui concerne le modèle EJB, le conteneur EJB est embarqué dans le serveur objets (dans la majorité des cas, serveur d'application et serveur d'objets ne font qu'un).

Par ailleurs, il est possible de combiner le moteur de persistance d'une implémentation JDO ou d'un outil de mapping objet-relationnel avec le modèle EJB. Ainsi, il est possible par exemple d'utiliser des EJB session manipulant des objets métier JDO ou Oracle9iAS TopLink, ou bien des EJB entité BMP dont la persistance est assurée à travers JDO ou Oracle9iAS TopLink. Avec certains serveurs d'EJB, il est même possible d'utiliser Oracle9iAS TopLink comme moteur de persistance pour les EJB entité CMP. Le schéma ci-dessous illustre ce scénario :




Les différents scénarios d'architecture mettent en évidence les différents niveaux de flexibilité offerts par les différents modèles pour la persistance d'objets métier. Alors que la norme EJB implique le recours à l'architecture 4 tiers, les modèles proposés par les implémentations JDO et les outils de mapping objet-relationnel comme Oracle9iAS TopLink peuvent s'accommoder de tous les types d'architecture, et peuvent également être combinés au modèle EJB en architecture 4 tiers.

# 3. Les solutions de persistance objet en architecture J2EE

## 3.1 Alternatives de persistance d'objets métier dans une architecture J2EE

Au sein de l'architecture J2EE, on retrouve la norme EJB, qui fait figure de standard incontournable et exclusif pour la modélisation d'objets métier persistants.

De fait, tel est bien le discours tenu par Sun Microsystems, et relayé par les partisans du « tout J2EE ». Ce discours est également entretenu par certains éditeurs de serveurs d'application J2EE qui voient parfois d'un mauvais œil le développement de solutions complémentaires ou concurrentes à leurs produits dans le domaine de la persistance objet.

Le thème de la persistance est un sujet sensible dont les débats sont souvent empreints de dogmatisme et d'esprit partisan. Nous nous fixons comme objectif d'apporter a nos lecteurs des éléments de compréhension des implications des divers choix possibles.

Avec le recul et l'expérience accumulés depuis quelques années, force est de constater cependant qu'en l'état J2EE n'est pas nécessairement la panacée. D'autres normes co-existent avec J2EE, et certaines se posent même comme potentiellement concurrentes à certaines de ses fonctionnalités. C'est le cas notamment de JDO qui, précisons-le, a été développée sous l'égide de Sun, et qui se pose pourtant comme une alternative sérieuse au modèle EJB pour la persistance.

Dans le domaine de la persistance objet, des produits comme Oracle9iAS TopLink bénéficient de presque 10 ans d'expertise accumulée. A titre de comparaison, la norme EJB apparue en 1998, n'a commencé à devenir réellement exploitable qu'à partir de la version 2.0, finalisée au cours de l'été 2001. En ce qui concerne la norme JDO, la version 1.0 de la spécification a été publiée en mars 2002. La maturité des technologies employées pouvant être un élément clé de choix pour des applications d'entreprise, la jeunesse des technologies n'est pas nécessairement gage de robustesse et de performance.

Enfin, la norme J2EE impose une architecture 3 ou 4 tiers spécifique pour le développement d'applications, ce qui n'est pas le cas des alternatives comme Oracle9iAS TopLink ou les implémentations de JDO qui sont neutres par rapport au modèle d'architecture applicative. De plus, ces alternatives se posent non pas uniquement comme approches concurrentes à J2EE et au modèle EJB, mais également en tant que briques complémentaires dans une architecture multi-tiers J2EE pouvant mettre en œuvre le modèle d'application EJB.

En fait, il faut bien distinguer deux types de persistance objet :
- la persistance objet dans un cadre de *modélisation orientée composant* : c'est l'approche choisie par la norme EJB avec les EJB Entité.
- la persistance objet dans un cadre de *modélisation objet pure*: c'est l'approche choisie par les modèles JDO et Oracle9iAS TopLink ; cependant, ces modèles présentent également l'avantage de pouvoir être utilisés comme mécanismes d'implémentation de la persistance de composants métier de type EJB Entité.

C'est pourquoi nous avons voulu mener une étude comparative et objective des différentes alternatives pour la persistance d'objets métier en architecture multi-tiers J2EE. Cette étude procède d'une évaluation méthodique de la couverture fonctionnelle des différentes approches que nous avons évoquées :

- la persistance de composants métier à travers l**e modèle EJB** de J2EE
- la persistance d'objets métier à travers **le modèle JDO**
- la persistance d'objets métier à travers **Oracle9iAS TopLink**, l'un des outils de mapping objet/relationnel les plus évolués disponibles sur le marché :

En complément de ce comparatif, cette étude s'attache également à expliquer la **complémentarité** existant entre ces différentes approches : en effet, ces différents modèles ne sont pas forcément exclusifs, ils peuvent être combinés de façon à exploiter au mieux les caractéristiques de chacun dans une approche « hybride ».

### 3.1.1 Grille de critères de couverture fonctionnelle pour la persistance objet

Afin de pouvoir évaluer et comparer les fonctionnalités des différents modèles de persistance objet, nous avons établi une grille de critères détaillée qui regroupe en huit domaines les différents aspects de la problématique de persistance objet:

1. **Transparence de modélisation :**
   *Disposer du maximum de flexibilité dans la modélisation permet aux concepteurs de modéliser le domaine métier le plus fidèlement possible.*

2. **Support des sources de données pour la persistance :**
   *La versatilité de l'outil et son adaptabilité au contexte de l'entreprise sont fonction de sa capacité de prise en charge des principaux SGBD du marché.*
   *L'ouverture vers d'autres supports de persistance (fichiers XML, sources de données J2C) permet de maximiser le potentiel de l'outil.*

3. **Capacités de modélisation et de projection :**
   *La richesse de modélisation offerte et l'étendue des options disponibles pour la projection du modèle objet apportent souplesse de conception et simplicité de développement.*

4. **Langage de requêtage objet :**
   *La puissance et la richesse du langage de requêtage objet conditionnent le niveau de complexité d'analyse que peuvent offrir les applications.*

5. **Potentiel transactionnel :**
   *La prise en charge de fonctionnalités transactionnelles sophistiquées permet de mettre en œuvre des scénarios complexes et de réaliser des applications multi-utilisateurs fiables et performantes.*

6. **Cache objet et optimisations :**
   *La présence d'un cache objet distribué, la souplesse de gestion des accès concurrents, et la présence de mécanismes d'optimisation sophistiqués permettent des gains de performance conséquents et favorisent une meilleure montée en charge des applications.*

7. **Environnement de développement (IDE) :**
   *La simplicité de développement des objets métier, renforcée par la présence d'outils adaptés, permet des gains de productivité conséquents.*

8. **Déploiement et intégration :**
   *La versatilité d'utilisation du modèle au sein d'architectures diverses, la facilité de déploiement et d'intégration avec les serveurs d'application J2EE, et la complémentarité avec d'autres solutions de persistance déterminent le potentiel et la valeur du modèle pour l'entreprise.*

◆ **Remarques concernant l'évaluation des alternatives pour la persistance objet :**

▪ **Évaluation des fonctionnalités du modèle EJB basée sur le mode CMP :**

La norme EJB prévoit 2 approches pour la persistance des composants entité : la persistance gérée par le composant (BMP) de façon programmatique, et la persistance gérée par le conteneur (CMP) de façon déclarative. Seul le mode CMP présente un réel intérêt en tant que solution de persistance transparente comparable aux services offerts par Oracle9iAS TopLink et par les implémentations JDO. C'est pourquoi l'évaluation des fonctionnalités du modèle EJB pour la persistance est basée uniquement sur l'utilisation du mode CMP. Il faut toutefois signaler que la persistance gérée par le composant (BMP) peut tirer grand bénéfice de l'utilisation d'une solution de persistance du type de celles étudiées dans ce document (Oracle9iAS TopLink entre autres propose des classes conformes au modèle EJB et simplifiant l'implémentation de la persistance des composants BMP.)

▪ **Grille d'évaluation :**

Les critères d'évaluation, numérotés pour faciliter le travail de référence, sont répartis dans les huit domaines fonctionnels ou techniques identifiés. Chaque critère est assorti d'une description précisant quelles caractéristiques sont évaluées.

Au cours de l'évaluation des différentes solutions de persistance, une note sera attribuée à chaque critère afin de mesurer le niveau d'aptitude correspondant de la solution étudiée. Une échelle de trois valeurs sera utilisée :

0 : fonctionnalité non prise en charge
1 : fonctionnalité partiellement prise en charge
2 : fonctionnalité entièrement prise en charge

Par la suite, les notes attribuées seront comptabilisées par catégorie et traduites en pourcentages pour dresser un graphique de synthèse.

La description exhaustive de la grille de critères utilisée pour l'évaluation fonctionnelle est présentée en annexe 5.1.

# 3.2 Couverture fonctionnelle du standard EJB en matière de persistance

## 3.2.1 Historique

La première version de la norme EJB, apparue en 1998, jetait les bases d'un modèle de composants persistants transactionnels, les EJB Entité, mais leur support était défini comme optionnel. Bien que cette première mouture incorporait déjà les 2 modes de persistance (BMP – persistance gérée par le composant, et CMP – persistance gérée par le conteneur), le modèle était entaché de nombreuses lacunes. Peu performant et peu flexible, il souffrait par ailleurs de graves défauts au niveau du déploiement et de la portabilité entre implémentations.

Avec EJB 1.1, le support des EJB entité est devenu obligatoire, et de nombreux défauts ont été corrigés, notamment concernant la compatibilité et la portabilité entre implémentations. Les améliorations les plus notables ont concerné la gestion transactionnelle et la gestion de la sécurité, un nouveau modèle de déploiement standardisé (descripteurs XML), et la standardisation de l'utilisation de JNDI (comme service de nommage pour l'accès aux composants et aux diverses ressources du serveur d'objets) et de RMI-IIOP (pour l'accès distant aux composants).

Mais ce n'est qu'avec l'apparition mi-2001 de la version 2.0 de la norme que les lacunes les plus graves, situées au niveau du modèle de persistance, ont commencé à être comblées. La version 2.0 a en effet apporté un grand nombre d'améliorations :

- l'utilisation d'un schéma abstrait de persistance pour les EJB entité CMP.
- le support des relations d'association (1-1, 1-N, et N-N) inter-composants (CMR).
- la définition d'un langage de requêtage objet puissant : EJB-QL.
- l'apparition des interfaces locales, qui permettent de remédier aux problèmes de performance résultant des communications réseau entre applications et composants, ou entre les composants entre eux. Les interfaces locales, qui réalisent des appels intra-JVM, réduisent considérablement l'overhead de communication entre applications clientes et composants EJB hébergés dans un même processus.

Ces améliorations ont enfin permis d'envisager une approche de modélisation à granularité fine pour les EJB entité.

Enfin, la toute dernière version de la norme (2.1), finalisée au cours de l'été 2002, apporte quelques améliorations au langage EJB-QL : le support des clauses de tri et des fonctions d'agrégat.

### 3.2.2 Présentation du modèle EJB

La norme EJB définit une architecture de composants transactionnels qui couvre largement le spectre de la modélisation métier :

- les EJB session permettent de modéliser des processus métier
- les EJB orienté message (message-driven beans) permettent de modéliser des processus asynchrones
- enfin, les EJB entité permettent de modéliser des entités métier persistantes

L'approche de modélisation par composants adoptée par la norme EJB se révèle très intrusive au niveau du modèle métier, et induit à l'usage une technicité importante dans le développement d'applications d'entreprise. En effet, ces composants sont constitués de plusieurs éléments :

- une interface de fabrique (home interface), pour la gestion du cycle de vie du composant.
- une interface cliente, qui expose les méthodes métier du composant.
- une classe d'implémentation, qui contient les champs persistants, l'implémentation des méthodes métier, ainsi que l'implémentation de plusieurs méthodes callback nécessaires à l'interfaçage entre le composant et le conteneur EJB.
- des classes proxy côté client (stubs) et côté serveur (skeletons) : ces classes, spécifiques à chaque implémentation, contiennent la logique de bas-niveau permettant de déléguer les appels des applications à travers les interfaces cliente et de fabrique vers la classe d'implémentation de l'EJB. Ces classes sont générées automatiquement par un compilateur spécifique à chaque serveur d'EJB.

De plus, il existe deux variantes pour les interfaces cliente et de fabrique : interfaces distantes (qui permettent l'accès aux composants depuis des applications distantes), et les interfaces locales (qui permettent d'optimiser l'accès aux composants depuis des applications hébergées dans le serveur d'application où sont déployés ces composants).

En échange de cette complexité, les composants EJB bénéficient de services de haut niveau fournis par le conteneur d'EJB:

- **la gestion du cycle de vie des composants :** le conteneur contrôle le cycle de vie des composants manipulés par les applications.
- **la gestion du pooling des composants** : cette technique favorise une meilleure montée en charge de l'application par la réduction du coût lié au cycle de création/destruction des objets : au lieu d'instancier et de détruire les objets à la demande, un service de pooling construit (instancie) au lancement un nombre fixe d'objets. Lorsque l'application requiert la création d'un objet, un objet du pool lui est alloué ; puis, lorsque l'application détruit l'objet, l'objet n'est pas réellement détruit, mais restitué au pool pour être ensuite réaffecté à l'application lors de demandes de création ultérieures. De cette façon, les cycles de création/destruction d'objets, coûteux en performances, sont éliminés, et l'application gagne en rapidité d'exécution.

- **la gestion de l'accès distant** aux composants et de l'interopérabilité avec Corba à travers RMI-IIOP.

- **la prise en charge de la distribution des objets et de la transparence de localisation** : les objets peuvent être répliqués sur plusieurs serveurs en cluster, et le service de nommage JNDI fournit la transparence de localisation et d'accès aux composants.

- une **gestion transactionnelle** sophistiquée et transparente : le conteneur peut prendre en charge automatiquement la démarcation des transactions en fonction d'attributs transactionnels déclarés au déploiement sur les méthodes des composants.

- **la gestion de la persistance** pour les composants persistants appelés EJB entité (cet aspect est détaillé ci-après)

- **la prise en charge de la sécurité** : le conteneur gère la sécurité d'accès aux composants en fonction de règles définies lors du déploiement. Le modèle de sécurité est basé sur la définition de rôles de sécurité logiques, auxquels on associe les groupes et utilisateurs de l'application. Le niveau de granularité de la sécurité est très fin, et permet de définir les accès au niveau de chaque méthode métier des composants.

- **la gestion de la réplication et du fail-over** : lors du déploiement, les composants EJB peuvent être répliqués sur plusieurs serveurs, membres d'un cluster ; le service de réplication assure alors la reprise sur panne (fail-over) des appels aux composants : lorsqu'un application effectue un appel à un composant hébergé sur un serveur et que ce serveur tombe, l'appel peut dans certains cas être relayé de façon automatique et transparente sur un autre serveur du cluster.

### ❑ Objets métier persistants : EJB entité

Les EJB entité représentent une vue d'entités métier stockées en base de données relationnelle. Les composants peuvent comporter des attributs et des objets dépendants persistants, et possèdent une identité modélisée sous la forme d'une classe de clé primaire. Il est également possible de modéliser des relations d'association entre composants. En revanche, la prise en charge de l'héritage n'est pas assurée. Concernant les possibilités de requêtage objet, la norme ne propose qu'un modèle de requêtes statiques sous forme de méthodes de recherche (custom finders) sur les composants eux-mêmes. La démarcation des transactions, gérées par le conteneur, est configurée de façon déclarative, sur chaque méthode de l'EJB. Pour l'implémentation de la persistance, deux approches sont proposées :

- **persistance gérée par le composant (*bean-managed persistance*)** : le composant est responsable de l'implémentation de sa logique de persistance. La norme définit un certain nombre de méthodes callback (appelées par le conteneur aux moments opportuns) à travers lesquelles les interactions entre le composant et le gestionnaire de base de données doivent être implémentées (typiquement à l'aide de l'API JDBC). D'autre part, le composant est également responsable de l'implémentation des méthodes de recherche.

- **persistance gérée par le conteneur (*container-managed persistence*):** le composant délègue au conteneur EJB la prise en charge de sa persistance. Un schéma abstrait de persistance est utilisé pour décrire les attributs persistants du composant et ses relations d'association avec d'autres composants. L'outil de mapping objet/relationnel du serveur EJB permet de définir les mappings des attributs et des relations sur le support relationnel, et d'exprimer la logique des requêtes statiques à l'aide d'un langage de requêtage objet sophistiqué : EJB-QL.

L'utilisation du mode de persistance gérée par le composant (BMP) contribue à fortement alourdir le processus de développement, et présente en contre-partie bien peu d'avantages, si ce n'est de permettre l'utilisation des modèles JDO et Oracle9iAS TopLink qui peuvent faciliter considérablement l'implémentation de la logique de persistance. En ce domaine, Oracle9iAS TopLink fournit une classe de base spécialisée (BMPEntityBase) qui permet d'automatiser l'interaction entre le composant et l'infrastructure de persistance.

Dans le cadre de l'évaluation fonctionnelle du modèle EJB en matière de persistance, nous nous limiterons au mode de persistance gérée par le conteneur.

- **Cycle de développement avec le modèle EJB**

Le cycle de développement de composants EJB entité est relativement long et complexe :

- phase de conception : écriture des différentes composantes de l'EJB entité (classe d'implémentation, classe de clé primaire, interface de fabrique, et interface cliente).

- phase de configuration : utilisation d'un outil de configuration pour la définition des propriétés de déploiement de l'EJB : mapping objet/relationnel des attributs persistants, définition des attributs transactionnels et de sécurité sur les méthodes métier, définition de la logique des méthodes de recherche.

- phase de déploiement : compilation des classes de l'EJB, et génération des classes proxy côté client et côté serveur. Déploiement du composant sur le serveur d'application.

Il est à noter que les fournisseurs d'IDE Java et de serveur d'application Java proposent des assistants automatisant la gestion de ces différentes étapes ; néanmoins une parfaite compréhension de la part du développeur du rôle de chaque constituant est nécessaire.

En ce qui concerne le développement des applications clientes, le processus est également relativement lourd:

Au sein des applications, les objets persistants (EJB entité) ne sont pas manipulés comme des objets Java simples, mais comme des objets distants, auxquels on accède au travers d'interfaces proxy. Pour chaque type de composant EJB entité, les applications clientes doivent manipuler deux interfaces : l'interface de fabrique, et l'interface cliente. La récupération de l'interface de fabrique d'un EJB entité s'effectue par le biais d'un lookup sur le contexte JNDI du serveur d'objets. C'est à partir de cette interface de fabrique que s'effectue l'instanciation et la recherche des objets persistants : les méthodes de gestion du cycle de vie et de recherche des composants renvoient un proxy (l'interface cliente) sur chaque objet. Les appels de méthodes sur les objets eux-mêmes s'effectuent à travers cette interface cliente.

Enfin, le modèle EJB impose un modèle de programmation *mono-thread* : d'une part, il est interdit de faire usage de threads au sein d'un composant EJB ; d'autre part, l'accès aux composants EJB eux-mêmes est mono-thread. Pour gérer les accès concurrents, deux approches sont employées par les conteneurs EJB. La plus commune consiste à créer plusieurs instances d'un même EJB pour gérer les accès concurrents. La deuxième consiste à sérialiser les appels concurrents sur une seule instance, ce qui grève fortement les performances.

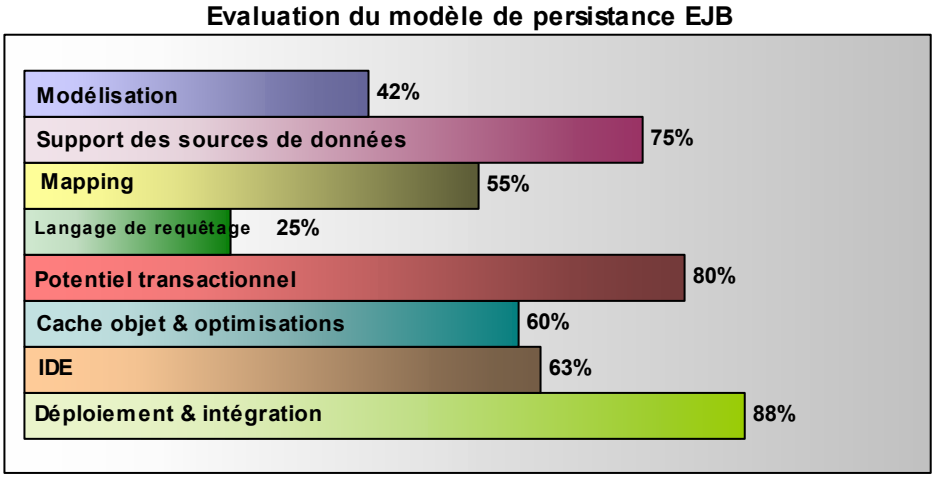
- **Flexibilité au niveau du choix d'architecture**

  La norme EJB s'inscrit entièrement dans la spécification J2EE, qui définit une architecture 4 tiers. Par conséquent, le modèle EJB n'est pas utilisable en architecture 2 tiers.

### 3.2.3 Synthèse des résultats de l'évaluation détaillée

La grille d'évaluation détaillée de la couverture fonctionnelle du modèle EJB en matière de persistance est présentée en annexe 5.2.

Le schéma ci-dessous présente la synthèse de l'évaluation fonctionnelle ; les notes, comptabilisées par catégorie, ont été traduites en pourcentages :




Ces résultats illustrent clairement les limitations du modèle EJB en termes de respect du paradigme objet, avec des capacités de modélisation et de requêtage réduites. On note également des capacités de projection (mapping) limitées, ainsi que des faiblesses au niveau du cache objet et des optimisations proposées. Enfin, ce modèle est limité en termes de déploiement à une architecture J2EE 4 tiers. En revanche, il bénéficie intrinsèquement d'un haut niveau d'intégration dans l'architecture J2EE, de la disponibilité d'outils de développement adaptés, et d'un potentiel transactionnel satisfaisant.

# 3.3 Couverture fonctionnelle de JDO en matière de persistance

## 3.3.1 Historique

Java Data Objects (JDO) est une spécification qui vise à définir un standard universel pour la persistance transparente d'objets Java simples. Cette spécification constitue l'adaptation au langage Java du standard de persistance transparente défini par l'ODMG (Object Database Management Group), un consortium de fournisseurs de bases de données objet formé avant l'avènement de Java, à l'époque où SmallTalk et C++ étaient les deux principaux langages orientés objet du marché.

Le développement du standard JDO s'est produit parallèlement à l'élaboration des normes EJB et J2C, et le groupe de travail JDO a toujours estimé qu'il était important de garder JDO en ligne avec ces standards. Cependant, l'implémentation d'un modèle de persistance non-intrusif, flexible et sophistiqué mais qui reste simple d'utilisation au sein des applications, a été la considération primordiale qui a guidé l'élaboration de cette norme.

La version 1.0 de JDO, développée sous l'égide du JCP[†] en tant que JSR[‡] 12, a été validée fin avril 2002.

## 3.3.2 Présentation de JDO

La spécification JDO se décompose en 2 grandes parties :

- L'interface d'implémentation (Implementation Interface, JDO SPI) définit les mécanismes que doivent implémenter les éditeurs afin d'assurer la prise en charge de la persistance des objets JDO et des fonctionnalités exposées aux applications à travers l'API JDO.

- L'interface applicative (Application Interface, JDO API) définit une API à l'intention des applications manipulant des objets JDO. A travers cette API, les applications accèdent au gestionnaire de persistance, qui leur permet d'instancier des objets persistants, de rechercher des objets en effectuant des requêtes, et d'exécuter des transactions.

Le modèle proposé par la norme JDO permet de manipuler des hiérarchies complexes d'objets Java sans avoir à se soucier des problématiques de persistance, qui sont gérées de manière totalement transparentes par l'implémentation JDO (le moteur de persistance qui implémente les spécifications et les interfaces définies dans la norme JDO ; en effet, tout comme EJB, rappelons que JDO est une norme et non un produit).

[†] *Java Community Process*.: instance chargée de guider les évolutions des technologies Java dirigée par Sun Microsystems et intégrant d'autres représentants des acteurs majeurs du monde Java. Les propositions d'évolution suivent un processus de spécification, d'élaboration et de validation par des groupes de travail.
[‡] *Java Specification Request* : demande de spécification Java dans le cadre du JCP, son étude est confiée à un groupe de travail chargé de son élaboration en vue d'une future normalisation.

Contrairement à la norme EJB, la spécification JDO est non-intrusive vis-à-vis du modèle objet, et impose très peu de contraintes. Par conséquent, JDO autorise une modélisation métier à granularité fine, basée sur des objets simples reflétant fidèlement le domaine métier. JDO permet l'utilisation sans restriction des mécanismes d'héritage et de composition (relations d'association entre objets) dans la conception objet.

JDO est également neutre vis-à-vis des référentiels de persistance ; la projection du modèle objet peut être réalisée sur un support de persistance quelconque : base de données relationnelle, base de données objet, fichiers binaires, documents XML, ou même au sein d'une application patrimoniale (ERP) ou mainframe.

De plus, la logique de persistance est entièrement découplée de la logique métier : aucun ajout de code spécifique à la persistance n'est requis au sein d'un objet JDO.

Les objets et attributs persistants sont déclarés dans un descripteur de déploiement XML standardisé. Cependant, le standard ne prévoit aucune règle pour la définition des mappings des attributs sur le support de persistance utilisé. Chaque implémentation JDO emploie donc une approche propriétaire, basée soit sur l'utilisation de « tags d'extension » dans le descripteur de déploiement JDO standard, soit sur l'utilisation d'un descripteur de déploiement additionnel à cet usage.

Pour implémenter la logique de persistance des objets JDO, la norme impose aux implémentations JDO le recours à la modification du code des objets. Les modifications nécessaires peuvent au choix s'effectuer sur le code source des classes JDO, ou directement sur le code compilé de ces classes. A cet effet, les implémentations JDO doivent fournir un outil de post-traitement de code spécifique (« code enhancer »).

A l'exécution, la persistance des objets JDO est contrôlée par un gestionnaire de persistance, qui intègre également un cache objet transactionnel. Avec JDO, contrairement à ce qui est pratiqué avec les EJB, la démarcation des transactions est effectuée explicitement par les applications. Il est d'ailleurs obligatoire d'inscrire les opérations de création et de mise à jour des objets persistants dans le cadre d'une transaction.

Pour permettre aux applications de rechercher des objets en fonction de critères spécifiques, la norme JDO définit un langage de requêtage objet appelé JDOQL (JDO Query Language). Ce langage, à la syntaxe proche de celle des expressions Java, permet de spécifier des clauses de recherche (filtres) et de tri, et permet la navigation à travers un graphe objet. Par contraste avec le modèle EJB, les requêtes ne sont pas définies de manière statique sur les composants lors du développement, mais construites de manière dynamique à l'exécution par les applications. D'autre part, JDOQL prend en charge les relations d'héritage de façon transparente, et permet par conséquent d'effectuer des requêtes dites polymorphes : les résultats de ce type de requêtes incluent non seulement des objets de la classe spécifiée, mais également les objets satisfaisant aux critères et appartenant à des classes dérivées.

Une architecture utilisant JDO pour la persistance objet est donc constituée des composantes suivantes :

- **le gestionnaire de persistance** (Persistence Manager) de l'implémentation JDO utilisée : il intègre le moteur de persistance et le cache transactionnel pour les objets. C'est à ses fonctionnalités que l'application fait appel à travers les appels à l'API applicative de JDO. En sus du gestionnaire de persistance, l'implémentation JDO doit également intégrer un moteur de requêtes JDOQL.

- **La source de données** : base de données relationnelle ou objet, ou autre support de stockage : utilisée pour faire persister le modèle objet de l'application, elle est gérée par le gestionnaire de persistance.

- **Les métadonnées XML de mapping des objets**, qui décrivent les classes et champs persistants du modèle objet ainsi que la source de données utilisée.

- **le modèle métier de l'application** : il s'agit du modèle de classes représentant les entités métier de l'application qui sont implémentés en tant qu'objets Java simples, et qui ont été compilés de manière à pouvoir devenir des objets persistants à travers JDO.

- **l'application** : elle manipule les objets persistants et effectue les appels nécessaires à l'API applicative de JDO pour instancier le gestionnaire de persistance, gérer les transactions, et lancer des requêtes.

- **Cycle de développement avec JDO :**

  Le cycle de développement de classes JDO se rapproche quelque peu du cycle de développement des EJB, et comporte plusieurs phases :

  - phase de développement : écriture des classes métier Java qui deviendront des objets JDO persistants.

  - phase de configuration du mapping : JDO ne définit aucune norme ou outil standard pour la configuration du mapping. Chaque implémentation JDO inclut donc son propre outil de mapping spécifique pour définir les méta-données XML des classes JDO et la projection des attributs des objets sur le support de persistance.

  - phase de post-traitement des classes JDO : utilisation d'un post-processeur de code source ou de code compilé (« *code enhancer* ») pour l'implémentation des mécanismes de persistance au seins des classes JDO.

  - phase de déploiement : elle diffère selon que l'on se trouve en environnement non-contrôlé (architecture 2 tiers) ou contrôlé (architecture 3 ou 4 tiers, intégration à un serveur d'application):

    - en environnement non-contrôlé (« *non-managed environment* »), les classes de l'implémentation JDO sont embarquées dans l'application elle-même.

    - en environnement contrôlé (« *managed environment* » , l'implémentation JDO est embarquée dans le serveur d'applications ou dans le serveur d'objets. Pour l'intégration avec les serveurs d'application J2EE, JDO s'appuie sur l'API J2C.

  En ce qui concerne le développement d'applications clientes, le modèle JDO diffère fortement du modèle EJB. Dans une application JDO, les objets persistants sont manipulés directement comme des objets Java simples (pas d'appels indirects de méthodes à travers des interfaces), et la démarcation des transactions s'effectue obligatoirement et explicitement par l'application, de manière programmatique.

  La première étape au sein d'une application JDO consiste à initialiser le moteur JDO (phase de « *bootstrapping* »). Cette opération s'effectue par la configuration d'une instance de fabrique de gestionnaire de persistance (*Persistence Manager Factory*). C'est à travers cette fabrique d'objets que l'application peut ensuite instancier un ou plusieurs objets gestionnaires de persistance (*Persistence Manager*).

Chaque gestionnaire de persistance crée et maintient une connexion vers le support de persistance, ainsi qu'un cache pour les objets persistants. Le gestionnaire de persistance est l'interface principale qui va permettre à l'application de manipuler des objets persistants, en lui offrant les fonctionnalités suivantes :

- l'instanciation d'objets persistants par création directe, ou par recherche d'objets existants sur le support de persistance.
- le contrôle des transactions : les opérations de création et de mise à jour des objets persistants doivent s'inscrire dans des transactions. Le gestionnaire de persistance fournit à l'application les fonctions nécessaires à la démarcation des transactions (création, validation, et annulation).
- la construction et l'exécution de requêtes pour la recherche d'objets.

Chaque gestionnaire de persistance n'est capable de gérer qu'une seule transaction à la fois sur les objets ; par conséquent, pour effectuer plusieurs transactions simultanément, il faudra instancier autant de gestionnaires de persistance que de transactions concurrentes.

Contrairement au modèle EJB, JDO permet les appels concurrents sur un même objet. En effet, il n'existe dans le cache qu'une seule copie de chaque objet, qui peut être accédée par plusieurs transactions de façon simultanée. Il est d'ailleurs primordial pour les applications JDO de recourir systématiquement à l'utilisation de transactions pour les mises à jour sur les objets : en effet, une application accédant à un objet du cache directement sans passer par une transaction peut corrompre l'état de cet objet.

- **Flexibilité au niveau du choix d'architecture**

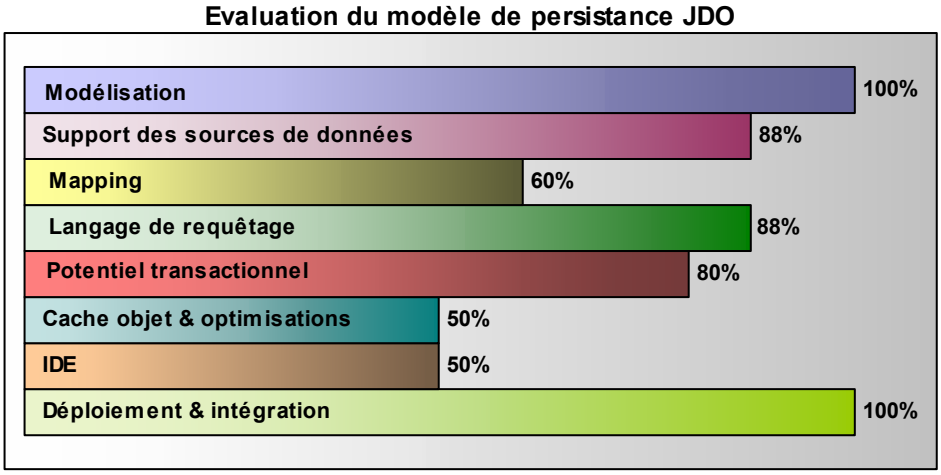
JDO offre une grande flexibilité en termes de choix d'architectures, et peut être utilisé aussi bien en architecture 2-tiers dans des applications autonomes (« standalone »), que dans des architectures 3 ou 4 tiers avec un serveur d'application J2EE.

D'autre part, JDO peut être facilement utilisé en tant que mécanisme de persistance pour des EJB entité BMP, où il apporte une forte valeur ajoutée.

### 3.3.3 Synthèse des résultats de l'évaluation

La grille d'évaluation détaillée de la couverture fonctionnelle du modèle JDO en matière de persistance est présentée en annexe 5.3.

Le schéma ci-dessous présente la synthèse de l'évaluation fonctionnelle ; les notes, comptabilisées par catégorie, ont été traduites en pourcentages :




Les notes obtenues par le modèle JDO sur les capacités de modélisation et de requêtage démontrent sa prise en charge complète du paradigme objet. On note également sa versatilité en termes de support de sources de données et de déploiement, ainsi que sa facilité d'intégration au sein d'une architecture J2EE. En revanche, les capacités de projection (mapping) de ce modèle restent moyennes : l'explication vient d'une part du support limité pour les relations d'association (pas de support de relations bi-directionnelles et de gestion de l'intégrité référentielle avec la prise en charge des suppressions en cascade), et d'autre part du faible niveau de compatibilité entre implémentations (mapping non-standardisé). Enfin, les implémentations JDO sont des produits encore jeunes. Bien que les produits disponibles intègrent un cache objet transactionnel sophistiqué, les possibilités de paramétrage et d'optimisation restent limitées.

# 3.4 Couverture fonctionnelle d' Oracle9iAS TopLink en matière de persistance

## 3.4.1 Historique

Oracle9iAS TopLink doit son existence et son nom à la société TOP (The Object People), fondée par des experts de l'objet au début des années 90. L'activité de TOP se partage à l'époque entre le consulting sur les problématiques objet et le développement en interne d'un framework de persistance pour ses clients. Ce framework, développé en langage SmallTalk, débouche sur le produit TopLink pour SmallTalk, commercialisé en 1994 et rapidement reconnu comme la référence en ce domaine.

L'avènement de Java et la révolution Internet dans la seconde moitié des années 90 bouleversent la cartographie du marché des technologies objet, et créent de nouveaux débouchés pour les solutions de persistance objet. Rapidement, la société TOP décide de porter TopLink sur la plate-forme Java. La première version de TopLink pour Java voit le jour début 1997.

Au cours de l'année 2000, la société WebGain, spécialisée dans les outils de développement Java, rachète la société TOP et son produit phare TopLink, qui est par la suite intégré à sa suite logicielle WebGain Studio.

Après 2 années au cours desquelles WebGain TopLink continue d'évoluer de manière importante (avec notamment la prise en compte des architecture EJB CMP), WebGain cesse brutalement ses activités commerciales.

En juin 2002, Oracle rachète à WebGain la technologie TopLink et les équipes techniques. Oracle intègre le produit à sa plate-forme J2EE Oracle9iAS , sous le nom de Oracle9iAS TopLink.

Aujourd'hui, après 2 rachats successifs, le développement de Oracle9iAS TopLink continue, et son avenir semble assuré au sein d'Oracle. Le produit actuel (version 9.0.3.2) est désormais l'aboutissement de près de 10 ans de réflexion, de développement, et d'expérience accumulés sur les problématiques de persistance objet et de mapping objet-relationnel auprès d'un très grand nombre de clients.

## 3.4.2 Présentation de la solution Oracle9iAS TopLink

Oracle9iAS TopLink est un produit éprouvé en matière de persistance de modèles objets complexes en base relationnelle. Très complète, la solution Oracle9iAS TopLink est constituée des briques suivantes :

- La « Java Foundation Library » : bibliothèque Java qui gère la persistance d'objets Java. Elle comprend le gestionnaire de persistance, le cache objet, le moniteur transactionnel, les mécanismes de définition du mapping objet/relationnel, et le moteur de requêtes (qui gére à la fois les expressions Java Oracle9iAS TopLink, les langages EJB-QL et SQL ainsi que les procédures stockées).

- Le « Mapping Workbench » : atelier graphique d'assistance a la définition des mappings entre un modèle objet et un schéma de base de donnée relationnelle.

- Deux bibliothèques spécifiques qui s'appuient sur la « Java Foundation Library », et qui permettent l'intégration du moteur de persistance Oracle9iAS TopLink avec deux des principaux serveurs d'application J2EE du marché :

  - « Oracle9iAS TopLink pour BEA WebLogic Server » offre un moteur de persistance basé sur Oracle9iAS TopLink pour les EJB entité CMP de WebLogic Server.

  - « Oracle9iAS TopLink pour IBM WebSphere » offre un moteur de persistance basé sur Oracle9iAS TopLink pour les EJB entité CMP de WebSphere Application Server.

  Le développement d'une bibliothèque similaire spécifique pour Oracle9i Application Server est en cours.

Oracle9iAS TopLink adopte une approche totalement non intrusive pour l'implémentation de la persistance objet (notamment de part l'utilisation des mécanismes d'introspection standard de Java), et permet une approche de modélisation à granularité fine utilisant les paradigmes de la conception objet : héritage, composition, et polymorphisme.

La logique de persistance est entièrement découplée de la logique métier et de la logique applicative. Aucun ajout de code spécifique à la persistance n'est requis, et contrairement à JDO, Oracle9iAS TopLink n'a pas recours à un mécanisme de post-traitement du code des objets métier pour l'implémentation des mécanismes de persistance. Les seules informations nécessaires à Oracle9iAS TopLink pour l'implémentation de la logique de persistance sont les mappings des attributs objet sur le support de persistance (Dans la pratique très majoritairement des bases de données relationnelles), qui sont stockés dans des descripteurs XML.

Bien qu'il soit axé en priorité sur la persistance objet en base de données relationnelle, Oracle9iAS TopLink peut également être utilisé avec d'autres supports de persistance. A cet effet, Oracle9iAS TopLink intègre un kit de développement (EIS SDK) qui permet de réaliser des adaptateurs Oracle9iAS TopLink pour la persistance dans des référentiels quelconques (applications patrimoniales ou mainframe, fichiers XML ou binaires, etc.). Oracle9iAS TopLink fournit une implémentation de référence d'un adaptateur développé avec son SDK pour la persistance d'objets en fichiers XML. A l'avenir, Oracle9iAS TopLink devrait proposer la prise en charge de la norme J2C pour l'interfaçage avec des supports de persistance divers (en particulier tous ce qui est stockage XML).

Au niveau applicatif, Oracle9iAS TopLink expose des APIs fonctionnellement similaires à celles de JDO pour l'instanciation du gestionnaire de persistance, la démarcation des transactions, et l'exécution de requêtes. De la même façon, le moteur Oracle9iAS TopLink associe au gestionnaire de persistance un cache objet transactionnel entièrement paramétrable et un puissant moteur de requêtes.

Pour la recherche d'objets, les applications utilisant Oracle9iAS TopLink peuvent tirer parti d'une grande variété et richesse de fonctionnalités: en effet, Oracle9iAS TopLink permet aux applications de construire des requêtes dynamiques et polymorphes à l'aide de 3 langages différents :

1. Approche objet :

   - les requêtes peuvent être exprimées avec le langage EJB-QL de la norme EJB et ce pour tous type d'objet.
   - Oracle9iAS TopLink intègre également un puissant framework de requêtage objet basé sur des expressions et des appels de méthodes Java : les « TOP Expressions. »

   - enfin, il existe un mode de requête « par l'exemple » (« *Query By Example* »), dans lequel on fournit une instance « modèle » des objets recherchés. Sur l'objet « modèle », on définit les valeurs recherchées pour tout ou partie des attributs de l'objet, et on peut également spécifier des conditions sur ces valeurs (à l'aide d'opérateurs similaires aux opérateurs SQL « LIKE » et « BETWEEN »).

2. Approche relationnelle :

   - les requêtes peuvent être exprimées en langage SQL, ou via des appels à des procédures stockées.

   - Bien que l'approche objet soit à privilégier, l'intégration de requêtes relationnelle peut être justifiée dans certains cas spécifiques.

A la diversité des langages de requêtage objet disponibles, Oracle9iAS TopLink ajoute également la prise en charge d'optimisations sophistiquées qui permettent d'améliorer notablement les performances lors de l'exécution des requêtes SQL sur la base relationnelle.

Au niveau d'une architecture à base de Oracle9iAS TopLink, on retrouve dans les diverses composantes les similarités avec le modèle JDO. Les applications accèdent à une Session Oracle9iAS TopLink (l'équivalent du gestionnaire de persistance de JDO) qui expose des fonctionnalités permettant d'instancier des objets persistants, d'effectuer des transactions, et d'exécuter des requêtes.

En fait, Oracle9iAS TopLink est partiellement compatible avec la spécification JDO, et implémente notamment l'API du gestionnaire de persistance de JDO. A l'heure actuelle cependant, Oracle9iAS TopLink n'est pas compatible avec l'interface d'implémentation (SPI) de la norme JDO, et ne prend pas non plus en charge le langage JDOQL (EJB-QL pouvant être utilisé comme langage de requêtage normalisé). En fait, Oracle s'inscrit pour l'instant dans une stratégie de veille active sur les développements de cette norme et l'intérêt qu'elle peut susciter auprès de ses clients.

Par rapport aux modèles EJB et JDO, et à la plupart des implémentations existantes basées sur ces normes, Oracle9iAS TopLink bénéficie d'une avance technologique considérable qui se remarque par sa très grande flexibilité et à travers les fonctionnalités sophistiquées qui sont prises en charge.

Parmi les fonctionnalités évoluées de Oracle9iAS TopLink, on citera la diversité des options de mapping disponibles, la gestion évoluée du cache, le support des transactions imbriquées, ainsi que les fonctionnalités sophistiquées offertes pour le requêtage.

- **Cycle de développement avec Oracle9iAS TopLink :**

  Par contraste avec les modèles de développement EJB et JDO, le cycle de développement/déploiement est réduit avec Oracle9iAS TopLink :

  - phase de développement : écriture et compilation des classes Java correspondant aux objets métier.

  - phase de mapping : utilisation de l'atelier de mapping Oracle9iAS TopLink Mapping Workbench pour la configuration du mapping objet/relationnel des classes persistantes. Les informations de mapping sont stockées au sein de fichiers XML (descripteurs XML), L'ensemble des caractéristiques relatives à une application (descripteurs XML, source de données, schéma relationnel) sont stockées dans un fichier Projet.

  - phase de déploiement : Oracle9iAS TopLink génère deux structures XML de déploiement :

    - le descripteur des mappings objet-relationnel

    - le descripteur des sessions (fichier sessions.xml) qui précise comment Oracle9iAS TopLink s'intègre dans l'environnement d'exécution (serveur d'application, base de données, etc.)

  En ce qui concerne le développement d'applications clientes, l'utilisation de Oracle9iAS TopLink se rapproche de l'utilisation de JDO : les applications manipulent les objets persistants directement comme des objets Java simples, et contrôlent le déroulement des transactions de façon explicite et programmatique.

  Au sein d'une application Oracle9iAS TopLink, la première étape consiste à instancier une session TopLink (TopLink Session), qui représente l'interface principale entre l'application, le framework Oracle9iAS TopLink, et le support de persistance. L'objet session établit et maintient une connexion avec la base de données des objets persistants, la liste des descripteurs de mapping correspondants, et gère un cache pour l'accès aux objets.

  Pour instancier l'objet session, la cinématique classique consiste à utiliser le Gestionnaire de Session (SessionManager). Ce dernier permet de récupérer (depuis le fichier sessions.xml) l'ensemble des caractéristiques de la session correspondant à l'application (fichier Projet, source de données, descripteurs XML de mapping, etc.).

  Une fois la session initialisée, l'application peut utiliser ses fonctionnalités pour instancier, lire, et mettre à jour des objets. La session offre également à l'application le moyen de gérer des transactions pour contrôler les opérations de mise à jour sur les objets à travers des objets spécialisés appelés « Units of Work ». Conceptuellement, un « Unit of Work » est l'encapsulation au niveau objet du concept de transaction. Il offre plus de souplesse pour les opérations de mise à jour, en simplifiant la démarcation transactionnelle, en réduisant les opérations de mise à jour en base de données au strict nécessaire, et en les ré-ordonnant automatiquement pour assurer l'intégrité référentielle et éviter les situations d'inter-blocage (« deadlocks »). D'autre part, grâce aux objets « Units of Work », il est possible d'effectuer plusieurs transactions en parallèle, et de réaliser des transactions imbriquées.

Pour la création et l'exécution de requêtes, l'objet session donne également accès au framework de requêtage de Oracle9iAS TopLink. Les objets « Queries » permettent de construire dynamiquement et d'exécuter des requêtes exprimées sous forme d'expressions Java, en langage EJB-QL, ou bien en langage SQL .

A l'instar de JDO, Oracle9iAS TopLink ne maintient qu'une seule copie de chaque objet dans le cache, et autorise les appels concurrents par plusieurs transactions sur les objets. Il est par conséquent primordial également pour les applications Oracle9iAS TopLink de recourir systématiquement à l'utilisation de transactions pour les mises à jour sur les objets : en effet, une application accédant à un objet du cache directement sans passer par une transaction peut potentiellement causer la corruption de l'état de cet objet.

- **Flexibilité au niveau du choix d'architecture**

Oracle9iAS TopLink offre une grande latitude quant aux modèles d'architecture supportés en déploiement. Les applications utilisant Oracle9iAS TopLink peuvent être déployées indifféremment en architecture 2-tiers dans des applications autonomes (« standalone »), ou dans des architectures 3 ou 4 tiers avec ou sans serveur d'application J2EE.

Les fonctionnalités de Oracle9iAS TopLink peuvent notamment être mises à contribution avec beaucoup de profit dans une architecture utilisant la norme EJB :

- avec des EJB session sans état ou avec état, Oracle9iAS TopLink permet d'encapsuler l'accès à des objets métier persistants. Dans ce scénario, on peut procéder à l'intégration entre Oracle9iAS TopLink et les services transactionnels du conteneur EJB à travers JTS.

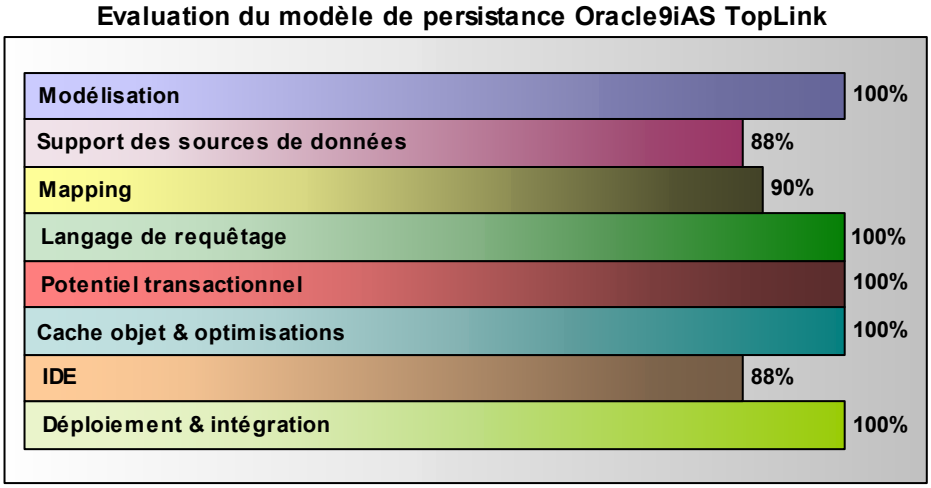
- avec des EJB entité BMP (persistance gérée par le composant), on peut utiliser les fonctionnalités de Oracle9iAS TopLink afin d'implémenter la gestion de la persistance du composant à moindre coût.

- avec des EJB entité CMP : Oracle9iAS TopLink peut être utilisé comme moteur de persistance CMP pour les composants EJB 2.0 ou 1.1 avec les conteneurs d'EJB d'IBM WebSphere Application Server et de BEA WebLogic Server. Oracle9iAS TopLink apporte également aux EJB CMP de WebLogic et de WebSphere des fonctionnalités à haute valeur ajoutée : nous pouvons citer entre autres la possibilité d'utiliser la notion de hiérarchie de beans, l'utilisation des séquences natives des SGBD, la prise en charge de requêtes dynamiques, la possibilité d'appeler des procédures stockées, la possibilité de combiner la persistance de beans et d'objets standard Java, etc.

### 3.4.3 Synthèse des résultats de l'évaluation

La grille d'évaluation détaillée de la couverture fonctionnelle du modèle Oracle9iAS TopLink en matière de persistance est présentée en annexe 5.4.

Le schéma ci-dessous présente la synthèse de l'évaluation fonctionnelle ; les notes, comptabilisées par catégorie, ont été traduites en pourcentages :




Le modèle Oracle9iAS TopLink excelle dans la plupart des domaines. On notera tout particulièrement ses possibilités étendues en matière de requêtage objet, ses capacités de mapping exhaustives, et un potentiel transactionnel remarquable. D'autre part, Oracle9iAS TopLink inclut un cache objet transactionnel de haut-niveau, qui offre des possibilités de paramétrage étendues et intègre de nombreuses optimisations pour l'amélioration des performances (optimisations au niveau de l'exécution des requêtes notamment). Enfin, ce modèle, facilement intégrable au sein d'une architecture J2EE, offre une grande versatilité pour le déploiement. Seuls points perfectibles, la richesse de l'IDE et le support des sources de données. Durant les phases de développement, un environnement de test pour les objets persistants serait un plus. D'autre part, la prise en charge des bases de données objet et des sources de données J2C serait également appréciable.

## 3.5 Perspectives de normalisation en matière de persistance objet

La persistance et le mapping objet-relationnel sont des problématiques extrêmement complexes, et partant, très difficiles à standardiser.

D'autre part, on assiste souvent à des querelles stériles entre partisans d'une norme ou d'une autre (les pro-JDO et les pro- EJB), inconditionnels de l'objet, et farouches défenseurs du modèle relationnel.

Si JDO semble à première vue bien placé pour devenir un standard incontournable, rien ne permet d'affirmer que cette norme détrônera l'architecture EJB ou les solutions de mapping objet-relationnel propriétaires mais éprouvées et bien installées comme Oracle9iAS TopLink. Il est par ailleurs intéressant de noter que Sun semble embarrassé de la dualité JDO et EJB et, loin de se poser en fervent défenseur de JDO, semble pour l'instant continuer à promouvoir le modèle EJB. Enfin, si JDO est pour l'instant relativement bien supporté par les éditeurs de bases de données orientées objet (dont les principaux proposent maintenant des implémentations JDO pour leurs produits), on constate une forte inertie de la part des éditeurs de SGBDR face à ce standard émergent. Dans l'intérêt de tous (utilisateurs et éditeurs) une convergence entre les travaux sur JDO et J2EE sur ce thème de la persistance, ainsi qu'une réelle adhésion du marché, nous semblent plus que souhaitables.

JDO souffre également de défauts de jeunesse bien compréhensibles, et la norme devra évoluer de façon très significative avant de pouvoir recueillir le soutien et l'implication nécessaires de la part des grands éditeurs. Pour l'heure, des acteurs majeurs comme IBM et Oracle semblent peu convaincus par JDO et adoptent une position très réservée vis-à-vis de cette norme, tout en restant attentifs à son évolution ainsi qu'à celle du marché. Ce « manque de maturité » de la norme se retrouve d'ailleurs dans les implémentations JDO disponibles sur le marché : la plupart des produits disponibles actuellement ne proposent qu'un support limité de la spécification, ne disposent pas de fonctionnalités sophistiquées pour la gestion du cache, et ne prennent pas pleinement en compte les spécificités des bases de données relationnelles.

D'autre part, si la spécification reste assez vague sur certains points qui mériteraient d'être clarifiés, elle insiste en revanche sur l'utilisation d'un mécanisme très discutable : le post-traitement de code Java (« *code enhancing* ») pour l'implémentation des mécanismes de persistance. Cette approche est extrêmement restrictive : des frameworks de persistance existants (dont Oracle9iAS TopLink) ont démontré que le recours à ce mécanisme n'était nullement nécessaire pour implémenter une gestion transparente de la persistance.

Parmi les axes de standardisation qui seraient les bienvenus, il apparaît important d'étudier la possibilité d'enrichir la grammaire XML utilisée pour les descripteurs de classes persistantes JDO, afin qu'elle prenne en compte et standardise la description des mappings des attributs des objets sur le support de persistance (en particulier pour le mapping en bases de données relationnelles).

Enfin, le langage JDOQL présente certains défauts, notamment la spécification des requêtes et des paramètres sous forme de chaînes de caractères, ce qui ôte toute possibilité de validation à la compilation. On peut également adresser les mêmes reproches au langage EJB-QL, et il est d'ailleurs étonnant de constater que ces langages de requêtage objet aient été basés tous deux sur une syntaxe textuelle. En effet, il aurait été plus naturel et plus avantageux de modéliser ces langages d'une façon purement objet, à travers les objets du

langage Java, pour bénéficier des avantages de validation à la compilation, et de performances à l'exécution.

En guise de conclusion, rappelons que la norme EJB, élaborée en 1998, a du subir bien des évolutions avant de pouvoir être réellement utilisable, qu'elle reste encore aujourd'hui entachée de certaines lacunes et imperfections, et que les implémentations de serveurs EJB sont encore loin d'être parfaites. Dès lors, on ne saurait que conseiller la plus grande circonspection quant à l'adoption précoce de standards émergents en matière de persistance objet.

# 4. Les atouts d' Oracle9iAS TopLink pour une architecture J2EE

L'évaluation des différentes solutions de persistance a permis de mettre en évidence les atouts de la solution Oracle9iAS TopLink en termes de richesse fonctionnelle et de fonctionnalités techniques sophistiquées pour l'optimisation des performances.

Si Oracle9iAS TopLink dispose d'avantages indéniables sur les autres modèles étudiés, il ne se pose pas pour autant uniquement en tant que solution concurrente. En effet, l'une des forces de Oracle9iAS TopLink réside dans sa complémentarité avec d'autres modèles existants. Ainsi, l'un des attraits de Oracle9iAS TopLink est sa capacité à pouvoir être employé de façon complémentaire à la norme EJB, en simplifiant la mise en œuvre de composants persistants, que ce soit en mode BMP (persistance gérée par le composant) ou en mode CMP (persistance gérée par le conteneur). En résumé, Oracle9iAS TopLink se positionne comme une solution s'adaptant a tout type d'architecture, et apportant de part sa maturité de la valeur ajoutée à chacune d'elle.

En tout état de cause, le choix d'une solution de persistance objet est une décision stratégique et délicate, qui peut avoir des conséquences sérieuses sur le déroulement et l'aboutissement d'un projet. L'adoption d'un produit comme Oracle9iAS TopLink peut donc susciter certaines réticences ou interrogations bien légitimes, auxquelles on se doit d'apporter des éléments de réponse argumentés. Par conséquent, il apparaît nécessaire de récapituler précisément les atouts de ce modèle face à d'autres alternatives, et de répondre aux principales interrogations qui peuvent être posées.

### 4.1.1 Principales interrogations et réticences face à l'adoption de Oracle9iAS TopLink

Parmi les interrogations ou les réticences face à l'utilisation de Oracle9iAS TopLink, les plus communes sont les suivantes :

- ***Au sein de J2EE, EJB assure déjà la persistance objet – Oracle9iAS TopLink est donc superflu***
  La vocation première de la norme EJB est de fournir une architecture de composants distribués pour les applications d'entreprise. A travers le modèle EJB entité, la norme propose un modèle de composants métier persistants et transactionnels.
  Cependant, le modèle des EJB entité est un modèle de composants, relativement lourd, et ne couvre pas les besoins de persistance d'objets Java légers. C'est d'ailleurs en partie pour couvrir ces besoins que la spécification JDO a vu le jour.
  Oracle9iAS TopLink est en partie comparable à JDO, et est utilisé en premier lieu pour modéliser des objets Java simples persistants. Par rapport aux implémentations JDO disponibles, Oracle9iAS TopLink bénéficie d'une grande maturité et de fonctionnalités sophistiquées et éprouvées.
  Cependant, même avec une architecture EJB entité, Oracle9iAS TopLink apporte également une forte valeur ajoutée : en effet, Oracle9iAS TopLink peut être facilement utilisé comme mécanisme de persistance des EJB entité BMP, et, avec certains serveurs d'application, peut également être utilisé pour la persistance d'EJB CMP.

L'utilisation conjuguée de Oracle9iAS TopLink avec le modèle EJB entité permet non seulement des gains de productivité appréciables, mais également des gains de performances, et apporte aussi une plus grande flexibilité au niveau des requêtes sur les objets (support des requêtes dynamiques).

- ***Oracle9iAS TopLink n'est pas un standard, c'est une solution propriétaire***
  Comme nous l'avons vu, plusieurs standards de persistance objet co-existent, dont les normes EJB Entité et JDO. Ces normes sont encore très jeunes, et amenées à fortement évoluer : leur utilisation ne garantit donc pas forcément la pérennité des applications face au changement.
  Oracle9iAS TopLink est un framework certes propriétaire, mais parfaitement intégré avec les standards de J2EE, notamment avec les normes JDBC, JTA/JTS, et EJB. Oracle9iAS TopLink implémente également partiellement la norme JDO. Ainsi, une application J2EE utilisant un modèle d'objets Java persistants avec Oracle9iAS TopLink peut indifféremment fonctionner sur Oracle 9IAS, IBM WebSphere Application Server ou BEA WebLogic Application Server avec Oracle9iAS TopLink pour Java.
  D'autre part, avec de bonnes pratiques de développement, il est possible de clairement identifier et de minimiser dans le code les dépendances aux caractéristiques propriétaires d'Oracle9iAS TopLink.
  Enfin, Oracle9iAS TopLink bénéficie d'une large base de clients installée et de nombreuses années d'existence. C'est un produit éprouvé, aux qualités reconnues, qui a traversé avec succès deux rachats successifs. L'avenir du produit, désormais entre les mains d'Oracle, semble plus que jamais assuré.

- ***Le coût d'acquisition de Oracle9iAS TopLink est trop élevé***
  Oracle9iAS TopLink est un produit élaboré qui offre des fonctionnalités de haut-niveau pour la mise en place de la persistance au sein d'architectures complexes.
  S'il est vrai que son coût de déploiement est élevé, il reste inférieur aux coûts de déploiement d'un serveur d'EJB, et comparable à ceux d'une solution JDO.
  D'autre part, il faut rappeler qu' au coût des licences de déploiement d'un projet, il faut ajouter le coût des développements. Les économies que permet de réaliser Oracle9iAS TopLink au niveau du développement d'une couche de persistance d'objets métier permettent de compenser rapidement le coût de déploiement du produit. Néanmoins, si le modèle d'objets est très simple et si la durée de vie de l'application est faible, Les gains apportés par Oracle9iAS TopLink ne compenseront pas son coût d'acquisition.
  Il faut également rappeler que la licence TopLink est incluse avec le serveur d'application Oracle9i Application Server, et qu'il n'y a par conséquent aucun surcoût dans ce cas.

- ***Oracle9iAS TopLink est un produit complexe à maîtriser***
  En vérité, c'est la problématique de la persistance objet en elle-même qui est complexe. Loin de compliquer cette problématique, Oracle9iAS TopLink permet bien au contraire de la simplifier pour le développeur, et de la rendre aussi transparente que possible. D'autre part, par comparaison, la modélisation d'objets Oracle9iAS TopLink et la réalisation d'applications utilisant les fonctionnalités de Oracle9iAS TopLink sont des tâches beaucoup moins complexes que la réalisation de composants EJB et leur utilisation au sein d'applications. D'autre part, en ce qui concerne la réalisation d'applications à base d'EJB, Oracle9iAS TopLink contribue précisément à réduire la complexité de la couche de persistance, et à favoriser la portabilité des développements entre serveurs d'applications.

### 4.1.2 Principaux atouts de Oracle9iAS TopLink en tant que solution de persistance objet

Le pragmatisme et le souci d'adaptabilité et d'efficacité qui ont présidé à la conception et à l'évolution de Oracle9iAS TopLink se retrouvent dans chacun des mécanismes mis en œuvre, et dans chaque fonctionnalité offerte par le produit. L'ensemble de ces caractéristiques confère à Oracle9iAS TopLink de précieux atouts:

- **Un haut degré de portabilité**

  En tant que produit 100% Java, Oracle9iAS TopLink bénéficie d'un haut niveau de portabilité entre plates-formes. Il peut être utilisé sur toutes les plates-formes Java certifiées : la majorité des systèmes Unix et Linux, ainsi que sous Windows (NT et 2000). Cette portabilité s'applique non seulement au framework Oracle9iAS TopLink, mais également à l'atelier de mapping graphique (Mapping Workbench) .

  Enfin, Oracle9iAS TopLink est compatible avec la plupart des SGBDR existants pour lesquels il existe un pilote JDBC.

- **Des performances élevées**

  Les mécanismes sophistiqués mis en œuvre par Oracle9iAS TopLink apportent un haut niveau de performances lors de la manipulation d'objets Java persistants.

  En premier lieu, le cache objet transactionnel intégré à Oracle9iAS TopLink offre un niveau de paramétrage très fin. Il est possible de configurer des modes de fonctionnement et une taille de cache pour chaque type d'objet, et de choisir le meilleur compromis entre performances et quantité de mémoire utilisée.

  D'autre part, plusieurs mécanismes spécifiques sont mis en œuvre pour accroître les performances lors de la manipulation de larges quantités d'objets, ou de graphes d'objets complexes à travers des requêtes. Oracle9iAS TopLink utilise au maximum les capacités du SGBD pour optimiser les performances. Outre l'utilisation d'un pool de connexions, on peut également noter l'emploi des curseurs (pour optimiser la récupération des résultats d'une requête), ainsi que l'utilisation de séquences pour la gestion de l'identité objet.

  Enfin, Oracle9iAS TopLink intègre une fonctionnalité unique, un service de *profiling* évolué, qui permet d'obtenir des statistiques très précises sur les performances du cache et des requêtes.

- **Une productivité importante**

  La productivité représente très souvent un critère différentiateur en matière de développement d'applications d'entreprise. Ce critère est d'autant plus important dans le monde Java où se côtoient plusieurs technologies et plusieurs produits .

  Dans le domaine de la persistance objet tout particulièrement, la productivité est un facteur très important. En effet, la réalisation d'une couche de persistance, si elle n'est pas assistée par l'emploi d'une technologie appropriée, peut représenter jusqu'à 40% des développements liés à un projet.

L'utilisation d'une solution de persistance objet à travers les modèles EJB, JDO ou Oracle9iAS TopLink permet de réduire le cycle de développement de manière considérable. Le développement de composants EJB, s'il reste assez lourd, est néanmoins facilité de nos jours avec l'utilisation d'environnements de développement offrant des fonctionnalités dédiées. Cependant, le modèle Oracle9iAS TopLink, tout comme le modèle JDO, permet une productivité bien supérieure au modèle EJB, car il autorise la modélisation des entités métier persistantes comme des objets Java simples. De ce fait, la modélisation des objets métier et le développement des applications peuvent s'effectuer avec n'importe quel environnement de développement Java classique.

En ce qui concerne le mapping objet-relationnel, l'atelier de mapping de Oracle9iAS TopLink, de part sa facilité d'utilisation, permet des gains de productivité importants.

- **Des fonctionnalités de requêtage sophistiquées**

  L'un des atouts majeurs de Oracle9iAS TopLink réside dans la diversité et la puissance de ses fonctionnalités de requêtage. En effet, le framework offre aux applications 3 langages de requêtage différents : SQL, EJB-QL, et un langage propriétaire objet qui permet d'exprimer des requêtes sous forme d'expressions et d'appels de méthodes Java. Tous ces langages peuvent être utilisés par les applications pour créer des requêtes dynamiques à l'exécution. De plus, il est possible d'exécuter des requêtes sur les objets du cache en mémoire, ce qui offre un niveau de performance bien plus élevé que l'exécution des requêtes sur la base de données.

- **Une intégration poussée avec les serveurs d'application J2EE**

  Oracle9iAS TopLink peut être facilement intégré à n'importe quel serveur d'application J2EE du marché. Les fonctionnalités de Oracle9iAS TopLink peuvent être utilisées non seulement pour la persistance d'objets Java simples, mais également pour la persistance de composants EJB : Oracle9iAS TopLink permet d'implémenter la logique de persistance de composants EJB entité BMP, et peut également être utilisé avec certains serveurs d'applications (IBM WebSphere Application Server, BEA WebLogic Server, et bientôt Oracle9i Application Server) comme moteur de persistance pour les EJB entité CMP.

### 4.1.3 Conclusion

Oracle9iAS TopLink offre une solution de persistance objet complète et éprouvée en environnement Java, qui s'accommode de tout type d'architecture. Facilement intégrable au sein d'une architecture J2EE 3 ou 4 tiers, Oracle9iAS TopLink se présente non pas seulement comme une alternative, mais comme une solution complémentaire au modèle EJB.

Les principaux atouts de Oracle9iAS TopLink peuvent être résumés par les trois principes fondateurs sur lesquels repose le produit :

- **Flexibilité** : Oracle9iAS TopLink n'impose pratiquement aucune restriction, que ce soit au niveau de la modélisation, au niveau du mapping, au niveau du SGBD, ou au niveau de l'architecture applicative. De même, Oracle9iAS TopLink n'impose aucune restriction quant à la manipulation des données. Plusieurs langages de requêtage sont supportés, et Oracle9iAS TopLink permet de tirer parti du meilleur des approches relationnelle et objet.

- **Productivité** : l'un des principaux objectifs de Oracle9iAS TopLink est de simplifier au maximum l'implémentation de la couche de persistance objet au sein des applications. Pierre angulaire de ce principe, l'outil de mapping (Mapping Workbench) de Oracle9iAS TopLink offre un haut niveau de productivité lors de la phase de mapping objet-relationnel. D'autre part, la simplicité du modèle de programmation applicatif, la flexibilité au niveau de l'architecture, et la puissance des fonctionnalités offertes, notamment en matière de requêtage et de gestion transactionnelle, apportent une grande souplesse et un haut niveau de productivité lors de la réalisation d'applications. Ainsi, que ce soit au niveau de la modélisation, du mapping objet-relationnel, ou de la conception des applications, les gains de productivité potentiels sont considérables.

- **Performance** : le souci de performance est également central dans Oracle9iAS TopLink. Les différentes techniques mises en œuvre se justifient principalement par le gain de performances qu'elles permettent de réaliser. Les possibilités de paramétrage très fines, disponibles non seulement au niveau du cache objet, mais également au niveau des requêtes, et au niveau de la définition des mappings eux-mêmes, permettent d'adapter de façon optimale le comportement de la couche de persistance en fonction de chaque type d'application.

# 5. Annexes

## 5.1 Grille de critères pour l'évaluation détaillée des solutions de persistance objet

| COUVERTURE FONCTIONNELLE EN MATIERE DE PERSISTANCE OBJET<br>–<br>Grille de critères pour l'évaluation | |
|---|---|
| **1. TRANSPARENCE DE MODELISATION** | |
| **Critère** | **Description** |
| **1.1** Niveau d'intrusion | Le mécanisme de persistance doit être le moins intrusif et le plus transparent possible au niveau du code des objets métier d'une part, et du code applicatif d'autre part. |
| **1.2** Restrictions sur les types de données pouvant être persistants | Le mécanisme de persistance impose-t-il des restrictions quant au type des objets Java pouvant être persistés ? |
| **1.3** Gestion de l'identité objet | Existe-t-il des mécanismes pour assurer la gestion et l'intégrité de l'identité des objets ?<br>Quelles sont les contraintes imposées par le mécanisme de persistance au niveau de l'identité des objets ? |
| **1.4** Support des objets persistants à granularité fine | L'architecture de persistance conditionne le niveau de granularité utilisable pour la modélisation des objets persistants. Une infrastructure lourde et intrusive limitera la modélisation à des entités à granularité large, tandis qu'une infrastructure légère et flexible permet d'envisager une modélisation à granularité fine. |
| **1.5** Support de l'héritage | Le mécanisme de persistance autorise-t-il l'utilisation de relations d'héritage dans le modèle objet ? |
| **1.6** Support des associations entre objets | Le mécanisme de persistance permet-il l'utilisation de relations d'association entre objets (relations 1-1, 1-N et N-N) au sein du modèle objet ? |
| **2. SUPPORT DES SOURCES DE DONNEES POUR LA PERSISTANCE** | |
| **Critère** | **Description** |
| **2.1** Support des SGBDR pour le mapping | L'outil doit supporter les principaux SGBDR du marché. |
| **2.2** Support de schémas existants en SGBDR | L'outil doit être capable d'assurer la projection du modèle objet sur un modèle relationnel existant (contraintes arbitraires de mapping). |
| **2.3** Génération automatique du schéma métier en SGBDR | L'outil doit être capable de générer automatiquement un schéma relationnel adapté à la projection du modèle objet. |
| **2.4** Support des sources de données J2C et d'autres référentiels pour le mapping (bases de données objet, fichiers XML, annuaires LDAP, etc.) | L'outil permet-il d'employer des référentiels autres que les SGBDR pour la persistance (notamment à travers des adaptateurs de ressource J2C) ? |
| **3. CAPACITES DE MODELISATION ET DE PROJECTION (Mapping)** | |
| **Critère** | **Description** |
| **3.1** Types de données supportés pour les mappings | Quels sont les types de données (classes et primitives du langage Java) qui sont supportés pour les champs persistants ? Quelles sont les éventuelles restrictions ou contraintes ? |
| **3.2** Support du mapping multi-table | Possibilité de répartir les attributs d'un objet dans les champs de plusieurs tables pour le mapping en SGBD. |

| | |
|---|---|
| **3.3** Support du mapping par sérialisation | Possibilité de mapper des objets par sérialisation dans des champs binaires (champs BLOB) en SGBD. |
| **3.4** Prise en charge de conversions de types de données et de transformations spécifiques | L'outil doit prendre en charge automatiquement les conversions entre types des objets et types de la source de données. Dans la mesure du possible, l'outil doit autoriser le choix d'un type de données particulier pour la persistance et effectuer toute conversion de valeur nécessaire (exemple : mapping d'un type 'Date' vers un type numérique, ou d'un type 'décimal' vers un type « chaîne de caractères »).<br>D'autre part, l'outil peut proposer la réalisation de transformations spécifiques. Par exemple, il peut proposer le mapping d'un type énuméré vers un type numérique ou alphanumérique. (Ex : mapping de l'énumération { féminin, masculin } sur un type numérique { 1, 2 } ou caractère { 'F', 'M' }. |
| **3.5** Gestion automatique de clés techniques pour la projection de l'identité objet | L'outil doit supporter le concept de « clés techniques » (« surrogate keys ») et proposer la gestion automatique de l'identité objet à travers la génération automatique et paramétrable (type de l'identifiant, utilisation de séquences, etc.) d' OIDs (object ids). |
| **3.6** Prise en compte de l'héritage | L'outil de mapping doit être capable de projeter les relations d'héritage du modèle objet sur le support de persistance.<br>Par exemple, en SGBDR, plusieurs stratégies peuvent être proposées pour le mapping d'une hiérarchie de classes:<br>- une table par hiérarchie de classes<br>- une table par classe (abstraite ou concrète)<br>- une table par classe concrète |
| **3.7** Prise en compte des relations d'association entre objets. Support de relations uni- et bi-directionnelles avec différentes cardinalités: 1-1, 1-N, N-N. | L'outil de mapping doit être capable de projeter des associations entre objets sur le support de persistance. Il doit être capable de gérer différents types de cardinalité (1-1, 1-N, N-N) et supporter des associations unidirectionnelles et bidirectionnelles.<br>La mise à jour des relations d'association, et le maintien de l'intégrité référentielle doivent être gérés de façon transparente par le moteur de persistance.<br>Le moteur de persistance doit également offrir la meilleure flexibilité possible quant au type de classes sous-jacentes pouvant être utilisées pour la représentation des associations (Collections Java, vecteurs, tableaux, etc.). |
| **3.8** Prise en compte des suppressions en cascade (relations d'association 1-N) | L'outil de mapping doit proposer la prise en charge optionnelle des suppressions en cascade pour les objets liés par une relation d'association de cardinalité 1-N : lorsque cette option est activée, la suppression d'un objet du côté unaire de la relation entraîne la suppression automatique du ou des objets liés du côté n-aire de la relation. |
| **3.9** Portabilité du mécanisme de description du mapping | Quel est le mécanisme de description du mapping employé (dictionnaire de données, classes Java, fichiers XML, etc.). Ce mécanisme répond-il à un standard, est-il inter-opérable avec d'autres implémentations ? |
| **3.10** Compatibilité au niveau binaire | Possibilité de réutiliser une application / un modèle d'objets persistants pour les déployer vers un autre environnement , ou de modifier les mappings (projection sur un autre support de persistance) sans recompilation. |
| **4. LANGAGE DE REQUETAGE OBJET** | |
| **Critère** | **Description** |
| **4.1** Présence d'un ou plusieurs langages de requêtage objet. | L'outil doit inclure un langage de requêtage permettant la navigation à travers un graphe objet, l'expression de clauses de sélection (conditions), et de tri des résultats. |
| **4.2** Possibilité d'effectuer des requêtes dynamiques | L'outil doit proposer un mécanisme permettant de générer des requêtes dynamiquement à l'exécution, par programmation. |
| **4.3** Support des requêtes polymorphes | Possibilité de spécifier des requêtes qui renverront des objets d'une classe donnée ainsi que de ses sous-classes. (Exemple : une sélection d'objets de type Compte Bancaire doit pouvoir retourner des instances des sous-classes différentes, telles que Compte Courant, Compte Joint, Compte Épargne, etc.) |
| **4.4** Optimisations apportées par le moteur de requêtes | Le moteur de requêtes peut offrir des fonctionnalités permettant l'optimisation des requêtes : par exemple, la possibilité d'effectuer les requêtes en mémoire sur le cache objet, l'utilisation de curseurs pour la navigation par lots à travers les résultats des requêtes, ou l'utilisation d'un algorithme de lecture de groupes d'objets par lot (« batch reading »). |

| 5. POTENTIEL TRANSACTIONNEL | |
|---|---|
| **Critère** | **Description** |
| **5.1** Prise en compte des transactions au niveau objet par le gestionnaire de persistance | Le gestionnaire de persistance doit obligatoirement prendre en compte le support des transactions au niveau **objet** (et non pas au niveau « données » en SGBDR). |
| **5.2** Flexibilité pour la démarcation des transactions | Le gestionnaire de persistance doit permettre de contrôler finement la démarcation des transactions. La démarcation des transactions peut être déclarative ou programmatique. |
| **5.3** Support des transactions linéaires | Le gestionnaire de persistance doit supporter le modèle de transactions linéaires |
| **5.4** Support des transactions imbriquées | Le gestionnaire de persistance peut optionnellement proposer le modèle de transactions imbriquées |
| **5.5** Support des transactions distribuées et du protocole de validation à 2 phases et interopérabilité avec d'autres moniteurs transactionnels | L'outil intègre-t-il, ou permet-il de s'interfacer avec un moniteur transactionnel via JTA afin de faire participer ses transactions à des transactions distribuées ? (Transactions X/Open DTP ou Corba OTS) |
| **6. CACHE OBJET ET OPTIMISATIONS** | |
| **Critère** | **Description** |
| **6.1** Présence d'un cache objet transactionnel | Le moteur de persistance doit intégrer un cache objet transactionnel, qui peut être paramétrable |
| **6.2** Gestion de l'accès concurrent à travers des verrous partagés et exclusifs sur les objets | Le cache doit inclure un mécanisme de gestion des accès concurrents sur les instances d'objets basé sur la pose de verrous sur les enregistrements de la source de données. Deux types d'accès concurrents doivent être gérés :<br>- mode exclusif ou « pessimiste » : pose d'un verrou en écriture, signifie que seule la transaction ayant verrouillé l'objet peut accéder à son état, que ce soit en lecture ou en écriture.<br>- mode partagé ou « optimiste » : aucun verrou n'est posé sur les données, ce qui signifie que plusieurs transactions peuvent accéder en lecture et en écriture à l'état d'un objet. Ce mode est dit « optimiste » car il repose sur l'hypothèse qu'entre l'accès à l'état de l'objet et la modification de cet état par l'application, aucune autre transaction ne modifie l'état de cet objet. Cependant, le risque de violation d'intégrité des données par modification concurrente doit être pris en compte, et géré : il est nécessaire de mettre en œuvre un mécanisme de détection *a posteriori* de l'invalidité de l'hypothèse de départ (aucune modification concurrente) et de résolution des collisions (rollback de la transaction courante si l'objet a été modifié par une transaction concurrente).<br>La possibilité de personnaliser les stratégies de verrouillage est un plus. |
| **6.3** Gestion du clustering et d'un cache distribué | Le produit permet-il de distribuer les objets sur plusieurs serveurs, et est-il capable de gérer un cache distribué ? |
| **6.4** Possibilités de paramétrage du cache objet | L'outil intègre-t-il un système de configuration du cache objet ? |
| **6.5** Optimisations de lecture et de parcours du graphe objet | Le moteur de persistance intègre-t-il des fonctionnalités d'optimisation pour la lecture de graphes objets : lazy loading, batch reading, indirection, etc. |
| **7. ENVIRONNEMENT DE DEVELOPPEMENT (IDE)** | |
| **Critère** | **Description** |
| **7.1** Présence d'un IDE, ou intégration avec un environnement de développement tiers | L'outil propose-t-il un IDE, ou peut-il être intégré à un ou plusieurs environnements de développement tiers? |
| **7.2** Présence d'un assistant visuel pour la configuration du mapping objet/relationnel | L'outil propose-t-il un outil qui permette de configurer graphiquement la projection des objets sur la base de données ? |
| **7.3** Présence d'un environnement de test | L'outil doit comporter un environnement de test afin de permettre de tester la persistance des objets avant leur déploiement |
| **7.4** Présence d'un profiler | L'outil intègre-t-il un outil de profiling permettant de mesurer les performances des opérations de persistance et de requêtage ? |

| 8. DEPLOIEMENT ET INTEGRATION | |
|---|---|
| **Critère** | **Description** |
| **8.1** Modèles de déploiement par tiers supportés | Quels sont les différents modèles d'architecture supportés pour le déploiement :<br>- architecture 2 tiers<br>- architecture 3 tiers<br>- architecture 4 tiers |
| **8.2** Intégration dans les serveurs d'application | Le service de persistance de l'outil peut-il être embarqué dans un serveur d'application J2EE ? |
| **8.3** Niveau d'intégration dans la norme J2EE | Quel est le niveau d'interopérabilité entre l'outil de mapping , le moteur de persistance et ses services (gestion des transactions) et les standards de J2EE ? Exemple, support transactionnel réalisé à travers JTA/JTS, support de JDBC et de J2C pour l'accès aux sources de données, et éventuellement possibilité d'utiliser le service de persistance comme moteur de persistance pour les EJB entité. |
| **8.4** Complémentarité avec d'autres standards de persistance | Ex : possibilité d'utiliser le moteur un moteur de persistance (Oracle9iAS TopLink, JDO) avec les EJB entité BMP et/ou CMP<br>Compatibilité de Oracle9iAS TopLink avec certaines interfaces de JDO |

# 5.2 Résultats de l'évaluation détaillée pour le modèle EJB

| Modèle de persistance EJB 2.x Entité (CMP) | | |
|---|---|---|
| **1. TRANSPARENCE DE MODELISATION** | | |
| **Critère** | **Note** | **Commentaire** |
| **1.1** Niveau d'intrusion | 0 | Le niveau d'intrusion inhérent à la norme EJB est -élevé : les EJB entité ne sont pas des objets Java simples, mais des composants développés selon un modèle pré-établi :<br>- nécessité de développer 3 à 4 classes différentes (interface de fabrique, interface cliente, classe d'implémentation, et classe de clé primaire) pour chaque composant métier.<br>- nécessité d'implémenter des interfaces, comprenant des méthodes de callback et d'attributs techniques dans la classe d'implémentation du composant. |
| **1.2** Restrictions sur les types de données pouvant être persistants | 0 | Le modèle EJB impose un modèle de composants persistants spécifique. Ce modèle ne permet pas de faire persister des objets Java simples. |
| **1.3** Gestion de l'identité objet | 2 | La gestion de l'identité des EJB repose sur la définition de classes de clés primaires: à chaque composant EJB doit être associée une classe de clé primaire. Le conteneur assure l'unicité des objets en fonction de leur clé primaire. Cette unicité est également assurée pour des objets déployés sur plusieurs conteneurs EJB en cluster.<br>L'identité d'une instance d'EJB est fixe : une fois créé, le composant ne doit pas modifier son identité (sa clé primaire). |
| **1.4** Support des objets persistants à granularité fine | 1 | Avec l'apparition des interfaces clientes locales, la gestion des associations inter-composants, et la gestion des objets dépendants (*dependent value classes*), les EJB entité permettent d'envisager une modélisation à granularité fine. Cette affirmation doit cependant être nuancée : la manipulation de collections importantes d'objets par exemple s'avère peu performante du fait de l'éclatement d'une entité en plusieurs objets Java. Il reste donc préférable avec les EJB de conserver une approche de modélisation à granularité assez large. |
| **1.5** Support de l'héritage | 0 | La spécification EJB interdit l'utilisation des mécanismes d'héritage sur les composants. L'explication relève d'une considération technique : les composants sont instanciés à travers l'interface de fabrique (EJBHome) qui renvoie une interface cliente (EJBObject ou EJBLocalObject) vers le composant ; or, le mécanisme de conversion de type utilisé (type narrowing de Corba) interdit les « down-cast » ou les « up-casts » de l'interface cliente vers une super-interface ou une interface dérivée. Ceci interdit en particulier la mise en œuvre de requêtes polymorphes dans les méthodes de recherche (« finder methods »). D'autre part, d'autres considérations limitent fortement la possibilité de l'utilisation de l'héritage : contraintes liées à la nécessité de gérer des arbres d'héritage multiples (classe d'implémentation et interfaces), et problème spécifique de l'héritage des classes de clé primaire.<br>Bien que certains éditeurs de serveurs EJB évoquent la possibilité d'utiliser des mécanismes d'héritage de « contournement » en EJB, il faut bien se rendre à l'évidence: la norme EJB ne permet pas en l'état l'utilisation simple et intuitive de relations d'héritage pour la modélisation des composants EJB. |
| **1.6** Support des associations entre objets | 2 | La norme EJB propose la gestion automatique de relations inter-composants gérées par le conteneur (CMR). Les associations de cardinalité 1-1, 1-N, et N-N sont supportées, de manière uni-directionnelle ou bi-directionnelle. Pour les relations à cardinalité n, le conteneur impose l'utilisation des classes standard de Java2 implémentant l'interface *Collection*. Le conteneur gère automatiquement l'intégrité référentielle des relations d'association, et propose la gestion automatique des suppressions en cascade (*cascade deletes*) pour les relations de cardinalité 1-n. |

| 2. SUPPORT DES SOURCES DE DONNEES POUR LA PERSISTANCE | | |
|---|---|---|
| **Critère** | **Note** | **Commentaire** |
| **2.1** Support des SGBDR pour le mapping | 2 | Les serveurs d'EJB du marché permettent l'utilisation de tout SGBDR disposant d'un pilote JDBC 2.0 implémentant l'interface *DataSource*. Le support des transactions distribuées requiert un SGBDR compatible avec le protocole de validation à 2 phases XA, dont le pilote JDBC expose l'interface *XaDataSource*. |
| **2.2** Support de schémas existants en SGBDR | 2 | Les principaux serveurs d'EJB du marché permettent d'utiliser un schéma de base de données existant pour la projection. |
| **2.3** Génération automatique du schéma métier en SGBDR | 2 | Les principaux serveurs d'EJB du marché sont dotés d'un outil de projection capable de générer automatiquement un schéma de base de données pour la persistance d'un composant EJB |
| **2.4** Support des sources de données J 2C et d'autres référentiels pour le mapping (bases de données objet, fichiers XML, annuaires LDAP, etc.) | 0 | La norme EJB ne donne aucune précision quant à la nature des supports de persistance pouvant être utilisés, et laisse entière latitude aux implémentations à ce sujet. Cependant, la spécification prend comme hypothèse générique l'utilisation d'un SGBDR.<br>La majorité des serveurs d'EJB actuels permettent uniquement l'emploi des SGBDR comme support de persistance. |
| **3. CAPACITES DE MODELISATION ET DE PROJECTION (Mapping)** | | |
| **Critère** | **Note** | **Commentaire** |
| **3.1** Types de données supportés pour les mappings | 2 | La norme EJB permet l'utilisation de la plupart des objets Java en tant qu'attributs persistants : primitives Java et classes « wrappers » correspondantes pour les mappings simples d'attributs de bases de données, et objets sérialisables (pour les objets dépendants) qui sont mappés vers des champs binaires (BLOB) en SGBDR. |
| **3.2** Support du mapping multi-table | 0 | La spécification EJB ne couvre pas les détails techniques de la projection du modèle EJB sur le support de persistance. Les capacités de projection sont donc fonction des implémentations de serveurs EJB.<br>En pratique, la grande majorité des serveurs d'EJB ne prennent pas en charge le mapping multi-table. |
| **3.3** Support du mapping par sérialisation | 2 | La norme EJB 2.0 introduit le concept d'objets dépendants (*dependent value classes*) qui peuvent être utilisés en tant que champs persistants d'un objet. Il est clairement précisé que les objets dépendants doivent obligatoirement être sérialisables. En fait, ce qui est sous-entendu par la spécification, et mis en pratique par les implémentations des éditeurs, est que ces objets sont destinés à être sérialisés dans des champs binaires (de type BLOB). |
| **3.4** Prise en charge de conversions de types de données et de transformations spécifiques | 0 | Les modules de mapping proposés avec les serveurs EJB utilisent des correspondances fixes entre types objet et types de données en SGBDR. Ils n'autorisent pas de conversions arbitraires de types ou de transformations spécifiques entre valeurs des attributs objet et valeurs stockées sur le support relationnel. |
| **3.5** Gestion automatique de clés techniques pour la projection de l'identité objet | 2 | La norme EJB prévoit la possibilité de déférer la définition de la clé primaire des composants à la phase de déploiement. Dans ce cas, la plupart des serveurs d'EJB proposent au déploiement la gestion transparente de l'identité objet en générant des identifiants objet techniques (surrogate keys). Certaines implémentations permettent également l'utilisation de tables de séquence pour la génération automatique des identifiants objet., et l'utilisation de séquences natives de certains SGBD. |
| **3.6** Prise en compte de l'héritage | 0 | L'utilisation des mécanismes d'héritage dans la modélisation EJB est impossible. |
| **3.7** Prise en compte des relations d'association entre objets. Support de relations uni- et bi-directionnelles avec différentes cardinalités: 1-1, 1-N, N-N. | 2 | Les associations de cardinalité 1-1, 1-N, et N-N sont supportées, de manière uni-directionnelle ou bi-directionnelle. Pour les relations à cardinalité n, la norme impose l'utilisation des classes standard de Java2 implémentant l'interface *Collection*. |
| **3.8** Prise en compte des suppressions en cascade (relations d'association 1-N) | 2 | Les suppressions en cascade pour les EJB entité dans une relation 1-N peuvent être prises en charge automatiquement par le conteneur (réglage optionnel dans le descripteur de déploiement de l'EJB). |
| **3.9** Portabilité du mécanisme de description du mapping | 0 | La description du mapping objet-relationnel n'est pas standardisée par la norme EJB. La plupart des implémentations EJB se basent sur l'utilisation de descripteurs XML pour le mapping, mais les schémas utilisés ne sont pas portables d'une implémentation à l'autre. |

| | | |
|---|---|---|
| **3.10** Compatibilité au niveau binaire | 1 | Les composants EJB sont partiellement portables au niveau binaire entre implémentations: la classe d'implémentation et les interfaces sont portables entre implémentations, mais les classes proxy (stubs et skeletons) doivent être régénérées. |

## 4. LANGAGE DE REQUETAGE OBJET

| Critère | Note | Commentaire |
|---|---|---|
| **4.1** Présence d'un ou plusieurs langages de requêtage objet. | 2 | La spécification EJB inclut un langage de requêtage objet basé sur SQL : EJB-QL. Ce langage permet le parcours de graphes et l'utilisation de conditions logiques. La norme EJB 2.1 apporte des améliorations : support des clauses de tri et support de fonctions d'agrégat (average, minimum, maximum, etc.) |
| **4.2** Possibilité d'effectuer des requêtes dynamiques | 0 | Non. La logique des requêtes est spécifiée durant la phase de déploiement du composant, dans le langage EJB-QL. Les requêtes, paramétrables, sont implémentées sous forme de méthodes de recherche (custom finders) sur les composants eux-mêmes.<br>La norme EJB ne prévoit aucun mécanisme pour l'exécution de requêtes dynamiques à partir des applications clientes. |
| **4.3** Support des requêtes polymorphes | 0 | Non (pas de support de l'héritage). |
| **4.4** Optimisations apportées par le moteur de requêtes | 0 | La norme EJB ne couvre pas l'utilisation de techniques d'optimisation pour l'exécution des requêtes.<br>Les implémentations disponibles sur le marché ne proposent pas d'optimisations pour l'exécution des requêtes. |

## 5. POTENTIEL TRANSACTIONNEL

| Critère | Note | Commentaire |
|---|---|---|
| **5.1** Prise en compte des transactions au niveau objet par le gestionnaire de persistance | 2 | Le conteneur d'EJB gère les transactions au niveau des composants EJB. Le niveau de granularité des transactions est assez fin (au niveau appel de méthode métier). |
| **5.2** Flexibilité pour la démarcation des transactions | 2 | La norme EJB propose 2 modes de démarcation des transactions :<br>- déclarative, au niveau du déploiement. Il est possible de choisir parmi plusieurs attributs transactionnels pour adapter finement la démarcation au niveau de chaque méthode métier des composants EJB. La gestion des transactions est alors effectuée automatiquement par le conteneur d'EJB.<br>- programmatique : la démarcation des transactions est à la charge du composant qui utilise l'API JTA pour contrôler explicitement le déroulement de ses transactions.<br>Le contexte transactionnel peut également être propagé du client vers le composant, ce qui permet de contrôler la démarcation des transactions au niveau de l'application.<br>Pour les EJB entité (objets persistants), seul le mode de démarcation déclaratif avec gestion par le conteneur est possible. Cependant, on peut utiliser des EJB Session avec démarcation programmatique des transactions comme façades vers des EJB entité. |
| **5.3** Support des transactions linéaires | 2 | Oui. |
| **5.4** Support des transactions imbriquées | 0 | Non – ni la norme EJB 2.0, ni la spécification JTA 1.0, ni l'API JDBC 2.0 ne permettent l'utilisation de transactions imbriquées. |
| **5.5** Support des transactions distribuées et du protocole de validation à 2 phases | 2 | Oui. Le conteneur ou serveur EJB intègre un moniteur transactionnel qui implémente l'API JTS ce qui apporte l'interopérabilité avec les moniteurs transactionnels Corba OTS. Les composants EJB peuvent participer à des transactions distribuées (mettant en œuvre plusieurs processus ou plusieurs sources de données). |

| 6. CACHE OBJET TRANSACTIONNEL ET OPTIMISATIONS | | |
|---|---|---|
| **Critère** | **Note** | **Commentaire** |
| **6.1** Présence d'un cache objet transactionnel | 1 | La spécification EJB précise que les conteneurs d'EJB utilisent un pool d'instances pour chaque type de composants EJB, et peuvent mettre en œuvre un cache pour maintenir l'état des instances entre plusieurs appels successifs. La spécification prévoit 3 mécanismes de synchronisation des transactions entre le pool d'instances et le support de persistance sous-jacent : les « commit options » A, B, et C (voir critère 6.2)<br>Cependant, ce système de pooling d'instance et de synchronisation transactionnelle ne peut pas réellement être assimilé à un véritable cache objet transactionnel. En particulier, les instances dans le cache ne peuvent être partagées entre plusieurs clients (une copie de l'instance n'est accédée que par un seul client à la fois, cf. critère 6.2 ci-dessous). |
| **6.2** Gestion de l'accès concurrent à travers des verrous partagés et exclusifs sur les objets | 2 | La spécification EJB définit deux méthodes de gestion des accès concurrents, qui mettent en œuvre un ou plusieurs des trois modes de synchronisation du cache (appelés « commit options ») également définis par la norme:<br>- mode d'accès exclusif ou « pessimiste » : les données de l'objet en base sont verrouillées en mode exclusif durant l'exécution de chaque transaction. Une seule instance de l'EJB est activée et les accès concurrents à cette instance sont sérialisés. L'instance reste activée dans le cache entre plusieurs invocation successives. Ce scénario correspond au mode de synchronisation « commit option A ».<br>- mode d'accès partagé ou « optimiste » : le serveur ne verrouille les données que lors de la synchronisation de l'état de l'objet avec l'état sur le support de persistance. Plusieurs instances de l'EJB peuvent être activées pour traiter les accès concurrents. Cet scénario correspond à l'utilisation de l'un des modes de synchronisation « commit option B » (l'instance reste en cache entre 2 transactions successives) ou « commit option C » (l'état de l'objet n'est pas conservé en cache entre 2 invocations successives). |
| **6.3** Gestion du clustering et d'un cache distribué | 1 | La norme EJB permet la distribution des objets sur des clusters de serveurs. En revanche, elle ne couvre pas la problématique de l'implémentation d'un cache distribué en mode cluster. Seuls quelques rares serveurs d'EJB du marché incluent une fonctionnalité de cache distribué. |
| **6.4** Possibilités de paramétrage du cache objet | 1 | La plupart des serveurs d'EJB n'autorise qu'un paramétrage rudimentaire :<br>- taille du pool d'instances pour chaque type d'EJB déployé (maximum d'instances d'un type d'EJB à autoriser dans le pool).<br>- mode de synchronisation du cache : le conteneur peut supporter un ou plusieurs modes de synchronisation (les « commit options » décrits au critère 6.2). |
| **6.5** Optimisations de lecture et de parcours du graphe objet | 1 | La spécification EJB évoque de façon succincte certaines optimisations possibles (lazy loading des relations d'association par exemple) pouvant être mises en œuvre dans les serveurs d'EJB, mais ne prescrit aucune règle.<br>Les principales implémentations de serveurs EJB du marché intègrent ce mécanisme de lazy loading pour les relations d'association entre composants. |
| **7. ENVIRONNEMENT DE DEVELOPPEMENT (IDE)** | | |
| **Critère** | **Note** | **Commentaire** |
| **7.1** Présence d'un IDE, ou intégration avec un environnement de développement tiers | 2 | La plupart des IDEs Java intègre des fonctionnalités spécifiques au développement des EJB. |
| **7.2** Présence d'un assistant visuel pour la configuration du mapping objet/relationnel | 2 | La plupart des serveurs EJB intègre un outil graphique de mapping O/R. |

| | | |
|---|---|---|
| **7.3** Présence d'un environnement de test | 1 | Le test des composants EJB nécessite en général leur déploiement préalable sur le serveur d'objets.<br>Cependant, la plupart des IDEs Java intègre maintenant un environnement de test pour les EJB. |
| **7.4** Présence d'un profiler | 0 | Aucun des principaux serveurs d'EJB n'est livré avec un outil de profiling. Cette fonctionnalité n'est généralement pas non plus disponible dans les IDEs. |

## 8. DEPLOIEMENT ET INTEGRATION

| Critère | Note | Commentaire |
|---|---|---|
| **8.1** Modèles de déploiement par tiers supportés | 1 | Le modèle EJB n'est utilisable qu'en architecture J2EE 4 tiers. |
| **8.2** Intégration dans les serveurs d'application | 2 | La plupart des serveurs d'application J2EE intègre un serveur d'EJB.<br>Il est également possible d'utiliser en théorie n'importe quel serveur d'EJB avec n'importe quel serveur d'application J2EE. Dans ce cas cependant, pour des raisons de performance, il est préférable que serveur d'application et serveur d'EJB soient exécutés au sein d'une même machine virtuelle (collocation). Cela n'est possible qu'avec certaines combinaisons de serveurs d'application et d'objets. |
| **8.3** Niveau d'intégration dans la norme J2EE | 2 | La norme EJB fait partie intégrale de la spécification J2EE, et fait usage d'autres normes de cette architecture : JNDI, JDBC, JTA, JTS, et RMI-IIOP en particulier. |
| **8.4** Complémentarité avec d'autres standards de persistance | 2 | Avec le modèle EJB BMP, il est possible d'utiliser d'autres standards de persistance pour mettre en œuvre la logique de persistance spécifique des composants.<br>D'autre part, certains conteneurs d'EJB offrent une architecture « pluggable CMP » qui permet de remplacer le moteur de persistance utilisé pour les EJB CMP par un mécanisme tiers. Toutefois, il n'existe à ce jour aucune interface normalisée entre moteur de persistance tiers et conteneur EJB. De ce fait, les architectures « pluggable CMP » existantes sont propriétaires à chaque éditeur de serveur EJB. |

# 5.3 Résultats de l'évaluation détaillée pour le modèle JDO

| Modèle de persistance JDO | | |
|---|---|---|
| **1. TRANSPARENCE DE MODELISATION** | | |
| **Critère** | **Note** | **Commentaire** |
| **1.1** Niveau d'intrusion | 2 | Très faible. JDO n'impose aucune contrainte (héritage de classe ou d'interface, attributs techniques, ou appels d'APIs) sur le modèle objet métier, dans lequel on peut faire libre usage des mécanismes d'héritage et de composition. Les seules obligations pour les classes persistantes sont :<br>- d'implémenter un constructeur par défaut sans arguments<br>- pour les classes pour lesquelles on souhaite mettre en œuvre la gestion de l'identité par l'application (voir critère 1.3), de créer une classe de clé primaire similaire à la classe de clé primaire d'un EJB.<br><br>Il faut cependant noter que le mécanisme qui permet d'assurer la gestion transparente de la persistance en JDO est basé sur le post-traitement du code source ou du code compilé des classes (« code enhancer »). La gêne occasionnée par ce mécanisme reste néanmoins minime.<br><br>Dans certains scénarios, on peut choisir de sacrifier le bénéfice de la persistance transparente à l'implémentation de fonctionnalités particulières. A cet effet, il est possible d'implémenter dans des objets JDO une ou plusieurs des 4 méthodes call-backs de l'API JDO : jdoPostLoad(), jdoPreStore(), jdoPreClear(), et jdoPostDelete(). L'utilisation typique de ces callbacks est l'implémentation de fonctionnalités de validation des données, et de l'implémentation de suppressions en cascade, ou d'autres contraintes. |
| **1.2** Restrictions sur les types de données pouvant être persistants | 2 | JDO n'impose pratiquement aucune restriction quant aux classes pouvant être persistantes. En ce qui concerne les attributs d'objets persistants, on peut utiliser les types de base (primitives et classe wrappers, chaînes de caractères, etc.…), ainsi que les classes conteneurs (Set, ArrayList, Vector, Hashtable, etc.…).<br>La spécification différencie cependant 2 catégories d'objets :<br>- les objets de 1ère catégorie (FCO – first-class objects), dotés d'une identité JDO : on y trouve tous les objets du modèle métier<br>- les objets de 2nde catégorie (SCO – second-class objects) : on y trouve les attributs d'un FCO qui ne peuvent être persistés que comme partie intégrante (attributs) de cet objet, et parfois les objets dépendants d'un FCO.<br>Les règles de classification des FCO et SCO varient entre implémentations. Généralement, on trouve dans les SCO des objets tels que : tableaux, classes wrappers (Integer, Double, etc.), String, et classes conteneurs (Vector, Hashtable, Set, etc.). Dans certains cas également, des objets dépendants FCO peuvent être persistés comme SCO.<br>Seuls certains objets système (comme Thread, ou Socket par exemple) ne peuvent pas être rendus persistants, que ce soit en tant que FCO ou SCO. |

| | | |
|---|---|---|
| **1.3** Gestion de l'identité objet | 2 | JDO offre 2 mécanismes principaux de gestion de l'identité objet :<br>- identité à la charge de l'application (application identity): dans ce cas de figure, pour chaque objet métier persistant, on doit définir une classe de clé primaire associée. Le mécanisme est similaire à (et interopérable avec) celui utilisé pour les EJB entité.<br>- identité gérée par le gestionnaire de persistance (data store identity) : c'est le mécanisme utilisé par défaut. Lorsqu'un objet est rendu persistant, une identité lui est attribuée par l'implémentation JDO sur le support de persistance. Ce mécanisme est typiquement utilisé pour gérer l'identité des objets dépendants.<br>Il existe un mécanisme supplémentaire particulier : l'identité non-durable (non-durable identity). L'identité des objets n'est assurée qu'en mémoire, mais pas sur le support de persistance où des objets « identiques » (valeurs d'attributs égales) peuvent exister (doublons). L'avantage apporté par ce mécanisme est sa performance très élevée lors des opérations d'insertion et de mise à jour « en bloc » sur le support de persistance. L'utilisation typique de ce mécanisme est la manipulation d'objets représentant des enregistrements de fichiers de journalisation. Dans ce scénario, il peut y avoir une grande quantité de « doublons », et la considération primordiale est la performance des opérations d'insertion et de mises à jour en bloc. |
| **1.4** Support des objets persistants à granularité fine | 2 | Les capacités sophistiquées de modélisation offertes par JDO permettent de mettre en œuvre sans problème un modèle objet à granularité fine au sein d'applications. |
| **1.5** Support de l'héritage | 2 | JDO apporte une prise en charge complète de l'héritage. Au sein d'une hiérarchie de classes, il est possible de mêler classes persistantes et classes non-persistantes. |
| **1.6** Support des associations entre objets | 2 | JDO permet l'utilisation des mécanismes standard de Java de composition entre objets.<br>Les objets dépendants peuvent dans certains cas être traités comme FCO, ou bien comme SCO. |
| **2. SUPPORT DES SOURCES DE DONNEES POUR LA PERSISTANCE** | | |
| **Critère** | **Note** | **Commentaire** |
| **2.1** Support des SGBDR pour le mapping | 2 | La spécification JDO est neutre vis-à-vis de la nature des supports de persistance sous-jacents utilisés par les implémentations.<br>La plupart des implémentations JDO permet l'utilisation des principaux SGBD du marché. |
| **2.2** Support de schémas existants en SGBDR | 2 | La plupart des implémentations JDO permet l'utilisation de schémas existants pour les objets persistants utilisant le mode de gestion de l'identité applicative (« application identity »). |
| **2.3** Génération automatique du schéma métier en SGBDR | 2 | La plupart des implémentations JDO permet la génération automatique du schéma relationnel pour la persistance d'objets (que ce soit en mode « application identity » ou en mode « datastore identity »). |
| **2.4** Support des sources de données J2C et d'autres référentiels pour le mapping (bases de données objet, fichiers XML, annuaires LDAP, etc.) | 1 | la spécification JDO prévoit l'utilisation d'autres supports de persistance que les SGBD, et recommande l'utilisation de connecteurs J2C pour réaliser l'interface vers d'autres sources de données comme les bases de données objet ou les applications d'entreprise (ERP, MOM) ou mainframe.<br>En pratique, certaines implémentations JDO permettent l'utilisation d'une ou plusieurs bases de données objet comme support de persistance, ainsi que la persistance en fichiers binaires ou XML dans certains cas.<br>En revanche, il n'existe à ce jour pratiquement aucune implémentation offrant la prise en charge d'autres sources de données à travers J2C. |
| **3. CAPACITES DE MODELISATION ET DE PROJECTION (Mapping)** | | |
| **Critère** | **Note** | **Commentaire** |
| **3.1** Types de données supportés pour les mappings | 2 | JDO permet l'utilisation de tous les types de base Java (primitives, classes standard), des collections (à travers les classes conteneur standard : HashSet, ArrayList, Vector, HashTable), ainsi que toute classe sérialisable. Le support des tableaux, optionnel dans la norme, est cependant souvent pris en charge par les implémentations. |

| | | |
|---|---|---|
| **3.2** Support du mapping multi-table | 0 | La norme JDO ne couvre pas la problématique du mapping, et il existe peu d'implémentations susceptibles de proposer cette fonctionnalité. |
| **3.3** Support du mapping par sérialisation | 2 | La norme JDO sous-entend que le mécanisme de sérialisation peut être utilisé pour la persistance d'objets de 2nde catégorie (ne disposant pas d'identité JDO), ainsi que dans certains cas, de manière optionnelle, pour des objets de 1ère catégorie (objets dépendants notamment).<br>La plupart des implémentations JDO utilise la sérialisation en champ binaire (BLOB) pour le stockage d'objets de 2nde catégorie en SGBD. En ce qui concerne les objets dépendants, la plupart des implémentations permettent également le recours à cette solution. |
| **3.4** Prise en charge de conversions de types de données et de transformations spécifiques | 1 | La spécification JDO ne couvre pas la problématique de conversion de types et de transformations spécifiques.<br>Il existe cependant quelques rares implémentations JDO qui offrent la possibilité de réaliser des conversions et des transformations arbitraires, en implémentant les conversions ou transformations à travers des classes Java. |
| **3.5** Gestion automatique de clés techniques pour la projection de l'identité objet | 2 | La gestion de l'identité des objets persistants peut être déléguée à l'implémentation JDO (mode « data store identity ») qui gère automatiquement l'identité des objets à travers des clés techniques.<br>JDO ne prévoit pas le support des tables de séquences ou des séquences natives des SGBD pour les clés techniques. Cependant, les implémentations sont libres de proposer des extensions pour la gestion des séquences, et certaines prennent en charge l'utilisation de tables de séquence. |
| **3.6** Prise en compte de l'héritage | 2 | La projection des relations d'héritage est prévue par la norme, mais aucune prescription n'est faite quant au mécanisme de projection employé. |
| **3.7** Prise en compte des relations d'association entre objets. Support de relations uni- et bi-directionnelles avec différentes cardinalités: 1-1, 1-N, N-N. | 1 | La spécification JDO prévoit la prise en compte des relations de type 1-1 ,1-N, et N-N. Cependant, la norme ne couvre pas la prise en charge des relations n-aires bi-directionnelles, ni la gestion de l'intégrité référentielle des relations.<br>Cette lacune empêche en particulier d'envisager l'utilisation de JDO en tant que mécanisme de persistance sous-jacent dans des conteneurs EJB pour le mode CMP des EJB entité définissant des relations d'association (CMR).<br>La gestion de la directionnalité et de l'intégrité référentielles pour les relations n-aires peut cependant être implémentée manuellement grâce aux mécanismes de callback disponibles sur les objets JDO (utilisation des callbacks jdoPreStore() et jdoPreDelete()). |
| **3.8** Prise en compte des suppressions en cascade (relations d'association 1-N) | 0 | La prise en charge des suppressions en cascade dans les relations n-aires n'est possible qu'à travers l'implémentation des mécanismes nécessaires dans le callback jdoPostDelete() des objets JDO. Cette solution présentant une grande complexité, et nuisant au caractère transparent de la persistance, est à déconseiller. |
| **3.9** Portabilité du mécanisme de description du mapping | 0 | La spécification JDO définit une grammaire XML standard pour la description des méta-données de persistance sur les classes JDO.<br>Cependant, les éléments définis par cette grammaire concernent uniquement la description des classes et champs persistants ainsi que de la gestion de l'identité des objets. Les éléments permettant de décrire le mapping des objets sur le support de persistance ne sont pas définis.<br>La spécification prévoit que les éléments de description du mapping, propriétaires à chaque implémentation, sont décrits à travers un attribut spécial permettant de gérer des extensions propriétaires (tag <extension>). Cet élément permet aussi de spécifier des informations relatives à certains mécanismes propriétaires mis en œuvre par les différentes implémentations de JDO.<br><br>Par conséquent, bien que toutes les implémentations JDO utilisent un fichier XML standardisé pour décrire le mapping objet/support de persistance, la description du mapping varie d'une implémentation à l'autre. |

| | | |
|---|---|---|
| **3.10** Compatibilité au niveau binaire | 2 | La norme JDO impose aux éditeurs d'assurer la compatibilité binaire des classes JDO entre implémentations. Ainsi, une classe persistante compilée est portable d'une implémentation JDO à l'autre sans recompilation. De la même façon, en cas de changement du support de persistance utilisé, si les méta-données de mapping doivent être régénérées, en revanche aucune recompilation des classes persistantes n'est nécessaire. |
| **4. LANGAGE DE REQUETAGE OBJET** | | |
| **Critère** | **Note** | **Commentaire** |
| **4.1** Présence d'un ou plusieurs langages de requêtage objet. | 2 | La spécification JDO définit un langage de requêtage doté d'une syntaxe très simple et relativement puissante : JDOQL. Ce langage permet de spécifier des conditions de sélection (filtres) exprimées comme expressions conditionnelles Java, et permet la navigation à travers le graphe objet. Enfin, il est également possible de spécifier des clauses de tri (méthode setOrdering() sur l'objet Query).<br>En revanche, JDOQL ne comporte aucune fonction d'agrégat (maximum, count, average, etc.). |
| **4.2** Possibilité d'effectuer des requêtes dynamiques | 2 | Le modèle JDO permet la création de requêtes dynamiques à l'exécution. Les requêtes ne sont pas définies sur les objets persistants (comme c'est le cas pour les EJB), mais par les applications en langage JDOQL. |
| **4.3** Support des requêtes polymorphes | 2 | JDO offre un contrôle très fin sur les requêtes. Les requêtes dynamiques peuvent être au choix polymorphes, ou restreintes aux objets d'une classe donnée sans prise en compte des objets de classes dérivées (second paramètre de la méthode getExtent() sur un objet Extent). |
| **4.4** Optimisations apportées par le moteur de requêtes | 1 | La spécification JDO ne couvre pas l'implémentation de techniques d'optimisation pour les requêtes. A l'heure actuelle, rares sont les implémentations JDO à proposer des optimisations. |
| **5. POTENTIEL TRANSACTIONNEL** | | |
| **Critère** | **Note** | **Commentaire** |
| **5.1** Prise en compte des transactions au niveau objet par le gestionnaire de persistance | 2 | La spécification JDO précise que les transactions sont gérées au niveau objet par le gestionnaire de persistance.<br>Dans le modèle JDO, chaque gestionnaire de persistance ne peut exécuter qu'une seule transaction simultanée (un seul objet Transaction par objet PersistenceManager).<br>Dans le cas où plusieurs transactions indépendantes doivent être exécutées en parallèle au sein d'une application, on doit instancier un gestionnaire de persistance par transaction concurrente.<br>Les applications contrôlent le déroulement des transactions par l'intermédiaire de fonctions d'API similaires à celles de JTA (begin, commit, rollback). |
| **5.2** Flexibilité pour la démarcation des transactions | 2 | Dans le modèle JDO, la démarcation des transactions est à la charge des applications clientes, et s'effectue à travers une instance du gestionnaire de persistance, par des fonctions d'API similaires à celles de l'API JTA standard de J2EE.<br>Il est également possible de synchroniser les transactions des gestionnaires de persistance JDO avec des transactions JTA. |
| **5.3** Support des transactions linéaires | 2 | Oui. |
| **5.4** Support des transactions imbriquées | 0 | Dans sa version actuelle, la spécification JDO ne supporte pas les transactions imbriquées. Leur support , ainsi que celui des points de synchronisation, est à l'étude pour une future version de la norme. |
| **5.5** Support des transactions distribuées et du protocole de validation à 2 phases | 2 | Dans une architecture 2-tiers avec une application JDO autonome (« standalone ») en environnement non-contrôlé (*non-managed environment*), on utilise l'API J2C pour coordonner les transactions entre gestionnaires de persistance JDO.<br>En environnement contrôlé (*managed environment*), avec le moteur JDO embarqué dans un serveur d'application J2EE, les transactions du gestionnaire de persistance JDO sont synchronisées avec le moniteur transactionnel du serveur d'applications, et on utilise typiquement des transactions JTA (compatibles avec le gestionnaire de persistance JDO). |

| 6. CACHE OBJET ET OPTIMISATIONS | | |
|---|---|---|
| **Critère** | **Note** | **Commentaire** |
| **6.1** Présence d'un cache objet transactionnel | 2 | La spécification JDO définit un modèle de cache objet transactionnel dont la gestion doit être automatisée et transparente. Cependant, la norme définit des fonctions d'API permettant de paramétrer le cache objet.<br>Le cache gère les transactions au niveau des instances, ainsi qu'au niveau des champs individuels d'une instance.<br>Le cache permet de gérer à la fois des objets persistants et transactionnels, et des objets persistants mais non-transactionnels De plus, chaque champ d'un objet peut être persistant ou non-persistant, et/ou transactionnel ou non-transactionnel.<br>Le cache JDO effectue également la synchronisation d'accès concurrents de plusieurs threads applicatives au niveau des objets et des champs individuels de chaque objet. |
| **6.2** Gestion de l'accès concurrent à travers des verrous partagés et exclusifs sur les objets | 2 | La norme JDO impose la prise en charge du mode « pessimiste » (verrous exclusifs) pour la gestion des accès concurrents.<br>Optionnellement, la norme recommande également le support du mode optimiste.<br>La plupart des implémentations JDO prend en charge ces 2 modes de gestion d'accès concurrents. |
| **6.3** Gestion du clustering et d'un cache distribué | 0 | La spécification JDO ne couvre pas la problématique de la distribution des objets, ou de la synchronisation de cache entre plusieurs gestionnaires de persistance JDO en architecture n-tiers. |
| **6.4** Possibilités de paramétrage du cache objet | 0 | La norme JDO ne couvre pas la possibilité de paramétrer le cache objet. |
| **6.5** Optimisations de lecture et de parcours du graphe objet | 1 | La norme JDO introduit le concept de « groupes d'attributs à charger » (fetch groups) afin d'optimiser la lecture des objets.<br>Par défaut, le chargement d'un objet est différé (« lazy loading » ou « just-in-time reading ») jusqu'à ce qu'on accède à l'un de ses attributs. A ce moment-là, le gestionnaire de persistance charge un groupe d'attributs par défaut (default fetch group) qui contient les attributs des types Java suivants : primitives, types Java de base (numériques alphanumériques, dates), et références vers d'autres objets. Dans le descripteur de persistance de chaque classe, il est possible de modifier les attributs qui seront inclus dans ce « default fetch group ». |
| **7. ENVIRONNEMENT DE DEVELOPPEMENT (IDE)** | | |
| **Critère** | **Note** | **Commentaire** |
| **7.1** Présence d'un IDE, ou intégration avec un environnement de développement tiers | 2 | Le modèle JDO ne requiert aucun environnement de développement particulier, et la modélisation des objets peut s'effectuer à l'aide d'un IDE classique du marché. |
| **7.2** Présence d'un assistant visuel pour la configuration du mapping objet/relationnel | 1 | Les outils fournis varient selon les implémentations JDO. Cependant, certaines implémentations JDO incluent un outil de mapping graphique. |
| **7.3** Présence d'un environnement de test | 1 | Les outils fournis varient selon les implémentations JDO. Cependant, le modèle JDO autorise très facilement la création d'environnements de test pour les objets JDO. |
| **7.4** Présence d'un profiler | 0 | Les outils fournis varient selon les implémentations JDO. La norme JDO elle-même n'aborde pas le sujet du profiling. |
| **8. DEPLOIEMENT ET INTEGRATION** | | |
| **Critère** | **Note** | **Commentaire** |
| **8.1** Modèles de déploiement par tiers supportés | 2 | Dès l'origine, la norme JDO a été conçue afin que le moteur de persistance puisse être utilisable à la fois en environnement non contrôlé (architecture 2 tiers), et puisse également être intégré au sein d'environnements contrôlés (serveurs d'application) dans une architecture 3 ou 4 tiers. |

| | | |
|---|---|---|
| **8.2** Intégration dans les serveurs d'application | 2 | JDO est conçue pour permettre facilement l'intégration du moteur de persistance au sein d'un serveur d'application J2EE.<br>La norme précise que cette intégration doit s'effectuer à travers les mécanismes de J2C : l'implémentation JDO est déployée en tant qu'adaptateur de ressource, et expose la classe de fabrique de gestionnaires de persistance dans le contexte JNDI. Ainsi, les applications (et composants EJB) peuvent récupérer une instance de gestionnaire de persistance JDO en effectuant un simple lookup JNDI. |
| **8.3** Niveau d'intégration dans la norme J2EE | 2 | Bien que ne faisant pas partie de la norme J2EE, JDO fait usage de nombreuses APIs de cette norme et s'interface naturellement avec les architectures J2EE à travers J2C. |
| **8.4** Complémentarité avec d'autres standards de persistance | 2 | Grâce à l'intégration à travers J2C d'une implémentation JDO dans un serveur d'application J2EE, on peut bénéficier des fonctionnalités de JDO au sein du modèle EJB.<br>JDO peut être utilisée à des fins de persistance objet :<br>- avec des EJB session sans état ou avec état, en mode de démarcation transactionnelle gérée par le conteneur ou par le composant<br>- avec des EJB entité BMP (persistance gérée par le composant) : l'utilisation de JDO permet de simplifier de façon très significative l'implémentation de la persistance d'un EJB entité BMP, et d'offrir des fonctionnalités sophistiquées.<br><br>Il est également envisageable de réaliser un conteneur EJB qui utiliserait JDO comme moteur de persistance pour les EJB entité CMP, ou bien de réaliser un moteur de persistance JDO qui pourrait être utilisé avec des serveurs d'EJB du marché qui possèdent une architecture « pluggable CMP ».<br>Malheureusement, en l'état actuel de la norme et des implémentations, cela semble peu réalisable. En effet, dans sa version actuelle, la norme ne couvre pas la gestion de l'intégrité référentielle des relations d'association 1-n uni ou bi-directionnelles. Or, la prise en charge de cet aspect est nécessaire à la gestion des relations inter-composants entre EJB CMP (container-managed relationships). |

# 5.4 Résultats de l'évaluation détaillée pour Oracle9iAS TopLink

| Modèle de persistance Oracle9iAS TopLink | | |
|---|---|---|
| **1. TRANSPARENCE DE MODELISATION** | | |
| **Critère** | **Note** | **Commentaire** |
| **1.1** Niveau d'intrusion | 2 | Très faible. Oracle9iAS TopLink n'impose aucune contrainte sur le modèle objet métier, dans lequel on peut faire librement usage des mécanismes d'héritage et de composition.<br>La seule exigence de Oracle9iAS TopLink est que les classes persistantes soient dotées d'un constructeur par défaut sans arguments.<br>Contrairement à JDO, l'implémentation de la persistance est entièrement transparente : Oracle9iAS TopLink n'effectue aucune modification sur le code compilé des classes.<br>Comme avec la norme JDO, il est possible de sacrifier au bénéfice de la persistance transparente pour implémenter à travers des méthodes callbacks des traitements en réponse à certains événements du cycle de vie des objets (« Oracle9iAS TopLink events ») : mise à jour, suppression, insertion, etc. Ceci permet par exemple d'implémenter des mécanismes de synchronisation avec d'autres services ou sous-systèmes, ou de gérer le cycle de vie d'attributs non-persistants des objets. |
| **1.2** Restrictions sur les types de données pouvant être persistants | 2 | Oracle9iAS TopLink n'impose pratiquement aucune restriction quant aux classes pouvant être persistantes. Toutes les primitives et classes Java de base peuvent être utilisés en tant qu'attributs persistants, de même que les références objet, les classes conteneur (Set, ArrayList, Vector, etc.), ainsi que les tableaux.<br>Seuls certains objets systèmes (Thread, Socket, etc..) ne sont pas pris en charge. |
| **1.3** Gestion de l'identité objet | 2 | Oracle9iAS TopLink gère l'identité objet à travers un ou plusieurs attributs de clé primaire. Contrairement aux exigences des modèle EJB ou JDO, Oracle9iAS TopLink ne requiert pas la définition d'une classe de clé primaire.<br>La gestion de l'identité objet peut être partiellement déléguée à Oracle9iAS TopLink : les objets doivent obligatoirement comporter un ou plusieurs attributs de clé primaire, mais Oracle9iAS TopLink peut prendre en charge l'attribution des identifiants de clé primaire automatiquement.<br>Oracle9iAS TopLink permet également le mapping d'objets ou de collections d'objets dépendants ne disposant pas d'identité. |
| **1.4** Support des objets persistants à granularité fine | 2 | Les capacités avancées de modélisation offertes par Oracle9iAS TopLink, ainsi que les performances apportées par le cache transactionnel et les diverses optimisations, permettent de mettre en œuvre sans problèmes un modèle objet à granularité fine au sein des applications. |
| **1.5** Support de l'héritage | 2 | Oracle9iAS TopLink autorise l'utilisation des mécanismes d'héritage, et permet de projeter un arbre d'héritage sur le support de persistance. |
| **1.6** Support des associations entre objets | 2 | Oracle9iAS TopLink assure une prise en charge extrêmement complète des relations d'association entre objets. |
| **2. SUPPORT DES SOURCES DE DONNEES POUR LA PERSISTANCE** | | |
| **Critère** | **Note** | **Commentaire** |
| **2.1** Support des SGBDR pour le mapping | 2 | Oracle9iAS TopLink prend en charge tous les SGBD disposant d'un pilote JDBC 2.0. Pour le support des transactions distribuées à travers JTA/JTS, Oracle9iAS TopLink requiert un pilote JDBC implémentant l'interface XaDataSource. |
| **2.2** Support de schémas existants en SGBDR | 2 | Oracle9iAS TopLink n'impose aucune contrainte sur le schéma de persistance et permet d'utiliser n'importe quel schéma relationnel existant. |

| | | |
|---|---|---|
| **2.3** Génération automatique du schéma métier en SGBDR | 2 | Oracle9iAS TopLink est capable de générer un schéma relationnel adapté à la projection du modèle objet. |
| **2.4** Support des sources de données J2C et d'autres référentiels pour le mapping (bases de données objet, fichiers XML, annuaires LDAP, etc.) | 1 | Oracle9iAS TopLink inclut un kit de développement (SDK) qui permet d'utiliser Oracle9iAS TopLink avec d'autres supports de persistance que les SGBD. L'interfaçage entre Oracle9iAS TopLink et le support de persistance tiers nécessite le développement des classes nécessaires à l'aide du SDK.<br>Oracle9iAS TopLink propose d'ailleurs un mécanisme de persistance en fichiers XML développé à l'aide de ce kit.<br>Il faut cependant bien réaliser que le développement d'un « adaptateur » à l'aide du SDK de Oracle9iAS TopLink est une tâche relativement complexe et coûteuse.<br>D'autre part, Oracle9iAS TopLink devrait proposer le support de sources de données J2C dans une prochaine version. |
| **3. CAPACITES DE MODELISATION ET DE PROJECTION (Mapping)** | | |
| **Critère** | **Note** | **Commentaire** |
| **3.1** Types de données supportés pour les mappings | 2 | Oracle9iAS TopLink permet le mapping de tous les types Java de base (primitives, classes usuelles) ainsi que des collections (classes conteneur ArrayList, Vector, Hashtable, etc. .), ainsi que des tableaux.<br>Pour le mapping des tableaux, Oracle9iAS TopLink permet de tirer parti des fonctionnalités spécifiques du SGBD Oracle : champ tableau de taille variable (VArray) et table imbriquée (Nested Table). |
| **3.2** Support du mapping multi-table | 2 | Oracle9iAS TopLink autorise la projection d'un objet sur plusieurs tables. La projection d'un arbre d'héritage peut également faire appel à plusieurs tables. |
| **3.3** Support du mapping par sérialisation | 2 | Oracle9iAS TopLink permet le mapping d'objets dépendants par sérialisation dans des champs binaires (BLOB) en SGBD. |
| **3.4** Prise en charge de conversions de types de données et de transformations spécifiques | 2 | Oracle9iAS TopLink offre une riche panoplie d'options pour la conversion de types et la transformation de valeurs entre le modèle objet et le modèle relationnel. Il est par exemple possible de stocker des nombres comme chaînes de caractères, ou de spécifier des mappings d'énumérations.<br>Par programmation, il est également possible de spécifier des transformations complexes entre valeurs d'attributs Java et valeurs stockées en base, celles-ci étant prises en compte par Oracle9iAS TopLink à l'exécution. |
| **3.5** Gestion automatique de clés techniques pour la projection de l'identité objet | 2 | Oracle9iAS TopLink propose la gestion automatique de clés techniques pour les objets à travers des tables de séquence, ou les séquences natives des SGBD (a l'exception de DB2). |
| **3.6** Prise en compte de l'héritage | 2 | Oracle9iAS TopLink propose 2 approches différentes pour la projection de relations d'héritage :<br>- une table par hiérarchie de classes. Cette approche permet d'optimiser les accès au détriment d'une augmentation de la taille des données en base.<br>- une table par classe de la hiérarchie (sans duplication d'attributs hérités) : cette approche permet d'optimiser l'espace utilisé en base au détriment de la performance d'accès (jointures entre plusieurs tables)<br>Il est également possible de ne mapper qu'une classe spécifique au sein d'une hiérarchie. Dans ce cas, Oracle9iAS TopLink permet de mapper les attributs hérités de la (ou des) superclasse(s). En fait Oracle9iAS TopLink propose un héritage des descripteurs de mapping qui peut être dé-corrélé de la hiérarchie de classe Java.<br>Enfin, il est également possible de définir des mappings d'interface. Ces mappings n'établissent aucune correspondance entre attributs d'objet et champs en SGBD, mais permettent d'utiliser des interfaces en lieu et place de classes concrètes pour la création et l'exécution de requêtes, ce qui apporte une plus grande flexibilité. |

| | | |
|---|---|---|
| **3.7** Prise en compte des relations d'association entre objets. Support de relations uni- et bi-directionnelles avec différentes cardinalités: 1-1, 1-N, N-N. | 2 | Oracle9iAS TopLink permet la projection de tout type de relations d'association entre objets, et prend en charge les relations unidirectionnelles et bi-directionnelles, de cardinalités 1-1, 1-N, ou N-N. Les relations peuvent être de deux types :<br>- relations privées: dans ce cas, il y a une relation d'appartenance entre objet(s) source(s) et objet(s) cible(s). Dans ce mode, Oracle9iAS TopLink effectue automatiquement une suppression en cascade si l'objet source est détruit.<br>- relations indépendantes (par défaut) : dans ce type de relations, les objets existent indépendamment, et la suppression d'un objet « source » n'entraîne pas forcément la suppression d'un objet « cible »..<br>Dans tous les cas, Oracle9iAS TopLink assure l'intégrité référentielle des relations.<br>De plus, Oracle9iAS TopLink prend en charge un type particulier de relation : les relations de type agrégat objet (« aggregate object mapping») utilisés pour modéliser :<br>- soit des relations à cardinalité strictement 1-1. Dans ce cas, les objets source et cible sont mappés dans un même enregistrement au sein d'une même table.<br>- soit des relations de type « 1-n » avec la même sémantique, excepté que ce type de mapping utilise 2 tables au lieu d'une.<br>Enfin, Oracle9iAS TopLink permet le mapping d'objets dépendants par sérialisation dans la même table que l'objet propriétaire, ainsi que le mapping de collections d'objets dépendants dans une table séparée. |
| **3.8** Prise en compte des suppressions en cascade (relations d'association 1-N) | 2 | Oracle9iAS TopLink peut assurer la gestion de l'intégrité référentielle et des suppressions en cascade dans des relations 1-1 1-n, ou n-n. Oracle9iAS TopLink qualifie ce type de relations de relations privées (« private relationships »). |
| **3.9** Portabilité du mécanisme de description du mapping | 0 | Oracle9iAS TopLink utilise un mécanisme propriétaire pour le mapping objet/relationnel. Les informations de mapping sont stockées au sein de fichiers XML.<br>Précisons que dans le monde Java, il n'existe malheureusement aucun standard régissant la définition du mapping objet-relationnel, que ce soit avec les normes EJB ou JDO, ou avec des produits comme Oracle9iAS TopLink. |
| **3.10** Compatibilité au niveau binaire | 2 | Contrairement aux approches employées par les modèles EJB et JDO, Oracle9iAS TopLink n'effectue aucune modification sur le code source ou le code compilé des classes persistantes, et ne génère aucune classe d'interface de bas-niveau.<br>Une application développée avec Oracle9iAS TopLink est compatible binaire d'un environnement à un autre. Le changement du support de persistance utilisé ne nécessite aucune recompilation des classes persistantes. |

| 4. LANGAGE DE REQUETAGE OBJET | | |
|---|---|---|
| **Critère** | **Note** | **Commentaire** |
| **4.1** Présence d'un ou plusieurs langages de requêtage objet. | 2 | Oracle9iAS TopLink offre une très grande flexibilité au niveau du requêtage objet : pas moins de 3 langages différents sont utilisables pour la création de requêtes dynamiques paramétrables:<br><br>- un langage de requêtage propriétaire entièrement orienté objet (Expression Builder) basé sur la combinaison d'appels de méthodes sur des objets Java pour l'expression de filtres de sélection, de navigation à travers le graphe objet, et d'expression de clauses de tri.<br>Ce langage de requêtage propose également un mode de création de requêtes par l'exemple (QBE – Query By Example), et permet la création de requêtes de type « édition d'états » (report queries) avec utilisation de fonctions d'agrégat (max, average, count, …) et de regroupement (group by).<br><br>- EJB-QL : Oracle9iAS TopLink permet la création de requêtes dynamiques exprimées dans le langage EJB-QL de la norme EJB.<br><br>- SQL : Oracle9iAS TopLink permet la création de requêtes en langage SQL. Il est également possible d'invoquer des procédures stockées du SGBD.<br><br>Par ailleurs, le langage de requêtage propriétaire de Oracle9iAS TopLink permet non seulement d'effectuer des requêtes de lecture d'objets, mais également de mise à jour et de suppression d'objets. |
| **4.2** Possibilité d'effectuer des requêtes dynamiques | 2 | Oracle9iAS TopLink offre la possibilité de définir des requêtes dynamiques à l'exécution. Les requêtes ne sont pas définies sur les objets persistants (comme c'est le cas pour les EJB), mais par les applications. Tous les mécanismes de requêtage proposés par Oracle9iAS TopLink (Expression Java, EJB-QL, SQL) peuvent être employés pour effectuer des requêtes dynamiques. |
| **4.3** Support des requêtes polymorphes | 2 | De la même manière que JDO, Oracle9iAS TopLink supporte les requêtes polymorphes, et permet au choix de restreindre la recherche aux objets d'une classe donnée dans l'arbre d'héritage ou d'inclure les objets des classes dérivées dans les résultats. |
| **4.4** Optimisations apportées par le moteur de requêtes | 2 | Oracle9iAS TopLink implémente un très grand nombre de techniques d'optimisation. Parmi celles-ci, on peut citer :<br>- La possibilité d'exécuter les requêtes en mémoire sur le cache objet uniquement (pas d'appels en SGBD).<br>- l'utilisation de curseurs pour la navigation à travers les résultats d'une requête |
| **5. POTENTIEL TRANSACTIONNEL** | | |
| **Critère** | **Note** | **Commentaire** |
| **5.1** Prise en compte des transactions au niveau objet par le gestionnaire de persistance | 2 | Oracle9iAS TopLink définit un concept de transaction au niveau objet : les « Units of Work ». Chaque « Unit of Work » contrôle de manière isolée les changements apportés au niveau d'un graphe objet, et ne met à jour lors de la validation de la transaction que les attributs de objets dont l'état a été modifié. |
| **5.2** Flexibilité pour la démarcation des transactions | 2 | Avec Oracle9iAS TopLink, la démarcation des transactions est à la charge des applications clientes. |
| **5.3** Support des transactions linéaires | 2 | Oracle9iAS TopLink prend en charge le modèle de transactions linéaires à travers 2 mécanismes :<br>- l'utilisation des transactions standard de JDBC (locales) et de JTA (locales ou distribuées)<br>- l'utilisation de transaction optimisées spécifiques à Oracle9iAS TopLink et appelées « Units of work » qui proposent des fonctionnalités à valeur ajoutée sur les transactions standard (voir critère 5.4) |

| | | |
|---|---|---|
| **5.4** Support des transactions imbriquées | 2 | Oracle9iAS TopLink autorise l'imbrication logique de transactions mais les regroupe au sein d'une seule et même transaction : en effet, JDBC ne permet pas l'utilisation de transactions imbriquées.<br><br>Cependant, Oracle9iAS TopLink offre un autre mécanisme transactionnel, propriétaire, qui lui prend réellement en charge le modèle des transactions imbriquées : les « Units of work ».<br><br>Comparés aux transactions classiques JDBC, les « Units of work » apportent les avantages suivants :<br>- support du parallélisme (plusieurs transactions concurrentes) et de l'imbrication (une transaction peut contenir des sous-transactions indépendantes).<br>- optimisation des mises à jour : les mises à jour ont une granularité au niveau des champs individuels des enregistrements : seuls les champs effectivement modifiés sont mis à jour en base.<br>- maintien de l'intégrité référentielle, avec ré-ordonnancement optimisé des opérations d'insertion, de mise à jour et de suppression, et résolution des relations bi-directionnelles.<br>- ré-ordonnancement des opérations d'accès aux tables lors d'opérations de mises à jour multiples, de façon à éviter les deadlocks potentiels. |
| **5.5** Support des transactions distribuées et du protocole de validation à 2 phases | 2 | Les transactions Oracle9iAS TopLink peuvent impliquer plusieurs objets répartis sur plusieurs bases de données. Oracle9iAS TopLink inclut un mécanisme de validation à 2 étapes propriétaire pour gérer les mises à jour transactionnelles sur plusieurs SGBD.<br>Par ailleurs Oracle9iAS TopLink peut s'interfacer avec un moniteur transactionnel externe à travers JTS (par exemple, avec le gestionnaire transactionnel d'un serveur d'application J2EE). Ce mécanisme permet de synchroniser les transactions de Oracle9iAS TopLink avec le moniteur transactionnel tiers et de faire participer l'application Oracle9iAS TopLink à des transactions distribuées XA. |
| **6. CACHE OBJET ET OPTIMISATIONS** | | |
| **Critère** | **Note** | **Commentaire** |
| **6.1** Présence d'un cache objet transactionnel | 2 | Oracle9iAS TopLink inclut un cache transactionnel partagé en lecture et écriture entièrement paramétrable qui met en œuvre des mécanismes sophistiqués (dont la synchronisation entre plusieurs applications distribuées).<br><br>et est d'ailleurs l'instigateur d'une spécification ouverte de cache objet développée sous l'égide du JCP : la spécification JCACHE (JSR 107) dont Oracle9iAS TopLink pourra tirer bénéfice. |
| **6.2** Gestion de l'accès concurrent à travers des verrous partagés et exclusifs sur les objets | 2 | Oracle9iAS TopLink permet l'utilisation de différentes méthodes de gestion des accès concurrents, au niveau de chaque classe persistante ou au niveau de chaque requête. Les principales options sont :<br>- l'utilisation de verrous exclusifs en mode pessimiste<br>- l'utilisation de verrous partagés en mode optimiste<br>Pour le mode optimiste, Oracle9iAS TopLink offre plusieurs stratégies de vérification paramétrables:<br>- vérification par « versioning » : Oracle9iAS TopLink peut utiliser un champ contenant un numéro de version numérique (incrémenté à chaque mise à jour).<br>- vérification par estampillage temporel (*time-stamping*) : Oracle9iAS TopLink utilise un champ contenant la date et l'heure de dernière modification de l'enregistrement.<br>- vérification au niveau des champs : Oracle9iAS TopLink permet la vérification des modifications par comparaison des valeurs d'un ou plusieurs champs de l'enregistrement avec les données en cache : on peut choisir d'effectuer la comparaison sur un ou plusieurs champs particuliers, sur l'ensemble des champs, ou bien sur le sous-ensemble des champs modifiés par la mise à jour. |

| | | |
|---|---|---|
| **6.3** Gestion du clustering et d'un cache distribué | 2 | En environnement distribué, Oracle9iAS TopLink offre un mécanisme de synchronisation de cache entre serveurs.<br>Le service de synchronisation du cache distribué peut utiliser au choix un mécanisme de notification multicast synchrone ou asynchrone entre sessions Oracle9iAS TopLink s'appuyant sur RMI, JMS, ou CORBA-IIOP. |
| **6.4** Possibilités de paramétrage du cache objet | 2 | Les possibilités de paramétrage du cache objet sont très étendues et permettent de déterminer finement le meilleur compromis entre performances et consommation de mémoire suivant les applications.<br>Différentes règles de gestion du cache peuvent être fixées pour chaque classe persistante (ainsi que pour les EJB entité si Oracle9iAS TopLink est utilisé pour la persistance CMP). |
| **6.5** Optimisations de lecture et de parcours du graphe objet | 2 | Un des principes directeurs de Oracle9iAS TopLink est de minimiser et d'optimiser les interactions avec la base. Pour ce faire, un certain nombre d'optimisations sont mises en œuvre, parmi lesquelles on peut citer :<br>- l'optimisation des appels au SGBD pour la lecture d'un groupe d'objets et d'objets dépendants associés (« batch reading », ou problématique d'optimisation des « N+1 appels au SGBD » en « 1+1 appels »)<br>- l'indirection, ou lecture « juste-à-temps » : lors de la lecture d'un objet, seuls ses attributs directs sont chargés en mémoire. Les objets affiliés par des relations d'association ne sont chargés qu'au moment où ils sont explicitement référencés dans le code. |
| **7. ENVIRONNEMENT DE DEVELOPPEMENT (IDE)** | | |
| **Critère** | **Note** | **Commentaire** |
| **7.1** Présence d'un IDE, ou intégration avec un environnement de développement tiers | 2 | Oracle9iAS TopLink n'imposant aucune contrainte spécifique sur la modélisation des objets, on peut utiliser un environnement de développement classique quelconque.<br>Par ailleurs, l'outil de mapping intégré permet également si on le souhaite de créer le modèle objet de l'application (création de classes avec un assistant visuel). |
| **7.2** Présence d'un assistant visuel pour la configuration du mapping objet/relationnel | 2 | L'atelier de mapping objet-relationnel graphique de Oracle9iAS TopLink (Mapping Workbench) est très complet. Les opérations principales prises en charge par cet outil sont :<br>- la création automatique d'un schéma relationnel pour la projection du modèle objet<br>- la projection du modèle objet sur un modèle relationnel existant<br>- la création automatique d'un modèle objet à partir d'un schéma relationnel existant<br>Les descripteurs de mapping au format XML sont générés de façon automatique par l'outil.<br>Toutes les opérations sont réalisées à travers une interface graphique sophistiquée et productive. |
| **7.3** Présence d'un environnement de test | 1 | La version actuelle de Oracle9iAS TopLink n'intègre pas de fonctionnalités de test unitaire pour les objets persistants Cependant, cette fonctionnalité sera disponible dans la prochaine version du produit, et est déjà implémentée dans la pré-version. |
| **7.4** Présence d'un profiler | 2 | Oracle9iAS TopLink intègre un service de journalisation de statistiques de performances de haut niveau. Ce profiler peut logger soit les ordres SQL bruts, soit un résumé d'informations de performance sur chaque requête exécutée. Il est également possible de générer un résumé d'informations de performances sur une session Oracle9iAS TopLink complète.<br>Parmi les informations qui peuvent être enregistrées, on peut citer :<br>- le type de la requête et ses arguments<br>- le temps d'exécution de la requête<br>- le nombre d'objets affectés, le nombre d'appels réussis en cache (« cache hits »), et le nombre d'objets manipulés par seconde<br>- les temps passés en cache, dans la préparation et dans l'exécution de la requête SQL, et dans la récupération des résultats |

| 8. DEPLOIEMENT ET INTEGRATION | | |
|---|---|---|
| **Critère** | **Note** | **Commentaire** |
| **8.1** Modèles de déploiement par tiers supportés | 2 | Oracle9iAS TopLink offre une grande versatilité et peut être utilisé aussi bien en architecture 2 tiers (applications autonomes) qu'en architecture 3 ou 4 tiers (applications embarquées dans un serveur d'application J2EE).<br>Pour accéder aux données stockées en SGBD pour la persistance du modèle objet, les applications Oracle9iAS TopLink doivent instancier des connexions à travers des objets Session.<br>Oracle9iAS TopLink offre trois types d'objets session qui s'inscrivent dans des scénarios d'utilisation différent :<br>- classiquement, en architecture 2-tiers, chaque application cliente utilise une session Oracle9iAS TopLink privée par SGBD. Ce modèle correspond à l'utilisation de « Database sessions » dans Oracle9iAS TopLink.<br>- en architecture 3-tiers, les applications Oracle9iAS TopLink embarquées dans un serveur d'application peuvent partager des sessions Oracle9iAS TopLink selon un modèle client/serveur. On instancie alors une session serveur par SGBD (« Server session ») et les applications récupèrent une session client (« Client session ») à partir de la session serveur. Les sessions serveur peuvent être gérées par un gestionnaire de sessions (Session Manager) qui utilise un fichier XML de configuration où les propriétés des sessions serveur sont définies.<br>- Enfin, Oracle9iAS TopLink propose également un mécanisme d'accès distant similaire à celui des EJB : les sessions distantes (« Remote Sessions ») qui permettent à des applications clientes de communiquer avec des sessions Oracle9iAS TopLink serveur instanciées sur des serveurs distants. A travers ce mécanisme, les applications clientes peuvent accéder aux objets persistants sur le serveur distant. Cependant, l'utilisation de ce modèle est fortement déconseillée, car il souffre des mêmes problèmes de performance et de scalabilité que le mode d'accès distant aux composants EJB. |
| **8.2** Intégration dans les serveurs d'application | 2 | L'intégration de Oracle9iAS TopLink aux serveurs d'application J2EE s'effectue très simplement en intégrant les bibliothèques de Oracle9iAS TopLink à celles du serveur d'application. Pour ce qui est du déploiement des applications utilisant Oracle9iAS TopLink, les descripteurs de mapping et les éléments de configuration s'intègrent parfaitement dans des les structures de déploiement J2EE (archives .ear,.war ou .jar).<br>Au niveau de l'utilisation de Oracle9iAS TopLink au sein d'applications déployées sur un serveur J2EE, on aura avantage à utiliser le modèle d'utilisation « trois-tiers » client/serveur de Oracle9iAS TopLink. Un fichier de configuration (sessions.xml) sera utilisé pour configurer le gestionnaire de sessions de Oracle9iAS TopLink (Session Manager) . Les applications pourront ensuite instancier des sessions serveur à partir du gestionnaire de sessions. et client. L'utilisation des sessions client/serveur de Oracle9iAS TopLink apporte les avantages suivants :<br>- pooling de connexions<br>- cache partagé entre sessions clientes d'une même session serveur. |
| **8.3** Niveau d'intégration dans la norme J2EE | 2 | Bien qu'étant conçu selon une architecture propriétaire, Oracle9iAS TopLink respecte les standards Java et s'intègre naturellement au sein d'une architecture J2EE. |

| | | |
|---|---|---|
| **8.4** Complémentarité avec d'autres standards de persistance | 2 | Les fonctionnalités de Oracle9iAS TopLink peuvent être mises à profit avec avantage dans une architecture utilisant la norme EJB. Oracle9iAS TopLink apporte des fonctionnalités complémentaires pour la persistance objet, utilisables avec tous les types de composants :<br>- avec des EJB session sans état ou avec état, dans lequel l'accès au SGBD est encapsulé à travers Oracle9iAS TopLink.<br>- avec des EJB entité BMP (persistance gérée par le composant) : la logique de persistance est alors encapsulée à travers Oracle9iAS TopLink.<br>- avec des EJB entité CMP : Oracle9iAS TopLink peut être utilisé comme moteur de persistance CMP pour les composants EJB 2.0 ou 1.1 avec les conteneurs d'EJB de WebSphere et de WebLogic. Oracle9iAS TopLink apporte également aux EJB CMP de WebLogic et de WebSphere une fonctionnalité à haute valeur ajoutée : la prise en charge de requêtes dynamiques. Les requêtes dynamiques sur les EJB peuvent être créées par les applications en langage EJB-QL, mais aussi en langage SQL, ou à l'aide du framework de requêtage propriétaire de Oracle9iAS TopLink (TOP Expressions).<br><br>Enfin, Oracle9iAS TopLink est partiellement compatible avec la norme JDO : il implémente les interfaces du gestionnaire de persistance des transactions, et étend l'interface de requêtes pour implémenter son framework de requêtage. En revanche, Oracle9iAS TopLink n'inclut pas le support de JDOQL. |

Une étude réalisée par

# Groupe SQLI

**(Département R&D)**

préparée pour

**Oracle**

**Auteur**
Laurent Denanot

**Contributeurs**
Nicolas Farges
Habib Guergachi

**Publication:**
RC4 — avril 2003

| USA | SIEGE EUROPEEN | SUISSE |
|---|---|---|
| TechMetrix Research<br>76 Bedford Street, suite 33<br>Lexington<br>MA 02420 | Groupe SQLI<br>268, av. du Président Wilson<br>93200<br>LA PLAINE SAINT-DENIS<br>FRANCE | SQLI<br>Chemin de la Rueyre<br>116-118<br>CH-1020 RENENS |
| Tel.: +1 781-890-3900<br>Fax: +1 781-240.0502 | Tel.: + 33 1 55 93 26 00<br>Fax: + 33 1 55 93 26 01 | Tel.: + 41 (0) 21 637 72 30<br>Fax : + 41 (0) 21 637 72 31 |

http://www.sqli.com
info@sqli.com