Réutilisation du code de programmation dans un environnement informationnel : une approche basée sur les modèles orientés objet et service

par

Nicolas de Bray

Essai présenté au CeFTI
En vue de l'obtention du grade de maître en technologies de l'information
(maîtrise en génie logiciel incluant un cheminement de type cours en technologie de l'information)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, février 2013

Sommaire

La problématique abordée dans cet essai est liée aux coûts de développement logiciel élevés qu'il n'est pas rare de constater dans le domaine de l'intelligence d'affaires. En effet, les données, provenant de sources et systèmes hétérogènes, s'y doivent d'être normalisées afin d'être rendues exploitables et cela engendre de tels coûts que l'intelligence d'affaires est souvent remise en question au profit de l'exploitation.

Compte tenu de cette problématique, l'objectif de cet essai est d'explorer une avenue possible consistant à maximiser la réutilisation du code de programmation au niveau de l'entrepôt de données afin de réduire les coûts de développements logiciels.

Pour ce faire, l'hypothèse de travail qui a été posée est que les fonctionnalités mises à disposition par le système de gestion de bases de données peuvent être évaluées et classifiées à l'aide des modèles de la programmation orientée objet et de l'architecture orientée service. L'intérêt de cette démarche est que ces modèles sont réputés pour favoriser la réutilisation, que ce soit celle du code de programmation ou celle du service. Une telle évaluation permettrait de cerner les fonctionnalités qui sont les plus porteuses de réutilisation, voire les plus économiques à utiliser.

Il est intéressant de mentionner au passage que la méthodologie élaborée dans le cadre de cet essai peut être reprise pour l'analyse de n'importe quel système de gestion de bases de données, voire de n'importe quel système tout court puisqu'il suffit à la limite de trouver un référentiel auquel comparer les fonctionnalités.

L’analyse des résultats révèle des écarts importants dans l’évaluation des fonctionnalités, ce qui permet de valider l’hypothèse de départ et du même coup de faire le classement des fonctionnalités disponibles en regard de leur potentiel de réutilisabilité.

De plus, une recommandation s’impose étant donné que la fonctionnalité au meilleur potentiel de réutilisation est également la plus récente. En effet, lorsque vient le moment de faire le choix des fonctionnalités à utiliser, il ne saurait être question de négliger l’importance de l’étude des possibilités offertes en programmation, dut-elle paraître un brin théorique. Il serait imprudent de ne se fier qu’à l’expérience des programmeurs et intervenants car l’expérience est fondée sur le passé alors que les technologies de l’information évoluent si rapidement.

Enfin, pour conclure, cette analyse, bien que fondée sur l’utilisation de modèles théoriques, recèle un aspect pratique très important. En effet, le classement des fonctionnalités qui résulte de la vérification de l’hypothèse est en lien direct avec la problématique de départ et se veut un outil de décision pratique pour le choix des fonctionnalités à utiliser et le contrôle des coûts de développement logiciel.

Remerciements

Je tiens à remercier ici M. Pierre-Martin Tardif, directeur pédagogique, qui a su, par son expérience et ses connaissances, m'apporter de valeureux conseils et ce, malgré mes questions qui n'étaient pas toujours aussi valables.

Je remercie également Mme Martine Dubé, directrice professionnelle, qui m'apporta son soutien avec enthousiasme en plus d'initier la réflexion qui mena au choix du sujet du présent ouvrage.

Enfin, je tiens à remercier M. Claude Cardinal, directeur adjoint du Centre de formation en technologies de l'information (CeFTI) de l'Université de Sherbrooke, qui a su m'encourager à persister dans la voie de la maîtrise à un moment où les forces commençaient à me faire défaut.

Table des matières

Liste des tableaux

Glossaire

Application	Logiciel qui permet de réaliser une ou plusieurs tâches ou fonctions
Cohérence	Absence de contradiction entre divers éléments
Déclencheur	Dispositif logiciel déclenché en fonction d'événements
Entrepôt de données	Base de données utilisée pour stocker des informations provenant de bases de données opérationnelles et fournir une aide à la décision en entreprise
Évolutivité	Capacité d'évolution
Format de données	Représentation de données sous forme de nombres binaires
Idempotence	Caractéristique d’une action qui peut être effectuée une ou plusieurs fois avec le même résultat
Interface	Dispositif permettant des échanges et interactions entre différents acteurs
Langage de requête	Langage informatique utilisé pour effectuer des requêtes sur des bases de données ou autres systèmes d'information
Langage procédural	Paradigme de programmation basé sur le concept d'appel procédural
Macro	Ensemble d'instructions qui sont représentées dans un format abrégé
Maintenance	Action de modifier un logiciel, après sa mise en œuvre, pour le corriger, l’améliorer ou l’adapter
Méta-donnée	Donnée servant à définir ou décrire une autre donnée
Méthode	Fonction faisant partie de l'interface d'un objet
Normaliser	Action de standardiser les données pour les rendre compatibles

Objet	Conteneur symbolique créé à partir d'un modèle appelé classe ou prototype, duquel il hérite les comportements et les caractéristiques
Procédure	Fonction ne renvoyant pas de valeur
Programmation	Ensemble des activités qui permettent l'écriture des programmes informatiques
Ressource	Composants d'un système virtuel incluant les fichiers, les connexions au réseau, et les zones de mémoire
Réutilisation	Action de reprendre un code logiciel existant pour le réemployer, éventuellement en l'adaptant, dans un contexte présentant certaines similarités avec le programme d'origine
Scalabilité	Capacité d'un dispositif informatique à s'adapter à un changement d'ordre de grandeur de la demande
Service	Fonctionnalité mise à disposition par un composant logiciel pour assurer une tâche particulière
Structure de données	Structure logique destinée à contenir des données, afin de leur donner une organisation permettant de simplifier leur traitement
Système informationnel	Ensemble organisé de ressources (matériels, logiciels, personnels, données et procédures) qui permet de collecter, regrouper, classifier, traiter et diffuser de l'information sur un environnement donné
Type	Un type définit les valeurs que peut prendre une donnée, ainsi que les opérateurs qui peuvent lui être appliqués
Vue	Une vue, dans une base de données, est une synthèse d'une requête d'interrogation de la base qui peut être considérée comme une table virtuelle

Liste des sigles, des symboles et des acronymes

ANSI	*American National Standards Institute* : institut national américain des normes
API	*Application Programming Interface* : interface de programmation
BTEQ	*Basic TEradata Query* : application client
CRUD	*Create, read, update and delete* : créer, lire, mettre à jour et supprimer
C et C++	Langages de programmation
Co	Compatibilité
D	Déclencheur
DBA	*Database Administrator* : administrateur de base de données
E	Étendue
EI	Écart interquartile
FDU	Fonction définie par l'utilisateur
Java	Langage de programmation orienté objet
M	Macro
MDU	Méthode définie par l'utilisateur
OOP	*Object-Oriented Programming* : programmation orientée objet
PS	Procédure stockée
PSX	Procédure stockée externe
SGBD	Système de gestion de base de données
SmallTalk	Langage de programmation orienté objet
SOA	*Service-Oriented Architecture* : architecture orientée services
SQL	*Structured Query Language* : langage de requête structurée
TDU	Type défini par l'utilisateur
TMDU	Type et méthode définis par l'utilisateur
V	Vue
VM	Vue matérialisée

Introduction

L'intelligence d'affaires est un domaine en pleine expansion et est devenu le « nerf de la guerre » dans bien des domaines où l'utilisation de l'information joue un rôle primordial. Cette intelligence d'affaires se fonde sur l'exploitation de l'information, plus précisément sur l'exploitation de données tirées d'un entrepôt de données.

Cependant, les coûts de développement et d'entretien d'un entrepôt de données peuvent être très élevés car les données, provenant la plupart du temps de systèmes hétérogènes, doivent y être normalisées afin de les rendre exploitables. Ces coûts sont alors tels qu'ils nuisent à l'évolution de l'entrepôt de données, voire même à la justification de son existence car, s'il est vrai que l'intelligence d'affaires peut jouer un rôle déterminant pour une entreprise, il n'en demeure pas moins qu'elle n'est pas essentielle aux opérations. Or, lorsque survient une période financière difficile, il n'est pas rare que les entreprises lâchent du lest et concentrent leur énergie sur le domaine opérationnel.

C'est dans un tel contexte qu'il apparaît important pour une entreprise de réduire les coûts d'exploitation de son entrepôt de données sans nuire au maintien ni au développement de l'intelligence d'affaires.

Dans le domaine d'un entrepôt de données comme dans beaucoup d'autres domaines, il y a plusieurs aspects qui peuvent être optimisés mais, dans le cas présent, c'est celui des coûts de développement et plus précisément ceux de la programmation qui sera examiné. Plus exactement, il sera question de chercher un moyen d'optimiser la réutilisation, que ce soit par la réutilisation du code de programmation ou par la réutilisation des services.

Heureusement le concept de réutilisation n'est pas nouveau et a déjà été exploré. En effet, les termes d'objet et de service sont autant de concepts voire de modèles qui permettent de servir, entres autres, celui de la réutilisation.

En fait, les modèles de programmation orientée objet (OOP) et d'architecture orientée service (SOA) servent très bien la réutilisation et ce n'est sans doute pas un hasard si les systèmes informationnels évoluent en ce sens aujourd'hui, notamment pour ces mêmes raisons d'optimisation et de réutilisation. [20][1]

Dans cet essai, l'étude de la question de la réutilisation sera donc abordée par le biais des modèles OOP et SOA. Puisque ces modèles ont prouvé leurs valeurs quant à la dite réutilisation, il est permis de se demander si une fonctionnalité qui respecterait bon nombre des caractéristiques de ces modèles ne serait pas en mesure d'offrir de meilleures possibilités de réutilisation.

Plus précisément, il s'agit de chercher à déterminer s'il est possible d'utiliser les modèles OOP et SOA afin d'évaluer les fonctionnalités du système de gestion de bases de données (SGBD) d'un entrepôt de données, de les comparer entre elles et de les classer en ordre de préférence. Ainsi, en sachant quelles fonctionnalités respectent au mieux les modèles OOP et SOA, il sera peut-être possible de mieux choisir quelles fonctionnalités employer et parvenir ainsi à une meilleure réutilisation. Les fonctionnalités évaluées sont celles permettant les création, sélection, mise à jour et suppression de données (CRUD).

La méthodologie utilisée pour répondre à ces questions devra définir quelles fonctionnalités étudier, quels principes retenir des modèles OOP et SOA. Puis il faudra, à partir des définitions de ces principes, formuler des questions qui, confrontées à chacune des fonctionnalités, serviront à les évaluer et à les noter à l'aide d'une échelle qu'il aura aussi fallut constituer au préalable. S'il s'avère ainsi possible d'évaluer les fonctionnalités, il sera alors aussi possible de les classer par ordre de préférence et le but sera atteint.

Ces définitions des fonctionnalités et principes seront faites à l'aide d'une documentation crédible et pertinente composée principalement d'ouvrages universitaires, de littérature spécialisée et de publications officielles du fournisseur du SGBD Teradata.

Enfin, pour clore cette introduction et puisque cet essai porte sur la réutilisation, il semble intéressant d'en relever les bénéfices afin de venir étoffer du même coup les valeurs et pertinence de l'essai lui-même.

En effet, si la réutilisation du code de programmation est un bénéfice important amené par le biais de l'héritage et du polymorphisme pour le modèle OOP et par le biais de la réutilisation du service pour le modèle SOA, [11] la réutilisation elle-même induit de nombreux bénéfices en terme d'effort et de contrôle des coûts [20].

Plus précisément, la réutilisation procure des avantages tels un développement plus rapide, une augmentation de la qualité du code, des changements plus faciles à opérer, une meilleure productivité, une réduction des coûts de maintenance, des délais de livraison plus courts, une complexité réduite et une cohérence accrue des applications et architectures logicielles, la réduction des défauts et des risques [28] et, en bout de ligne, une plus grande agilité des systèmes d'information. [4]

Chapitre 1

Hypothèse et méthodologie de travail

Voici présentées l'hypothèse de travail ainsi que la méthodologie qui sera employée pour vérifier cette première.

1.1 Hypothèse

L'hypothèse de cet essai est que les principes des modèles OOP et SOA peuvent servir à l'évaluation des fonctionnalités CRUD qui sont mises à disposition par le SGBD Teradata, laquelle évaluation permettra ensuite de classifier ces fonctionnalités.

1.2 Méthodologie

La méthodologie proposée pour la vérification de l'hypothèse en est une de comparaison. En effet, pour déterminer si les modèles OOP et SOA peuvent servir à évaluer les fonctionnalités CRUD du SGBD Teradata, il faut pouvoir comparer les caractéristiques de ces fonctionnalités avec chacun des principes constituant les fondements théoriques de ces modèles.

En établissant ce comparatif, il sera possible de déterminer quelles fonctionnalités respectent quels principes et à quel degré et ainsi de savoir si les modèles OOP et SOA peuvent servir à évaluer ces fonctionnalités d'une façon significative.

Pour réaliser cette comparaison, il faut déterminer les principes qui servent de critères d'évaluation, l'échelle de mesure qui permet de quantifier ces évaluations, les sujets qui sont à évaluer et puis enfin la façon d'analyser les résultats.

La détermination des critères consiste à consulter la littérature et identifier à partir de plusieurs sources les principes de chaque modèle qui semblent faire consensus.

La détermination de l'échelle doit être faite en gardant à l'esprit que cette échelle se veut nuancée tout en demeurant pertinente afin de permettre des mesures intermédiaires significatives et compréhensibles.

La détermination des sujets ou des fonctionnalités se fait en consultant directement la documentation officielle du SGBD Teradata.

L'analyse des résultats se fait en agrégeant les mesures de compatibilité par les différentes dimensions disponibles : fonctionnalités, principes et modèles. Ceci permet de dégager des mesures statistiques diverses (moyenne, étendue, écart interquartile) qui aident à déterminer de façon certaine si les modèles OOP et SOA peuvent servir à évaluer les fonctionnalités CRUD du SGBD Teradata.

Afin d'estimer la validité de cette méthodologie, il est possible de procéder avec une simple déduction logique ou syllogisme d'Aristote. Ainsi, sachant que certaines fonctionnalités sont, aux sens entendus dans le propos de cet essai, offertes comme des services et/ou implémentées comme des objets, il est possible d'énoncer le syllogisme suivant : les fonctionnalités peuvent être des objets ou des services; or puisque les objets et services sont compatibles avec les modèles OOP et SOA; alors les fonctionnalités peuvent être compatibles avec les modèles OOP et SOA.

Bien que ce syllogisme permet de démontrer une partie de la méthodologie exposée, il est possible, en partant du même postulat, de compléter le raisonnement et d'en démontrer la valeur complète.

En effet, en supposant que des services et objets soient évalués par cette méthodologie et que les résultats de cette évaluation soient compilés et analysés, il en ressortirait que ces services et objets sont totalement compatibles avec les modèles OOP et SOA. De ce constat serait alors déduit que ces modèles peuvent servir à évaluer ces fonctionnalités et que l'hypothèse de départ est vérifiée sans qu'il ne soit possible de la vérifier avec plus de vigueur.

Or, sachant que les fonctionnalités peuvent être des services et/ou des objets, il reste à démontrer à quel point ces fonctionnalités sont compatibles avec les modèles OOP et SOA, ce qui est déjà en soit une forme de validation de l'hypothèse de départ puisqu'il est logiquement exclu que ces fonctionnalités leurs soient incompatibles.

Enfin, afin de savoir si un principe retenu est pertinent pour l'évaluation des fonctionnalités, il est permis de penser qu'une fonctionnalité qui respecte un tant soit peu ce principe en cautionne d'emblée la pertinence puisque celui-ci aura permis l'évaluation de cette fonctionnalité. Cela est d'autant plus vrai si plusieurs fonctionnalités respectent à des degrés divers un principe donné.

En bref, la validation de l'hypothèse découle du fait qu'il est possible de déterminer qu'une fonctionnalité respecte à divers degrés *x* nombre des principes retenus des modèles OOP et SOA et qu'il en va de même pour l'ensemble des fonctionnalités.

Chapitre 2

Limites, portée et contexte de l'essai

Voici les limites, portée et contexte de l'essai qui viennent en définir l'environnement, les composantes, les thèmes importants, ce que cet essai couvre et ne couvre pas.

2.1 Limites de l'essai

Cet essai se limite à la validation de l'hypothèse de départ qui est de vérifier si les principes des modèles OOP et SOA peuvent servir à l'évaluation et au classement des fonctionnalités CRUD du SGBD Teradata.

Il n'est pas question ici de prétendre constituer un guide décisionnel complet permettant de décider sans autre considération de quelles fonctionnalités privilégier l'emploi.

En effet, s'il résulte de cet essai un classement des fonctionnalités, il demeure que les fonctionnalités examinées ne se valent pas toutes, ne sont pas toutes interchangeables et ne peuvent pas être considérées hors de tout contexte d'utilisation.

2.2 Le choix des modèles OOP et SOA

Le choix des modèles OOP et SOA pour l'évaluation des fonctionnalités du SGBD s'est naturellement imposé au fil des lectures préparatoires à cet essai. Ce choix reflète simplement les tendances de deux domaines qui composent ensemble le contexte dans lequel vient s'inscrire cet essai, c'est à dire les domaines de l'intelligence d'affaires et des SGBD de grande envergure.

La tendance en intelligence d'affaires est d'intégrer au mieux l'architecture orientée services car celle-ci procure de nombreux avantages dans ce domaine parmi lesquels figure la réutilisation, pour ne citer que celui-là.

> « *A Service Oriented Architecture (SOA) may provide benefits such as promoting re-use, the ability of combining services in order to create new composite applications, the use of decoupled services through a standard interface, supplying the technological approach in order to develop solutions of Business Intelligence (BI).* » [1]([4], p. 1)

Quant à la tendance du côté des systèmes de gestion de bases de données de grande envergure, cette tendance s'inscrit maintenant dans la voie de l'intégration des types objets au modèle relationnel traditionnel. En clair, les concepts de la OOP ont été portés vers les SGBD et, avec eux, de nombreux avantages dont celui encore une fois non négligeable de la réutilisation.

> « *The desire to represent complex objects has led to the development of object-oriented (OO) systems.[…] Object-oriented databases employ a data model that supports object-oriented features […] and abstract data types. […]The strong connection between application and database results in less code, more natural data structures, and better maintainability and reusability of code.* » [2] ([13], p. 2)

2.3 Pourquoi deux modèles au lieu d'un seul

Le choix d'utiliser deux modèles au lieu d'un seul pour l'analyse des fonctionnalités du SGBD s'est présenté de lui-même, sans préméditation. Puisque ces deux modèles induisent la

---

[1] Une architecture orientée services peut apporter des bénéfices comme la réutilisation, la possibilité de combiner des services dans le but de créer de nouvelles applications composites, l'utilisation de services découplés au travers d'un interface standard, fournissant l'approche technologique dans le but de développer des solutions d'intelligence d'affaires. (traduction libre)

[2] Le désir de représenter des objets complexes a conduit à l'élaboration de systèmes orientés objet (OO) [...] Les bases de données orientées objet emploient un modèle de données qui prend en charge des fonctionnalités orientées objet [...] et des types abstraits de données. [...] Le lien étroit entre l'application et la base de données résulte en moins de code, des structures de données plus naturelles, et une meilleure maintenance et réutilisabilité du code. (traduction libre)

réutilisation, bien qu'à des niveaux différents, il apparaissait opportun de les utiliser tous les deux.

Cette réutilisation s'exprimant justement à des niveaux différents permettait de penser que l'analyse serait ainsi plus complète, que l'évaluation des fonctionnalités représenterait par le fait même encore plus de valeur et que les modèles OOP et SOA puissent se révéler complémentaires dans le contexte de cet essai.

2.4 Quel code de programmation réutiliser?

Dans un entrepôt de données, le code de programmation le plus couramment utilisé est probablement celui qui a trait à l'exploitation des données, du moins c'est forcément une partie importante du code de programmation qui y est produit ou utilisé.

Dans un même ordre d'idée, il est probable que les meilleures opportunités d'amélioration de la réutilisation du code de programmation résident du côté du code permettant cette exploitation des données.

Or, puisqu'un SGBD met à disposition ses propres fonctionnalités permettant l'exploitation de ses données, il est naturel de se demander si ces fonctionnalités offertes sont utilisées de façon optimale et si des améliorations ne pourraient pas être apportées de ce côté-là.

2.5 Définition de la réutilisation

La réutilisation peut se définir d'une façon générale comme étant l'usage de concepts et d'objets précédemment acquis dans une nouvelle situation [41] ou encore comme étant l'usage de logiciels ou connaissances logicielles existants pour construire de nouveaux logiciels. [16] Cette réutilisation consiste alors en la représentation à différents niveaux d'abstraction de l'information liée au développement, le stockage de cette représentation pour référence future,

la liaison entre les nouvelles et anciennes situations, la duplication des objets et des actions déjà développés ainsi que leur adaptation pour répondre à de nouveaux besoins. [40]

La définition de la réutilisation étant ainsi exposée, il est à préciser que ce concept de réutilisation s'articulera de deux manières différentes dans le contexte du présent essai. D'une part, il désignera la réutilisation du code de programmation en ce qui concerne le modèle OOP et, d'autre part, il désignera la réutilisation du service pour le modèle SOA.

Bien que ces deux applications du concept de la réutilisation soient distinctes, il demeure qu'elles sont intimement liées puisqu'un service se compose de code de programmation. La différence s'exprime donc ici en terme de niveau : le code de programmation étant au plus bas niveau et le service à un niveau supérieur.

Puisque la réutilisation prend ici deux significations différentes selon que le modèle appliqué soit OOP ou SOA, il apparaît d'autant plus intéressant de les utiliser conjointement afin de réaliser ainsi une évaluation élargie à plus d'un niveau.

2.6 Le SGBD comme seul dispensateur de services

Il convient de statuer que le dispensateur de services soit le SGBD puisque, d'une part, le modèle SOA induit, par son principe de faible couplage, la simplification et la centralisation des composantes et que, d'autre part, un SGBD est conçu pour être son propre dispensateur de services et gérer l'offre de services permettant d'accéder à ses données. Ainsi, puisque le modèle SOA sert de référence et qu'il s'agit d'évaluer les fonctionnalités d'un SGBD, il faut voir le SGBD comme étant le seul dispensateur de services.

Seules les fonctionnalités offertes par le SGBD sont évaluées et les fonctionnalités nécessitant l'utilisation d'applications tierces sont ignorées, tel que les fonctionnalités offertes par une application client. Le présent essai portera sur les fonctionnalités offertes par le SGBD.

2.7 Définition d'une fonctionnalité

Dans le cadre de cet essai, le terme « fonctionnalité » signifiera une façon d'exploiter ou d'interroger les bases de données du SGBD, ce qui est bien différent du terme fonction qui a une signification bien précise en programmation.

Il s'agit de fonctionnalités propres au langage de requête structuré (SQL), supportées par le SGBD en place et définies comme étant des objets de la base de données au même titre que peut l'être une table.

2.8 Les fonctionnalités CRUD

Les fonctionnalités évaluées sont celles permettant l'exploitation des données et plus précisément celles qui permettent d'exécuter, en tout ou en partie, les opérations impliquées par le concept de CRUD, soit les opérations de création, sélection, mise à jour et suppression de données. [24]

2.9 Le SGBD Teradata

Un entrepôt de données requiert beaucoup de traitement et ses besoins en terme de performance sont élevés. C'est la raison pour laquelle le système de gestion de base de données en place dans le cas qui nous intéresse est fondé sur le modèle relationnel auquel ont été ajoutés de nouveaux types objet. Plus précisément, il s'agit d'un système Teradata.

Teradata est un SGBD de type relationnel qui a été enrichi des types objets et de leurs fonctionnalités associées. Mis à part ses capacités de calcul particulières qui en font un SGBD aux performances telles qu'il est surtout destiné aux systèmes de grande envergure, son implémentation des fonctions SQL demeure dans les normes standards.

Ainsi, les fonctionnalités qui sont retenues aux fins d'évaluation sont toutes des fonctionnalités faisant partie des normes SQL standards de l'Institut national américain des normes (ANSI) que Teradata aura implémentées telles quelles ou avec de légères différences.

Ce faisant, la méthodologie utilisée dans la présente évaluation est tout à fait générique et portable à d'autres SGBD bien connus qui implémentent les normes SQL ANSI tels Oracle, Microsoft SQL Server, DB2, Sybase pour ne nommer que ceux-là. Cependant, puisque l'implémentation des normes ANSI SQL diffère d'un SGBD à un autre, il faut se référer à la documentation officielle du fournisseur avant de réaliser l'évaluation à nouveau pour un autre SGBD.

Enfin, chaque fonctionnalité évaluée est annotée quant au respect des normes SQL ANSI et il sera donc possible de ne pas refaire l'évaluation d'une fonctionnalité pour un autre SGBD si l'implémentation de cette fonctionnalité y respecte également les normes.

2.10 Les fonctionnalités disponibles dans l'environnement de Teradata

En consultant la documentation officielle du SGBD Teradata, il a été possible de dénombrer neuf fonctionnalités étant définies comme des objets de la base de données et permettant d'exploiter les données. [36] Ce sont précisément ces fonctionnalités qui seront évaluées.

1. Les déclencheurs (D)
2. Les fonctions définies par l'utilisateur (FDU)
3. Les macros (M)
4. Les méthodes définies par l'utilisateur (MDU)
5. Les procédures stockées (PS)
6. Les procédures stockées externes (PSX)
7. Les types définis par l'utilisateur (TDU)
8. Les vues (V)
9. Les vues matérialisées (VM)

2.11 La nature des fonctionnalités retenues du SGBD Teradata

Les fonctionnalités retenues aux fins d'évaluation sont inspirées des normes SQL ANSI, si elles ne s'y conforment pas tout simplement. Ce sont donc des fonctionnalités usuelles des SGBD, même si leurs implémentations peuvent différer selon les SGBD.

Ce sont aussi des fonctionnalités natives des SGBD, c'est-à-dire qu'elles résident dans les SGBD et y sont stockées sous forme d'objets, au même titre que des tables par exemple.

De plus, ces fonctionnalités peuvent être déjà programmées et offertes à travers une librairie fournie avec le SGBD ou encore devoir faire l'objet de développement et faire alors partie intégrante de librairies programmées sur mesure pour les besoins utilisateur.

Ces fonctionnalités sont en fait des programmes conçus sur mesure ou non et offerts en tant que services par le SGBD où elles résident.

En clair, la nature bipartite de ces fonctionnalités qui seront évaluées dans cet essai et qui sont autant des programmes que des services vient appuyer le choix d'utiliser les deux modèles OOP et SOA pour effectuer l'évaluation.

2.12 La norme SQL ANSI

Le langage SQL fait l'objet d'une norme de l'ANSI depuis 1986. Malgré cela, des problèmes de portabilité du code SQL existent encore entre les SGBD en raison d'un manque de respect de la norme ou de ses interprétations différentes, que ce soit à cause de la grande taille de la norme et de ses spécifications incomplètes ou encore à cause des contraintes de compatibilité avec les systèmes existants.

2.13 L'évolution des SGBD

Aujourd'hui, il coexiste plusieurs modèles de SGBD fondés sur le modèle relationnel qui a été introduit par le Dr. E.F. Codd dans les années 1970 et qui a fait ses preuves depuis.

Avec l'avènement de l'Internet, de nouveaux formats de données plus complexes ont cependant été requis pour répondre à la demande (image, vidéos, etc.) et le modèle orienté objet a vu le jour sur les bases des principes de la programmation orientée objet. Dorénavant, de nouveaux types « riches » de données étaient rendus disponibles et même l'intégration du langage de requête était parfaitement imbriquée dans les langages de programmation usuels tels C++ , Java, SmallTalk, délaissant au passage le traditionnel SQL.

Si ce modèle orienté objet était très attrayant pour ses nombreuses qualités intrinsèques au concept orienté objet (encapsulation, héritage, liaison dynamique et polymorphisme), il n'en demeure pas moins qu'il a prouvé avoir ses limites en termes de performance.

En effet, l'optimisation des requêtes s'y trouvait grandement complexifiée, sa « scalabilité » ou sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande laissait à désirer et il s'est avéré incapable de soutenir les systèmes à grande échelle tels les entrepôts de données.

La solution pour les systèmes à grande échelle semblait donc résider dans la fusion des modèles relationnel et orienté objet et de nombreux vendeurs se sont en effet mis à offrir de nouveaux systèmes basés sur un nouveau modèle relationnel-objet, lequel étendait son support pour intégrer les types objet ainsi que les fonctionnalités qui sont attendus de tels types. [13]

Chapitre 3

Détermination des principes

Ce chapitre couvre la cueillette, la mise en commun des sources, la compilation, la sélection et la définition des principes des modèles OOP et SOA, lesquels serviront ensuite à l'élaboration des critères d'évaluation des fonctionnalités.

3.1 Le choix des principes à retenir des modèles OOP et SOA

Les fonctionnalités sont comparées aux modèles OOP et SOA selon les principes qui caractérisent ces modèles. Cependant, puisque la littérature diverge à cet effet, il faut trouver un moyen de faire un choix.

Pour opérer ce choix, il a été décidé de consulter plusieurs sources et ouvrages de références, d'en extraire les principes liés au modèle en cours et d'arrêter d'accumuler ces sources au moment où une impression de consensus semble apparaître parmi celles-ci. Enfin, les principes retenus sont ceux qui sont partagés par au moins la moitié de ces sources. De cette façon, il est possible de dégager un consensus quant aux principes qui sont retenus de ces modèles et qui servent de critères dans l'évaluation des fonctionnalités du SGBD.

3.2 La pondération des principes retenus

Puisqu'il est difficile de trouver des principes qui font l'unanimité parmi les auteurs et qu'il faut retenir ceux qui sont cités par au moins la moitié des sources afin d'obtenir une série de principes « généralement admis », il serait encore plus difficile de les pondérer.

En effet, attribuer un poids propre à chacun des principes selon l'importance que lui accorde chaque auteur semble relever des travaux d'Hercule. Pour cette raison, les principes retenus des modèles OOP et SOA se verront tous attribués le même poids.

3.3 Les principes OOP

Les principes fondamentaux du modèle OOP diffèrent légèrement selon les sources : pour cette raison six sources ont été consultées et les principes retenus puis définis sont ceux cités par au moins la moitié de ces sources.

Voici énumérés les principes fondamentaux du modèle OOP selon ces six différentes sources.

1. Abstraction, héritage, liaison dynamique et polymorphisme. [27]
2. Encapsulation, héritage, liaison dynamique, polymorphisme et récursion. [39]
3. Encapsulation, héritage et liaison dynamique. [44]
4. Encapsulation, héritage et polymorphisme. [7]
5. Encapsulation, héritage, liaison dynamique et polymorphisme. [26]
6. Composition, encapsulation, héritage et polymorphisme. [48]

3.4 Encapsulation

L'encapsulation est une technique pour minimiser les interdépendances entre des modules en définissant des interfaces externes stricts. [43] L'encapsulation sert également à cacher et protéger la structure interne d'un objet de manière à ce qu'aucun autre objet ne puisse altérer de façon imprévue ses opérations ou ses attributs. [8]

3.5 Héritage

L'héritage est un mécanisme pour classifier les objets et leur permettre d'hériter et de réutiliser les méthodes et propriétés d'objets parents. En programmation orientée objet, l'héritage est la première source de réutilisation du code. [25]

3.6 Liaison dynamique

La liaison dynamique signifie que la méthode qui sera exécutée par un objet suite à un appel de procédure ne sera connue qu'au moment de son exécution. [22]

3.7 Polymorphisme

Le polymorphisme signifie qu'une même entité, comme par exemple une méthode, peut prendre plusieurs formes et faire référence à plus d'un objet dans le temps. [22]

3.8 Les principes SOA

Les principes fondamentaux du modèle SOA diffèrent d'avantage selon les sources que ceux du modèle OOP : pour cette raison huit sources ont été consultées et les principes retenus puis définis sont ceux cités par au moins la moitié de ces sources.

Voici énumérés les principes fondamentaux du modèle SOA selon ces huit différentes sources.

1. Absence d'état, abstraction, autonomie, composabilité, contrat de service, découvrabilité, faible couplage, réutilisabilité. [14]
2. Absence d'état, abstraction, autonomie, composabilité, contrat de service, découvrabilité, faible couplage, réutilisabilité. [47]

3. Absence d’état, autonomie, composabilité, découvrabilité, faible couplage, granularité, idempotence, interopérabilité, réutilisabilité, technicité. [21]
4. Abstraction, composabilité, cohérence, contrat de service, encapsulation, évolutivité, extensibilité, granularité, interopérabilité, réutilisabilité, robustesse, universalité. [50]
5. Absence d’état, asynchronicité, autonomie, auto-descriptivité, composabilité, découvrabilité, faible couplage, gouvernance, granularité. [19]
6. Abstraction, autonomie, composant, composabilité, contrat de service, faible couplage, découvrabilité, encapsulation, flexibilité, granularité, interopérabilité, modularité, réutilisabilité. [2]
7. Découvrabilité, faible couplage, distributivité, interopérabilité, modularité, séparation par sujet, interchangeabilité. [3]
8. Absence d’état, abstraction, classification, composabilité, découvrabilité, faible couplage, fonctionnement par message, identité. [18]

3.9 Absence d’état

Un service doit minimiser la consommation de ressources en déléguant la gestion des informations d’état quand cela est nécessaire. [14] La gestion des informations d'état excessive peut compromettre la disponibilité d'un service et saper son potentiel d'évolutivité. Les services sont conçus pour rester en état uniquement si cela est nécessaire. L'application de ce principe exige que l'architecture technologique fournisse la gestion d'état. [14]

3.10 Abstraction

Le principe d’abstraction consiste à ne révéler d’un service que l’information nécessaire à son utilisation. Mis à part sa description dans le contrat de service, la logique ou l’implémentation du service est cachée. [14]

3.11 Autonomie

Le service a le contrôle sur sa logique qui est encapsulée. Il n'y a aucun besoin d'intervention humaine lors de son exécution. [2]

3.12 Composabilité

N'importe quel service doit être capable d'être combiné à d'autres services pour créer un service de plus haut niveau. [2]

3.13 Contrat de service

Chaque service décrit son but et ses moyens à travers le contrat de service. [14] La partie fondamentale du contrat de service consiste à décrire l'interface technique du service, son API. [15] Chaque interface de service définit ainsi un contrat qui sert à gérer les communications entre les services. [2]

3.14 Découvrabilité

Les services sont fournis avec des métadonnées de communication par lesquelles ils peuvent être efficacement découverts et interprétés. [14]

3.15 Faible couplage

Les dépendances entre les services sont réduites au minimum de sorte qu'un changement dans un service ne requiert pas de changement dans les services qui lui sont liés. [2]

3.16 Granularité

Cette caractéristique décrit à quel point une architecture orientée service est découpée en services et sous-services. Plus la granularité est fine, plus il y a de services et sous-services. [2]

3.17 Interopérabilité

Chaque service devrait avoir un interface bien défini qui puisse décrire le service et aider à l'interopérabilité entre les différents services ainsi qu'entre les différentes plateformes et technologies. [2] Les services réalisent l'interopérabilité soit en adhérant à des normes d'interfaces, soit en passant par un « courtier » de services capable de traduire d'un interface à un autre en temps réel. [42]

3.18 Réutilisabilité

Les services doivent être réutilisables pour le développement d'une multitude d'applications. [2]

Cette réutilisabilité n'est cependant pas le seul facteur pouvant favoriser la réutilisation telle que définie à la section 2.5. En effet, la réutilisabilité est certes une caractéristique facilitant la réutilisation mais il n'en demeure pas moins que chacun des autres principes des modèles OOP et SOA peut aussi être qualifié de facteur facilitant. Par exemple, il apparaît évident que l'héritage, le polymorphisme, le contrat de service ou la composabilité sont autant de principes qui favorisent cette réutilisation définie à la section 2.5, laquelle est bien plutôt un avantage induit par l'application des modèles OOP et SOA qu'un simple et unique principe du modèle SOA.

Chapitre 4

Construction de l'évaluation

Suite aux définitions des principes des modèles OOP et SOA, il faut définir les énoncés ou critères qui serviront à l'évaluation des fonctionnalités ainsi que l'échelle et la méthode d'évaluation qui serviront à noter ces fonctionnalités en regard de ces critères.

4.1 Échelle de Likert

Afin d'évaluer les fonctionnalités du SGBD, il convient de se doter d'une échelle qui permette de faire quelques nuances.

Une échelle de Likert sera donc utilisée afin de noter les fonctionnalités évaluées. Cette échelle comportera cinq niveaux et permettra de refléter le degré de conformité ou de compatibilité de chacune des fonctionnalités en regard de chacun des principes retenus des modèles OOP et SOA.

L'échelle proposée présente des valeurs de 0 à 4, la première signifiant un degré de compatibilité nul et la seconde une entière compatibilité avec, entre ces deux extrêmes, trois valeurs permettant de représenter sommairement les nuances de faible, moyenne et grande compatibilités.

Cette échelle se veut impaire ou asymétrique et à cinq niveaux afin de refléter avec plus de justesse les degrés de compatibilité estimés.

Tableau 4.1 Échelle d'évaluation des fonctionnalités

Note	Description du niveau de compatibilité
4	Entièrement compatible
3	Très compatible
2	Moyennement compatible
1	Peu compatible
0	Incompatible

Tableau 4.2 Énoncés OOP

Principe	Énoncé
Encapsulation	Une fonctionnalité rend-t-elle inaltérable son contenu?
Héritage	Une fonctionnalité peut-elle hériter du code de programmation d'une autre fonctionnalité?
Liaison dynamique	Une fonctionnalité permet-elle d'en appeler une autre dynamiquement?
Polymorphisme	Une fonctionnalité peut-elle prendre plusieurs formes?

Tableau 4.3 Énoncés SOA

Principe	**Énoncé**
Absence d'état	Une fonctionnalité est-elle sans état ou, autrement dit, est-ce que l'interface auquel elle adhère se charge de la gestion des états?
Abstraction	Une fonctionnalité permet-elle de cacher son fonctionnement interne?
Autonomie	Une fois lancée, une fonctionnalité a-t-elle besoin d'intervention humaine pour s'exécuter?
Composabilité	Une fonctionnalité peut-elle être combinée avec une autre du même type pour en créer une de plus haut niveau?
Contrat de service	Une fonctionnalité peut-elle avoir une documentation complète la décrivant?
Découvrabilité	Une fonctionnalité permet-elle de fournir sur demande des informations nécessaires à son utilisation?
Faible couplage	Une fonctionnalité a-t-elle un minimum de dépendance?
Granularité	Une fonctionnalité peut-elle aisément être exécutée par une autre du même type et être définie en sous-fonctionnalité?
Interopérabilité	Une fonctionnalité peut-elle être exécutée par le biais d'un interface ou langage de programmation standard tel SQL?
Réutilisabilité	Une fonctionnalité peut-elle être implantée une seule fois et réutilisée plusieurs fois dans des contextes différents, pour des applications différentes?

4.2 Méthode d’évaluation

La méthode utilisée pour évaluer les fonctionnalités et leur attribuer les notes présentées au tableau 4.1 en regard des énoncés des tableaux 4.2 et 4.3 est une méthode élaborée de manière à faire en sorte qu’un maximum de subjectivité soit écarté.

D’une part, les énoncés tirés des principes des modèles OOP et SOA ont été construits avec un souci de clarté et de concision afin d’éviter les ambiguïtés et ont également été adaptés au contexte de cet essai. De plus, ils ont été élaborés sous forme de questions dichotomiques auxquelles, à défaut de pouvoir répondre par la positive ou la négative nette, il sera tout de même relativement aisé de répondre en apportant quelques nuances.

D’autre part, ces énoncés ou questions sont amenés à évaluer des fonctionnalités qui ont été préalablement documentées par les publications officielles du SGBD Teradata de même que par des publications complémentaires universitaires et spécialisées. Chaque fonctionnalité est ainsi décrite par une documentation exhaustive parmi laquelle chaque énoncé devrait aisément trouver sa réponse.

Enfin, lorsque la réponse aux énoncés ou questions ne peut pas s’exprimer par la positive ou la négative nette, c’est-à-dire par l’entière compatibilité ou incompatibilité de la fonctionnalité, il demeure néanmoins tout à fait possible, tel que cité plus haut, d’exprimer les autres niveaux de l’échelle présentée au tableau 4.1. En effet, si une fonctionnalité est presque entièrement compatible ou, en d’autres termes, entièrement compatible mais avec un bémol, elle sera notée comme ayant étant très compatible. De la même façon, une fonctionnalité qui se verrait à peu de chose près totalement incompatible serait qualifiée de peu compatible. Et, finalement, une fonctionnalité qui ne pourrait pas être jugée ni presque compatible ni presque incompatible serait alors considérée comme étant moyennement compatible. En procédant de cette manière, il apparaît possible d’évaluer efficacement les fonctionnalités avec un minimum de subjectivité.

Chapitre 5

Évaluation et classement des fonctionnalités

Voici les évaluations des fonctionnalités présentées dans des tableaux synthèse. Les fonctionnalités sont classées en ordre alphabétique sans égard à d'autres considérations. Ces évaluations utilisent toutes l'échelle de Likert présentée au tableau 4.1.

5.1 Les déclencheurs

Voici l'évaluation de la fonctionnalité des déclencheurs telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les déclencheurs Teradata sont une implémentation de la norme SQL ANSI. Ils contiennent que du code SQL. Un déclencheur est une fonction activée par un événement tel une opération sur une table. Les déclencheurs sont surtout utiles pour la maintenance des tables. [10]

Tableau 5.1 Évaluation de la fonctionnalité des déclencheurs

Principe	Note	Principe	Note
Encapsulation	4	Composabilité	0
Héritage	0	Contrat de service	0
Liaison dynamique	0	Découvrabilité	0
Polymorphisme	0	Faible couplage	1[3]
Absence d'état	4	Granularité	0[4]
Abstraction	4	Interopérabilité	0[5]
Autonomie	4	Réutilisabilité	0

Sources : NCR Corporation (6), (2003), p. 122-123, Coffing, T. (2011), p. 459-465, Anonyme, (2012).

5.2 Les fonctions définies par l'utilisateur

Voici l'évaluation de la fonctionnalité des fonctions définies par l'utilisateur telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les fonctions définies par l'utilisateur s'intègrent au langage SQL, comme une fonction standard SQL. [46] Cependant, elles contiennent du code en langages C ou C++. Les fonctions définies par l'utilisateur Teradata ne sont pas une implémentation de la norme SQL ANSI.

[3] Un déclencheur est couplé à une table.

[4] Un déclencheur ne peut être exécuté que par la table à laquelle il est lié.

[5] Un déclencheur est indirectement exécuté via la table à laquelle il est lié.

Bien que l'objet de cet essai soit d'évaluer les fonctions disponibles dans le SGBD Teradata, il est intéressant de noter que les fonctions définies par l'utilisateur peuvent voir leurs possibilités d'utilisation étendues avec l'emploi du modèle de programmation SQL-MapReduce, lequel est rendu disponible grâce notamment au produit Aster Data de Teradata. En effet, cela permet les principes suivants : polymorphisme, liaison dynamique, composabilité, découvrabilité, contrat de service et granularité. [17] [49]

Tableau 5.2 Évaluation de la fonctionnalité des fonctions définies par l'utilisateur

Principe	**Note**	**Principe**	**Note**
Encapsulation	4	Composabilité	0
Héritage	0	Contrat de service	0
Liaison dynamique	0	Découvrabilité	0
Polymorphisme	0	Faible couplage	3[6]
Absence d'état	4[7]	Granularité	0
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	4

Sources : Tunnell, B. (2010), p. 2, NCR Corporation (2), (2005), p. 17-19, NCR Corporation (3), (2010), p. 13-16, Danesi, L. (2010), Friedman, E., Pawlowski, P. et Cieslewicz, J. (2009), p. 1413.

6 Le code en langage C ou C++ doit résider dans une librairie externe au SGBD.

7 L'absence d'état tient compte des multiples langages SQL, C et C++.

5.3 Macros

Voici l'évaluation de la fonctionnalité des macros telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les macros Teradata ne sont pas une implémentation de la norme SQL ANSI. Elles contiennent que du code SQL.

Tableau 5.3 Évaluation de la fonctionnalité de la macro

Principe	**Note**	**Principe**	**Note**
Encapsulation	4	Composabilité	4
Héritage	0	Contrat de service	0
Liaison dynamique	1[8]	Découvrabilité	1[9]
Polymorphisme	1[10]	Faible couplage	4
Absence d'état	4	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	4

Sources : Shet, V. (2010), p. 1, Coffing, T. (2011), p. 345, NCR Corporation (6), (2003), p. 19, 156-158.

[8] L'inclusion de logique conditionnelle dans une macro implique son exécution à partir du programme BTEQ.

[9] La découvrabilité d'une macro se limite à la confirmation de la présence d'information retournée.

[10] L'inclusion de logique conditionnelle dans une macro implique son exécution à partir du programme BTEQ.

5.4 Procédures stockées

Voici l'évaluation de la fonctionnalité des procédures stockées telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les procédures stockées Teradata sont une implémentation de la norme SQL ANSI. Elles contiennent que du code SQL.

Tableau 5.4 Évaluation de la fonctionnalité des procédures stockées

Principe	Note	Principe	Note
Encapsulation	4	Composabilité	4
Héritage	0	Contrat de service	0
Liaison dynamique	2[11]	Découvrabilité	2[12]
Polymorphisme	2[13]	Faible couplage	4
Absence d'état	4	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	4

Sources : NCR Corporation (6), (2003), p. 160-166, NCR Corporation (7), (2010), p. 524-537, NCR Corporation (5), (2003), p. 155, Charucki, (2012), p. 2.

11 L'inclusion de logique conditionnelle et de SQL dynamique ne vaut pas la liaison dynamique purement objet.

12 Les informations disponibles concernent seulement les paramètres et leurs attributs.

13 Sans être polymorphique, une PS peut être dynamique.

5.5 Les procédures stockées externes

Voici l'évaluation de la fonctionnalité des procédures stockées externes telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les procédures stockées externes Teradata ne sont pas une implémentation de la norme SQL ANSI. Elles contiennent du code C, C++ ou Java.

Tableau 5.5 Évaluation de la fonctionnalité des procédures stockées externes

Principe	Note	Principe	Note
Encapsulation	4	Composabilité	4
Héritage	0	Contrat de service	0
Liaison dynamique	3[14]	Découvrabilité	2[15]
Polymorphisme	2[16]	Faible couplage	3[17]
Absence d'état	4[18]	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	4

Sources : NCR Corporation (1), (2005), p. 170, NCR Corporation (4), (2012), p. 15, NCR Corporation (2), (2005), p. 109-110, Charucki, (2012), p. 2.

14 La logique conditionnelle est optimale, sans équivaloir à la liaison dynamique purement objet.

15 Les informations disponibles concernent seulement les paramètres et leurs attributs.

16 Sans être polymorphique, une PSX peut être dynamique.

17 Le code en langage C, C++ ou Java doit résider dans une librairie externe au SGBD.

18 L'absence d'état tient compte des multiples langages SQL, C, C++ et Java.

5.6 Les types et méthodes définis par l'utilisateur

Voici l'évaluation des fonctionnalités des types et méthodes définis par l'utilisateur tels qu'on les retrouve dans le SGBD Teradata en regard de leur compatibilité avec les principes retenus des modèles OOP et SOA.

Les types et méthodes définis par l'utilisateur Teradata ne sont pas une implémentation de la norme SQL ANSI. Les méthodes contiennent du code C ou C++.

Bien que les types et méthodes définis par l'utilisateur soient deux objets distincts du système Teradata, ils forment néanmoins une paire, au même titre que les objets et méthodes en programmation orientée objet.

Si les types définis par l'utilisateur peuvent être fonctionnels sans méthode définie par l'utilisateur, une méthode ne peut exister sans être reliée à un type. [31] Pour cette raison, ces deux objets sont évalués en même temps.

Tableau 5.6 Évaluation des fonctionnalités des types et méthodes définis par l'utilisateur

Principe	Note	Principe	Note
Encapsulation	4	Composabilité	4
Héritage	4	Contrat de service	4[19]
Liaison dynamique	4	Découvrabilité	4
Polymorphisme	4	Faible couplage	3[20]
Absence d'état	4[21]	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	4

Sources : NCR Corporation (3), (2010), p. 145-152, NCR Corporation (9), (2010), p. 74, Lawandus, G. (2005), p. 1-15, Zeidenstein, (2001), NCR Corporation (10), (2010), p. 239.

5.7 Les vues

Voici l'évaluation de la fonctionnalité des vues telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les vues Teradata ne sont pas une implémentation de la norme SQL ANSI. Elles contiennent que du code SQL.

19 Une méthode standard peut être implantée, voire héritée, afin de révéler l'information nécessaire du type.

20 Le code en langage C ou C++ doit résider dans une librairie externe au SGBD.

21 L'absence d'état tient compte des multiples langages SQL, C et C++.

Il est à noter que les vues qui sont intéressantes du point de vue de la réutilisation du code sont celles qui peuvent représenter des requêtes courantes et donc un avantage à être considérées comme des sous-requêtes. [52] Dans un tel cas, on parlera souvent de « vues intermédiaires » pour désigner des sous-vues.

Tableau 5.7 Évaluation de la fonctionnalité des vues

Principe	**Note**	**Principe**	**Note**
Encapsulation	4	Composabilité	4
Héritage	0	Contrat de service	0
Liaison dynamique	0	Découvrabilité	0
Polymorphisme	0	Faible couplage	4
Absence d'état	4	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	3[22]

Source : NCR Corporation (6), (2003), p. 120

5.8 Les vues matérialisées

Voici l'évaluation de la fonctionnalité des vues matérialisées telles qu'on les retrouve dans le SGBD Teradata en regard de sa compatibilité avec les principes retenus des modèles OOP et SOA.

Les vues matérialisées Teradata ne sont pas une implémentation de la norme SQL ANSI. Elles contiennent que du code SQL.

[22] La réutilisabilité est réduite faute de paramétrisation.

Dans un contexte de réutilisation du code et donc d'utilisation de vues intermédiaires et de requêtes imbriquées, des questions de performances entrent en jeu et l'utilisation de « vues matérialisées » devient nécessaire. Pour le SGBD Teradata, ces vues matérialisées sont désignées par le terme *Join Index*. [52]

Tableau 5.8 Évaluation de la fonctionnalité des vues matérialisées

Principe	**Note**	**Principe**	**Note**
Encapsulation	4	Composabilité	4
Héritage	0	Contrat de service	0
Liaison dynamique	0	Découvrabilité	0
Polymorphisme	0	Faible couplage	4
Absence d'état	4	Granularité	4
Abstraction	4	Interopérabilité	4
Autonomie	4	Réutilisabilité	2[23]

Sources : NCR Corporation (6), (2003), p. 120, Zheng,W. (2010), Au, G. et Ellmann, C. (2002), p. 2-28

5.9 Ensemble des résultats et classement des fonctionnalités

Voici les fonctionnalités et leur pointage permettant de mesurer leur degré de respect des principes retenus des modèles OOP et SOA, la note maximale étant de 56.

[23] La réutilisabilité est réduite faute de paramétrisation et d'ajout de jointure.

Tableau 5.9 Ensemble des résultats

	Note de chaque fonctionnalité							
Principe	**D**	**FDU**	**M**	**PS**	**PSX**	**TMDU**	**V**	**VM**
Encapsulation	4	4	4	4	4	4	4	4
Héritage	0	0	0	0	0	4	0	0
Liaison dynamique	0	0	1	2	3	4	0	0
Polymorphisme	0	0	1	2	2	4	0	0
Absence d'état	4	4	4	4	4	4	4	4
Abstraction	4	4	4	4	4	4	4	4
Autonomie	4	4	4	4	4	4	4	4
Composabilité	0	0	4	4	4	4	4	4
Contrat de service	0	0	0	0	0	4	0	0
Découvrabilité	0	0	1	2	2	4	0	0
Faible couplage	1	3	4	4	3	3	4	4
Granularité	0	0	4	4	4	4	4	4
Interopérabilité	0	4	4	4	4	4	4	4
Réutilisabilité	0	4	4	4	4	4	3	2
Total	17	27	39	42	42	55	35	34

Chapitre 6

Interprétation des résultats

Voici les validation, compilation, manipulation et interprétation des résultats des mesures de compatibilité des principes des modèles OOP et SOA sur les fonctionnalités CRUD du SGBD Teradata. Ces résultats, présentés sous plusieurs angles, permettront de mieux aborder la question de la validation de l'hypothèse de travail ainsi que de présenter quelques autres analyses intéressantes.

6.1 Validité des résultats

La méthodologie de comparaison utilisée pour la vérification de l'hypothèse, c'est-à-dire pour déterminer si les modèles OOP et SOA servent à évaluer les fonctionnalités CRUD du SGBD Teradata, semble avoir été efficace puisqu'elle permet de comparer et de noter les caractéristiques de ces fonctionnalités avec chacun des principes constituant les fondements théoriques de ces modèles.

À l'aide des critères d'évaluation issus des principes et de l'échelle de mesure utilisée pour quantifier les évaluations, il a en effet été possible de déterminer quelles fonctionnalités respectent quels principes et à quel degré et ainsi de savoir si les modèles OOP et SOA pouvaient servir à évaluer ces fonctionnalités d'une façon significative.

Les critères d'évaluation tirés de plusieurs sources de la littérature spécialisée se sont donc avérés pertinents pour l'évaluation des fonctionnalités. Quant à l'échelle de mesure, elle était suffisamment nuancée tout en demeurant pertinente afin de permettre des mesures intermédiaires significatives et compréhensibles.

Enfin, les résultats de cette évaluation présentés au chapitre 5 ont tous été validés et approuvés par un expert administrateur de base de données (DBA) Teradata. Il est à mentionner que cet expert a du même coup approuvé les choix des fonctionnalités CRUD Teradata présentés au chapitre 2, la détermination des principes OOP et SOA du chapitre 3 ainsi que l'élaboration des critères d'évaluation faite au chapitre 4.

Cette validation des résultats de l'évaluation par un expert DBA a permis de corriger le tir et de réajuster quelques évaluations à la hausse ou à la baisse. Plus précisément, seulement cinq évaluations sur un total de 112 ont été ajustées, soit un peu plus de 4%.

La valeur ajoutée à la validité des résultats de cette participation de l'expert DBA apparaît très importante et revêtir de multiples aspects. D'une part, l'expert DBA représente l'expérience et la connaissance par l'utilisation de ces petits détails qui échappent aux documentation officielle et littérature spécialisée. Ainsi l'évaluation aura-t-elle été faite par la documentation et l'expérience si l'on peut dire. D'autre part, le fait que seulement 4% des évaluations aient été retouchées vient du même coup cautionner la valeur de cette documentation et de l'évaluation qui aura été faite sur les bases de celle-ci. Enfin, puisqu'un SGBD Teradata est destiné aux systèmes de grandes envergures, il est raisonnable de croire que l'expertise de son DBA doive aller de pair.

6.2 De la pertinence de l'échelle de mesure

S'il apparaît évident, suite aux analyses précédentes, que les principes des modèles OOP et SOA peuvent s'appliquer aux fonctionnalités CRUD du SGBD Teradata et servir à les évaluer, il est par contre possible de se demander si l'échelle utilisée pour ce faire est bien adaptée.

En effet, à l'aide du tableau ici-bas, il est possible de constater que les deux valeurs extrêmes de cette échelle qui représentent 40% des choix possibles accaparent à elles-seules plus de

86% des mesures, laissant donc moins de 14% de ces mesures aux trois autres valeurs de l'échelle, lesquelles représentent pourtant 60% des choix possibles.

Cette constatation n'amène tout de même pas à se demander si une évaluation binaire n'eut pas suffit à la tâche puisque la présente échelle aura tout de même permis d'apporter quelques nuances appréciables là où c'était nécessaire mais le point mérite tout de même d'être soulevé.

Tableau 6.1 Nombre de mesures par niveau de compatibilité

Niveau de compatibilité	Total	%
Entièrement compatible	65	58,0
Très compatible	5	4,4
Moyennement compatible	6	5,4
Peu compatible	4	3,5
Incompatible	32	28,5

6.3 Agrégation par fonctionnalité et classement des fonctionnalités

L'agrégation par fonctionnalité des résultats des mesures de compatibilité permet d'effectuer un premier classement des fonctionnalités, par ordre décroissant de respect de l'ensemble des principes des modèles OOP et SOA.

Les totaux affichés montrent clairement qu'il y a des fonctionnalités qui respectent mieux que d'autres l'ensemble des principes des modèles OOP et SOA et qu'il peut même y avoir une dispersion importante des résultats. Cette dispersion semble tout à fait normale car, après tout, une fonctionnalité comme TMDU est de type objet et est donc fortement évaluée par les principes du modèle OOP.

Les autres colonnes affichent les étendue (E), écart interquartile (EI) et compatibilité (Co).

Tableau 6.2 Agrégation par fonctionnalité, classement et moyennes

Fonctionnalité	**Total**	**E**	**EI**	**Co**
TMDU	55	1	0	3,93
PS	42	4	2	3,00
PSX	42	4	2	3,00
M	39	4	3	2,79
V	35	4	4	2,50
VM	34	4	4	2,43
FDU	27	4	4	1,93
D	17	4	1	1,21
Moyenne	36,625	3,625	2,5	2,60

6.4 Total par fonctionnalité

La pertinence des principes des modèles OOP et SOA semble d'abord se vérifier avec la moyenne des totaux ci-haut puisque celle-ci, à 36,625, se situe à un niveau assez élevé à mi-chemin entre les valeurs 28 et 42, soit les valeurs qu'auraient obtenues des fonctionnalités étant respectivement moyennement compatible et très compatible avec les principes des modèles OOP et SOA.

6.5 Étendue par fonctionnalité

L'étendue est exprimée par l'écart entre les valeurs extrêmes des résultats. Les différentes valeurs obtenues de l'agrégation par fonctionnalité des mesures de compatibilité offrent une étendue moyenne importante de 3,625, soit de 90,6% pour l'ensemble des fonctionnalités, le maximum étant de 4.

Cette étendue importante indique que les fonctionnalités ont été individuellement évaluées de façon fort variée par chacun des principes des modèles OOP et SOA, ce qui constitue un second pas vers la confirmation de ces principes en tant que mesures valables.

En effet, cette étendue permet de ne pas douter de la possibilité d'évaluer ces fonctionnalités en regard des principes des modèles OOP et SOA puisque ces principes, dans l'ensemble, se sont différemment appliqués pour chaque fonctionnalité.

6.6 Écart interquartile par fonctionnalité

L'écart interquartile est l'étendue obtenue en retranchant le quart des résultats extrêmes, c'est-à-dire 25% des résultats les plus élevés et 25% des résultats les plus bas. L'écart interquartile représente alors l'étendue de la moitié des résultats.

Ainsi, si les mesures extrêmes sont enlevées des résultats de chacune des fonctionnalités, il est encore permis de constater que chaque fonctionnalité est évaluée avec des variations de mesure moyennes de 2,5 sur 4, soit l'équivalent de 62,5%.

6.7 Compatibilité par fonctionnalité

Les mesures de compatibilité agrégées par fonctionnalité permettent d'en calculer les compatibilités pour l'ensemble des principes des modèles OOP et SOA. Il est possible de constater que la compatibilité moyenne de toutes les fonctionnalités envers l'ensemble des principes est assez élevée avec un chiffre de 2,60 plus près du « très compatible » que du « moyennement compatible », soit à 65% pour être précis.

Tableau 6.3 Autres mesures statistiques sur l'ensemble des fonctionnalités

Mesure	Valeur
E	38
EI	8

6.8 Étendue de l'ensemble des fonctionnalités

L'étendue de l'ensemble des totaux des fonctionnalités permet de constater que les mesures de compatibilité ne sont pas que variées pour chacune des fonctionnalités, mais aussi entre ces fonctionnalités elles-mêmes. En effet, l'étendue maximale possible étant de 56 points, il appert que l'étendue actuelle de 38 points entre les minimum de D et maximum de TMDU représente une proportion non négligeable de 67,85% de cette première. Cette étendue est donc importante car, pour être maximale, il aurait fallu qu'une fonctionnalité soit totalement

compatible avec chacun des quatorze principes des modèles OOP et SOA et qu'une autre leur soit en même temps totalement incompatible, ce qui était improbable.

6.9 Écart interquartile de l'ensemble des fonctionnalités

L'écart interquartile sur l'ensemble des résultats agrégés par fonctionnalité affiche une valeur restreinte de huit, ce qui fait en sorte de situer la moitié des résultats autour de la moyenne, soit entre des valeurs de compatibilité équivalentes à environ 51% et 79%.

6.10 Ventilation de l'ensemble des fonctionnalités

Considérant les valeurs numériques correspondantes aux niveaux de compatibilité présentés au tableau 4.1, il est aisé de se rendre compte que les résultats agrégés par fonctionnalité affichent une bonne ventilation entre ces différentes valeurs. En effet, le niveau d'incompatible est le seul qui n'y soit pas représenté, même de loin.

Cette bonne ventilation des résultats implique que les principes des modèles OOP et SOA permettent non seulement d'évaluer les fonctionnalités mais qu'en plus ils permettent de les évaluer distinctement les unes des autres, d'une manière qui soit articulée. Ceci, encore une fois, vient militer en faveur de la validation de l'hypothèse de départ.

6.11 Agrégation par principe et classement des principes

L'agrégation par principe des mesures de compatibilité permet d'effectuer un premier classement par ordre décroissant de ces principes. Ce tableau, basé sur le total des points que chaque principe a pu appliquer à l'ensemble des fonctionnalités évaluées, permet d'illustrer le niveau de compatibilité qu'a l'ensemble des fonctionnalités avec chaque principe.

Ce classement indique que certains principes s’appliquent mieux que d’autres à l’ensemble des fonctionnalités évaluées et, en d’autres mots, semblent donc plus pertinents.

Tableau 6.4 Agrégation par principe, classement et moyennes

Principe	Total	E	EI	Co	Qualité
Absence d'état	32	0	0	4	Rassembleur
Abstraction	32	0	0	4	Rassembleur
Autonomie	32	0	0	4	Rassembleur
Encapsulation	32	0	0	4	Rassembleur
Interopérabilité	28	4	0	3,5	Rassembleur
Faible couplage	26	3	1	3,25	Rassembleur
Réutilisabilité	25	4	1	3,13	Rassembleur
Composabilité	24	4	0	3	Rassembleur
Granularité	24	4	0	3	Rassembleur
Liaison dynamique	10	4	2	1,25	Différenciateur
Découvrabilité	9	4	2	1,13	Différenciateur
Polymorphisme	9	4	2	1,13	Différenciateur
Contrat de service	4	4	0	0,5	Différenciateur
Héritage	4	4	0	0,5	Différenciateur
Moyenne	20,79	2,78	0,57	2,60	---

6.12 Total par principe

L'agrégation des mesures par principe indique une moyenne assez élevée de 20,79 (64,9%) qui se situe à mi-chemin entre les valeurs 16 et 24, soit les valeurs qu'auraient obtenues des principes étant respectivement moyennement compatible et très compatible avec l'ensemble des fonctionnalités.

6.13 Étendue par principe

L'étendue moyenne des principes est de 2,78 (69,5%), ce qui laisse faussement supposer des étendues variées pour chacun des principes car il est possible de constater qu'il n'en est rien puisque certaines étendues sont beaucoup plus importantes que d'autres. Mais cela n'a pas grande importance dans le cas des principes comme il sera démontré ici-bas.

En effet, comme c'est le cas pour l'étendue des mesures agrégées par fonctionnalité, il semble naturel de penser et c'est valable qu'un principe affichant une étendue certaine est d'emblée pertinent puisqu'il permet effectivement de mesurer différemment les fonctionnalités entre elles.

Cependant, à l'inverse, le peu ou l'absence d'étendue pour un principe ne l'invalide pas non plus car un principe peut très bien s'appliquer de la même façon à toutes les fonctionnalités sans pour autant perdre de sa pertinence.

Par exemple, les principes des abstraction, autonomie, encapsulation et absence d'état n'offrent aucune variation de mesures dans leurs évaluations des fonctionnalités mais, par contre, ces mesures sont toutes au maximum, ce qui veut dire que les fonctionnalités sont toutes entièrement compatibles avec ces principes.

6.14 Écart interquartile par principe

La moyenne de l'écart interquartile de chacun des principes indique, qu'une fois les mesures extrêmes enlevées, chaque principe a tendance à présenter une évaluation constante et très peu dispersée des fonctionnalités, avec une variation d'environ 14%.

6.15 Compatibilité par principe

La compatibilité de chacun des principes est très parlante dans ce tableau. Il est en effet possible de constater que toutes les fonctionnalités sont totalement compatibles avec quatre d'entre eux alors que deux autres principes n'offrent qu'une très maigre compatibilité pour l'ensemble des fonctionnalités.

En moyenne, la compatibilité d'un principe se situe à peu près à mi-chemin entre le moyennement compatible et le très compatible (65%), ce qui ne semble pas mauvais pour la validation de l'hypothèse de départ.

6.16 Qualité des principes

Si certains principes ont une compatibilité très faible et semblent par le fait même ne pas être très pertinents en tant que mesures pour l'ensemble des fonctionnalités, il demeure que ces principes offrent une étendue maximale, ce qui signifie qu'ils sont au moins entièrement compatibles avec certaines fonctionnalités.

Or, un principe qui s'applique ainsi à une minorité de fonctionnalités, voire à une seule, permet du même coup à celle-ci de se démarquer du lot. Un tel principe devient alors différenciateur et revêt par le fait-même une importance toute nouvelle.

Ainsi, il est très intéressant de constater que les principes d'héritage, contrat de service, polymorphisme, découvrabilité et liaison dynamique ne s'appliquent qu'à une minorité de fonctionnalités et qu'ils permettent donc de les différencier.

Cette qualité des principes a été illustrée dans la colonne qualité du tableau 6.4 où les principes sont qualifiés de « différenciateur » et « rassembleur » par opposition.

Tableau 6.5 Autres mesures statistiques sur l'ensemble des principes

Mesure	Valeur
E	28
EI	19

6.17 Étendue de l'ensemble des principes

L'étendue existante entre les principes est de 28 et représente un niveau très élevé, soit plus de 87% du maximum possible. Cela confirme que certains principes offrent des compatibilités totales ou presque nulles à l'ensemble des fonctionnalités.

6.18 Écart interquartile de l'ensemble des principes

Une fois les extrêmes enlevés, il reste encore un très important écart de 19 entre les valeurs extrêmes restantes, soit l'équivalent de 59%. Cela signifie que la moitié des principes offrent encore des niveaux de compatibilité très variés à l'ensemble des fonctionnalités.

6.19 Compatibilité des principes par modèle

Les principes utilisés proviennent de deux modèles différents, les modèles OOP et SOA. Il apparaît intéressant de voir quelles sont les compatibilités pour chacun de ces modèles.

Tableau 6.6 Mesures statistiques des compatibilités des principes par modèle

Mesure	OOP	SOA
Moyenne	1,72	2,95
E	3,5	3,5

6.20 Moyenne de la compatibilité des principes par modèle

Les moyennes des compatibilités des principes de chacun des modèles font en sorte de qualifier le modèle OOP à un peu plus de la mi-chemin entre la faible compatibilité et la moyenne compatibilité, soit à 43%, et le modèle SOA à la haute compatibilité, soit à 73,75%. En d'autres mots, l'ensemble des fonctionnalités est près de la moyenne compatibilité avec le modèle OOP et hautement compatible avec le modèle SOA.

6.21 Étendue de la compatibilité des principes par modèle

Les étendues des deux modèles sont identiques et presque maximales. Ceci indique que les deux modèles ont eu chacun au moins deux principes totalement et presque aucunement compatibles avec l'ensemble des fonctionnalités. Ces principes sont l'encapsulation et l'héritage pour le modèle OOP et les abstraction, autonomie, absence d'état et contrat de

service pour le modèle SOA. L'étendue seule, puisqu'identique, ne permet donc pas de distinguer les modèles.

L'étendue seule ne permet pas de distinguer les modèles mais, en conjonction avec des moyennes très différentes, cette similitude devient révélatrice de la présence d'un principe à très forte compatibilité parmi ceux du modèle OOP.

6.22 Compatibilité des fonctionnalités par modèle

Au lieu de mettre les modèles en relation avec les compatibilités des principes, il est encore plus révélateur de les mettre en relation avec les compatibilités des fonctionnalités. Pour ce faire, il faut départager les mesures de compatibilité des fonctionnalités entre les modèles.

Tableau 6.7 Compatibilité moyenne des fonctionnalités par modèle

	Compatibilité	
Fonctionnalité	**OOP**	**SOA**
TMDU	4	3,9
PS	2	3,4
PSX	2,25	3,3
M	1,5	3,3
V	1	3,1
VM	1	3
FDU	1	2,3
D	1	1,3

Tableau 6.8 Mesures statistiques par modèle des compatibilités des fonctionnalités

Mesure	OOP	SOA
Moyenne	1,72	2,95
E	3	2,6

6.23 Moyenne de la compatibilité des fonctionnalités par modèle

Les moyennes des compatibilités des fonctionnalités par modèle sont identiques à celles des principes du tableau 6.6 et c'est bien normal puisqu'il s'agit des mêmes mesures moyennes mais assemblées autrement.

6.24 Étendue de la compatibilité des fonctionnalités par modèle

L'étendue plus large des compatibilités des fonctionnalités avec le modèle OOP, en conjonction avec une moyenne beaucoup plus faible, indique la présence d'une fonctionnalité à très forte compatibilité avec les principes du modèle OOP. Cette fonctionnalité est la TMDU dont les très fortes compatibilités sont dans le tableau 6.7.

6.25 Qualité des modèles

Il est permis de constater qu'un seul modèle a accordé une note parfaite à une fonctionnalité : c'est-à-dire que la fonctionnalité en question est totalement compatible avec tous ses principes et il s'agit du modèle OOP et de la fonctionnalité TMDU.

Compte tenu de la faible compatibilité des autres fonctionnalités avec le modèle OOP, il est permis de constater que ce modèle devient différenciateur lorsqu'il s'agit de la fonctionnalité TMDU.

Bien que cette même fonctionnalité obtienne la note presque parfaite avec le modèle SOA, celui-ci demeure tout de même rassembleur puisque l'ensemble des autres fonctionnalités lui est aussi hautement compatible.

6.26 Résumé et conclusion de l'analyse des résultats

En résumé, l'analyse des résultats a permis de dégager plusieurs constats qui, tous réunis, permettent de ne pas douter de la pertinence de se servir des modèles OOP et SOA pour évaluer les fonctionnalités CRUD du SGBD Teradata et d'ainsi confirmer l'hypothèse de travail.

En effet, en agrégeant les mesures de compatibilité par fonctionnalité, il est permis de constater que celles-ci sont toutes compatibles avec l'ensemble des principes et qu'il n'y en a pas une seule qui n'y soit moins que peu compatible. Que ce soit d'un point de vue individuel ou d'ensemble, les mesures de compatibilité des fonctionnalités sont bien variées et l'ensemble des principes permet donc de mesurer les fonctionnalités efficacement et avec diversité.

D'un autre côté, en agrégeant les mesures de compatibilité par principe, il est permis de constater que ceux-ci sont tous compatibles avec l'ensemble des fonctionnalités mais que ces compatibilités sont beaucoup plus diversifiées et dispersées que pour les fonctionnalités. D'un extrême à l'autre, certains principes sont totalement compatibles avec toutes les fonctionnalités alors que d'autres ne sont que peu compatibles avec l'ensemble des fonctionnalités. À la limite certains principes sont compatibles, et alors totalement, avec une

seule fonctionnalité et permettent par le fait même de la différencier du lot, mais aucun principe est incompatible avec l'ensemble des fonctionnalités.

Cette grande différence de compatibilité qui n'atteint jamais l'incompatibilité fait en sorte de camper les principes en deux types : les plus compatibles étant les principes rassembleurs et les moins compatibles étant les principes différenciateurs. Ces derniers permettant de différencier une fonctionnalité des autres acquièrent alors une importance certaine. Bref, tous les principes sont donc pertinents et l'hypothèse de travail est vérifiée.

Enfin, pour ce qui est des modèles, il apparaît clairement que le modèle SOA est plus rassembleur que différenciateur alors que le modèle OOP est davantage différenciateur. Ce dernier constat est vérifié par le fait que la fonctionnalité TMDU est la seule qui soit de type objet, la seule qui a donc pu profiter pleinement de cette différenciation et, par conséquent, celle qui a obtenu le plus haut score de compatibilité.

Conclusion

Les coûts d'un entrepôt de données étant très élevés, il apparaissait nécessaire de chercher à les réduire et la voie de la réutilisation du code de programmation semblait faire du sens à première vue.

Les fonctionnalités CRUD offertes par le SGBD semblaient constituer une bonne opportunité à cet effet étant donné leur fréquence d'utilisation dans un environnement informationnel. Ainsi, s'il pouvait être trouvé une manière pertinente d'en aborder l'étude, peut-être pourrait-il aussi être trouvé une manière de mieux les utiliser afin d'en favoriser la réutilisation et de réduire ainsi les coûts de programmation.

Les tendances de l'évolution de l'intelligence d'affaires, des entrepôts de données et des SGBD ainsi que les réputations des modèles OOP et SOA ont tôt fait d'inspirer une manière d'aborder l'étude de ces fonctionnalités. En effet, il est apparu pertinent d'évaluer ces fonctionnalités à l'aide des modèles OOP et SOA car, s'il s'avérait qu'une fonctionnalité respecte les principes de ces modèles et leur soit compatible, alors peut-être que cette fonctionnalité pourrait offrir semblables avantages en ce qui a trait à la réutilisation du code. De plus, les réutilisations induites par ces deux modèles n'étant pas du même niveau, l'une étant au niveau du code de programmation et l'autre au niveau du service, la profondeur de l'analyse s'en trouverait accentuée.

L'hypothèse de travail qui a ensuite été formulée pour mener à bien cette étude proposait donc de vérifier si les principes des modèles OOP et SOA pouvaient servir à l'évaluation et à la classification des fonctionnalités CRUD qui sont mises à disposition par le SGBD Teradata, lequel a été choisi pour cette étude car il convient aux systèmes de grande envergure.

La méthodologie élaborée pour mener à bien cette étude en était une de comparaison et consistait à comparer les caractéristiques de chacune des fonctionnalités avec chacun des principes constituant les fondements théoriques des modèles OOP et SOA.

Pour ce faire, il a fallu consulter diverses sources documentaires pour rassembler des principes qui fassent consensus, définir ces principes, élaborer des critères d'évaluation à partir de ces principes, déterminer les fonctionnalités à évaluer, déterminer une échelle d'évaluation qui puisse servir à noter les fonctionnalités en regard de leur compatibilité avec chacun des principes et enfin, présenter les résultats d'analyse de diverses manières, selon différentes dimensions et mesures statistiques afin de fonder la réflexion devant mener à la vérification de l'hypothèse de départ.

Tout ceci étant fait, il a été permis de vérifier l'hypothèse de départ à savoir qu'il est possible d'évaluer et de classifier les fonctionnalités CRUD du SGBD Teradata à l'aide des modèles OOP et SOA.

De plus, les résultats d'analyse ont permis de dégager une compatibilité plus que satisfaisante des fonctionnalités envers les principes retenus de ces modèles de même qu'un résultat inattendu : certains principes et le modèle SOA se sont révélés rassembleurs alors que, par opposition, certains autres principes ainsi que le modèle OOP se sont révélés différenciateurs.

Au final, ces résultats ont même fait apparaître les modèles OOP et SOA comme étant complémentaires car, non seulement s'appliquent-ils à des niveaux qui leurs sont propres, mais, comme il a été mentionné ci-haut, ils se sont aussi distingués l'un de l'autre en étant rassembleur et différenciateur.

Pour faire suite à cet essai et en guise de développement futur, il conviendrait certainement de compléter la distinction et le classement des fonctionnalités selon le contexte d'utilisation. En effet, une telle analyse constituerait un parfait complément à la présente.

Une autre possibilité très intéressante serait de simplement utiliser le classement des fonctionnalités présenté au tableau 6.2 afin de construire une grille d'évaluation qui permettrait à une entité de faire le point sur ses habitudes de programmation. Partant de cet état des lieux, cette entité serait alors peut-être en mesure de dégager des opportunités et de privilégier l'utilisation de fonctionnalités permettant une meilleure réutilisation, le tout dans le but de réduire les coûts de développement.

Liste des références

[1] Abadi, M. et Cardelli L., *A Theory of Objects*, 1e éd., Springer-Verlag, New York, 1996, 396 p.

[2] Al-kofahi, M., *Service Oriented Architecture (SOA) Security Models*, ProQuest Dissertations & Theses, Ann Arbor, 2011, 141 p.

[3] Altman, R., *SOA Overview and Guide to SOA Research*, http://my.gartner.com/portal/server.pt?open=512&objID=260&mode=2&PageID=3460702&id=1396514&ref=seo, 10 août 2012.

[4] Andreescu, A. et Mircea, M., *Combining actual trends in software systems for business management*, CompSysTech '08 Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, vol. 9, 2008. p. 1-6.

[5] Anonyme, *Teradata Triggers*, http://teradatafaqs.blogspot.ca/p/tutorial.html, 1er novembre 2012.

[6] Au, G. et Ellmann, C., *Introduction to Materialized Views In Teradata*, http://tensupport.com/library/materialview.pdf, 9 octobre 2012.

[7] Bersini, H., *La programmation orientée objet - Cours et exercices en UML 2 avec Java 6, C# 4, C++, Python, PHP 5 et LinQ*, 5e éd., Eyrolles, Paris, 2011, 644 p.

[8] Burton, P. et Bruhn, R., *Using UML to facilitate the teaching of object-oriented systems analysis and design*, Journal of Computing Sciences in Colleges, vol. 19, no. 3, janvier 2004, p. 278-290.

[9] Charucki, R., *Event-Based Applications*, http://www.teradata.com/resources/white-papers/event-based-applications-eb4334/?type=WP, 10 août 2012.

[10] Coffing, T., *Tera-Tom on Teradata SQL V12-V13*, Coffing Publishing, Middletown, 2011, 543 p.

[11] Dan, A., Johnson, R., et Carrato, T., SOA service reuse by design, SDSOA '08 Proceedings of the 2nd international workshop on Systems development in SOA environments, 2008, p. 25-28.

[12] Danesi, L., *Extend the possibilities - Choosing the extensibility feature allows for database behavior change and customization*, http://teradatamagazine.com/v10n02/Tech2Tech/Extend-the-possibilities/, 23 octobre 2012.

[13] Devarakonda, R., *Object-relational database systems: the road ahead*, Crossroads, Vol. 7, No. 3, 2001, p. 15-18.

[14] Erl, T., *SOA Principles of Service Design*, 1e éd., Prentice Hall, Boston, 2007, 608 p.

[15] Erl, T., *SOA Glossary Service Contract*, http://www.soaglossary.com/service_contract.php, 10 octobre 2012.

[16] Frakes W. B. et Kang K., *Software Reuse Research: Status and Future*, IEEE Transactions on Software Engineering, vol. 31, no. 7, juillet 2005, p.529.

[17] Friedman, E., Pawlowski, P. et Cieslewicz, J., *SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions*, Proc. VLDB Endow, vol. 2, no. 2 août 2009, p. 1402-1413.

[18] Graham, I., *Requirements Modelling and Specification for Service Oriented Architecture*, 1e éd., Wiley, Chichester, 2008, 320 p.

[19] Holley K. et Arsanjani A., *100 SOA Questions: Asked and Answered*, 1e éd., Prentice Hall, Boston, 2010, 304 p.

[20] Janies, L. *The great convergence, Aligning master data management, service oriented architecture and the active data warehouse to improve business agility*, Teradata magazine, September 2007, p. 1-2.

[21] Josuttis, N., *SOA in Practice, The Art of Distributed System Design*, 1e éd., O'Reilly, Sebastopol, 2009, 344 p.

[22] Korson, T. et McGregor, J., *Understanding object-oriented: a unifying paradigm*, Communications of the ACM, vol. 33, no. 9, septembre 1990, p. 40-60.

[23] Lawandus, G., *User-defined Types, Modeling the Real World with UDTs*, http://www.teradata.com/white-papers/User-defined-Types-EB4622, 10 juillet 2012.

[24] Martin, J., *Managing the database environment*, Prentice Hall, Upper Saddle River, 1983, 464 p.

[25] Matthews, S. et Grove, C., Applying object-oriented concepts to documentation, dans *Proceedings of the 10th annual international conference on Systems documentation (SIGDOC '92)*, ACM, New York, 1992, p. 265-271.

[26] Meyer, B., *Conception et programmation orientées objet*, 1e éd., Eyrolles, Paris, 2008, 1223 p.

[27] Mitchell, J., *Concepts in programming languages*, 1e éd., Cambridge University Press, Cambridge, 2002, 540 p.

[28] Mohagheghi, P. et Conradi, R., *An empirical investigation of software reuse benefits in a large telecom product*, ACM Transactions on Software Engineering and Methodology, vol. 17, no. 3, juin 2008, p. 13:1-13:31.

[29] NCR Corporation, *SQL Reference: Data Definition Statements*, El Segundo, 2005, 1308 p.

[30] NCR Corporation, *SQL Reference : UDF and External Stored Procedure Programming*, El Segundo, 2005, 236 p.

[31] NCR Corporation, *SQL Reference : UDF, UDM, and External Stored Procedure Programming*, El Segundo, 2010, 436 p.

[32] NCR Corporation, *SQL Stored Procedures and Embedded SQL*, El Segundo, 2012, 476 p.

[33] NCR Corporation, *Teradata Parallel Transporter Application Programming Interface*, El Segundo, 2010, 196 p.

[34] NCR Corporation, *Teradata RDBMS SQL Reference : Volume 1 - Fundamentals*, El Segundo, 2003, 230 p.

[35] NCR Corporation, *Teradata RDBMS SQL Reference : Volume 6 - Data Manipulation Statements*, El Segundo, 2003, 738 p.

[36] NCR Corporation, *Introduction to Teradata*, El Segundo, 2010, 218 p.

[37] NCR Corporation, *SQL Fundamentals*, El Segundo, 2010, 316 p.

[38] NCR Corporation, *SQL Data Types and Literals*, El Segundo, 2010, 340 p.

[39] Pierce, B., *Types and Programming Languages*, 1e éd., MIT Press, Cambridge, 2002, 645 p.

[40] Prieto-Diaz, R., *Classification of reusable modules. Software Reusability: Concepts and Models*, Addison-Wesley Pub. Co., New York, 1989, p. 1: 99-123.

[41] Prieto-Diaz, R. et Freeman, P., *Classifying Software for Reusability*, IEEE Software, Janvier 1987, p. 6-16.

[42] Rouse, M., *Interoperability Definition*, http://searchsoa.techtarget.com/definition/interoperability, 10 octobre 2012.

[43] Schärli, N., Black, P., et Ducasse, S., Object-oriented encapsulation for dynamically typed languages, dans *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '04)*, ACM, New York, 2004, p. 130-149.

[44] Scott, M., *Programming language pragmatics*, 2e éd., Morgan Kaufmann, San Francisco, 2006, 912 p.

[45] Shet, V., *How to create and use Macro?*, http://www.teradatahelp.com/2010/09/macro-is-teradata-extension-to-ansi-sql.html, 28 août 2012.

[46] Tunnell, B., *Support for Multiple Languages*, http://www.teradatamagazine.com/uploadedFiles/Support-for-Multiple.pdf, 9 octobre 2012.

[47] Valipour H., AmirZafari B., Maleki N. et Daneshpour N., *A Brief Survey of Software Architecture Concepts and Service Oriented Architecture*, 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT), août 2009, p. 34-38.

[48] Weisfeld, M., *The Object-Oriented Thought Process*, 3e éd., Addison-Wesley Professional, Boston, 2008, 360 p.

[49] White, C., *MapReduce and the Data Scientist*, http://www.teradata.com/white-paper/MapReduce-and-the-Data-Scientist/, 23 octobre 2012.

[50] Wright, M. et Reynolds, A., *Oracle SOA Suite 11g R1 Developer's Guide*, 1e éd., Packt Publishing, Birmingham, 2010, 720 p.

[51] Zeidenstein, K., DB2's object-relational highlights: User-defined structured types and object views in DB2, http://www.ibm.com/developerworks/data/library/techarticle/zeidenstein/0108zeidenstein.html, 1er novembre 2012.

[52] Zheng, W., *Teradata Join Index*, http://teradata.weizheng.net/2010/10/teradata-join-index.html, 9 octobre 2012.

Annexe 1

Bibliographie

Borsadwala, T., *The enterprise alchemist*, http://www.teradatamagazine.com/v09n03/Tech2Tech/The-enterprise-alchemist/, 29 août 2012.

Ezquerro, R., *Discover Teradata Meta Data Services*, http://www.teradata.com/white-papers/Discover-Teradata-Meta-Data-Services-eb5360, 29 août 2012.

Kilaru, Haritha, *A pattern language for service-oriented architecture*, ProQuest Dissertations & Theses, 2006, 203 p.

Mohania, M., Avoiding Re-computation: View Adaptation in Data Warehouses, dans *In Proc. of 8 th International Database Workshop*, Hong Kong, 1997, p. 151-165.

NCR Corporation, *SQL External Routine Programming*, El Segundo, 2012, 746 p.

NCR Corporation, *Teradata JDBC Driver User Guide Chapter 2*, http://developer.teradata.com/doc/connectivity/jdbc/reference/current/jdbcug_chapter_2.html#CCHJHFD3, 26 août 2012.

NCR Corporation, *Teradata Meta Data Services*, http://www.teradata.com/tools-and-utilities/meta-data-services/, 29 août 2012.