



Module D314 – Document de conception

|  |  |  |
| --- | --- | --- |
| Suivi des versions | | |
| Date | **Version** | **Objet de la version** |
| 19/11/2015 | 01 | Création du document |
| 23/11/2015 | 02 | Ajout de la partie « Détails de la partie WS » |
|  |  |  |

Sommaire

[Description 3](#_Toc435703367)

[Architecture générale 3](#_Toc435703368)

[Détails de la partie Web-service 4](#_Toc435703369)

[Architecture 4](#_Toc435703370)

[Méthodes exposées et format des messages 4](#_Toc435703371)

[Gestion du cache 4](#_Toc435703372)

[Logs applicatives 4](#_Toc435703373)

[Logs techniques 4](#_Toc435703374)

[Installation et configuration 5](#_Toc435703375)

[Environnement utilisé 5](#_Toc435703376)

[Description du projet Netbeans 5](#_Toc435703377)

[Librairies tierces 5](#_Toc435703378)

[Tests et mise en œuvre 5](#_Toc435703379)

[Build et déploiement 5](#_Toc435703380)

[Méthodes de tests 5](#_Toc435703381)

[Détails de la partie client 5](#_Toc435703382)

[Architecture 5](#_Toc435703383)

[Méthodes appelées et contexte d’appel 5](#_Toc435703384)

[Logs applicatives 5](#_Toc435703385)

[Logs techniques 5](#_Toc435703386)

[Installation et configuration 5](#_Toc435703387)

[Environnement utilisé 5](#_Toc435703388)

[Description du projet Visual Studio 5](#_Toc435703389)

[Librairies tierces 6](#_Toc435703390)

[Tests et mise en œuvre 6](#_Toc435703391)

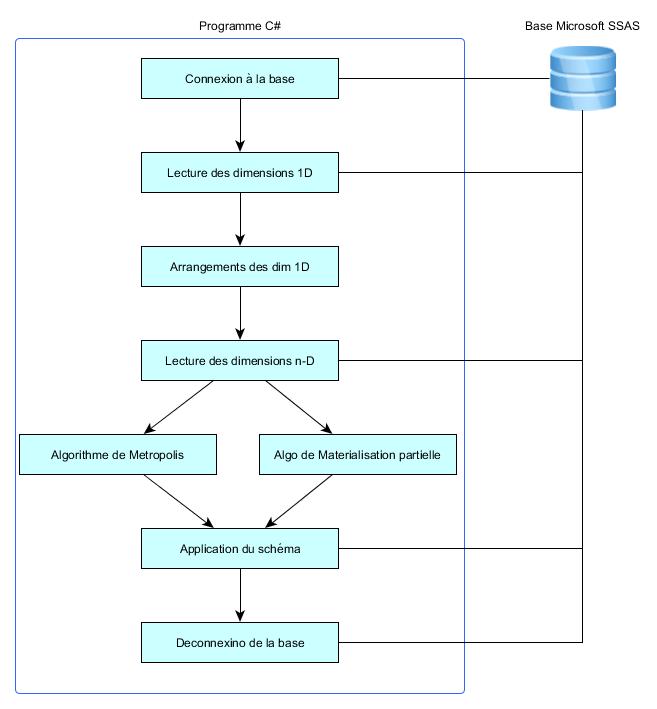
# 

# Description

Réalisé dans le cadre de notre projet d’étude de Master 2 SIID, le projet « **Optimisation des Données de l’Entrepôt (ODE)** » consiste à utiliser les techniques mathématiques vues dans le Master afin de construire l'entrepôt de données de manière optimal, en termes de temps de réponse à l’interrogation des cubes et d’occupation disque :

<https://github.com/M2SIID-ODE/Projet_ODE>

Dans le projet ODE, un seul programme en C# se charge à la fois des interactions avec la base de données SQL Server OLAP (Microsoft SSAS) et des calculs d’optimisations du schéma de base de données :



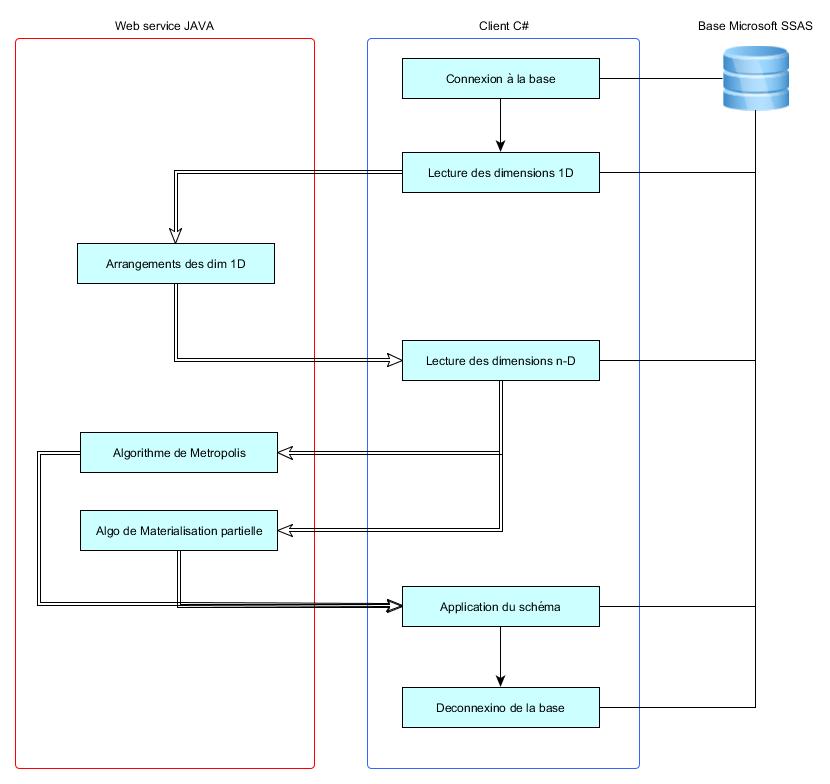
Dans le cadre du module D314 Web-service, nous avons migré les fonctionnalités « cœur de calcul » vers un Web-service Java. Ainsi, les processus consommateurs de ressources machine (Principalement charge CPU) peuvent être déportés sur un serveur pourvu d’une forte puissance de calcul, tandis que la base OLAP reste sur une machine plutôt orientée accès disque et mémoire RAM.

De plus, la portabilité du la JVM Oracle permettra une exploitation sur différents systèmes d’exploitation, par exemple sur la plateforme Amazon Web Service AWS- EC2 (Instances types C3 et C4 orientées calculs intensifs) tandis que la base OLAP reste cantonnée à l’environnement Windows.

# Architecture générale

Quatre méthodes ont été migrées vers un web-service Java :

* **Calcul de schéma optimal par la méthode de Metropolis**
* **Calcul de schéma optimal par la méthode de matérialisation partielle**
* **Arrangement des dimensions du cube relationnel de la base de données OLAP** (2 Méthodes liées)



# Détails de la partie Web-service

## Architecture

### Méthodes exposées

D \*\*\*\*

### Signatures et type

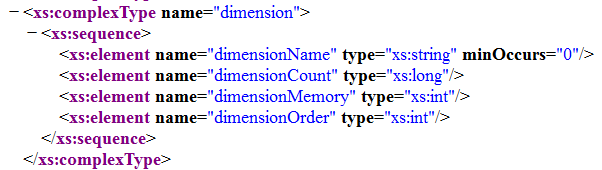
Les « signatures » des méthodes du web-service (Noms des méthodes, paramètres et retours) sont accessibles dans un unique fichier WSDL, auto-généré par le serveur Glassfish :

<http://127.0.0.1:8080/WebServiceOde/OdeServiceImplService?WSDL>

De plus, la description des types et des structures utilisés par le web-service est faite dans le schéma XSD, référencé dans le WSDL :

<http://127.0.0.1:8080/WebServiceOde/OdeServiceImplService?xsd=1>

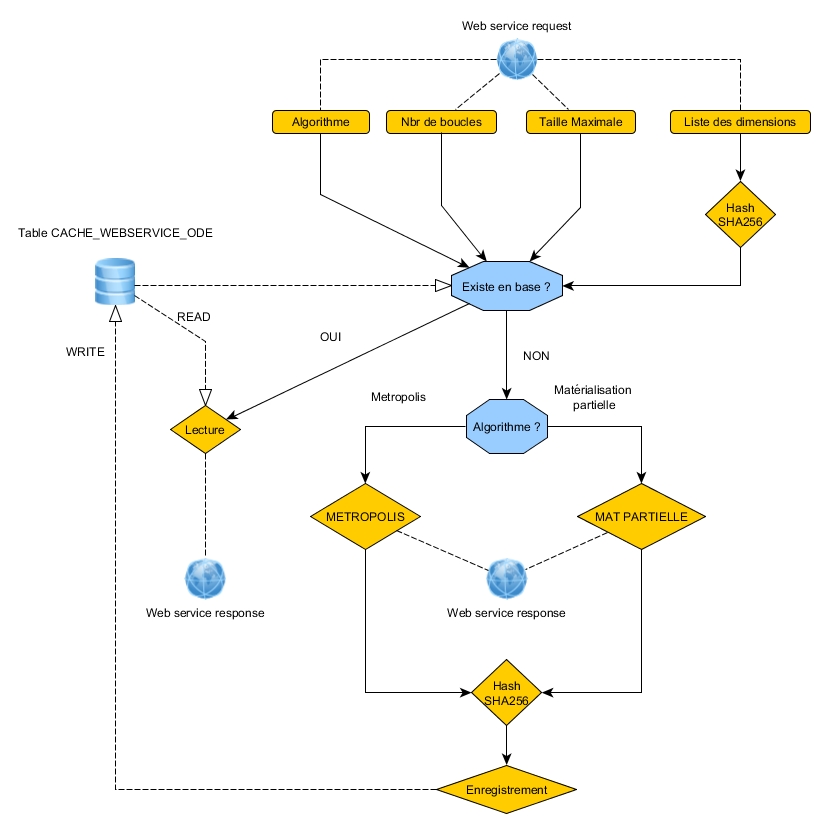
On y voit notamment la définition de la classe « Dimension », traité comme un type complexe, avec les membres PUBLIC définis dans les sources Java de la classe éponyme :



### Gestion du cache

En fonction du nombre de dimensions (Et du nombre de tirages aléatoires pour Metropolis) les calculs peuvent être longs, alors que le résultat attendu est déterministe. Même les tirages au sort sont « constants » entre deux exécutions, car le germe du générateur aléatoire est identique. Pour chaque algorithme : « Même entrées, même sorties »…

Voici la logique du cache que nous gérons dans le web-service :



Une fonction de remise à zéro du cache a été prévue dans le web-service afin de remettre à zéro manuellement le stockage du cache : La méthode ***cleanCache*** réalise un DELETE de l’ensemble de la table CACHE\_WEBSERVICE\_ODE. Dans le client C#, la même méthode peut-être appelée à la demande de l’utilisateur.

L’accès à la base de données SQLite peut se faire à partir d’un client graphique, comme ***SQLiteDatabaseBrowser*** présenté dans le devoir du D314 :

<http://sqlitebrowser.org/>

Pour mieux voir le mécanisme de cache lors de l’exécution du web-service, il faut se connecter au fichier de base de données ***CacheWebServiceOde.db*** déployé sur Glassfish, et non celui du projet NetBeans. Dans notre configuration de déploiement (Serveur Glassfish local sur Windows 7 Pro) ce fichier est stocké dans :

***\Sources\WebService\WebServiceOde\build\web\WEB-INF\classes***

### Logs applicatives

On utilise l’API existante ***log4j 1.2*** (Et non 2.X) afin de générer et gérer facilement les traces du programmes, car les sorties « Console » de NetBeans ne sont pas adaptés au contexte de production.

<https://logging.apache.org/log4j/1.2/>

Le niveau de « finesse » des logs générées et leur présentation peut être changée au travers du fichier ***log4j.properties***  situé :

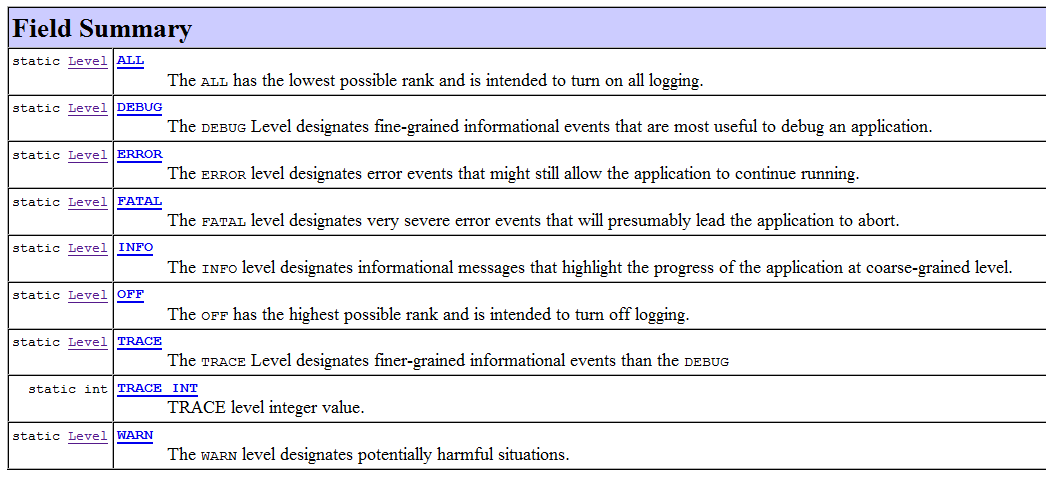
* Dans la section ***Web pages/WEB-INF/Classes*** du projet NetBeans.
* Dans ***\Sources\WebService\WebServiceOde\build\web\WEB-INF\classes*** du projet déployé

Par exemple, nous avons réalisé le débogage avec le niveau DEBUG, dans un fichier de sortie :



Dans l’API log4j, 4 niveaux de finesse sont disponibles, cf. sa Javadoc sur :

<https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html>



En situation de production, le niveau de finesse doit être augmenté à ERROR ou WARN pour ne pas surcharger le serveur (Même si la taille des logs applicatives est réduite à 5 Mo, les logs au niveau DEBUG vont nécessiter des ressources I/O disques importantes, au détriment des performances du Web-service)

Le fichier de logs applicatives est actuellement généré à l’emplacement définit par le paramètre suivant du ***log4j.properties*** :



***NOTE****: Le processus de Build & Deploy décrit ci-dessous va écraser le fichier existant par celui du projet NetBeans.*

### Logs techniques

Indépendamment des logs applicatives, les logs techniques du serveur Glassfish contiennent tous les évènements du serveur (Lancement / Arrêt du serveur, Déploiement / Retraits de .WAR) ainsi que les logs complètes en cas d’exception Java. Les logs applicatives y sont également retranscrits, mais noyées sous un flot d’informations techniques, ce qui rend leur utilisation délicate.

Dans notre configuration de déploiement (Serveur Glassfish local sur Windows 7 Pro) ces logs techniques sont stockées dans :

C:\Users\olivier.essner\AppData\Roaming\NetBeans\8.0.2\config\GF\_4.1\domain1\logs\

***Note*** *: Les fichiers cachés de Windows doivent être affichés.*

## Installation et configuration

### Environnement utilisé

* **OS** : Microsoft Windows 7 Pro 64 bits FR
* **IDE** : Oracle NetBeans IDE 8.0.2 for Java EE
* **JDK** : Oracle JDK 1.8.0 u 60
* **Serveur applicatif** : Oracle Glassfish 4.1 (Fourni avec NetBeans)

### Description du projet NetBeans

* **Catégorie de projet** : Java Web
* **Projet** : Web application
* **Nom du projet** : OdeWebService
* Pas de répertoire dédié aux librairies
* **Serveur** : Glassfish Server 4.1
* **Java EE version** : Java EE 7 Web
* Pas de framework

### Librairies tierces

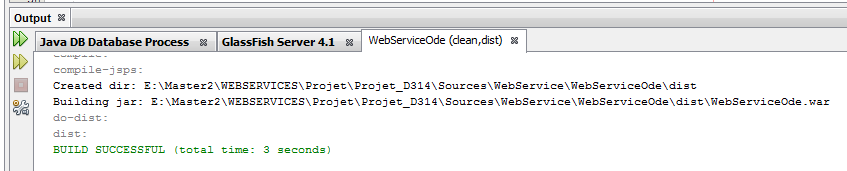
* **Log4j** < log4j-1.2.17> : Pour les logs applicatifs.
* **SqLite** < sqlite-jdbc-3.8.11.2> : Base de données embarquée.

## Tests et mise en œuvre

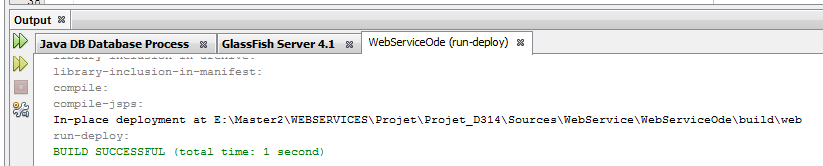
### Build et déploiement

Lancer NetBeans, puis ouvrir le projet ***\Sources\WebService\WebServiceOde***

Sur l’explorateur de solution de NetBeans, faire un clic-droit sur la racine ***WebServiceOde*** puis choisir « Clean & Build ». La fenêtre « Output » de NetBeans doit signaler un succès :



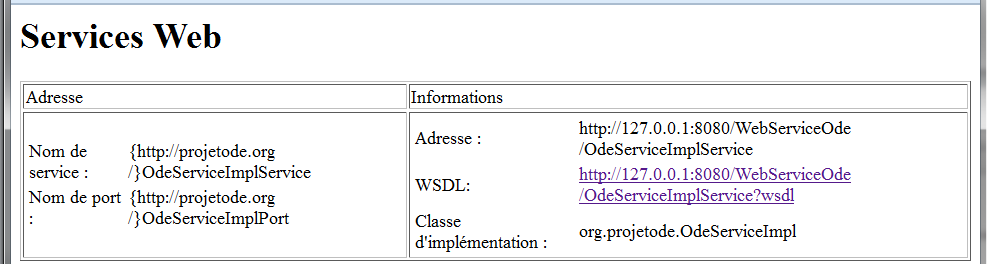
Sur l’explorateur de solution de NetBeans, faire un clic-droit sur la racine ***WebServiceOde*** puis choisir « Deploy ». La fenêtre « Output » de NetBeans doit signaler un succès, avec démarrage du serveur Glassfish si c’est le premier déploiement depuis le lancement de NetBeans :



On peut s’assurer du bon déploiement du Web-service en accédant à la page HTML auto-générée par Glassfish pour décrire les Web-service qui sont déployés et démarrés :

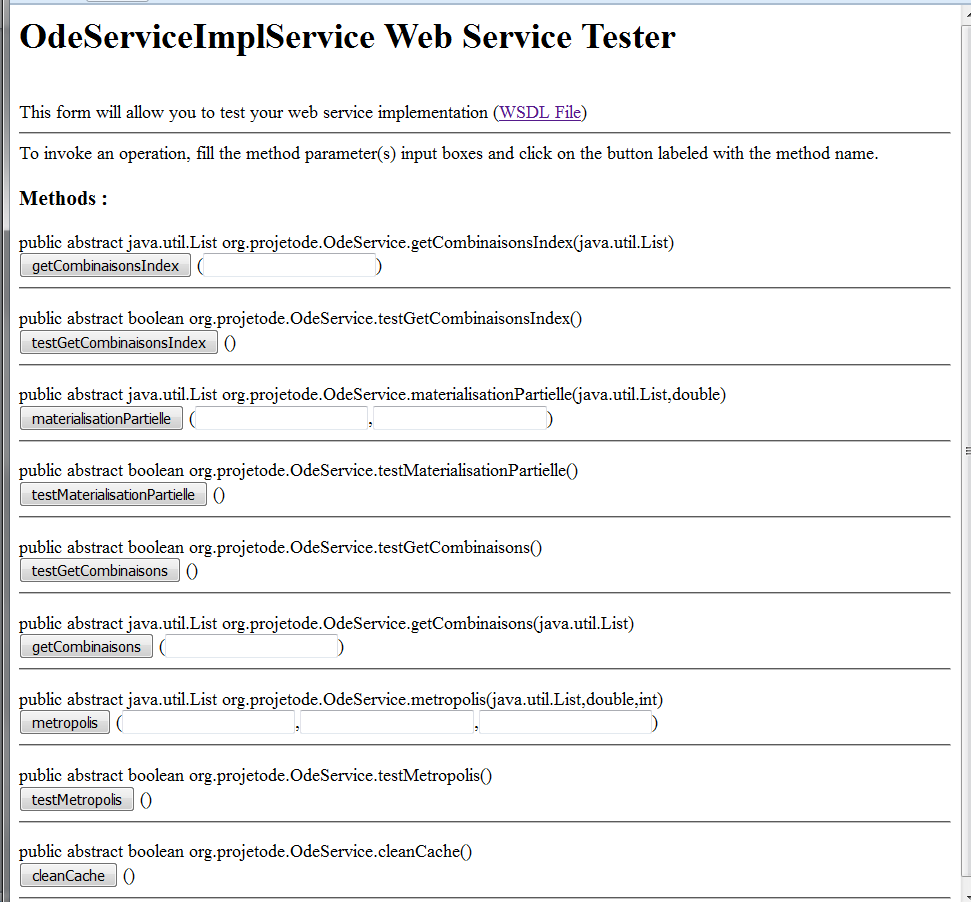
Résumé des web-services déployés sur le serveur applicatif :

<http://127.0.0.1:8080/WebServiceOde/OdeServiceImplService>



Liste détaillée des méthodes exposées sur :

<http://127.0.0.1:8080/WebServiceOde/OdeServiceImplService?tester>



En cas de problème (Page inaccessible ou service non-déployé) consulter les logs techniques, cf. Section « Logs techniques » plus haut.

### Méthodes de tests

Chaque méthode du Web-service (A l’exception du nettoyage de cache) dispose d’une fonction de test, qui simule la partie cliente : Les méthodes du Web-services sont appelées avec des valeurs de test prédéfinies.

|  |  |
| --- | --- |
| Méthode exposée | Méthode de test associée |
| getCombinaisonsIndex(java.util.List) | testGetCombinaisonsIndex() |
| materialisationPartielle(java.util.List,double) | testMaterialisationPartielle() |
| getCombinaisons(java.util.List) | testGetCombinaisons() |
| metropolis(java.util.List,double,int) | testMetropolis() |

Ces méthodes sont utiles pour tester le fonctionnement du web-service en l’absence de client, à partir de la seule page HTML de test générée par Glassfish sur :

<http://127.0.0.1:8080/WebServiceOde/OdeServiceImplService?tester>

Ces méthodes de test n’ont pas d’argument, car tout est prédéfini, et ne retournent qu’un booléen : 1 si tout s’est bien passé, 0 sinon. Le débogage du Web-service se fait alors au travers de ses logs applicatifs, cf. Section « Logs applicatives » plus haut.

# Détails de la partie client

## Architecture

### Méthodes appelées et contexte d’appel

D \*\*\*\*\*

### Logs applicatives

\*\*\*\*\*

### Logs techniques

\*\*\*\*\*

## Installation et configuration

### Environnement utilisé

* **OS** : Microsoft Windows \*\*\*\*\*
* **IDE** : Visual Studio 2015 Community Edition
* **Framework .NET** : \*\*\*\*\*

### Description du projet Visual Studio

* **Catégorie de projet** : \*\*\*\*\*
* **\*\*\*\*\***

### Librairies tierces

* \*\*\*\*\*
* \*\*\*\*\*

## Tests et mise en œuvre

D \*\*\*\*\*