

Université Pierre et Marie Curie



Paris IV

Environnements continentaux-hydrosciences

Géosciences

Sous la direction du Dr François Bouillé, UPMC

Rapport de stage de fin d'étude - Master 2 IASIG 2013-2014

**SYSTEME D'INFORMATION DU CONSEIL
GENERAL DES LANDES - MISE EN PLACE
D'UNE STRATEGIE D'HISTORISATION ET
D'UNE PASSERELLE UTILISATEUR**

Étudiant	Pierre Foicik
Cursus:	Master Professionel IASIG

Mont-de-Marsan, le 27 Novembre 2014

FRONTISPICE

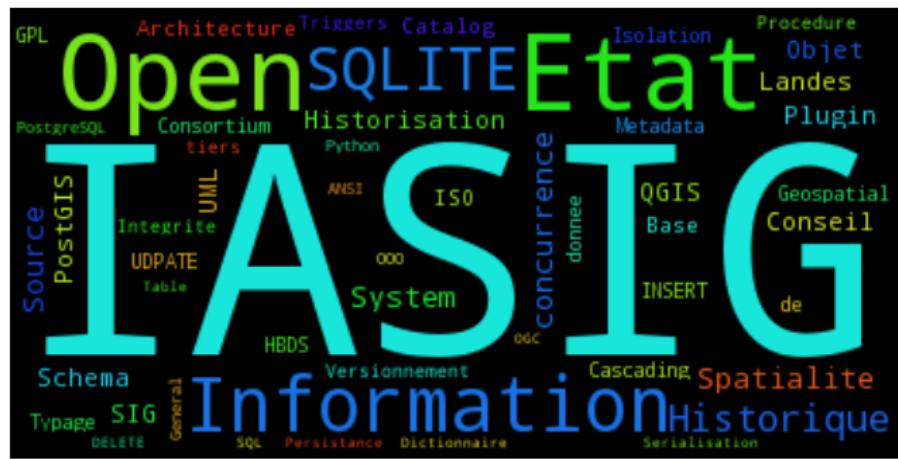


Table des Matières

A. INTRODUCTION.....	8
I. CONTEXTE D'ACCUEIL	9
B. SOLUTION TECHNIQUE ET ARCHITECTURE.....	10
I. DÉPLOIEMENT ET FONCTIONNALITÉS	10
II. CONCEPT ARCHITECTURAUX GÉNÉRAUX	12
a. <i>Côté serveur: Architecture centrée sur les données.....</i>	13
b. <i>Côté client : Architecture multi-couches (n-tiers) côté client.....</i>	14
i. La couche de présentation (« Presentation Layer »):.....	14
ii. La couche logique (« Business logic»):.....	15
iii. La couche de persistance des données:.....	15
III. IMPLÉMENTATION	17
a. <i>Architecture d'un Plugin QGIS.....</i>	17
b. <i>Structuration du Plugin QGIS « sig40 ».....</i>	19
c. <i>fichier de configuration.....</i>	19
d. <i>Paquetages principaux.....</i>	20
i. Le paquetage dbtools.OgeoDB.....	20
ii. Le paquetage dbtools.OSqlite.....	21
iii. Le paquetage dbtools.URI_Builder.....	22
iv. Le paquetage reSyncDB.....	22
v. Le paquetage action.ProjetAction.....	25
e. <i>Persistante du projet et référencement des objets.....</i>	27
IV. EXÉCUTION	30
C. SERVEUR DE BASE DE DONNEE POSTGRESQL/POSTGIS.....	36
I. FONDAMENTAUX ARCHITECTURAUX	37
II. ADMINISTRATION DU SERVEUR POSTGRESQL	37
a. <i>Authentification sur le réseau du Département des Landes.....</i>	37
b. <i>Méthode d'authentification sur le serveur PostgreSQL.....</i>	37
c. <i>Définition des rôles et privilèges.....</i>	38
III. L'INTERFACE CLIENT	38
a. <i>La librairie libpq:.....</i>	38
b. <i>Le Schéma d'information (introduction au catalogue système).....</i>	39
D. BASE DE DONNEE SQLITE/SPATIALITE.....	39
I. FONDAMENTAUX ARCHITECTURAUX	39
II. AUTHENTIFICATION	40
III. INTERFACE	40
E. MISE EN PLACE D'UN SYSTEME D'HISTORISATION.....	41
I. CONCEPTUALISATION -ARCHITECTURE DE LA BASE DE DONNÉES ET INTERACTIONS UTILISATEURS	41
II. POURQUOI DÉVELOPPER UNE STRATÉGIE D'HISTORISATION?	41
III. MISE EN PLACE D'UN SYSTÈME D'HISTORISATION – STRUCTURATION DE LA BDD ET STRATÉGIE SERVEUR	43
a. <i>Introduction- Modèle simplifié du système d'historisation.....</i>	43
b. <i>Modèle de données logique.....</i>	43
c. <i>Profil de Modèle de données UML (« Data Model Profile »).....</i>	45
d. <i>Cycle de vie d'un objet – Création et modification.....</i>	46
e. <i>Cycle de vie d'un objet – Suppression.....</i>	48

<i>f. Implantation des fonctionnalités dans les tables de la BD SIG40.....</i>	48
IV. MISE EN PLACE D'UN SYSTÈME D'HISTORISATION — ACCÈS ET GESTION DE LA CONCURRENCE	
MULTI-UTILISATEUR	52
<i>a. Conceptualisation.....</i>	52
<i>b. Isolation des transactions – Politique du serveur.....</i>	52
<i>c. Isolation des transactions – implémentation.....</i>	52
<i>d. Protection des champs et restriction des possibilités d'édition.....</i>	54
<i>e. Mécanisme d'édition concurrentielle des données partagées.....</i>	54
i. Extraction (ie : Migration PostgreSQL vers SQLITE local).....	55
ii. Édition.....	60
iii. Réconciliation.....	60
V. DÉVELOPPEMENTS	61
F. MANUEL D'UTILISATION DU PLUGIN SIG40.....	62
I. INSTALLATION	62
II. CONTEXTE	62
III. SÉQUENCE D'UTILISATION TYPIQUE	63
IV. DESCRIPTION DES FONCTIONNALITÉS	65
<i>a. Création d'un projet :.....</i>	65
i. Connexion à une base de donnée.....	65
ii. Génération des fichiers du projet.....	67
iii. Sélection des données sur le serveur.....	68
iv. Finalisation de l'extraction.....	68
<i>b. Session d'édition des données.....</i>	70
<i>c. Fin de session d'édition des données.....</i>	72
<i>d. Reprise d'un projet non complété.....</i>	72
<i>e. Réconciliation d'un projet avec le serveur.....</i>	72
G. CONCLUSION.....	73
H. ANNEXES.....	74
ANNEXE I: PROCESSUS D'HISTORISATION DE LA BDUNI	74
ANNEXE II: GRAPHE D'APPEL GÉNÉRAL DU PROJET	75
ANNEXE III: GRAPHE D'APPEL GÉNÉRAL DU PROJET	75
ANNEXE III: DIAGRAMME DE CLASSE UML DU PROJET	77
	77
I. BIBLIOGRAPHIE.....	80

Liste des Figures

Illustration 1: Diagramme de cas d'utilisation - Fonctionnement du middleware.....	11
Illustration 2: Diagramme de déploiement de la passerelle SIG40.....	12
Illustration 3: Projet SIG40 - Modèle simplifié d'architecture en couche d'abstraction concentrique.....	13
Illustration 4: Architecture minimale d'un plugin QGIS.....	17
Illustration 5: Arborescence du plugin sig40 et description de l'objet des fichiers et répertoires.....	18
Illustration 6: Graphe d'appel simplifié du projet.....	18
Illustration 7: Diagramme de classe UML du paquetage OGeoDB.....	21
Illustration 8: Diagramme de classe UML du paquetage OSqlite.....	22
Illustration 9: Diagramme de classe URI_Builder (avec héritage qgis.core explicité).....	22
Illustration 10: Diagramme de classe UML du paquetage reSyncDB.....	23
Illustration 11: Graphe d'appel des paquetages développés pour la synchronisation d'un projet.....	24
Illustration 12: Diagramme de classe UML du Paquetage ProjetAction.....	26
Illustration 13: Dictionnaire python et méthodes associées.....	27
Illustration 14: Description de la structure de conservation des métadonnées des tables de la BD sig40.....	28
Illustration 15: Diagramme de classe OProject (constitué uniquement de classe d'objet sérialisable). Une méthode de réinitialisation est omise sur le schéma.....	29
Illustration 16: Diagramme de classe HBDS du projet sig40.....	29
Illustration 17: Copie locale des données serveur.....	30
Illustration 18: Graphe d'enchaînement des modules et fonctions lors de la création d'un projet sig40 (la coloration (couleurs froides vers couleur chaudes, représentent le temps d'exécution croissant des fonctions. Les nombres indiqués sur les flèches représentant l'enchaînement indique le nombre d'appel à la fonction pointée).....	31
Illustration 19: Graphe d'enchaînement - Synchronisation après modification utilisateur (transaction de type UPDATE & DELETE).....	32
Illustration 20: Graphe d'enchaînement - Tentative de synchronisation d'un projet déjà synchronisé.....	33
Illustration 21: Graphe d'enchaînement - Abandon de synchronisation suite à la détection d'un conflit non résolu.....	34
Illustration 22: Graphe d'enchaînement - Synchronisation d'un projet sans aucune modification.....	35
Illustration 23: Le logo PostgreSQL.....	36
Illustration 24: Le logo SQLITE.....	39
Illustration 25: Classe d'objets (pseudo-HBDS) portant les métadonnées du mécanisme historisation.....	44
Illustration 26: Diagramme de classe (UML) prototype de la base de données SIG40 (les opérations d'affectation et de lecture sur les attributs sont assumée et ne sont pas représentées).....	45
Illustration 27: UML Data Model Profile - Conceptualisation de la Base de donnée SIG40.....	46
Illustration 28: Modélisation HBDS de la Base de donnée SIG 40 et de son système d'historisation.....	47
Illustration 29: Implantation et corps de la procédure automatisée (trigger) déclenché lors d'une insertion sur une table de la BDD.....	49
Illustration 30: Implantation et corps de la procédure automatisée (trigger) déclenché lors d'une mise à jour effectué sur un objet appartenant à une table de la BDD.....	49

Illustration 31: Diagramme d'activité schématique du déclenchements des « Triggers » PostgreSQL spécifique à l'historisation lors d'une transaction.....	51
Illustration 32: Mise en concurrence et édition sauvage de l'objet 1 par un utilisateur. L'utilisateur 2 a demandé une vérification de l'état de synchronisation avant l'édition de l'objet par l'utilisateur 1 entraînant une désynchronisation non détectée.....	53
Illustration 33: Code Pseudo-SQL introduisant une vérification de la synchronisation comme condition requise pour effectuer une mise à jour (implanté en utilisant une fonction de bloc de code anonyme Postgresql).....	54
Illustration 34: Représentation des cascades de Triggers lors d'édition de la base SQLITE.....	56
Illustration 35: Implantation et corps des triggers SQLITE exécutés par des transaction UPDATE.....	57
Illustration 36: Implantation et corps des triggers SQLITE exécutés par les procédures INSERT et DELETE.....	57
Illustration 37: Implantation et corps des triggers SQLITE spécifiques au référencement des objets au sein de zones de réconciliation, exécutés par les procédures UPDATE.....	58
Illustration 38: Diagramme d'activité - Référencement des objets lors de l'ajout d'une Zone de Réconciliation (ZR).....	58
Illustration 39: Modélisation HBDS du mécanisme de suivi des modifications utilisateurs.....	59
Illustration 40: Plugin SIG40 – Bouttons d'accès aux outils (a : ouverture d'un projet ; b : reprise d'un projet ; c : création d'une zone de réconciliation ; d : destruction de zones de réconciliation ; e : réconciliation avec le serveur de donnée distant).....	65
Illustration 41: Fenêtre de connexion - Connexion serveur validée.....	65
Illustration 42: Fenêtre de connexion - Connexion serveur impossible.....	66
Illustration 43: Fenêtre de connexion - Connexion base de donnée validée.....	66
Illustration 44: Message de connexion validation des paramètres de connexion.....	67
Illustration 45: Fenêtre principale du projet.....	67
Illustration 46: Dialogue de choix d'un fond de couche pour la sélection de zone cliente.....	68
Illustration 47: Fenêtre d'assistance à a sélection d'un extrait de la base de donnée (la sélection géographique) et sélection des couches à importer.....	69
Illustration 48: Exemple de projet chargé et prêt à l'édition.....	70
Illustration 49: Boîte de dialogue de création de zone de réconciliation.....	71
Illustration 50: Dialogue permettant la suppression de zones de réconciliation.....	71
Illustration 51: Graphe d'appel général du projet sig40.....	76
Illustration 52: Diagramme de classe UML du projet sig40 (incluant uniquement les paquetages développés, les bibliothèques importées ne sont pas représentées).....	78

Liste des Tables

Tableau 1: Création d'un rôle "groupe" et héritage sur utilisateur.....	38
---	----

Liste des Abréviations, Symboles et Anglicismes

IGN	Institut Géographique National
HBDS	Hypergraph-Based Data Structure
UML	Unified Modelling Language
DMP	Data Model Profile (UML)
SGBD	Système de Gestion de Base de Donnée
BD	Base de Donnée
Plugin	Extension logicielle
OGC	Open Geospatial Consortium
ISO	International Standards Organization
SIG	Système d'Information Géographique

A.INTRODUCTION

Ce document constitue le rapport de stage de fin d'études de la formation Master 2 « Informatique Appliquée au Systèmes d'Information Géographique (*LASIG*) » délivrée par l'Université Pierre et Marie Curie (Paris VI). Il ne constitue pas un rapport de spécifications techniques pour le développement du système d'information géographique (SIG) du Conseil Général des Landes (CG). Il s'attelle néanmoins à présenter et à formaliser certains aspect de la structuration de la base de donnée supportant le SIG et propose une architecture logicielle possible pour le développement d'une passerelle utilisateur.

Ce rapport présente donc les concepts et développements réalisés dans le cadre de l'implémentation d'une stratégie de versionnement (historisation) des données géographiques du Conseil Général. Il est étendu par la description des spécifications et l'implémentation d'une passerelle utilisateur pour permettre le déploiement de l'utilisation du SIG parmi les agents des différents services du CG. L'objectif principal est de permettre le partage, l'édition et la création de contenu sur le SIG du département, de manière efficace et sécurisé, sous la responsabilité d'un administrateur unique.

Au cours de 4 mois de stage de fin d'études réalisés au cours des mois de juin à novembre 2014, la structuration de la base de donnée a été testée et validée en collaboration avec l'ingénieur principal responsable du SIG Clotilde Mohsen. L'outil utilisateur a été prototypé et soumis à une phase de pré-test avant un premier déploiement sur quelques postes client.

L'implantation de cette stratégie a été réalisée intégralement de par l'utilisation d'outils logiciels libre (sous licence GPL 2.0). L'ensemble des développements présentés ici tombent de fait sous le coup de cette licence. La base de données est montée, structurée et servie par un serveur PostgreSQL (dans sa version 9.3 à jour au mois d'Octobre 2014). La passerelle utilisateur est constitué autour du logiciel libre SIG Qgis (version 2.2 minimum et 2.4 recommandée) sous la forme d'une extension « plugin » permettant :

1. L'extraction ciblée de donnée

2. L'édition complète et sécurisée
3. Synchronisation des modifications avec le serveur distant.
4. Pose les bases requises pour le développement d'une gestion avancée des synchronisation

Après avoir présenté les aspects techniques et concepts sous-jacent à cette réalisation (architecture et outils), le présent document s'attellera à détailler la mise en œuvre d'une stratégie de versionnement (historisation) et l'implémentation d'un outil d'édition dédié, réalisé sous la forme d'une extension du logiciel *QGIS*. Enfin, un manuel utilisateur est fourni traitant des principales fonctionnalités requises à l'utilisation par les agents du Conseil Général.

I. Contexte d'accueil

J'ai été accueilli au sein du département informatique du Conseil Général, encadré par Mme Clothilde Mohsen (ingénieur principal responsable du SIG), sous la responsabilité du directeur du service M. Jean-Michel Guillou, chef du service des systèmes informatiques et de l'administration électronique.

Mon stage s'inscrit dans l'application de la directive INSPIRE¹ qui défini un cadre pour assurer la diffusion et le partage de l'information géographique environnementale au niveau européen.

Le choix de conserver la maîtrise de la compétence SIG au sein du Conseil Général a été acté. Un plan d'action triennal initié et les choix techniques arrêtés. Ma participation à ce projet a consisté en la définition et l'implémentation d'une stratégie de versionnement des bases de données géographiques supportant des mécanismes d'accès sécurisés et le partage des données dans le cadre de leur édition.

¹ <http://inspire.ign.fr/directive/presentation>

B.SOLUTION TECHNIQUE ET ARCHITECTURE

I. Déploiement et fonctionnalités

L'extension développée permet de supporter l'édition des données du Système d'information Géographique (SIG) du Département des Landes en gérant de manière transparente la concurrence multi-utilisateurs et la protection des données (historisation et cloisonnement des compétences). Le diagramme de cas d'utilisation suivant (Illustration 1) synthétise les opérations supportées par l'application.

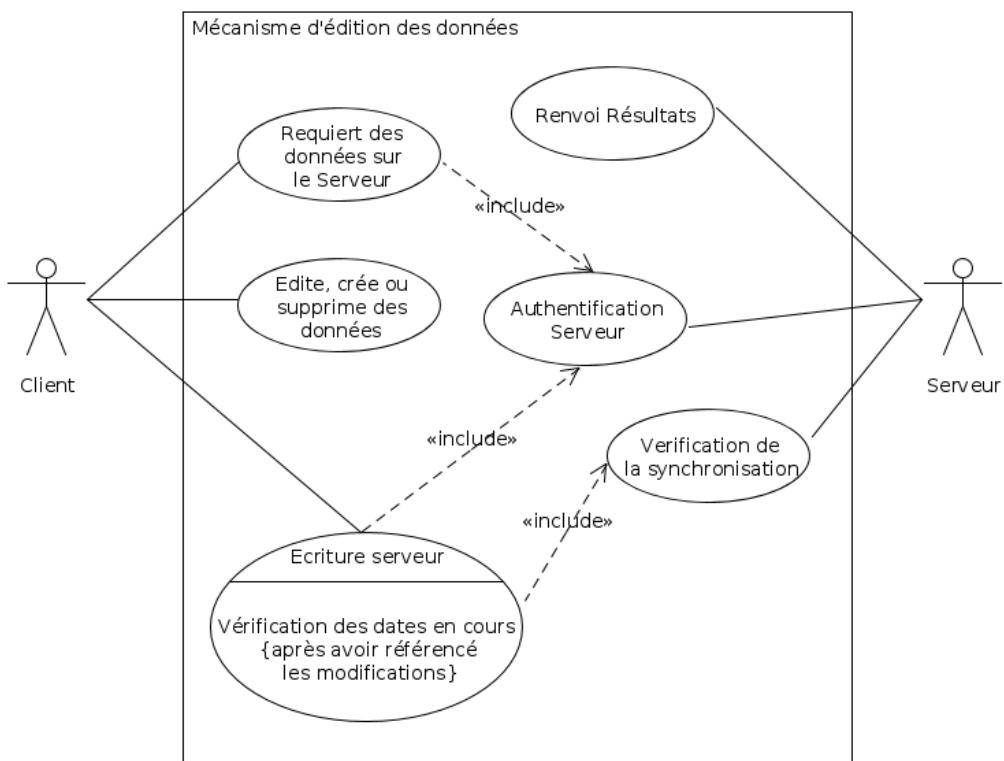


Illustration 1: Diagramme de cas d'utilisation - Fonctionnement du middleware

Le projet est implanté sur les différents composants du SIG tel que présenté par le diagramme ci-dessous (Illustration 2). Le serveur SIG40 sert une base de données *PostgreSQL* à des clients exécutant l'application *QGIS* (dans sa version 2.2 au minimum). L'application *QGIS* est étendue par le code source du Plugin ***sig40*** gérant une base de données locale *SQLITE*. Le code source de cette extension est écrit en ***Python 2.7*** et est complètement intégré au sein de l'interface du logiciel *QGIS*. Il fait appel uniquement aux bibliothèques standard Python 2.7 (*pyscopg2*², *sqlite3*³,..) aucune configuration n'est donc requise.

² <http://pythonhosted.org//psycopg2/>

³ <https://docs.python.org/2/library/sqlite3.html>

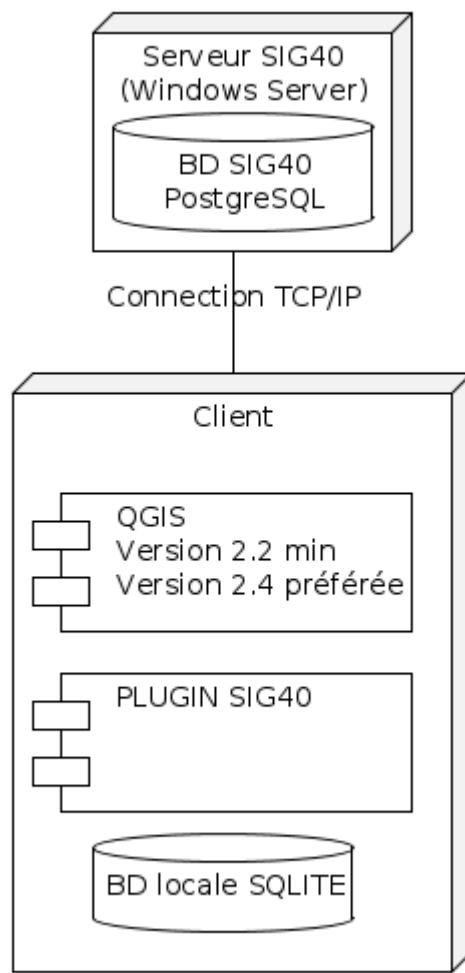


Illustration 2: Diagramme de déploiement de la passerelle SIG40

L'environnement d exécution requis ne nécessite donc aucune maintenance, les paquetages requis (machine virtuelle et bibliothèque python 2.7) étant installés de pair avec l'application *QGIS* à partir d'un exécutable unique.

II. Concept architecturaux généraux

La mise en œuvre de ce projet a nécessité l'intervention sur les composantes serveur ("back-end") et cliente ("front-end") pour déployer une véritable stratégie de versionnement. Ceci a été rendu possible par l'intervention dans une phase précoce du développement du SGBD du Département (structuration encore incomplète).

Les solutions back-end proposées sont basées sur des algorithmes de traitements faisant usage des outils avancés fournis par *PostgreSQL*. Il est néanmoins intéressant de

noter que les concepts sur lesquels sont basés le système de versionnement sont commun à tous les SGBD du marché, le concept et la solution proposés sont donc théoriquement portables vers un service de base de données différent.

Le projet fait appel à deux SGBD aux usages, cibles et niveaux de complexité très différents :

- base de données PostgreSQL exécuté sur un serveur Postgres nécessitant une gestion par administrateur, incluant la gestion personnalisée des priviléges utilisateur et permettant une gestion de la concurrence avancée.
- Fichier de base de données *SQLITE* – SGBD « minimaliste », « indépendant de toute plate-forme » ne nécessitant aucune administration ni serveur dédié. Il s'agit de fichiers se comportant comme un SGBD à part entière.

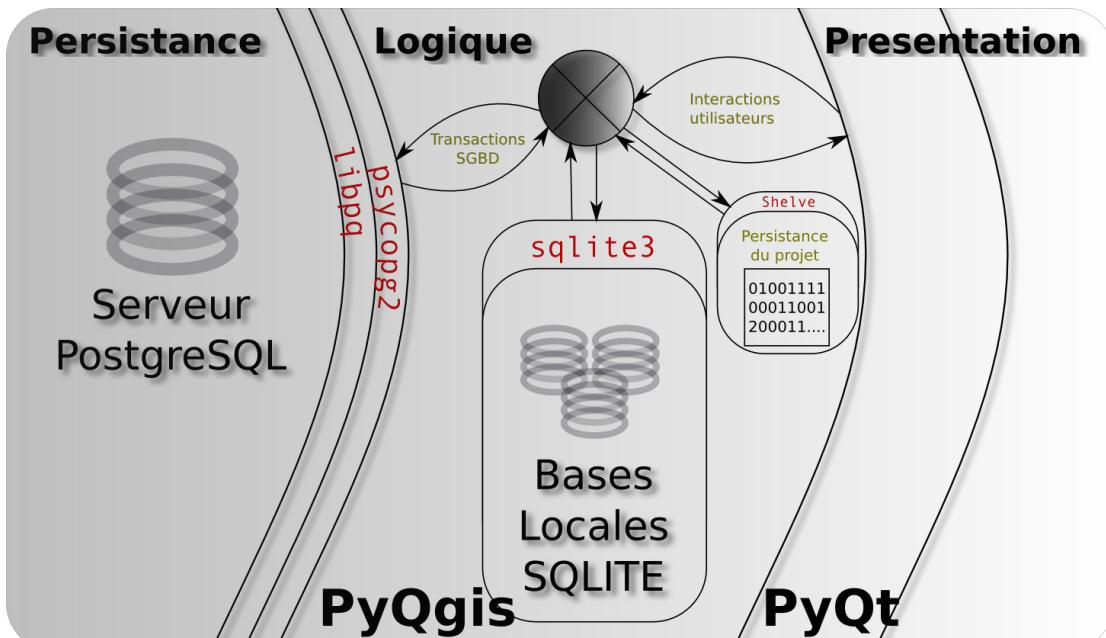


Illustration 3: Projet SIG40 - Modèle simplifié d'architecture en couche d'abstraction concentrique

a. Côté serveur: Architecture centrée sur les données

L'architecture globale du projet comporte un composant central de type système de gestion de base de donnée (SGBD). Dans cette architecture, des composants périphériques, encore appelés *clients*, interrogent et sont servis par le composant central (serveur de données). Le versionnement des données manipulées par les utilisateurs est réalisé par des procédures portées par la base de données elle-même. Il s'agit

d'une série de procédures automatisées développées conjointement avec la structuration des tables de la base.

L'utilisation de procédures portées et exécutées par le serveur de base de données constraint fortement le développement de la logique applicative front-end en donnant un cadre plus strict. Cette approche a également pour avantage de mettre la priorité sur la structuration de la base de donnée en amont de tout développement.

b. Côté client : Architecture multi-couches (*n-tiers*) côté client

Le composant périphérique repose sur une architecture 3-tiers classique, symbolisé en Illustration 3 ci-avant. Les champs d'interférences des couches d'applications logiques sont dédiés à :

- l'indexation de données
- au déploiement dynamique (implantation) des mécanismes de versionnement au sein de couches semi-persistantes
- à la génération et à la persistance d'un projet de versionnement (extraction/synchronisation)
- au pré-traitement des couches de rendu.

i. La couche de présentation (« *Presentation Layer* »):

La couche de présentation de ce projet constitue une extension de l'application libre *QGIS*, est donc entièrement montée sur l'interface préexistante du logiciel. L'intégralité des interactions sont programmées au travers de boîtes de dialogue et/ou d'outils développés en PyQT au travers d'un système de gestion de signaux utilisateurs connectés sur les fiches programmatiques de la couche logique. Les outils préexistants fournis par l'application sont également réutilisables. Cette approche permet ainsi un déploiement robuste et rapide sur les postes clients en tirant profit des structures et fonctionnalités déjà mises en place. D'un point de vue structure du projet, les *packages* (voir Illustration 5) correspondant à l'interface utilisateur sont bien séparés (voir paragraphe B.III.b), reconnaissables et éditables indépendamment des algorithmes de traitements.

ii. **La couche logique (« Business logic»):**

Le cœur de l'application (le plugin sig40) implémentant les algorithmes et structures créés pour le projet font partie de cet espace de nom. L'intégralité des fonctionnalités et structures proposées par le logiciel *QGIS* y sont accessibles (incluant donc les *packages qgis.core et qgis.gui*). Les fonctionnalités suivantes sont supportées:

- Le montage et la structuration du SGBD local, incluant l'insertion et l'activation de procédures automatisées de suivi des modifications utilisateurs locales (triggers *SQLITE*)
- la constitution des structures de données Ad-hoc (référencement des métadonnées du projet)
- La montage d'une abstraction suffisante pour permettre une utilisation transparente du SGBD sous-jacent à l'utilisateur.
- Le contrôle et la validation du déroulement des processus de synchronisation

Au sein de cette couche logique il a été implanté une **persistence « à l'échelle de la durée de vie du projet »**. Ceci pour répondre aux objectifs d'édition en sécurité de données sensibles et partagées. Ainsi une copie locale des données est réalisée sur la machine cliente et la persistence de certains objets du projet (**sérialisation d'une classe encapsulant les références aux métadonnées du projet**) est établie pour créer les conditions requises à l'historisation des actions utilisateur. La persistence locale des données est réalisée au travers de la création de bases de données *SQLITE* locales et la persistence du projet est établie en générant une pseudo-base de donnée objet (au travers des fonctionnalités de sérialisation Python « *Shelve* ») permettant la conservation de certains types d'objet intrinsèque au langage python.

Les spécifications de cette couche répondent à la structuration HBDS du logiciel présenté en Illustration 16 .

iii. **La couche de persistance des données:**

Comme énoncé ci-dessus, une des clefs de l'architecture de ce projet réside dans l'utilisation de deux couches de persistance pour donner un maximum de souplesse à l'utilisation. Ainsi il est établi une couche de persistance « locale » construite et synchronisable avec une couche de persistance distribuée (un serveur de base de données accessible à tous les acteurs du conseil général). Cette **couche de persistance « objectif »** partagée entre utilisateur est installée sur un serveur PostgreSQL distant. L'intégralité

des transactions l'affectant étant nécessairement traitées pour validation par le moteur de référencement de la couche logique. Les transactions avec ces couches de persistances sont réalisées selon les spécifications du langage *SQL* au travers de bibliothèques C (*libpq* et *sqlite*) interfacée par les bibliothèques python correspondante (*psycopg2* et *sqlite3*). La couche logique est complètement tributaire de la structuration des tables de la bases de données cible, il s'agit là d'un point d'amélioration de l'architecture à prévoir. Celui-ci est rappelé en tant que tel comme étant un développement important requis pour assurer la robustesse du projet à long terme dans le paragraphe dédié à et effet. Les modifications devrait prendre exemple sur le modèle de génération de la couche de persistance locale, qui est intégralement générée par les algorithmes de la couche logique.

III. Implémentation

a. Architecture d'un Plugin QGIS

QGIS est une application écrite en *C++* incorporant plus de 400 classes. Une grande partie de ces classes font l'objet de « Bindings » avec Python au travers de l'utilisation de SIP⁴. L'extension des fonctionnalités *QGIS* est réalisée au travers de l'addition de **codes source Python** interprétés lors de l'exécution du logiciel.

./Plugin	
init.py	-- Script requis contenant une seule méthode qui initialise la classe du plugin
plugin.py	-- L'implémentation du plugin qui gère chargement et execution des fonctionnalités
plugin_dialog.py	-- La/les fenêtres de dialogue(s) principale(s)
ui_plugin.py	-- Description des dialogues (eventuellement compilé par le compilateur interface pyuic4)
resources.py	-- Description des ressources additionnelles (éventuellement complié par le compilateur pyrcc4)
metadata.txt	-- Description du plugin (obligatoire)

Illustration 4: Architecture minimale d'un plugin QGIS

Les plugins *QGIS* sont typiquement empaquetés dans un répertoire unique contenant code source et ressources additionnelles éventuelles (icônes, fichier de configuration,...). Une structure typique minimale doit être respectée pour bénéficier des facilités de déploiement d'une extension *QGIS*. Au démarrage de *QGIS*, les plugins référencés voit la méthode *classfactory* de leur fichier *_init_.py* appelée. Si aucune erreur n'est rencontrée à l'interprétation du code source, la méthode *initGui* du script principal est appelée et le plugin est ajouté aux menus et à la barre d'outils le cas échéant. Toute erreur d'interprétation empêche la présentation du plugin au sein de l'interface du logiciel et le message d'erreur de l'interpréteur est émis à destination de l'utilisateur.

Ce modèle d'architecture s'enrichit de pair avec le niveau de complexité du code. Ainsi, les packages similaires peuvent être regroupés au sein de répertoires communs, les paquetages essentiels étant conservés à la racine du répertoire conteneur du plugin. L'Illustration 5 ci-après présente l'arborescence du plugin retenue pour le projet.

⁴ http://loc8.cc/ppg/py_sip

```

/sig40
    └── /action - Algorithmes
    └── /dbtools - Outils de découverte de la base de données
    └── /ui - Design des dialogues utilisateur
    └── /utils_sig40 - Boîte à outils
    └── /ressources - ressources additionnelles
        └── __init__.py -- Fichier d'initialisation
            sig40.py -- implementation des fonctionnalités
            sig40_dialog.py -- Création de projet
                Initialisation de la fenêtre de dialogue principale
            sig40_Reload.py -- Classe pour rechargement de projet
            sig40_reSync.py -- Classe pour synchronisation locale/serveur
            sig40_ZR_dialog -- Classe pour définition de Zone de réconciliation
                après la création de projet
            Global_mod.py -- Fichier portant les variables d'environnement par default
            Oprojet.py -- Classe d'objet python entièrement
                sérialisable pour la persistance du projet

```

Illustration 5: Arborescence du plugin sig40 et description de l'objet des fichiers et répertoires

L'articulation entre les classes principales du projet (incluses dans les 8 fichiers à la racine du répertoire du projet) est présentée ci dessous. Tous est dirigé vers la constitution de la classe **Oproject**, décrite plus en détail dans les paragraphes suivants.

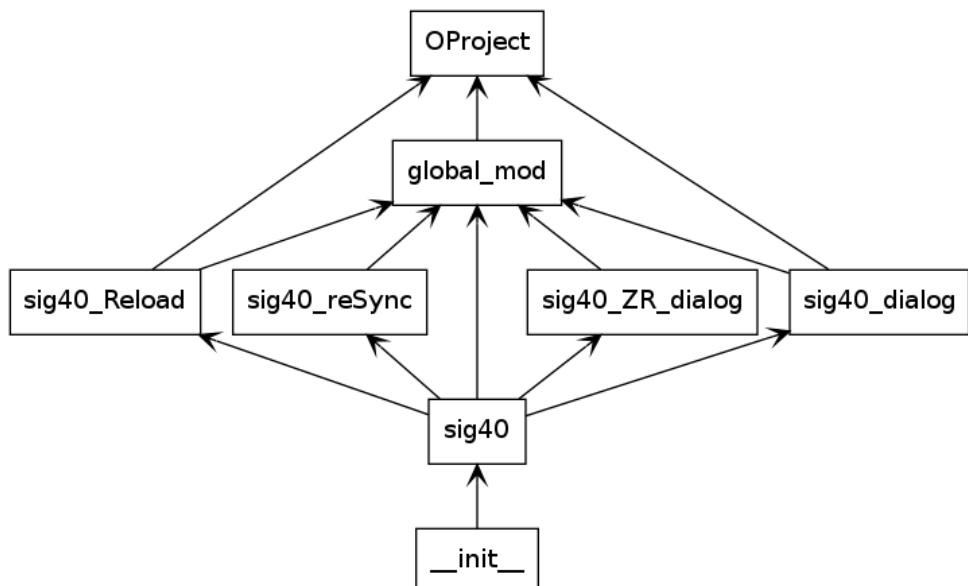


Illustration 6: Graphe d'appel simplifié du projet

b. Structuration du Plugin OGIS « sig40 »

Dans le cadre de ce projet , l'architecture est montée à partir de l'analyse du système présentée selon la nomenclature et les outils proposés par la méthode HBDS (cf Illustration 16 ci-après). Cette représentation non fonctionnelle du plugin en tant que système d'information permet de démarrer le prototypage des objets requis. Elle est enrichie et complétée lors du montage effectif des classes programmatiques (« *feed-back* ») de manière à représenter au mieux les développements réalisés. La structuration est articulée autour d'une classe **SIG40** réalisant l'indexation des métadonnées du projet, incluant :

- *Paramètres utilisateur*
- *Paramètres de connexion*
- *Référencement des tables et de leurs champs (dénomination et typage)*
- *Référencement de l'état de synchronisation du projet avec le serveur*

L'implémentation de ces éléments est présentée dans le paragraphe B.III ci-après.

L'exécution du plugin déclenche un processus de découverte des tables de la base de données et initie le mécanisme de suivi des modifications au travers de structures d'indexation.

c. fichier de configuration

Le fichier **Global_mod.py** contient une classe encapsulant les variables globales et paramètres de l'extension. Il est utilisé en tant que passerelle vers la classe persistante Oproject.

Ses attributs concernent quelques variables par défaut requises pour la tentative de connexion initiale sur le serveur de base de données. En particulier :

Bdbname : Le nom du fichier de base de donnée par défaut (postgres dans le cas d'un serveur PostgreSQL).

HostName : L'adresse du serveur sur le réseau (ou localement). Il peut s'agir d'un alias ou d'une adresse IP classique.

Cette classe contient également les champs de configuration requis pour la protection des champs de l'historisation :

immutable_field : Liste de chaîne de caractères, explicite, indiquant les champs à gérer dans les tables d'attribut

source_donnee_field_config : dictionnaire de chaînes de caractères pour fixer un type prédefini d'un champ des tables de données dans le cas de la gestion de types personnalisés.

*d. **Paquetages principaux***

L'intégralité du projet est écrit en se limitant aux bibliothèques standard *Python 2.7*, *PyQT4* et en faisant un maximum usage de l'API *QGIS*. L'environnement est fourni par défaut lors de l'installation de *QGIS* sur le poste client. Le diagramme de classe global du projet est présenté en Annexes 2 et 3.

i. **Le paquetage dbtools.OgeoDB**

Ce fichier contient les 5 classes gérant les fonctionnalités nécessitant des transactions avec la base de données et permet de construire les interactions entre QGIS et les couches de persistances nécessaires pour construire un projet **sig40**.

1. **OQgis** introduit les méthodes pour construire et gérer les structures QGIS requises pour réaliser l'intégration des données servies par un SGBD au sein d'un projet QGIS (en d'autres termes, réaliser la migration SGBD vers un format vectoriel géré par QGIS)
2. **LocalDB_utils** donne toutes les méthodes requises pour réaliser le montage et les transactions avec les bases de données *SQlite* locales.
3. **PGDB_utils** est une boîte à outils statiques sur une base de donnée *PostgreSQL*
4. **OgeoDB** distribue toutes les méthodes requises pour la découverte des métadonnées d'une base PostgreSQL, une connexion et les outils requis pour le référencement des tables au chargement d'un projet.

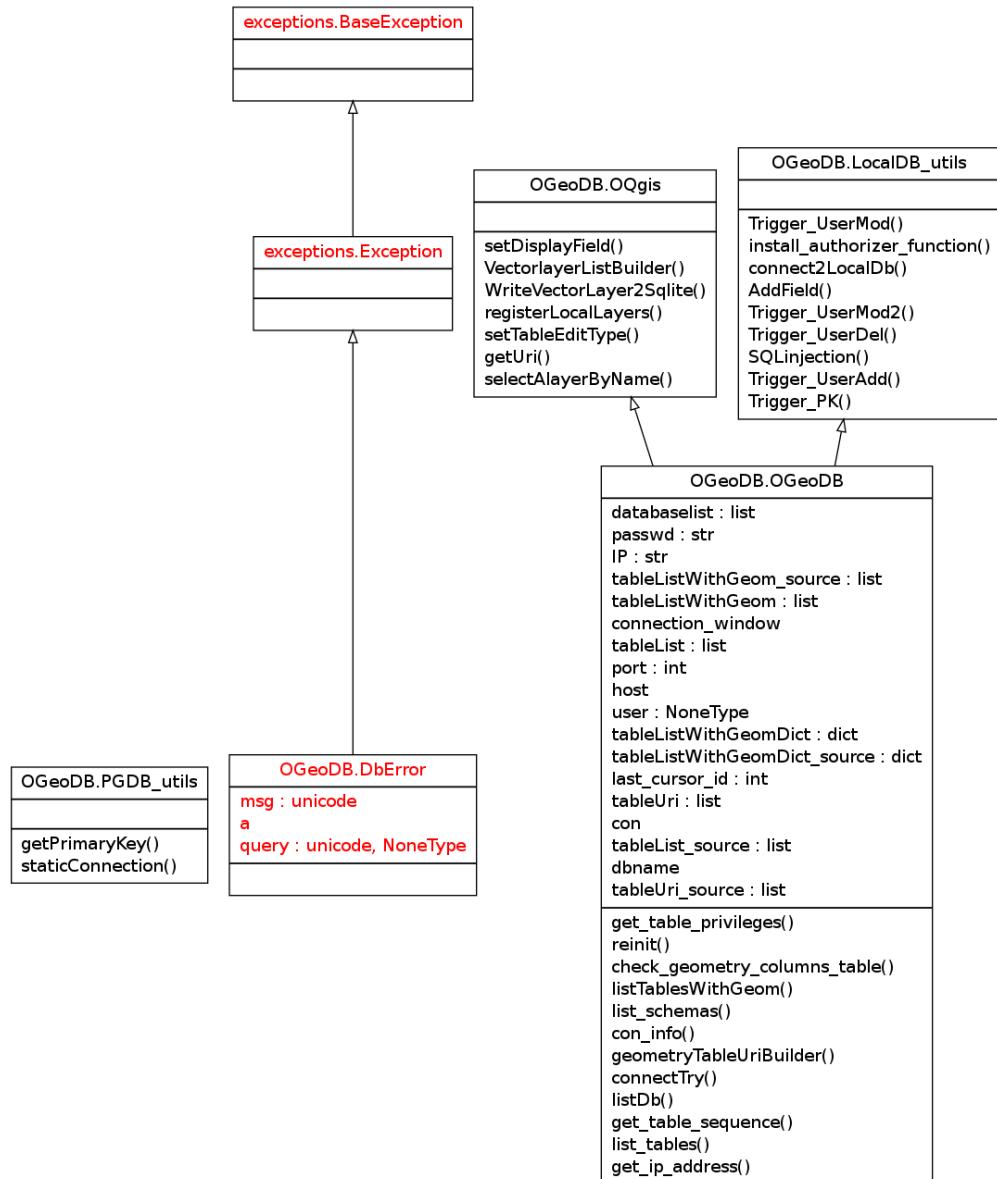
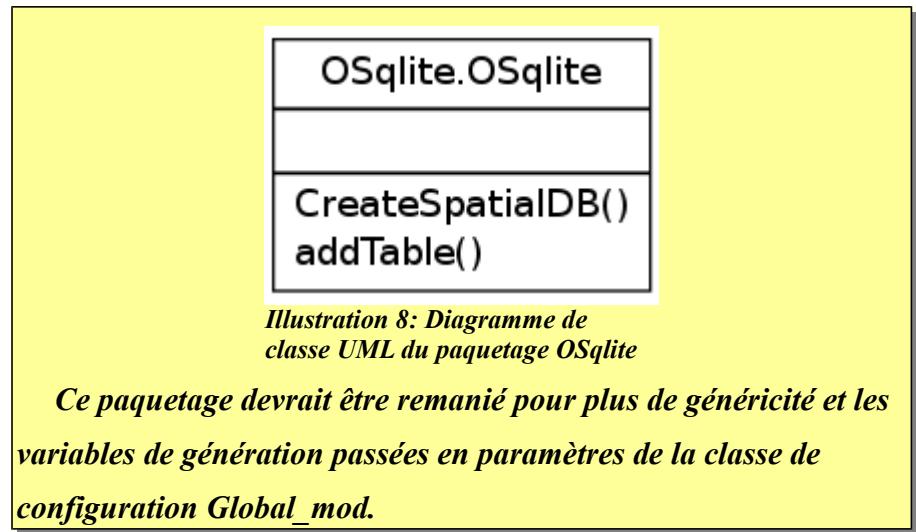


Illustration 7: Diagramme de classe UML du paquetage OGeoDB

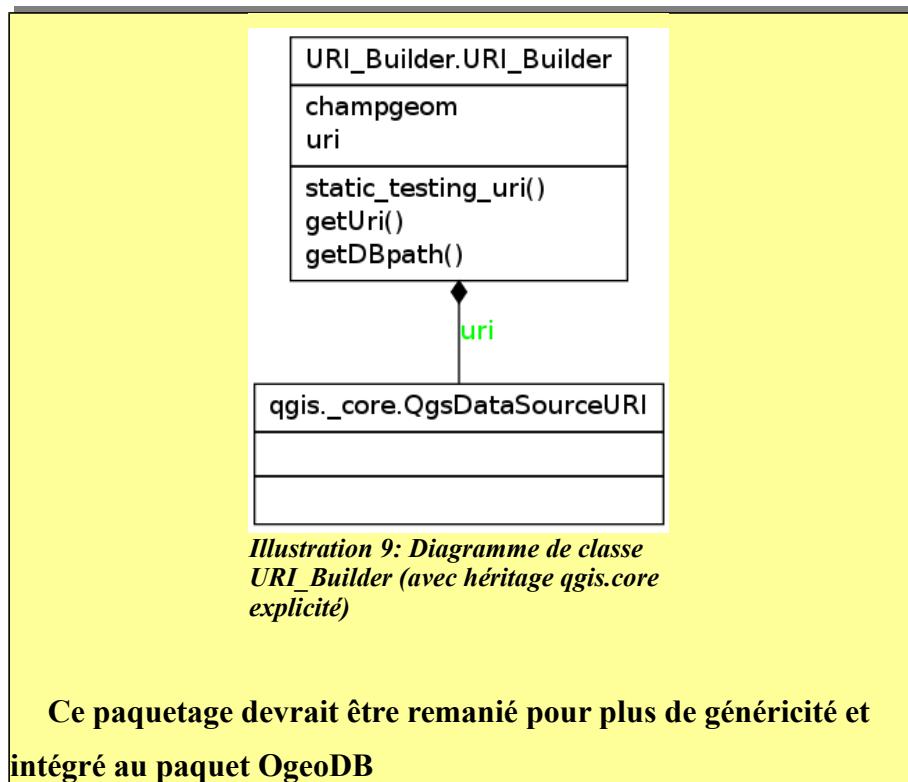
ii. Le paquetage dbtools.OSqlite

Ce paquetage permet la construction d'un fichier de base de données *SQLITE* étant du *SPATIALITE* requis pour la gestion des zones de réconciliation. Ce paquetage propose de générer le fichier de base de données de manière statique ou non.



- iii. Le paquetage dbtools.URI_Builder

Cette classe encapsule et surcharge un objet *QGIS QgsDataSourceURI*⁵ gérant les information de connexion requise par QGIS sur une table d'un SGBD.



- iv. Le paquetage reSyncDB

Ce paquetage contient tous les algorithmes et les fonctions requises pour synchroniser la base de données locale avec la base de données distante. Le graphe d'appel détaillé à ces fonctions est exposé dans le paragraphe B-IV dédié à l'exécution de l'implémentation Python du plugin.

⁵ Paquetage **qgis.core**; <http://qgis.org/api/classQgsDataSourceURI.html>

<code>sig40_reSync.reSyncDB</code>
<code>mapinstance</code> <code>qproject</code> <code>oproject</code> <code>loaded : bool</code>
<code>dict_to_html_table()</code> <code>generateReport()</code> <code>getNumrec()</code> <code>finalizeAndPublish()</code> <code>processDelete()</code> <code>reconciliate()</code> <code>dict_to_html_table_error()</code> <code>sendMailReport()</code> <code>referenceNewObject()</code> <code>writeProjectShelf()</code> <code>referenceUpdatedObject()</code> <code>referenceDeletedObject()</code> <code>trueDesync()</code> <code>processUpdate()</code> <code>processCreate()</code> <code>remove_img_tags()</code> <code>dict_to_html_table_oblivion()</code>

Illustration 10: Diagramme de classe UML du paquetage reSyncDB

1. **Reconciliate** : Classe encapsulant l'algorithme de réconciliation.
2. **ReferenceUpdatedObject** : référence les objets mis à jour au cours de la session utilisateur.
3. **ReferenceDeletedObject** : référence les objets supprimés au cours de la session utilisateur.
4. **ReferenceNewObject** : référence les objets créés au cours de la session utilisateur.
5. **ProcessUpdate** : fonction de traitement d'écriture des modifications autorisées sur la base de données distante – Concerne les objets mis à jour localement
6. **ProcessCreate** : fonction de traitement d'écriture des modifications autorisées sur la base de données distante – Concerne les objets créé localement

7. **ProcessDelete** : fonction de traitement d'écriture des modifications autorisées sur la base de données distante – Concerne les objets supprimés localement
8. **TrueDesync** : Fonction d'exploration et de référencement des désynchronisations

Les autres méthodes de la classe de réconciliation concerne le formatage des structures de référencement pour la génération du rapport de réconciliation *html* et l'émission des messages email.

Pour rappel, le schéma d'appel simplifié est présenté ci-après :

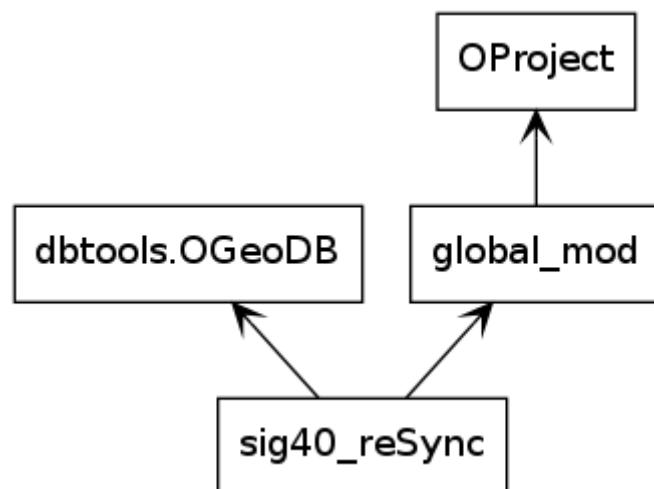


Illustration 11: Graphe d'appel des paquetages développés pour la synchronisation d'un projet

v. Le paquetage action.ProjetAction

Ce paquetage contient l'ensemble des algorithmes de traitements et outils requis pour la constitution et la modification d'un projet **sig40**. Il réalise également la connexion de ces algorithmes et outils avec les boutons et boîtes de dialogue utilisateur.

1. **CustomStaticTools** : Boîte à outil statique. Ne contient que la méthode permettant le centrage des fenêtres à l'affichage.
2. **YesNo** : Génère une boîte de dialogue dédiée à la suppression d'un projet
3. **NewProjectAction** : Encapsule les méthodes nécessaires à la constitution d'un projet ; Connecte les signaux utilisateur de la boîte de dialogue principale.
4. **GeoSelection** : Encapsule les méthodes requises pour réaliser une opération de sélection géographique interactive pour définir l'étendue et les tables extraites de la base de données distante.
5. **ConnectionParameters** : Classe permettant la définition a posteriori des paramètres de connexion utilisateur (non utilisée dans les algorithmes de traitement)
6. **ZRselection2** : Encapsule les méthodes requises pour réaliser une opération de création d'une nouvelle zone de réconciliation (ie : référencement des zones et des objets impactés par une édition utilisateur)
7. **QgsMapToolSelectPolygon** : Algorithme de sélection et de référencement d'objets. Défini un outil interactif de sélection par dessin d'une forme polygonale (sans îles).

Cette classe devrait être remaniée pour isoler l'outil de l'algorithme de traitement

8. **ZrDestroy** : Algorithme de gestion de la suppression du référencement d'objets préalablement référencés.
9. **Function_threader** : Classe d'exécution d'un thread avec passage d'arguments.

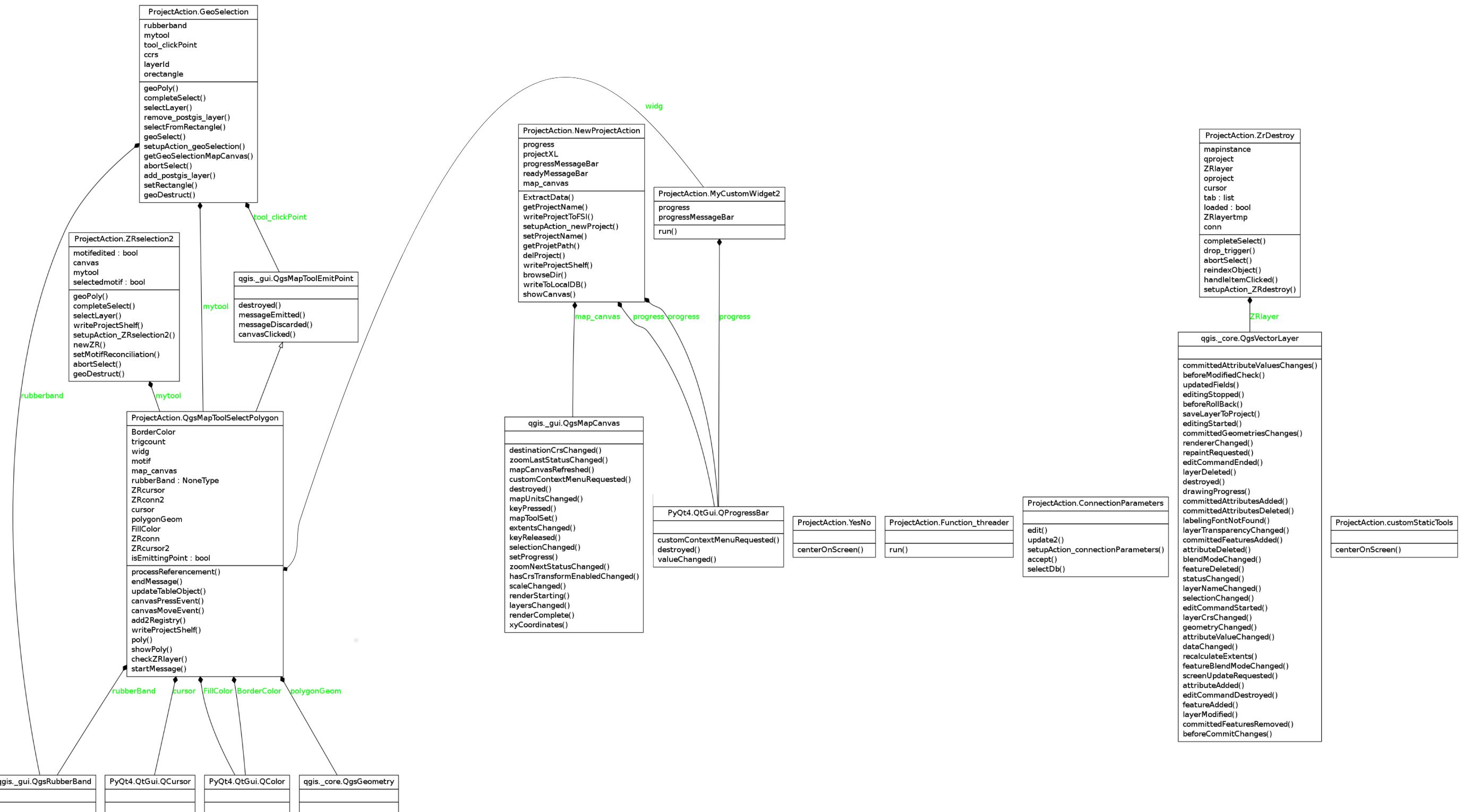


Illustration 12: Diagramme de classe UML du Paquetage ProjAction

e. Persistance du projet et référencement des objets

Comme présenté ci-avant, la persistance du projet permet à un utilisateur de faire évoluer son projet dans le temps en conservant des références sur les tables modifiées, les zones de réconciliations ajoutées et/ou supprimés... La généricité des traitements permet de s'abstenir d'avoir à décrire préalablement les tables importées depuis le serveur en construisant des structures référençant les métadonnées à partir des catalogues de la base de données distante. Les objets sont référencés au sein de dictionnaires Python (présentés en Illustration 13) présentant l'avantage de fournir des accès explicites à leur contenu au travers de clefs uniques.

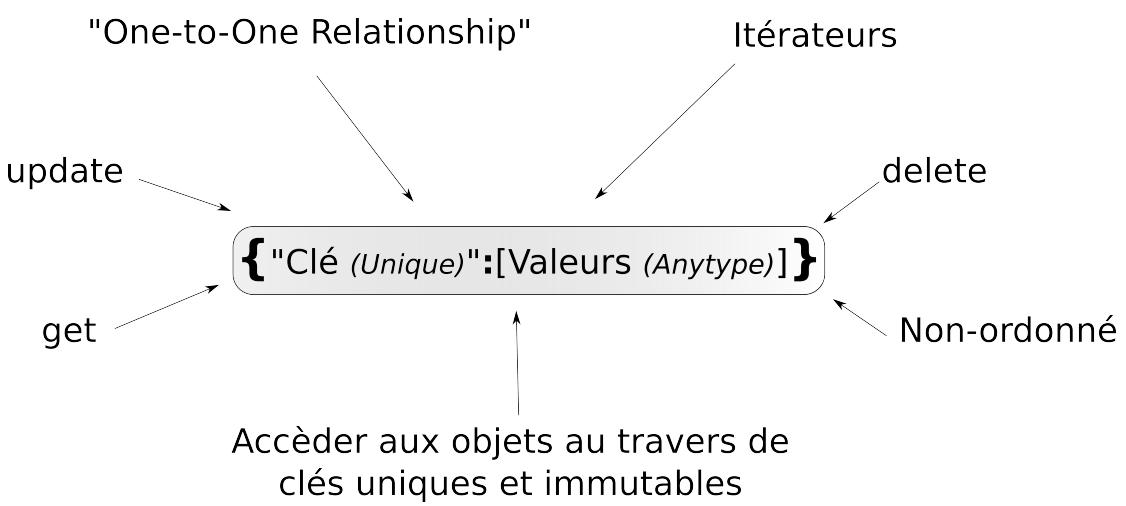
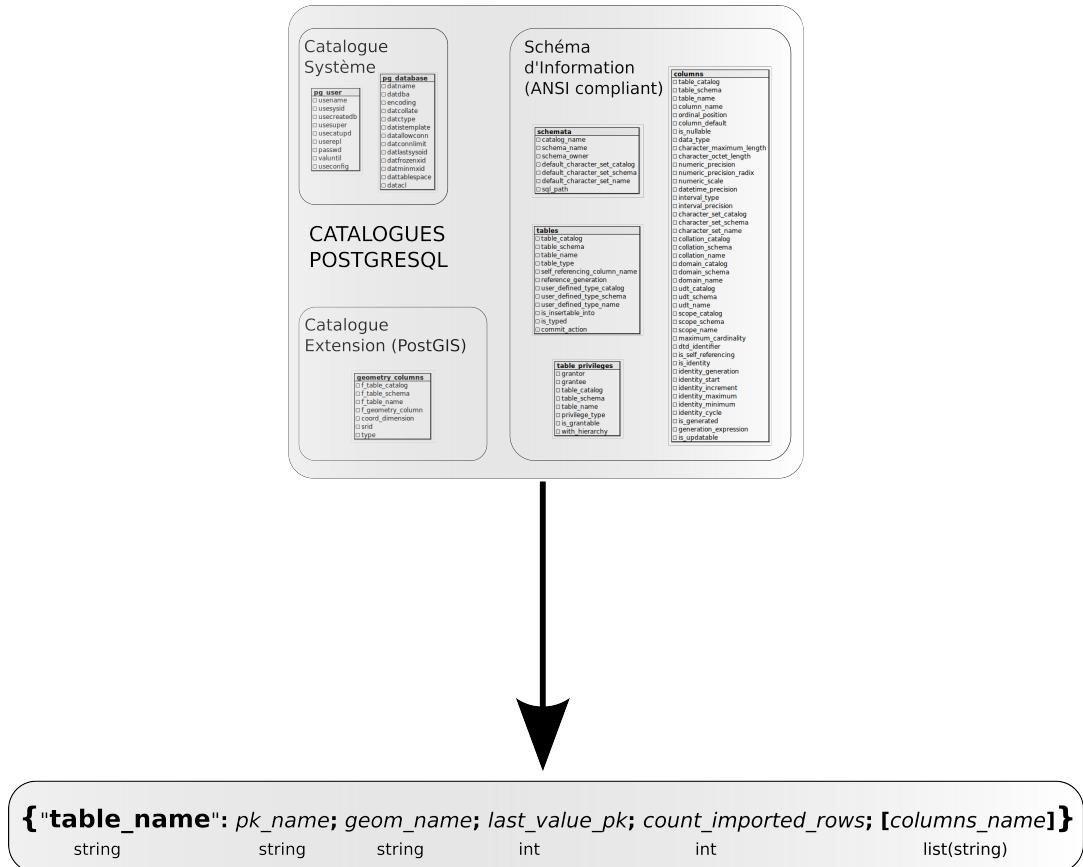


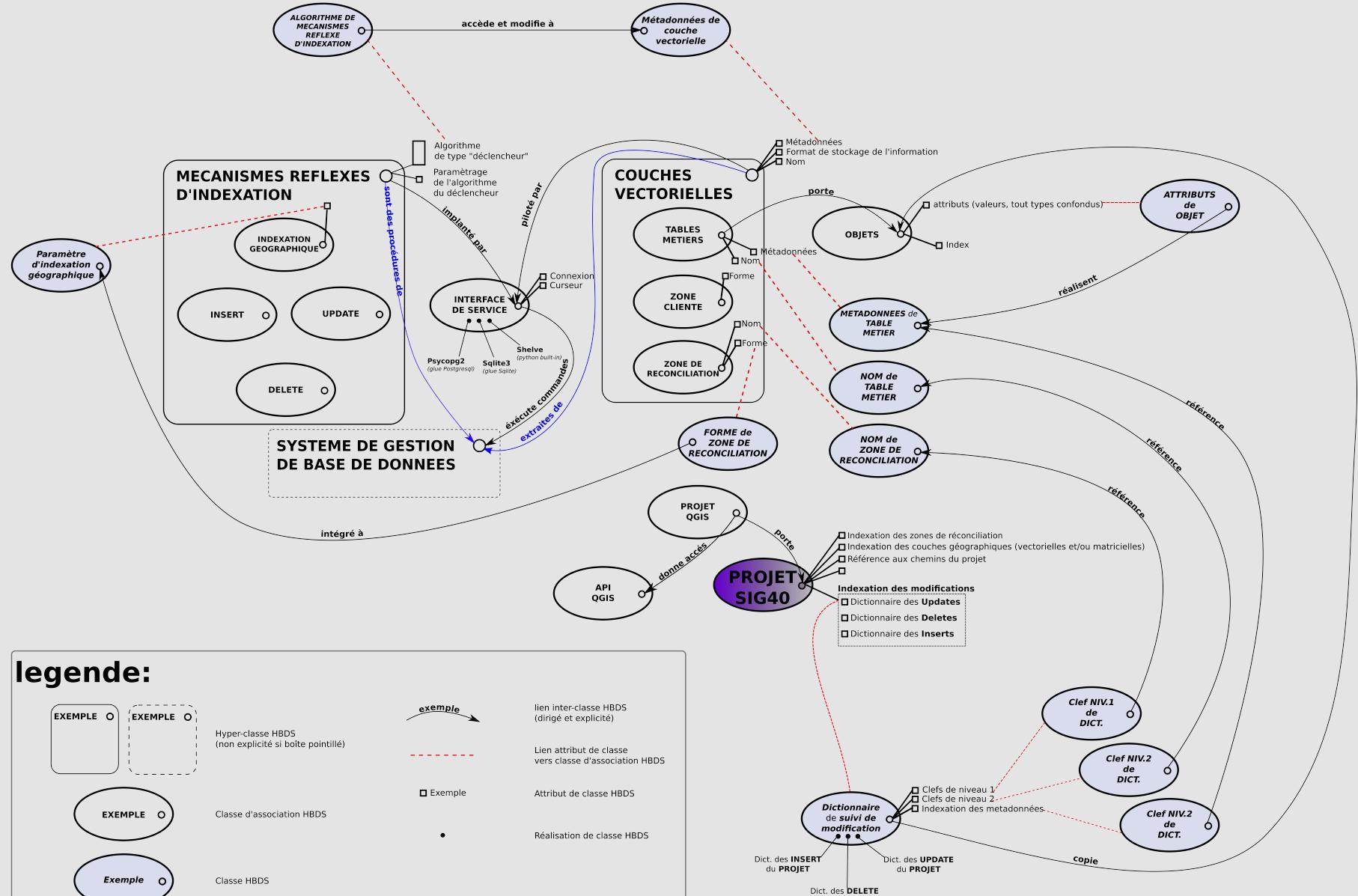
Illustration 13: Dictionnaire python et méthodes associées

Une entrée de ce dictionnaire constituant ainsi « l'atome » de description des tables importée depuis la base de données selon le schéma suivant (Illustration 14) :



Définition d'un dictionnaire indexant les propriétés des tables de la base de donnée
Illustration 14: Description de la structure de conservation des métadonnées des tables de la BD sig40

Ces structures référencent donc les noms et métadonnées des tables extraites du serveur (typage, nom des champs). Elles sont encapsulées en tant qu'attribut de la classe de persistance du projet (**sig40.Oproject**) qui va être sérialisée sous forme d'un fichier binaire instancié lors de la reprise d'une session d'édition.



MODELISATION HBDS DU DIAGRAMME DE CLASSE D'UN PROJET SIG40 (PLUGIN QGIS)

Illustration 16: Diagramme de classe HBDS du projet sig40

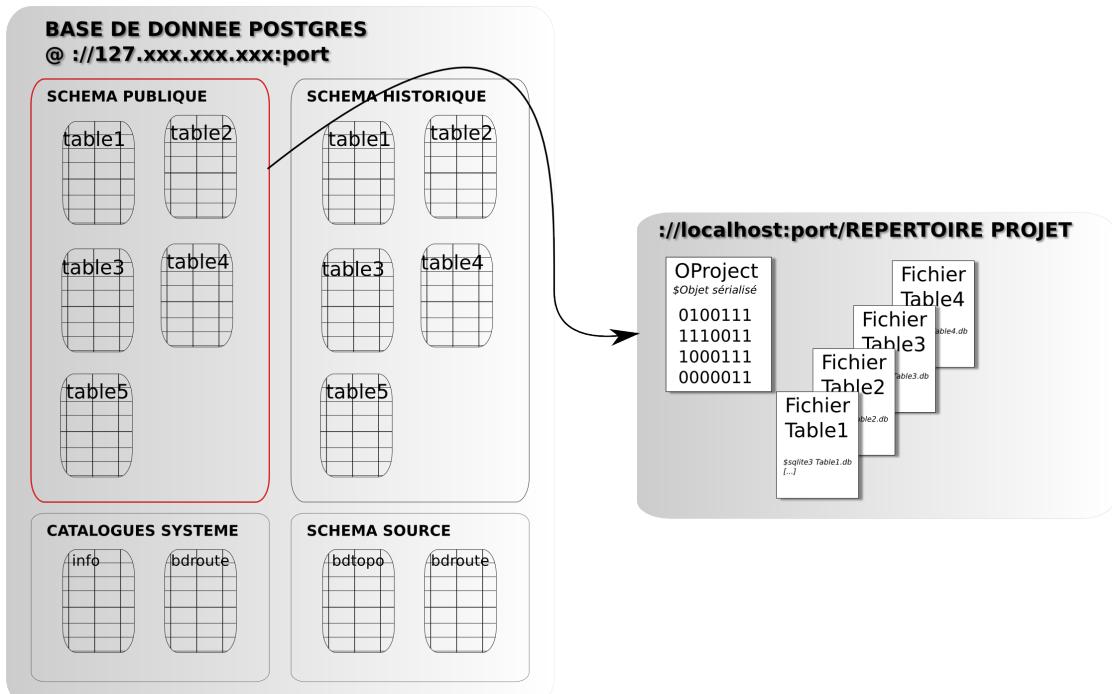


Illustration 17: Copie locale des données serveur

Localement, un répertoire est créé et dédié à l'écriture sur disque des données du projet Celui-ci contient la copie de l'extrait des tables du serveur sig40 sous forme de fichiers de base de données *SQLITE* à raison **d'une base par table extraite** du serveur PostgreSQL. Ces bases sont donc enregistrées au sein du répertoire sous la forme de fichiers individuels nommés selon la dénomination de la table parente sur le serveur distant.

IV. Exécution

L'exécution du plugin concerne la constitution d'un projet et la synchronisation des données après une session de modification. Les diagrammes suivants présentent les graphes d'enchaînement des modules et fonctions lors de :

- la constitution d'un nouveau projet sig40.
- La synchronisation d'un projet comportant des modifications utilisateurs
- L'abandon d'une synchronisation suite à un conflit entraînant la détection d'une désynchronisation.
- La tentative de synchronisation d'un projet n'ayant subi aucune modification
- La tentative de synchronisation d'un projet déjà synchronisé

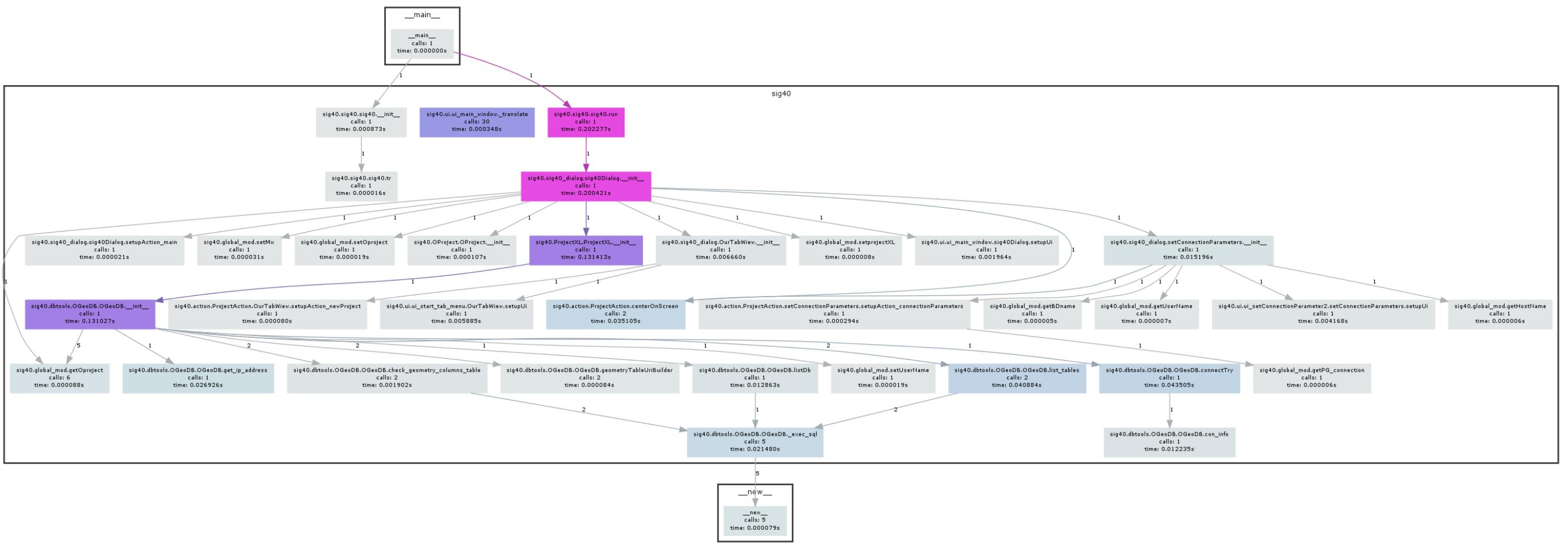


Illustration 18: Graphe d'enchaînement des modules et fonctions lors de la création d'un projet `sig40` (la coloration (couleurs froides vers couleur chaude), représentent le temps d'exécution croissant des fonctions. Les nombres indiqués sur les flèches représentent l'enchaînement indique le nombre d'appel à la fonction pointée)

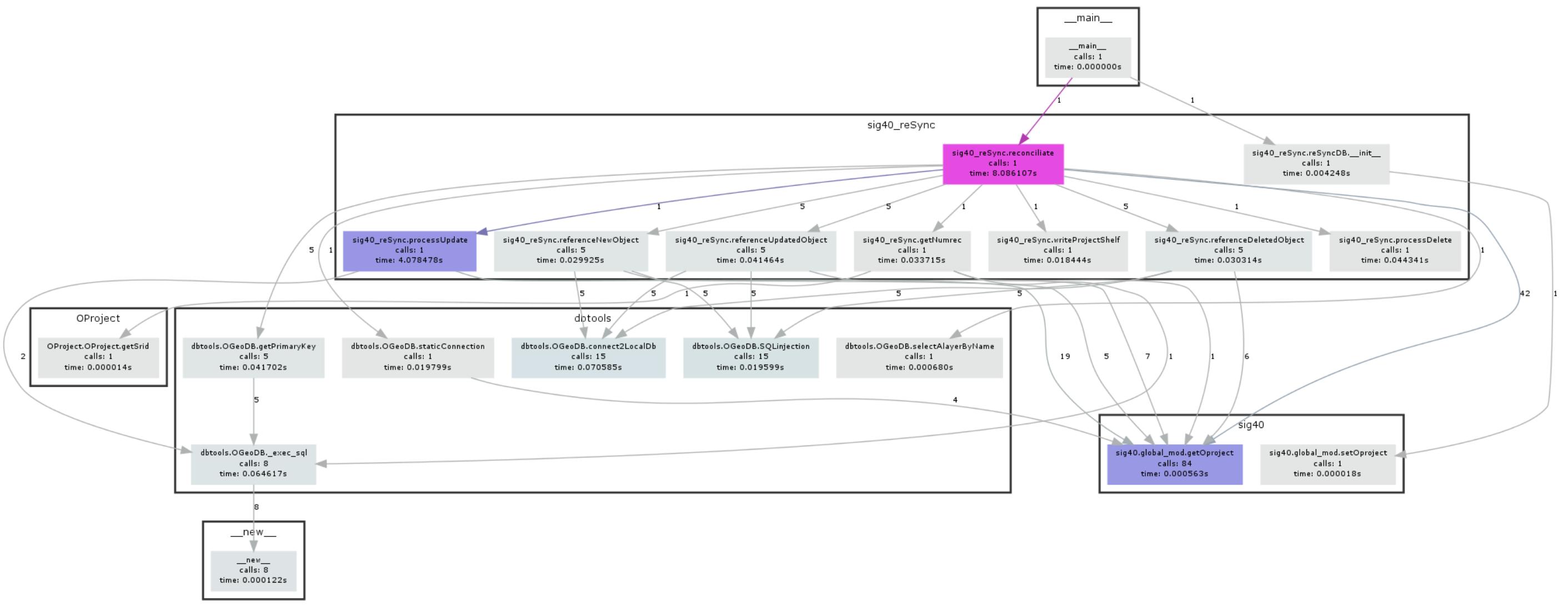
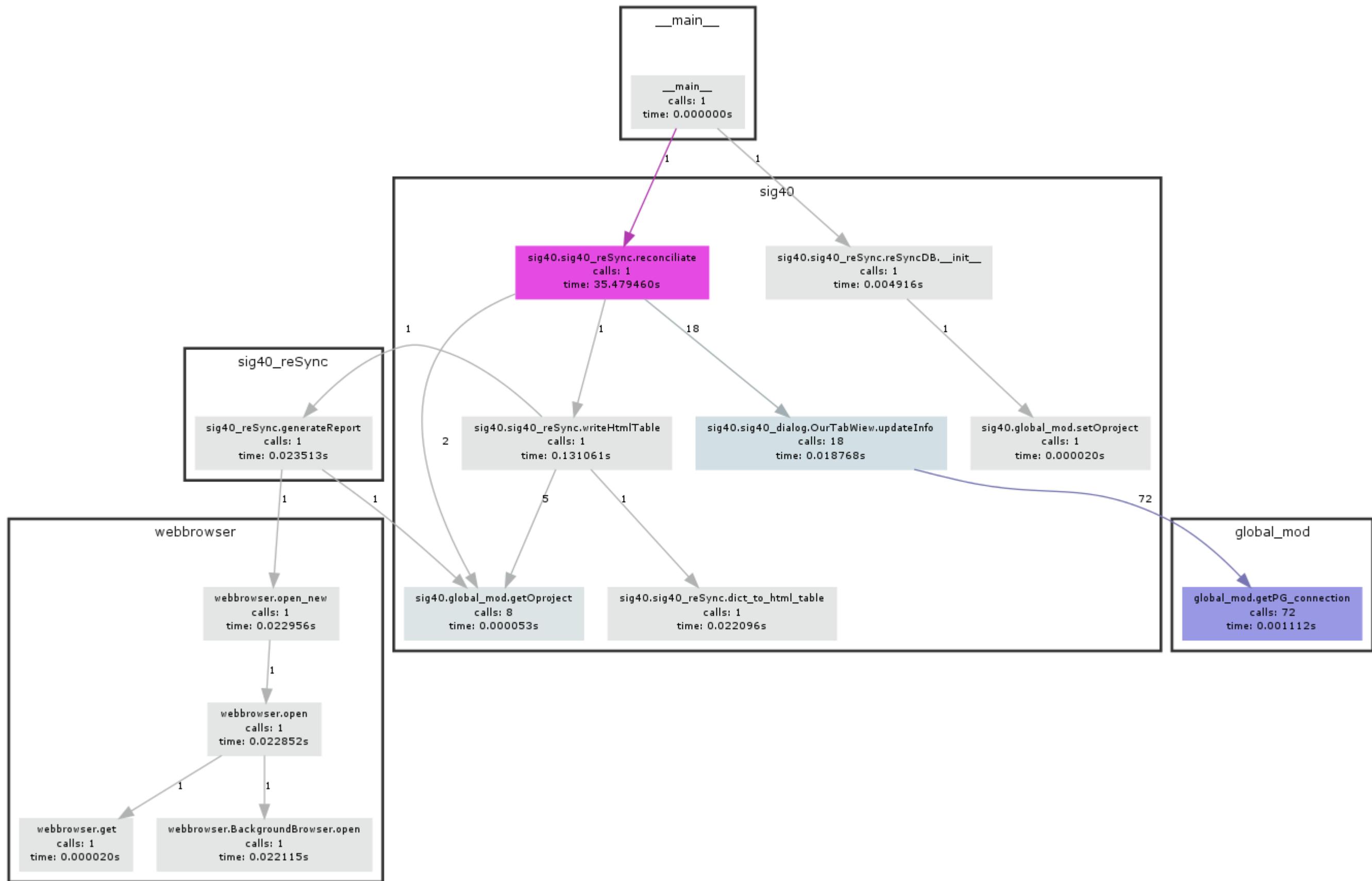


Illustration 19: Graphe d'enchaînement - Synchronisation après modification utilisateur (transaction de type UPDATE & DELETE)



Generated by Python Call Graph v1.0.1
<http://pycallgraph.slowchop.com>

Illustration 20: Graphe d'enchaînement - Tentative de synchronisation d'un projet déjà synchronisé

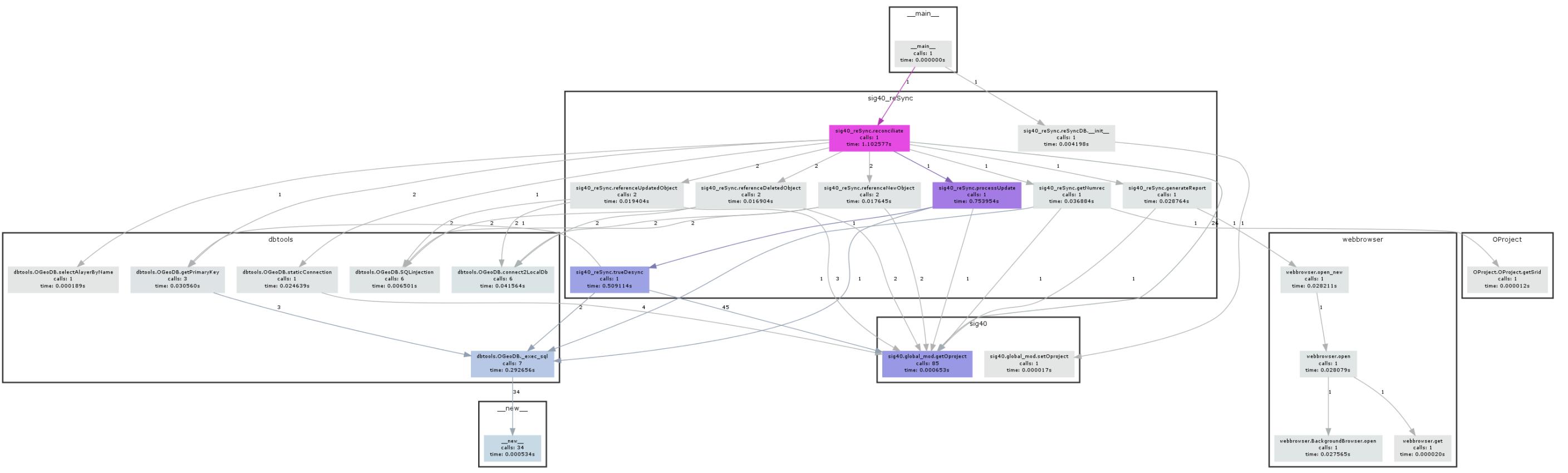
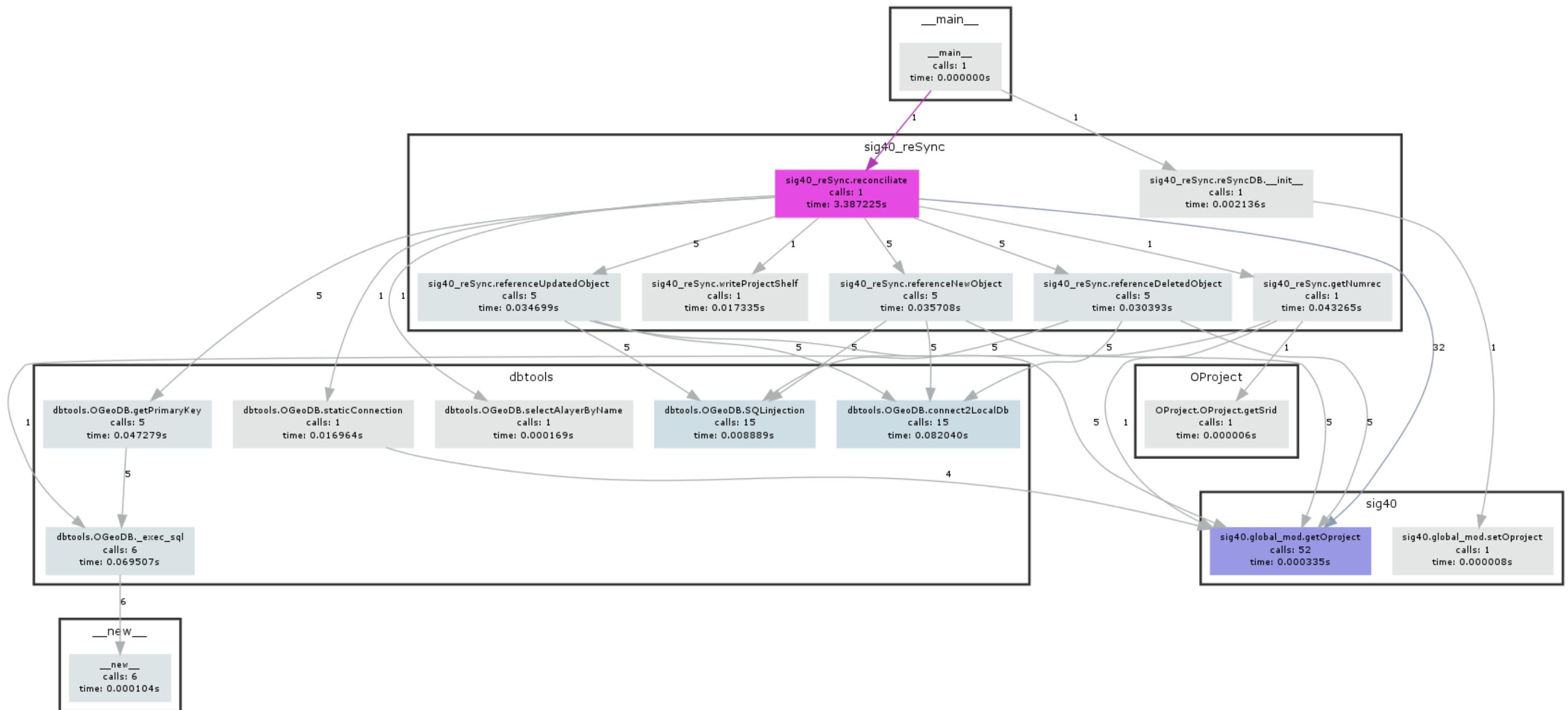


Illustration 21: Graphe d'enchaînement - Abandon de synchronisation suite à la détection d'un conflit non résolu



Generated by Python Call Graph v1.0.1
<http://pycallgraph.slowchop.com>

Illustration 22: Graphe d'enchaînement - Synchronisation d'un projet sans aucune modification

C. SERVEUR DE BASE DE DONNEE POSTGRES/POSTGIS

Tel que présenté par ses développeurs, PostgreSQL est un système de gestion de base de données Object-relationnel (object-relational database management system ORDBMS) basé sur POSTGRES 4,2 développé à l'université de Berkeley (Californie). PostgreSQL correspond ainsi à la poursuite *open-source* de ce projet.



Illustration 23: Le logo PostgreSQL

Il existe un certains nombre d'avantage à l'utilisation d'un SGBD libre :

- *Pas de coût de déploiement (et donc pas de sur-déploiement).*
- *Pas de coût caché (pas de licence additionnelle)*
- *Code source disponible et modifiable*
- *Compatibilité Windows/Linux/OS X*
- *Support amélioré au travers de la communauté d'utilisateur (professionnels, chercheurs, amateurs...)*
- *Bonne intégration avec de nombreux autres logiciels libre permettant d'en faire le SGDB référence.*

Parmi les fonctionnalités avancées proposées par ce logiciel, les développements réalisés pour ce projet font usage tout particulièrement :

- *des « Triggers » (déclencheurs automatisés)*
- *Du système de gestion de la concurrence entre utilisateurs*
- *Du système de sauvegarde de l'intégrité des transactions*
- *Des capacités d'extension (Types de données, écriture de fonction additionnelles,...)*
- *Exécution de blocs de code anonymes*

PostgreSQL est également étendu par la librairie PostGIS (version 2.1) de manière à permettre la manipulation d'objets géographiques selon une abstraction élevé. De nombreuses fonctions géométriques peuvent être appliquée directement par le serveur et une indexation spécifique des données est créée de manière à fournir un accès efficace à l'information. PostGIS étend PostgreSQL en ajoutant des types et outils pour la

manipulation d'objets vectoriels. Le typage de ces objets répond aux exigences posées par l'OGC (Open Geospatial Consortium) et l'ISO (International Standards Organization). Le support pour la manipulation des données géographiques raster est également implanté mais non abordé dans le cadre de ce projet.

I. **Fondamentaux architecturaux**

L'architecture logicielle proposée par PostgreSQL utilise le modèle client/serveur. Une session PostgreSQL consiste en une connexion traitée par le serveur dédié (le serveur *Postgres*) et le client qui contrôle les opérations à réaliser sur le serveur. Le contrôleur correspond à l'extension développée (*plugin QGIS*), qui permet de réaliser des opérations de sélection et d'écriture sur le serveur distant.

Dans le cadre de cette structuration client/serveur, le(s) client(s) et le(s) serveur(s) sont localisés physiquement sur des machines hôtes différentes. Au sein du Département des Landes, le serveur PostgreSQL est exécuté sur un serveur *DELL PowerEdge* dans un environnement « *Windows Server 2012* » virtualisé sur un OS Linux. Dans ce contexte applications clientes et serveur communiquent sans intervention, le serveur proposant un service (processus) toujours accessible, prêt à traiter les demandes de connexions.

II. **Administration du serveur PostgreSQL**

a. **Authentification sur le réseau du Département des Landes**

La connexion au réseau interne du CG est réalisée par mot de passe géré au niveau de l'OS lors du démarrage d'une session. L'adressage TCP/IP est dynamique (DHCP). Les adresses étant régénérées lors de l'absence d'une connexion durant une certaine période de temps (absence de l'utilisateur sur le réseau).

b. **Méthode d'authentification sur le serveur PostgreSQL**

La gestion des priviléges utilisateurs est établie sur plusieurs niveaux de sécurité. Elle repose tout d'abord sur la méthode d'authentification lors d'une demande de connexion au serveur.

L'identification est assurée par l'association d'un nom utilisateur sur le réseau du Département des Landes associé à l'adressage TCP/IP physiquement lié à un poste utilisateur connecté.

Dans ce contexte, et de part le faible nombre de la base d'utilisateurs potentiels du SIG du Département, les connections utilisateurs sont définies manuellement par l'administrateur de la base de donnée (création, mise à jour et suppression)

La méthode retenue est donc du type **Trust authentication** explicitée par l'administrateur pour chaque utilisateur au travers donc d'un couple (nom d'utilisateur/Adresse IP sur le réseau du CG).

c. Définition des rôles et privilèges

Un **rôle** est donc créé pour chaque nouvel utilisateur nécessitant un accès à la base de données. Selon le niveau d'utilisation anticipé par le gestionnaire de la base de données et selon les compétences métiers, ce rôle hérite d'un ou plusieurs privilèges de **groupes prédéfinis**.

```
CREATE ROLE pg_group_metier  
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER } [,...] | ALL [ PRIVILEGES ] } ON [  
TABLE ] table_metier [, ...] TO { [ GROUP ] pg_group_metier |  
PUBLIC } [, ...] [ WITH GRANT OPTION ]  
CREATE ROLE name_example ;  
GRANT pg_group_metier TO name_example;
```

Tableau 1: Crédit d'un rôle "groupe" et héritage sur utilisateur

La définition des « rôles groupés » est intriquée avec le développement de la structuration de la base de données. En effet l'étendue des privilèges attribués aux utilisateurs doivent bien correspondre au niveau d'interaction (**select/insert/update/delete**) et aux champs d'applications (tables impactées) requis par ses attributions.

III. L'interface client

a. La librairie libpq:

L'interface de programmation est réalisée au travers de la **librairie C libpq** qui donne les fonctions nécessaires au passage des requêtes vers le serveur et encapsule les résultats. Dans le cadre de ce projet, la **libpq** n'est jamais utilisée directement mais invoquée au travers du paquetage **psycopg2** réalisant la liaison (« Binding ») avec le langage **Python 2.7**

b. Le Schéma d'information (introduction au catalogue système)

Le catalogue Postgresql et plus particulièrement son *schéma d'information* constitue la porte d'entrée vers la découverte des objets de la base de données renseignant sur :

- Sa structure (bases de données, schéma et tables)
- Ses utilisateurs, explicitant leurs priviléges
- le typage des objets

Le schéma d'information est défini en tant que standard SQL, il constitue une vue standardisée du catalogue système PostgreSQL. Il n'est pas inscrit par défaut dans les chemins de recherche et son utilisation doit ainsi être explicite pour éviter des collisions sur l'espace des noms de la base de donnée.

D.BASE DE DONNEE SQLITE/SPATIALITE

SQLITE est une librairie logicielle implémentant un moteur transactionnel pour base de données SQL ne nécessitant pas de serveur ni de configuration. Il s'agit également d'une base de données appartenant au domaine public.



Illustration 24: Le logo SQLITE

SPATIALITE étend les types *SQLITE* en introduisant des fonctionnalités vectorielles comme la librairie *PostGIS* étendant le SGBD PostgreSQL.

I. Fondamentaux Architecturaux

Contrairement aux serveurs SGBD Postgres évoqués précédemment, les bases de données *SQLITE* ne reproduisent pas l'architecture serveur/client mais sont directement intégrées aux programmes tables, index, fonctions et catalogues systèmes sont directement intégrés au sein d'un fichier unique.

L'intégration directe aux programmes utilisant une base de données *SQLITE* la rend idéale pour développer des applications clientes sans partage entre utilisateur, fonctionnant hors ligne.

II. Authentification

SQLITE ne propose pas de système de gestion de droit et repose donc sur les priviléges donnés par l'OS à l'utilisateur sur le fichier au sein de son système de gestion de fichiers (écriture/lecture).

III. Interface

La bibliothèque SQLITE est écrite en C (respectant les normes ANSI) sans utilisation d'aucune autre bibliothèque externe. Il s'agit de la librairie *sqlite* dans sa version 3. Python inclus *SQLITE* dans sa bibliothèque standard depuis sa version 2.5 (paquetage *sqlite3*)

E.MISE EN PLACE D'UN SYSTEME D'HISTORISATION

I. Conceptualisation -Architecture de la base de données et interactions utilisateurs

L'historisation d'une base de données consiste à intégrer la dimension temporelle au cœur des mécanismes de mise à jour d'une base de donnée :

Il s'agit de ne jamais supprimer un objet de la base de donnée et de constituer un historique de ses états au fur et à mesure des modifications appliquées. Toute opération autorisée (modification ou création) résulte ainsi en la création d'un objet historique, comportant une référence à son parent, l'objet actuel, visible aux utilisateurs.

Les objets sont également **historisés selon un second niveau d'indexation « projet »**, qui permet de lier des objets ayant subis des modifications au cours d'une même session d'édition par un utilisateur unique.

L'ensemble des modifications concomitante réalisée par un utilisateur sur un projet commun est donc sélectionnable.

Les paragraphes suivants présentent le développement de la stratégie d'historisation :

- En rappelant tout d'abord ses objectifs et limitations
- En explicitant la structuration de la Base de données
- En explicitant les mécanismes de gestion de la concurrence

II. Pourquoi développer une stratégie d'historisation?

« des outils les accompagnants dans leurs démarches »

La généralisation de l'outil numérique et l'augmentation du volume de données à traiter , classifier et archiver par des utilisateurs aux compétences et métiers très différents doit être accompagné de la mise à disposition d'outil d'uniformisation des traite

ments et proposant une abstraction suffisamment élevée pour masquer la complexité des processus engagés. Les utilisateurs finaux des bases de données et des applications clientes doivent pouvoir se concentrer sur le cœur de leurs métiers et utiliser des outils les accompagnants dans leurs démarches plutôt que d'adapter leurs démarches à des outils.

« Une aide à la transmission »

Les problématiques de renseignement, mise à jour, création de données et enfin de partage d'une information fiable (maintenue régulièrement au fait des changements) constituent le cœur des concepts amenant au développement d'un système d'historisation. Ainsi la production de données. Cette structuration des connaissances est un atout important dans le partage de celles-ci. L'uniformisation de la structuration des connaissances accumulées se veut ainsi une aide à la transmission.

« Une garantie de l'intégrité de l'information »

Le partage de l'information entre corps de métiers différents, la dématérialisation des activités peuvent être sources de conflit sur le contenu des objets mis à disposition. La concurrence doit pouvoir être détectée, décrite et des mécanismes de mise en relation des utilisateurs imaginés dans l'objectif de garantir l'intégrité de l'information. La sauvegarde brute et régulière de l'information ne donne pas les clefs pour comprendre d'éventuels conflits et d'en déduire des méthodes de résolution appropriées (automatisées ou semi-automatisées).

III. Mise en place d'un système d'historisation – Structuration de la BDD et stratégie serveur

a. Introduction- Modèle simplifié du système d'historisation

Les travaux de développements présentés dans ce rapport sont inspirés des concepts appliqués à la structuration et au système d'historisation appliqué à la BDUni de l'IGN. La BDUni étant une **base de données de production** de données vecteurs portant sur la France entière et mettant en concurrence de nombreux utilisateurs manipulant des objets référencés géographiquement. Cette base de données constituant un des socles des produits commerciaux préparés par l'IGN (BDCarto®, BDTopo®,...).

La structure de la BDUni se divise en trois tables attributaires :

- Deux tables pour chaque classe d'objets :
 - Une table des objets actuels, servis aux utilisateurs
 - Une table des objets historiques
- Une Table pour l'indexation des mises à jours – la **table des Zones de réconciliations**, requise pour les opérations de réfraction vers des états passés.

Ces tables présentent un attribut (clé d'indexation) forçant des **contraintes d'unicité et d'intégrité** permettant de lier un objet à tout ou partie de son historique parallèlement aux attributs décrivant la mise à jour (typiquement, la date, l'utilisateur et le motif de mise à jour). Une représentation de cette structure logique telle qu'implantée dans la base de données du Département des Landes est présentée en Illustration 28 ci-après

Enfin la mise à jour de ces tables sur la base d'exécution de transaction de type **INSERT/UPDATE/DELETE** suite à des modifications par un utilisateur est intégralement laissée à la base de données par le biais de procédures automatisées, « triggers », programmées pour s'exécuter et réaliser les copies et mises à jour des champs propres à l'historisation sans intervention de l'utilisateur.

b. Modèle de données logique

Il est mis l'accent ici sur la structuration de l'information permettant le développement du système d'historisation. Le versionnement des objets est donc réalisé au travers de l'édition d'attributs spécifiques (Illustration 25) adossé à des procédures automatisées (dont les implantations SQL sont présentées en Illustration 28, 29 et 30) exécutées par le serveur de base de données.

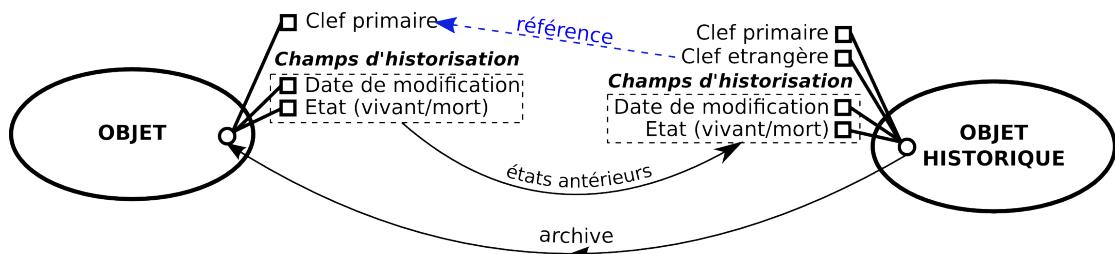


Illustration 25: Classe d'objets (pseudo-HBDS) portant les métadonnées du mécanisme historisation

Le mécanisme d'historisation repose donc sur la combinaison de 3 attributs :

1. La mise en place de relation inter-table reposant sur l'utilisation de clef étrangère. Relation 1 à plusieurs (objet → objet historique)
2. Un champ **date de modification** qui donne la dernière date de modification de l'objet considéré (ce champ étant non renseigné à la création d'un nouvel objet)
3. Un champ **d'état** qui indique si l'objet a été détruit (la date de sa destruction correspondant à la **date de modification** associée). Un objet n'est jamais physiquement détruit ni retiré de la table des objets actuels. Il est simplement rendu inaccessible par une sélection simple. Le dernier état d'un objet (et tous ses attributs métiers) et donc toujours accessible.

Additionnellement, un niveau de recherche est ajouté en définissant une classe d'indexation, nommée **zone de réconciliation** par le biais duquel sont attribués des numéros d'identification uniques permettant de lier les objets modifiés au sein d'un même projet.

Dans son application au SGBD du Département des Landes, ce modèle est appliqué à la définition des métadonnées et à la structuration des tables de la base de données.

les différents champs concernés par l'historisation ainsi que les référencements entre tables sont permis par des copies de clés primaires entre tables (unicité et intégrité). L'Illustration 26 ci-avant représente le diagramme de classe originel portant tous les attributs permettant une gestion du versionnement de la base de données incluant la gestion de la concurrence dans un contexte multiples-utilisateurs. Il est ainsi défini pour chaque classe «métier» une classe spécifique portant l'historique des objets associés. Il s'agit d'une relation 1 à plusieurs puisqu'à un objet on associe tous ses

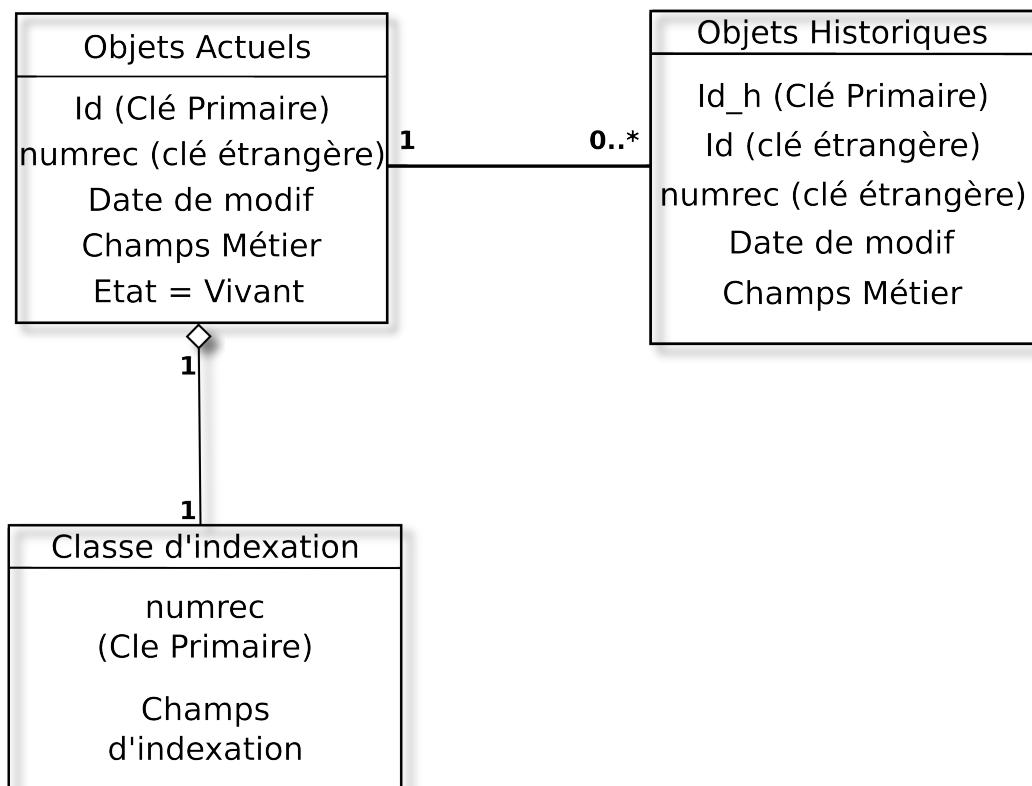


Illustration 26: Diagramme de classe (UML) prototype de la base de données SIG40 (les opérations d'affectation et de lecture sur les attributs sont assumée et ne sont pas représentées)

états antérieurs. Un troisième objet, « classe d'indexation » est également établi pour augmenter l'historisation des objets de relation de type « projet », permettant de **lier tous les objets ayant fait l'objet d'une modification/création/suppression concourante** et de référencer **l'utilisateur** à l'origine du « projet ».

c. Profil de Modèle de données UML (« Data Model Profile »)

L'Illustration 27 ci-après présente le *DMP UML* pour la BD SIG40. Cette présentation a pour intérêt de se rapprocher du modèle « physique » de la structure base telle qu'installée sur le serveur.

Les classes d'objets ont été cartographiées sur une représentation des classes de tables. Cette représentation conceptualise au plus près la structure et l'exécution de l'historisation de la base de données SIG40.

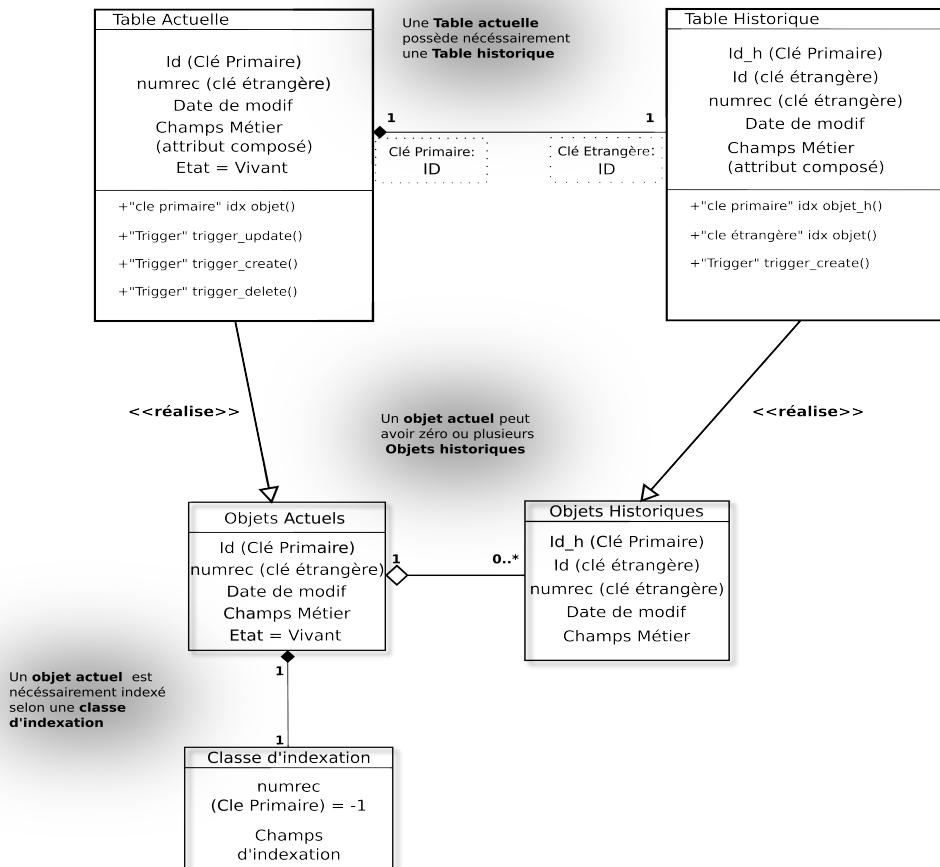
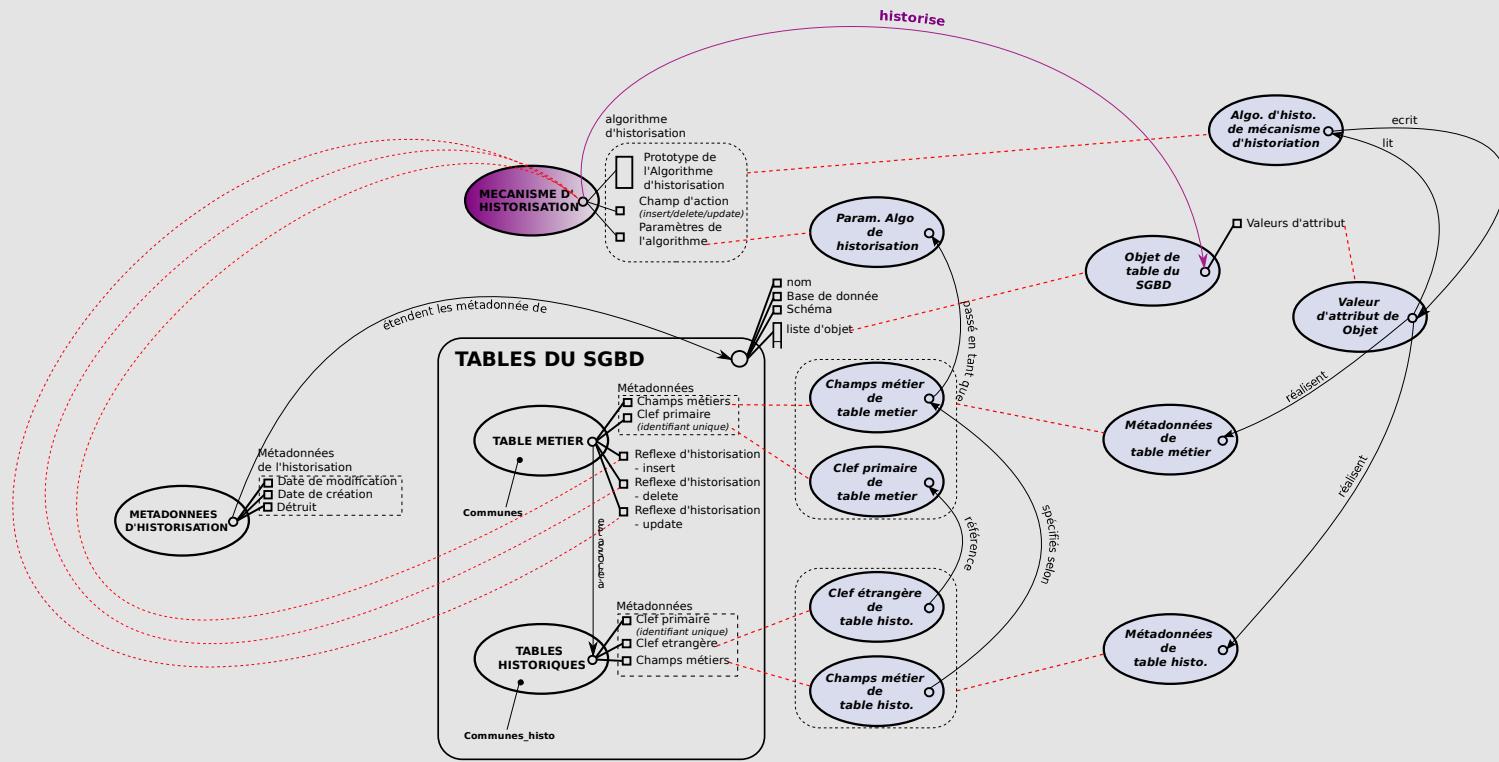


Illustration 27: UML Data Model Profile - Conceptualisation de la Base de donnée SIG40

d. Cycle de vie d'un objet – Crédit et modification

Dans le cadre de l'historisation **un objet de la base de donnée SIG40 est persistant physiquement sur le système une fois celui-ci créé**. Toute action de modification d'un objet touchant tout ou partie de ces attributs entraîne de fait l'inscription de son état antérieur dans la table d'historique associée. La réfraction de l'information actuelle depuis un état antérieur entre dans ce cadre.

L'inscription d'un objet dans sa table historique correspondante comporte l'inscription d'une clé étrangère qui renvoie à l'objet actuel. Cette mise en relation des tables par la distribution de ces clés uniques (primaires ou étrangères selon qu'elles indexent un objet appartenant à la table ou un objet appartenant à une autre table) permet simplement d'interroger la base de données pour récupérer tout ou partie des versions antérieures d'un objet.



legende:

EXEMPLE	EXEMPLE	Hyper-classe HBDS (non explicité si boîte pointillé)
EXEMPLE		Classe d'association HBDS
Exemple		Attribut de classe HBDS
	•	Réalisation de classe HBDS

STRUCTURATION DE LA BASE DE DONNEE SIG40 ET DE SON MECANISME D'HISTORISATION

e. Cycle de vie d'un objet – Suppression

Toute action de suppression d'un objet dans le cadre de l'historisation est traitée comme une extension du processus appliqué à une modification. L'objet est alors simplement marqué comme détruit mais persiste en tant qu'objet actuel, dans la table des objets actuels. Le champ **d'état** est dédié à cette opération. Ce champ devra donc faire partie intégrante des requêtes utilisateurs dans le cadre d'une extension systématique par une clause du type **WHERE NOT détruit** lors de sélection d'objets des tables pour ne traiter que les objets vivants. Réciproquement, cette approche permet de récupérer aisément les objets dans un état détruit de la base.

**Toute opération de suppression physique de l'intégralité
des références d'un objet dans la base sort du cadre de
l'historisation et nécessite l'intervention de administrateur
de SGBD**

f. Implantation des fonctionnalités dans les tables de la BD SIG40

Comme présenté précédemment dans l'architecture du projet, il a été choisi de développer des procédures implantées dans la base et exécutées automatiquement et systématiquement par le serveur. Ces procédures sont décrites comme des « triggers » selon la nomenclature usuelle des SGBD. Elles s'exécutent systématiquement lors de transactions appliquées à la base de données selon le prototype choisi (**INSERT** et **UPDATE**). Les « triggers » sont installés sous forme de procédures inscrites au langage *plpgsql* (SQL augmenté), 2 par tables, présentées ci-après en Illustration 29 et 30.

Ces procédures ont pour rôle de copier un état antérieur vers la table associée spécifique et de mettre à jour certaines valeurs datées des champs de l'historisation. (date création ou de modification). Dans un contexte client/serveur unique ces procédures permettent d'établir un historique cohérent de l'évolution de la base de données. On verra que la gestion de la concurrence entre utilisateurs d'une même base entraîne nécessairement une complexification du système et fera intervenir des processus externes, trop complexes pour être intégralement portés sous forme de triggers en raison de leurs limitations (effet « cascade »).

```
CREATE TRIGGER insert_trigger
BEFORE INSERT
ON table_name
FOR EACH ROW
EXECUTE PROCEDURE insert_nouvel_objet();
```

```
CREATE OR REPLACE FUNCTION insert_nouvel_objet()
RETURNS trigger AS
$BODY$
BEGIN
NEW.date_de_creation = current_timestamp;

RETURN NEW;
END;
```

Illustration 29: Implantation et corps de la procédure automatisée (trigger) déclenché lors d'une insertion sur une table de la BDD

```
CREATE TRIGGER update_trigger
BEFORE UPDATE
ON table_name
FOR EACH ROW
EXECUTE PROCEDURE historise();
```

```
CREATE OR REPLACE FUNCTION historise();
RETURNS trigger AS
$BODY$
BEGIN
-- Met à jour le champ permettant de vérifier la sync.
NEW.date_de_modification = current_timestamp;

-- Copie l'intréalité des champs de l'objet avant UPDATE
-- Vers la table correspondante du schéma historique

INSERT INTO historique.tabel_name(xxxx, yyyy,[...])
VALUES (OLD.xxxx, OLD.yyyyy, [...]);

RETURN NEW;
END;
```

Illustration 30: Implantation et corps de la procédure automatisée (trigger) déclenché lors d'une mise à jour effectué sur un objet appartenant à une table de la BDD

Leur exécution est explicitée par le diagramme d'activité supportant les mécanismes déclencheurs spécifiques au processus d'historisation (présenté ci-après en Illustration 31).

Il n'existe pas de trigger spécifique à des actions de type **DELETE**. En effet, il n'existe pas de suppression à proprement parler dans ce contexte d'historisation. Ainsi, les actions éventuelles de suppression d'objets par un utilisateur ne sont pas permises en tant que telles, mais modifiées en transactions **UPDATE** spécifiques. Elles répondent donc toujours au trigger correspondant de mise à jour. Le champ **d'état** dans la table des objets actuels fait référence à une éventuelle suppression de l'objet. La dernière version de celui-ci est donc toujours accessible dans la base de donnée mais le champ d'état renseigne d'éventuelles applications clientes lors des sélections réalisées par l'utilisateur.

L'objet actuel est intégralement dupliqué, indexé et interrogable pour future référence ou en cas de réfraction de l'information.

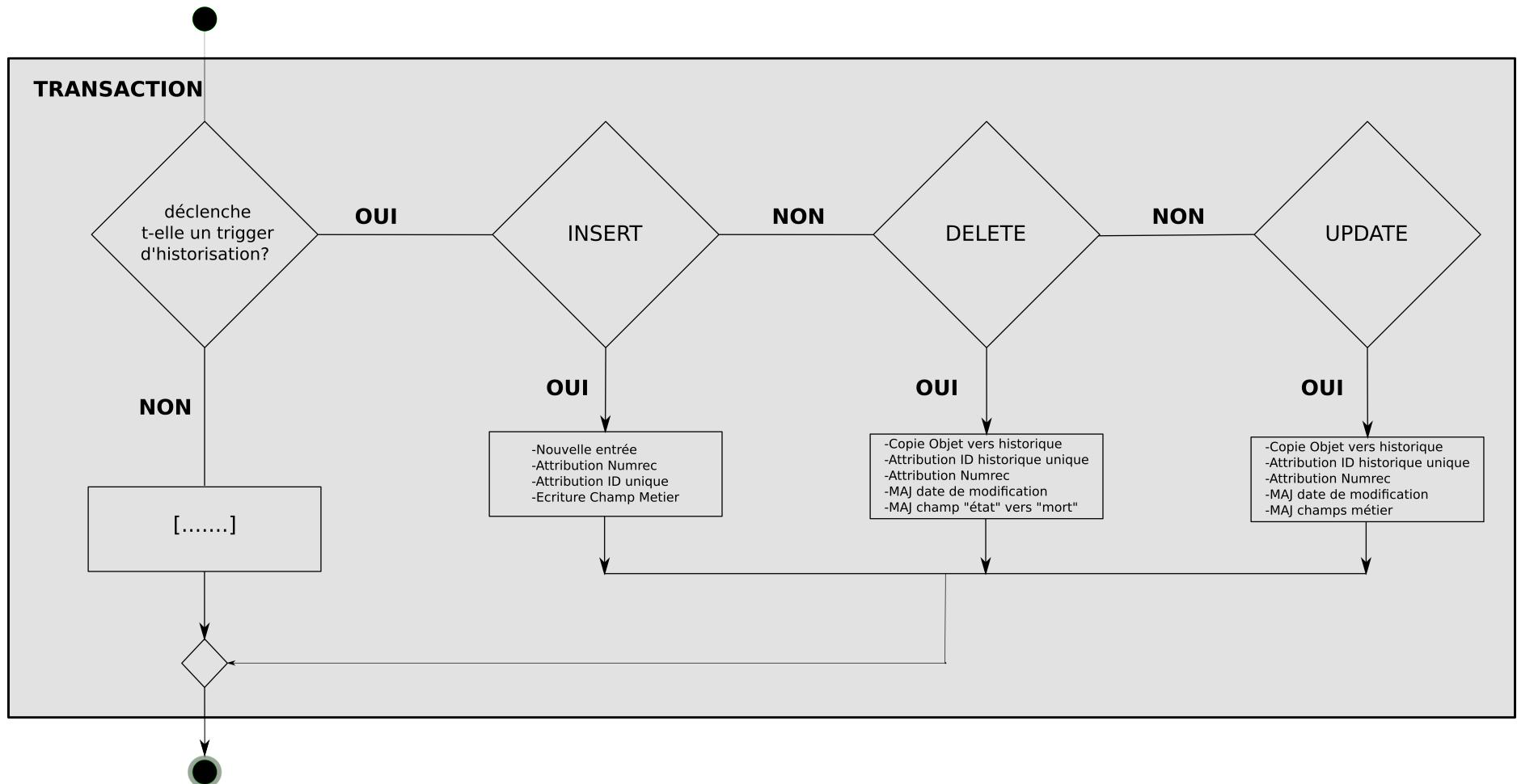


Illustration 31: Diagramme d'activité schématique du déclenchement des « Triggers » PostgreSQL spécifique à l'historisation lors d'une transaction

IV. Mise en place d'un système d'historisation – accès et gestion de la concurrence multi-utilisateur

a. Conceptualisation

La gestion de la concurrence entre utilisateurs multiples d'une même base de donnée a lieu sur plusieurs niveaux :

- Concurrence d'accès simultanés aux ressources du serveur (isolation des transactions SQL).
- Concurrence sur la modification d'objets partagés (Désynchronisation de ses sions utilisateur exécutée localement)

b. Isolation des transactions – Politique du serveur

Le niveau d'isolation pour le serveur SIG40 est défini en tant que « **Read Committed** » qui propose un niveau d'isolation partiel entre transactions concomitantes.

Les requêtes **SELECT** simples ne retournent uniquement les données « **committed** » à l'instant de la sélection tout en restant capable de voir les changements encore non appliqués par elle-même.

Les requêtes en **UPDATE/DELETE/etc..** prennent le même comportement mais sont mises en attente si une transaction est en cours d'exécution avant de s'exécuter elle-même (les modifications sont alors appliquées sur la version mise à jour des objets concernés et les clauses **WHERE** sont réévaluées).

c. Isolation des transactions – implémentation

Le niveau d'isolation « **Read Committed** » ne satisfait pas complètement les besoins d'isolation lors du processus de réconciliation. En effet, une réconciliation nécessite l'exécution d'une série de transactions qui laisse la possibilité d'édition concurrentes sauvage liée à l'empilement de requête en concurrence sur le même objet (re présenté en Illustration 32 ci-après). Il est nécessaire dans ce cas d'augmenter une requête en écriture d'une condition sur la date de modification.

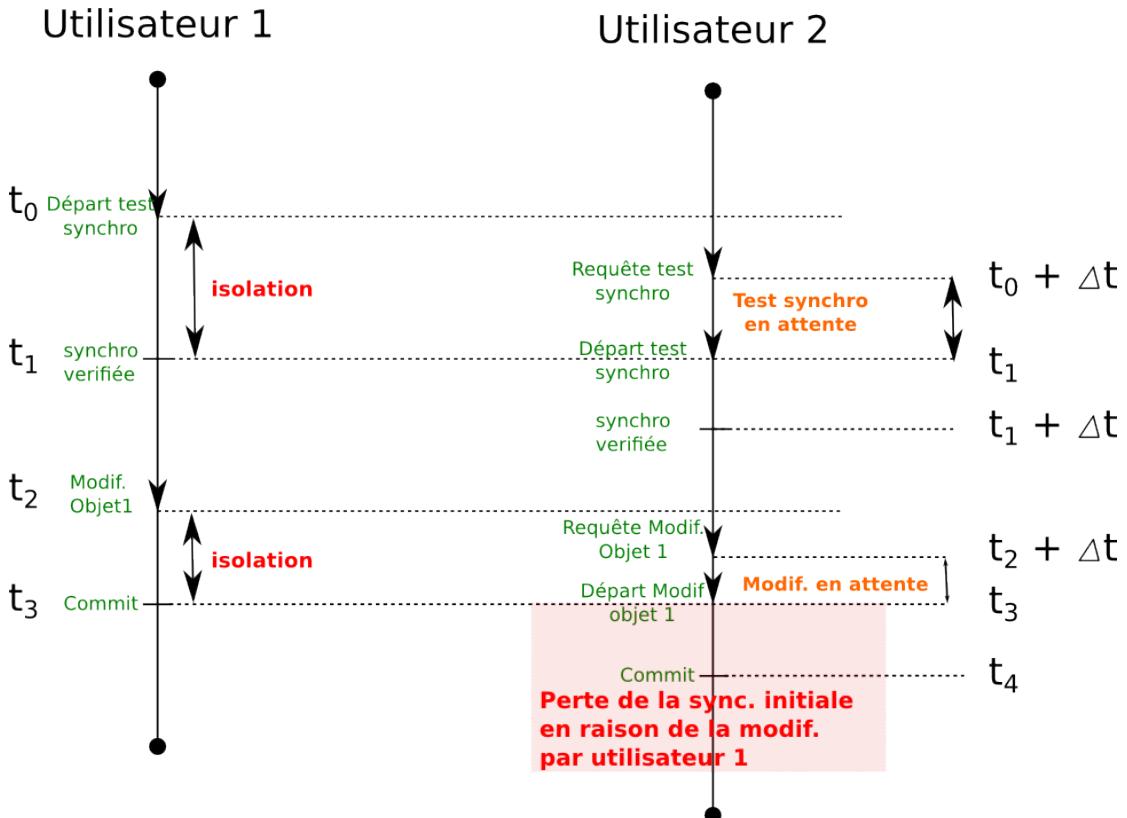


Illustration 32: Mise en concurrence et édition sauvage de l'objet 1 par un utilisateur. L'utilisateur 2 a demandé une vérification de l'état de synchronisation avant l'édition de l'objet par l'utilisateur 1 entraînant une désynchronisation non détectée

Si deux utilisateurs cherchent à modifier un même objet de manière concomitante, il existe une possibilité pour que la synchronisation pourtant testée en début de réconciliation soit perdue avant la transaction UPDATE suivante. Il s'agit du cas où un utilisateur 2 démarre une réconciliation avant l'exécution de la mise à jour effectuée par un utilisateur 1, premier arrivé. Il est donc nécessaire de planifier une procédure de vérification supplémentaire avant de démarrer l'**UPDATE** afin de vérifier et de s'assurer encore une fois de la synchronisation.

La comparaison des dates de modification est donc promue en tant que condition d'exécution d'une éventuelle mise à jour au sein même de la transaction pour s'assurer de l'isolation.

D'un point de mise en œuvre, cette requête est exécutée par le client sous la forme d'une injection d'un bloc de code anonyme de manière à pouvoir faire usage d'une requête conditionnelle non SQL (usage de IF non supporté par les spéc. SQL)

```

DO $$
    DECLARE sync BOOLEAN ;
    BEGIN
        sync = false ;
        --Test de la synchronisation au sein même de la transaction
        UPDATE
            IF (SELECT date_de_modification FROM table WHERE
            [condition,...]) =      date_de_modification_locale
            THEN
                sync =true ;
                UPDATE [...]
            END IF ;
        END$$

Illustration 33: Code Pseudo-SQL introduisant une vérification de la synchronisation comme condition requise pour effectuer une mise à jour (implanté en utilisant une fonction de bloc de code anonyme Postgresql)

```

d. Protection des champs et restriction des possibilités d'édition

Certains champs propres à l'historisation des données sur le serveur SIG40 n'ont pas vocation à être édités (mais présentent un intérêt à être exposés) et certains objets ne sont accessibles qu'en lecture seule selon le niveau de privilèges de l'utilisateur.

La réalisation de ces sécurités est effectuée par la politique de transmission de priviléges gérée directement par le serveur. Néanmoins, pour éviter de multiplier les éventuelles erreurs (rejet de transactions) lors du processus de réconciliation, les champs concernés et les droits utilisateurs sont analysés lors de l'intégration des tables aux projets et le gestionnaire du rendu est invoqué pour poser des blocages supplémentaires pour le confort d'utilisation (cette stratégie reste néanmoins non sécurisée, c'est à dire réversible et donc susceptible de mettre le code en échec). Le typage personnalisé de champ fait également l'objet de restriction, en proposant des listes prédefinies de valeurs à l'utilisateur.

e. Mécanisme d'édition concurrentielle des données partagées

L'historisation automatisée des modifications est réalisée par le déclenchement de procédures automatisées présentées précédemment. Il est néanmoins nécessaire d'aug-

menter cette approche par le développement d'une stratégie permettant une gestion concurrentielle des accès et modifications sur la base données. Cette gestion repose sur une analyse comparative **de l'état des objets** sur le serveur avec des objets modifiés et/ou créés localement.

i. Extraction (ie : Migration PostgreSQL vers SQLITE local)

L'extraction des données du serveur est réalisée par une série de requêtes **SELECT** répondant au niveau d'isolation de transactions appliquées sur la base de données (dans le cas d'un serveur Postgres, il s'agit d'un même niveau d'isolation sur toutes les bases portées par un même serveur).

Une copie locale des ces données est réalisée sur le disque du client en respectant (et indexant) les structures des tables serveur. Il s'agit d'une migration partielle des données vers des bases de données *SQLITE* étendues par les fonctions géographiques *SPATIALITE* pour conserver les propriétés spatiales des objets copiés.

Les tables copiées sont étendues par une série de champs propres à l'historisation locale et de « *triggers* » spécifiques aux manipulations réalisées par un utilisateur au cours d'une session d'édition. Ces champs sont au nombre de 5 :

- **userMod** (Booléen) – Vrai si l'objet a subi une modification
- **userAdd** (Booléen) – Vrai si l'objet a été ajouté
- **userDel** (Booléen) – Vrai si l'objet a été supprimé
- **Zrlocal** (Entier) – L'index de rattachement à une zone de réconciliation
- **ZrNew** (Entier) – Réservé pour calcul

« Générer automatiquement des valeurs sur les champs spécifiques de l'historisation ».

Ces champs spécifiques à l'historisation sont entièrement dépendant des « triggers » implantés dans chacune des tables. L'indexation des modifications est générée à partir de l'analyse de ces champs lors du processus de réconciliation (présenté ci-après). L'exécution des « triggers » est implicite et irrémédiable lors de transaction **INSERT / UPDATE / DELETE** avec la base de données. La finalité de leur exécution est de générer automatiquement des valeurs sur les champs spécifiques de l'historisation. Le remplissage des champs d'historisation est donc dynamique, les actions successives d'édition sont susceptibles de modifier plusieurs fois l'état des champs d'historisation.

« Effet de « cascade »

Il faut ajouter que l'exécution des « triggers » n'est pas ordonnée et sujette à un effet de « cascade » (représenté ci-après en Illustration 34), c'est à dire au déclenchement d'un « trigger » par un autre. L'écriture des procédures associées tient compte de cet aspect, leur nombre doit être réduit à un minimum et des conditions d'exécution peuvent/doivent être stipulées pour éviter de générer des inter-dépendances trop complexes. Enfin, il doit être noté que dans le cadre de l'utilisation d'une base de données *SQLITE*, les opérations **INSERT** et **DELETE** déclenchent de fait les « triggers portés » par des opérations **UPDATE**.

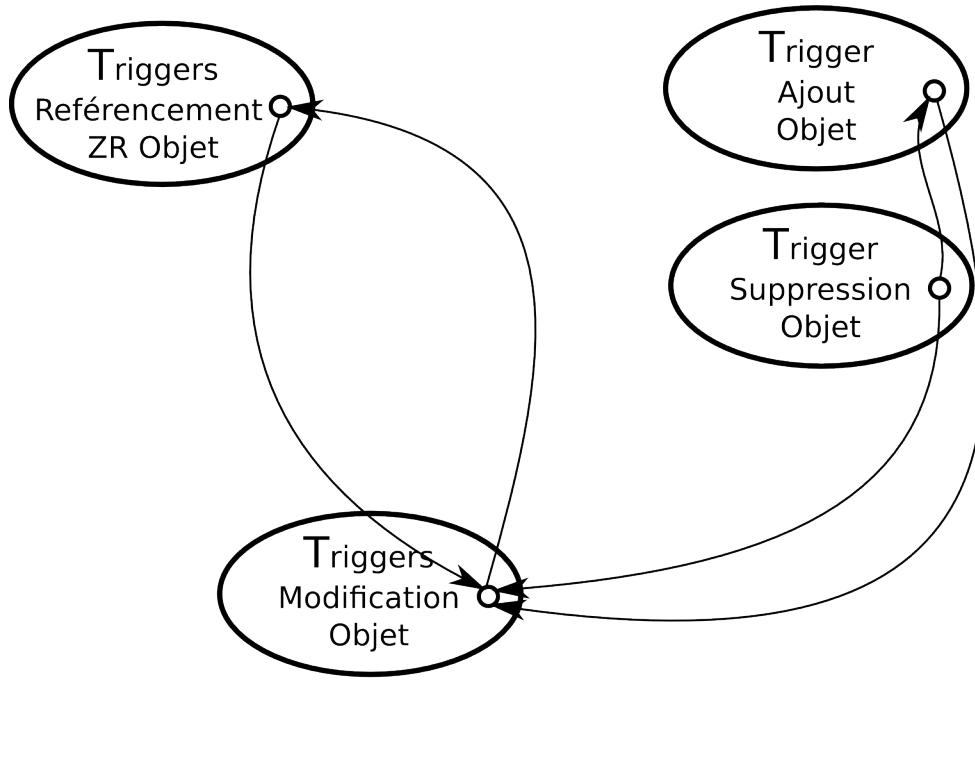


Illustration 34: Représentation des cascades de Triggers lors d'édition de la base *SQLITE*

Les algorithmes des triggers sont présentés ci-après sur les illustrations 35 et 36 en code pseudo SQL. Ces procédures sont créées spécifiquement pour chaque table des bases de données *SQLITE*, et certaines sont amenées à être réinitialisées. En effet la procédure de référencement des objets au sein des zones de réconciliations peut-être

étendue ou réduite au grès de l'ajout ou de la suppression de zones de réconciliation par l'utilisateur.

```
CREATE TRIGGER userMod_trig AFTER UPDATE ON db_name
BEGIN
    UPDATE table_name SET userMod = new.ZRnew % 2, ZRnew = old.ZRnew
    WHERE rowid = new.rowid AND new.ZRlocal != -999;
END;
```

```
CREATE TRIGGER userMod2_trig AFTER UPDATE ON db_name
BEGIN
    UPDATE table_name SET userMod = new.ZRnew %2, ZRnew = 1
    WHERE rowid = new.rowid AND new.ZRlocal = -999 ;
END;
```

Illustration 35: Implantation et corps des triggers SQLITE exécutés par des transaction UPDATE

```
CREATE TRIGGER userAdd_trig AFTER INSERT ON db_name
BEGIN
    UPDATE table_name SET userAdd = 1
    WHERE rowid = new.rowid
END;
```

```
CREATE TRIGGER userDel_trig BEFORE DELETE ON db_name
BEGIN
    INSERT INTO table_name ([liste des champs métier], userDel, original_pk, ZRlocal )
    SELECT [liste des champs métier],1, old.pk, old.ZRlocal FROM table_name
    WHERE rowid = old.rowid;
    UPDATE table_name SET userDel =1
    WHERE rowid = old.rowid;
END;
```

Illustration 36: Implantation et corps des triggers SQLITE exécutés par les procédures INSERT et DELETE

De manière à encadrer « l'effet de cascade », un champ dédié **ZrNew** permet d'empêcher de marquer un objet comme **modifié** lors de son référencement au sein d'une zone de réconciliation. En effet, le référencement passe nécessairement par une procédure **UPDATE** qui déclenche sa propre cascade de « trigger ». Or, le référencement ne doit pas être vu comme une modification de manière à limiter les écritures sur le serveur lors de la réconciliation (et limiter ainsi les possibilités de désynchronisation avec les objets sur le serveur distant).

```

CREATE TRIGGER MoveOutOfZR_trig AFTER UPDATE ON db_name
BEGIN
UPDATE table_name SET userMod=1, ZRlocal = -666"
WHERE
  (NOT st_intersects([géométrie de l'objet],[géométrie d'une Zone de réconciliation])
   AND rowid = new.rowid);
END;

```

```

CREATE TRIGGER MoveInsideZR_trig+ZRnumber AFTER UPDATE ON table_name
BEGIN
UPDATE tabel_name SET ZRlocal = ZRnumber
#If the current feature geometry is not null
WHERE isEmpty[géométrie d'une Zone de réconciliation] = 0
  AND st_intersects([géométrie de l'objet],[géométrie d'une Zone de réconciliation])
   AND rowid = new.rowid;
END;

```

Illustration 37: Implantation et corps des triggers SQLITE spécifiques au référencement des objets au sein de zones de réconciliation, exécutés par les procédures UPDATE

Le diagramme (Illustration 38) ci-dessous présente les actions engendrées par l'ajout d'une nouvelle zone de référencement au projet.

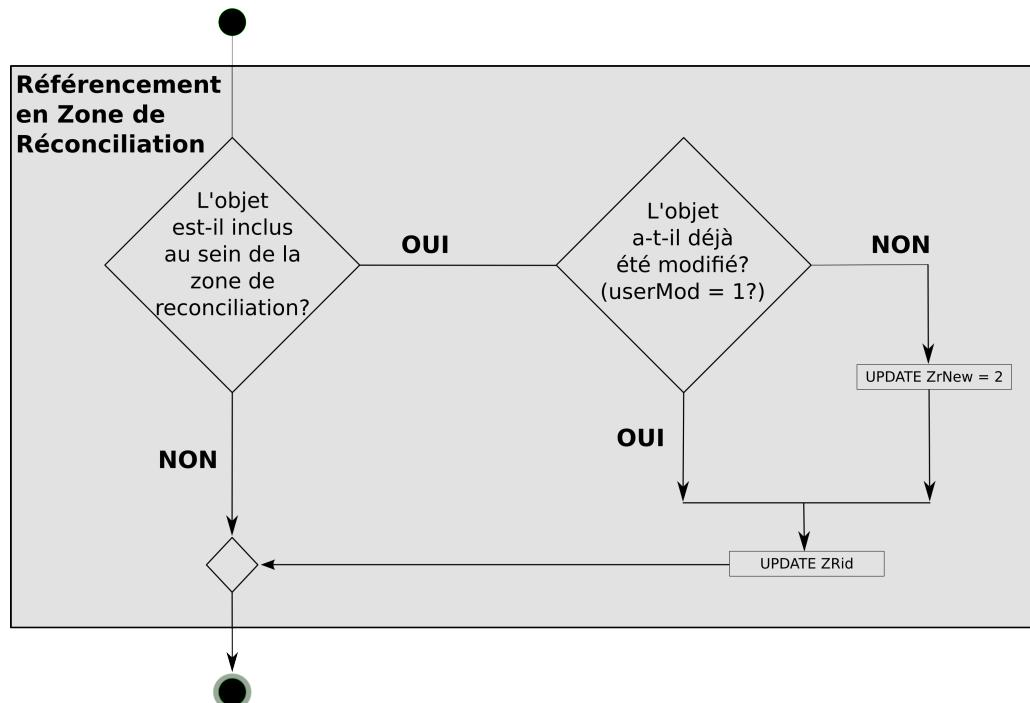
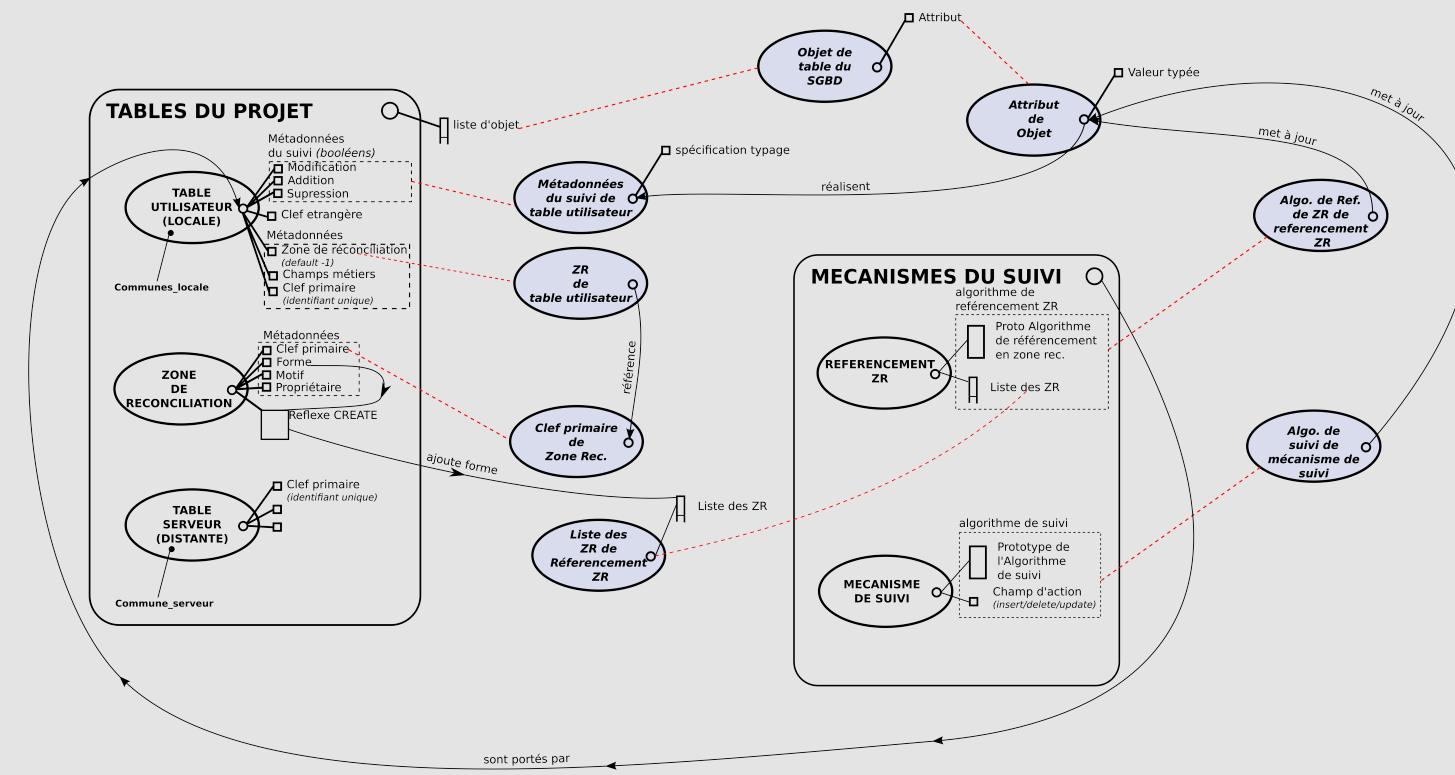


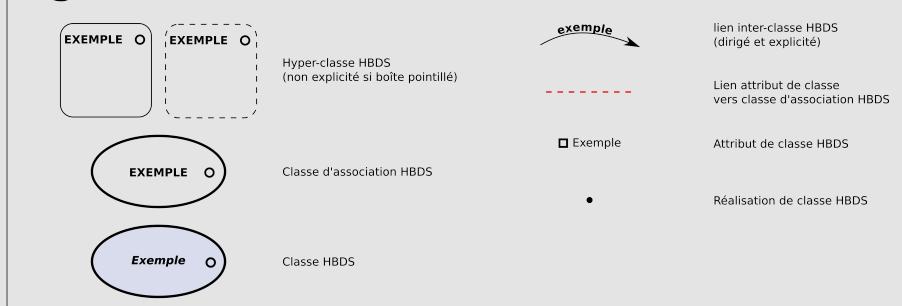
Illustration 38: Diagramme d'activité - Référencement des objets lors de l'ajout d'une Zone de Réconciliation (ZR).

Le champ **ZrNew** est mis à jour si l'objet est vierge de toute modification préalable. Ainsi, les conditions associées au « trigger » de mise à jour permettront de conserver la valeur **userMod** à son état initial si requis⁶.

⁶ Les algorithmes de "trigger UPDATE" calculent la valeur **userMod** sur la base **ZrNew Modulo 2**



legende:



STRUCTURATION DU MECANISME DE SUIVI DES MODIFICATIONS UTILISATEURS

Illustration 39: Modélisation HBDS du mécanisme de suivi des modifications utilisateurs

ii. Édition

Une session d'édition peut-être démarrée, interrompue et reprise. Toute action d'édition est permise au sein des tables chargées au démarrage d'un projet. En revanche un certain nombre de limitations doivent être posé :

- **Aucun ajout de table** ne saura être pris en compte au moment de la réconciliation avec le serveur. Il reste cependant possible d'adoindre des couches supplémentaires au projet sans altérer son intégrité.
- **Aucun ajout de champ** (colonnes) aux tables du projet ne sera pris en compte. Il s'agit de plus d'une action a priori rendue impossible de part les propriétés des bases de données *SQLITE*.
- Certaines modifications de valeurs de champs sont interdites. Il reste possible de contourner ces limitations mais aucune modification ne sera prise en compte.
- Le typage des objets doit être respecté.
- Un projet réconcilié (avec succès) ne peut plus être réutilisé. Il faudra alors créer un nouveau projet.
- Renommer les tables est interdit.

Au cours d'une session d'édition, l'utilisateur doit sélectionner les objets modifiés pour les inclure au sein de zones de réconciliation (classe d'indexation présenté en E.I et E.III.b). Ces zones d'indexation ont un objectif double :

- Limiter le nombre d'objets dont la synchronisation devra être vérifiée.
- Renseigner les métadonnées de la modification. C'est dire le/les motifs de modifications et inscription des informations projet(utilisateur et nom de projet).

iii. Réconciliation

Une réconciliation peut présenter plusieurs états selon son exécution :

- **Réussie** – Toutes les modifications sont inscrites sur le serveur. Le projet est gelé localement. Aucune modification ultérieure ne peut être admise. Un nouveau projet est requis.
- **Échec** – Des erreurs sont présentes (typage, non référencement ; désynchronisation des objets détectée). À ce stade du projet, seules les erreurs liées au typage des objets peuvent être corrigées directement par l'utilisateur avant de tenter une nouvelle synchronisation avec le serveur. Des conflits impliquant

une désynchronisation avec les objets de la base sont relevés et explicités en pointant les objets et les champs concernés ainsi que l'utilisateur avec lequel le conflit est levé. Une résolution manuelle est requise.

- **Partielle** – Certains objets non pas été référencés mais d'autres sont corrects et inscrits au serveur. Les objets oubliés doivent être réédités au sein d'un nouveau projet.

V. Développements

Le prototype de plugin réalisé répond aux concepts et spécifications présentés dans ce rapport. Il est fonctionnel mais présente quelques défauts et limitations qu'il s'agira de dépasser pour plus de robustesse à l'exécution et de souplesse pour l'utilisateur.

1. La structuration de la base de données Postgres est manuelle. L'ajout d'une table requiert le montage manuel de l'historisation (champs et procédures automatiques). Une fonctionnalité de montage automatique devrait être implémentée. Cet aspect pourrait probablement être développée en faisant usage des fonctionnalités objets de *PostgreSQL*, en particulier le concept d'héritage⁷ qui pourrait être appliqué à une table des champs de l'historisation.
2. Les triggers PostgreSQL sont explicités manuellement. Une procédure de montage automatique à partir des catalogues systèmes devrait être développée. Il s'agit de s'abstenir de l'entretien de ces procédures.
3. Un projet réconcilié devrait pouvoir être réutilisé tel-que pour ajouter de la souplesse à l'utilisation. Ce développement permettrait de mieux gérer les projets évoluant sur des agendas longs.
4. Un module de résolution des conflits de synchronisation devrait être développé pour permettre à un administrateur d'appliquer ces changements une fois le conflit résolu.
5. Un module de réfraction pour l'administrateur.

Enfin le volume de données dans ce système est appelé à augmenter rapidement. Une stratégie de sauvegarde incluant la gestion d'une historisation doit être développée pour purger les tables historiques d'informations obsolètes et/ou périmées

⁷ <http://www.postgresql.org/docs/9.3/static/tutorial-inheritance.html>

F.MANUEL D'U'TILISATION DU PLUGIN SIG40

Le Plugin sig40 se définit comme une extension du logiciel Qgis. Il est compatible avec les versions ≥ 2.2 de Qgis. Son installation n'est pas permise aux travers des dépôts officiels. L'ensemble des fichiers sources constituant le **plugin** doit être simplement copié dans le répertoire dédié au stockage des **plugins** (`../.qgis2`). A terme, le plugin devrait être distribué sous la forme d'une archive auto-installable. Un redémarrage du logiciel Qgis est requis pour recharger l'extension. Il est également recommandé (de manière optionnelle) de charger l'extension **Plugin Reloader** à partir des dépôts officiels. Cette extension permet de recharger le plugin en cas de comportement erratique.

I. Installation

L'installation consiste en la copie des fichiers sources et ressources du plugin dans le répertoire dédié à cet effet. Il s'agit sous environnement Windows de l'espace réservé à l'adresse suivante (dans le cadre d'une installation par défaut de *QGIS*) :

`c:\users\nom_utilisateur\.qgis2\python\plugins`

Une fois copié dans ce répertoire, le plugin sera automatiquement chargé au démarrage de *QGIS*. Une erreur au chargement du plugin indique probablement l'utilisation d'une version incompatible de *QGIS*. Contactez donc votre administrateur système pour résoudre le problème. Le plugin est recommandé pour la version 2.4. Mettez à jour votre version de *QGIS* *le plus rapidement possible*.

II. Contexte

Le **Plugin** (« extension ») **sig40** est destiné à accompagner l'édition des données stockées sur la base de données SIG du Département des Landes. Cette extension permet de **se conformer aux mécanismes d'historisation des modifications**, de reconnaître les modifications effectuées par les utilisateurs, de **valider les modifications** dans un contexte de **partage de l'information entre services** et de **reconnaître d'éventuels conflits** qui impliquent une mise en relation entre agents.

Il s'agit **d'extraire une vue restreinte de la base de données complète** (géographiquement et selon le niveau de priviléges de l'utilisateur) et d'en effectuer une copie sur le poste local le temps requis par le projet d'édition.

Historisation des modifications

Ensemble des mécanismes d'indexation des données permettant de suivre l'évolution de l'état des objets stockés dans la base de donnée incluant :

- Structuration de métadonnées dédiées
- Gestion de la concurrence entre utilisateurs

III. Séquence d'utilisation typique

Une séquence d'utilisation est axée autour de 5 actions prédéfinies, dont l'application est laissée au contrôle de l'utilisateur. Une description fonctionnelle des actions est présentée en paragraphe IV ci-après en simulant une session de création et d'édition de projet.

Une séquence d'utilisation type s'articule autour de :

1. Ouverture d'un projet
2. Définition d'une zone géographique et des jeux de données
3. Extraction des données vers le poste local et optionnellement définition d'une zone de réconciliation a priori.
4. Édition des données et définition de zone de réconciliation a posteriori
5. Réconciliation des données avec le serveur

La phase d'ouverture de projet permet d'établir une connexion sur le serveur, de découvrir et de sélectionner une base de donnée parmi celles disponibles. Le programme constitue alors les répertoires au sein desquels seront écrites l'intégralité des données demandées et nouvellement créées au cours d'une session d'édition.

Ouverture d'un projet

Un projet ne pouvant pas être redéfini une fois créé, il s'agit de bien définir le cadre de l'édition à venir.

La définition d'une zone géographique permet la sélection jeux de données à extraire parmi toutes les couches disponibles sur le serveur et d'établir les limites d'extension des jeux de données à extraire (définition d'une *zone cliente*) .

L'extraction des données consiste à réaliser une copie des jeux de données sur le poste local (au sein du répertoire *./current* du directoire).

Optionnellement il est possible dès ce stade du projet de définir une ou plusieurs *zones de réconciliation* qui engloberont les objets à éditer au cours du projet.

La phase d'édition des données peut-être réalisée en utilisant toutes les fonctions d'édition de Qgis (accès aux tables vectorielles, outils de sélection et de dessin, ...)

Important ; Edition des données :

les couches extraites lors de la phase de création du projet sont les seules qui se verront remonter en base (synchroniser) lors de la phase de réconciliation.

Il n'est pas possible d'ajouter ou de supprimer une table du projet dans un contexte d'historisation.

Il est en revanche possible d'ajouter une ou plusieurs couches (vecteurs, raster,...) à **titre de référence**, mais **ces données ne seront pas renvoyées vers le serveur (toute édition sera alors perdue)**

La phase de réconciliation des données consiste à réécrire sur le serveur les modifications réalisées au cours de la session d'édition. Les mécanismes de validation et de résolution de conflits sont invisibles à l'utilisateur, aucun paramétrage n'est requis. Un rapport de réconciliation est émis à la fin de la transaction à destination de l'utilisateur et de l'administrateur de la base de données. Ce rapport indique l'état du projet (synchronisé ou projet en conflit) et résume les actions réalisées et les points de conflit le cas échéant.

Ce que ce Plugin n'est pas :

Cette extension n'est pas un outil destiné à l'édition ou à la production de carte thématiques. Son utilisation doit être réservée à des sessions d'éditions de données.

IV. Description des fonctionnalités

Dans le cadre d'un chargement correct du plugin au démarrage de *QGIS*, les fonctionnalités sont accessibles directement dans la barre d'outils telles que présentées en Illustration 40 ci-après. 5 Actions sont possibles en fonction de l'état d'avancement de votre projet. Chaque action est accessible directement au travers de son icône correspondante.

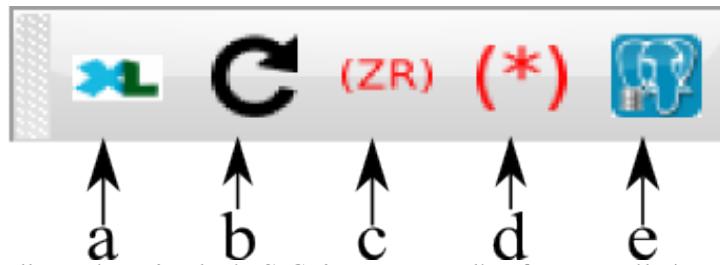


Illustration 40: Plugin SIG40 – Boutons d'accès aux outils (a : ouverture d'un projet ; b : reprise d'un projet ; c : création d'une zone de réconciliation ; d : destruction de zones de réconciliation ; e : réconciliation avec le serveur de donnée distant)

a. *Création d'un projet :*

i. Connexion à une base de données

L'icône **a** (logo du CG) permet d'initier un nouveau projet. La première étape est un tentative de connexion automatique sur le serveur dédié au données SIG. Aucune information n'est à renseigner. Cette étape permet de valider l'ouverture de la connexion.

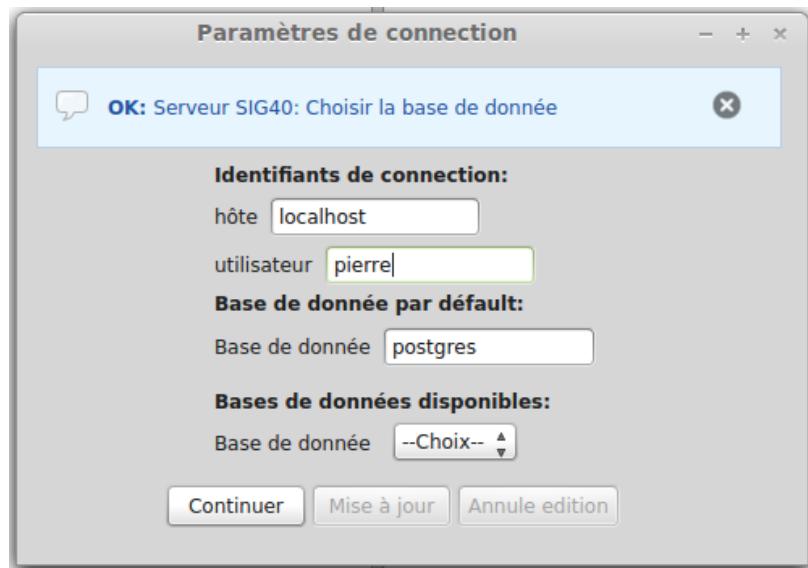


Illustration 41: Fenêtre de connexion - Connexion serveur validée

En cas de problème de connexion, un bandeau d'erreur (Illustration 42) apparaît et distribue un message indiquant l'origine probable de l'erreur. Veuillez transmettre cette information à votre administrateur si vous ne pouvez pas résoudre ce conflit par vous-même.



Illustration 42: Fenêtre de connexion - Connexion serveur impossible

Une fois la **connexion serveur validée**, veuillez choisir une base de données parmi celles disponibles au travers du menu déroulant en base de la fenêtre de connexion. Validez ensuite votre choix en cliquant sur le bouton **de mise à jour**.

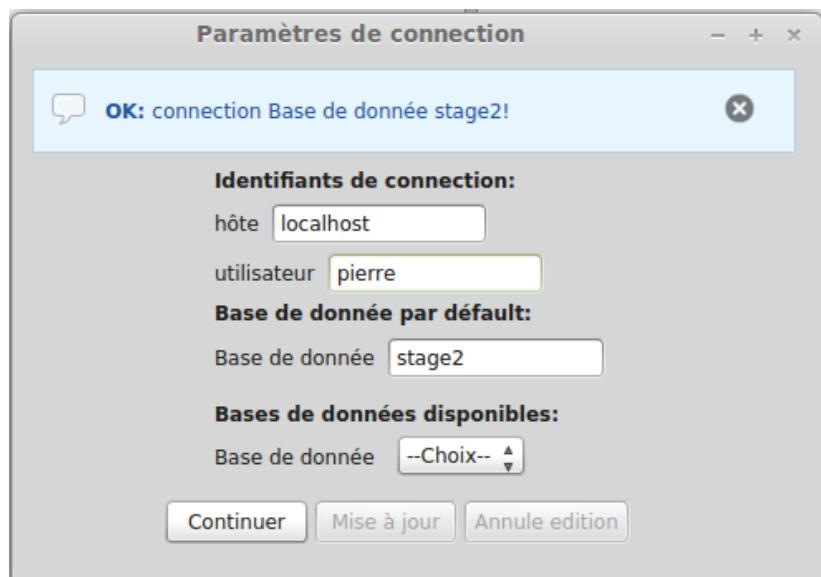


Illustration 43: Fenêtre de connexion - Connexion base de donnée validée

La boîte de dialogue est alors réinitialisée et indique la possibilité ou non de connexion. Cliquez sur **continuer** pour finaliser votre choix ou **réitérer l'étape précédente** en sélectionnant une base de données différente dans la liste déroulante.

Une connexion réussie **est indiquée visiblement dans QGIS par la barre de message** de la fenêtre cartographique comme présentée ci-dessous :



ii. Génération des fichiers du projet

La boîte de dialogue suivante (Illustration 45) s'ouvre alors automatiquement pour permettre de compléter les renseignements requis au montage d'un projet valide. Les étapes de création sont numérotées et la progression est guidée par un système d'activation/désactivation des actions autorisées/interdites.

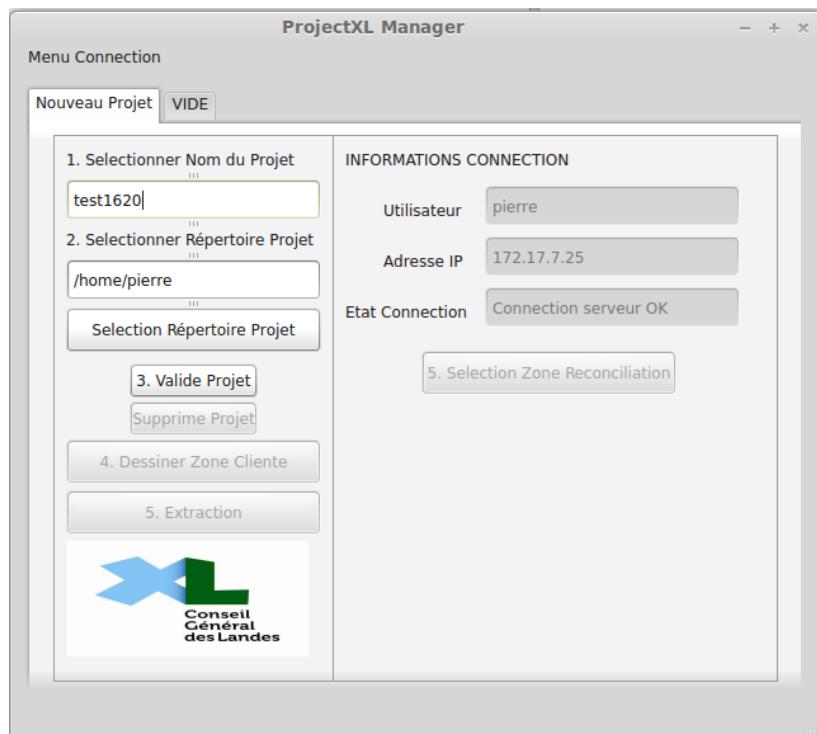


Illustration 45: Fenêtre principale du projet

1. Donnez un nom à votre projet
2. Sélectionnez votre répertoire de travail
3. Validez pour continuer

Les informations de connexion à droite sont affichées en cas de message d'erreur. Votre nom d'utilisateur doit correspondre à votre identifiant de connexion sur le réseau du CG. En cas d'erreur, veuillez transmettre ces informations à votre administrateur.

Notez-bien :

Les valeurs affichées dans les cadres d'information de connexion sont destinées à guider la résolution d'erreur par l'administrateur. Vos droits et priviléges d'utilisation sur la base de données sont définis sur la base du couple nom/adresse IP affiché.

iii. Sélection des données sur le serveur

L'entrée numéro 4 « **4. Dessiner Zone Client** » dans la fenêtre principale du plugin permet de réaliser votre sélection personnalisée sur les données du serveur. Il s'agit de :

1. Restreindre géographiquement l'étendue des données à sélectionner (consiste à dessiner un rectangle)
2. Restreindre les couches à extraire

Immédiatement après avoir cliqué sur le bouton de sélection de zone cliente, un dialogue vous demande choisir une couche pour vous aider à réaliser votre sélection géographique. **Choisissez une couche aux attributs facilement reconnaissable pour support du dessin de la limitation de la zone à extraire.**

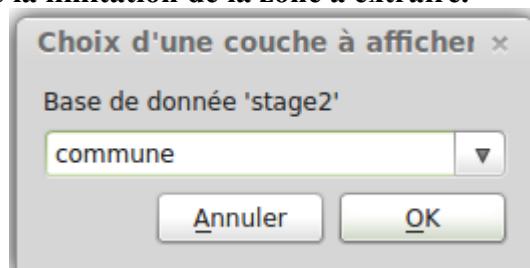


Illustration 46: Dialogue de choix d'un fond de couche pour la sélection de zone cliente

Une fois validé, le **dialogue de sélection géographique** (Illustration 47) est affiché et présente la couche sélectionnée comme fond de carte. Si vous désirez changer le fond de carte, cliquez sur annuler et réitérer l'action précédente.

Le tableau à gauche du dialogue vous permet de choisir les couches à importer en cochant/décochant les boîtes prévues à cet effet. L'icône en surbrillance au dessin de la fenêtre de présentation permet **d'activer l'outil de sélection**. Une fois l'outil activé cliquez en deux endroits distincts de la fenêtre pour définir votre rectangle de sélection. Un **aperçu de la zone sélectionnée** est alors présenté en surbrillance. Validez ensuite votre choix en cliquant sur le bouton **OK**. Il est possible de redéfinir cette zone à volonté en cliquant sur l'icône :

iv. Finalisation de l'extraction

Une fois validé, la boîte de dialogue principale est de nouveau affichée et vous demande de cliquer sur bouton **5.Extraction** pour continuer. Vous pouvez toujours à ce moment décider de revenir sur votre décision et redéfinir la zone de sélection.

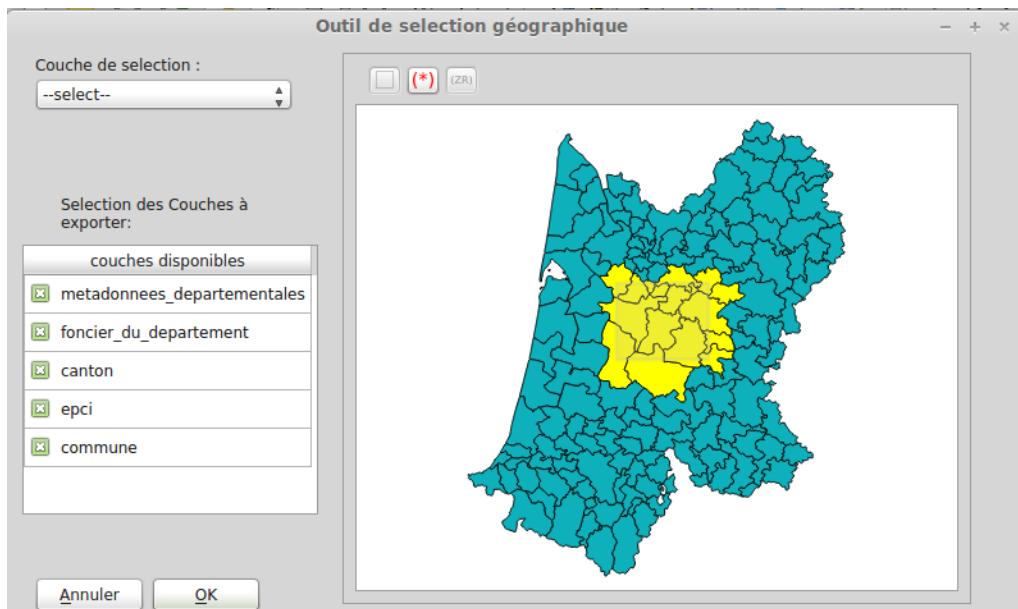


Illustration 47: Fenêtre d'assistance à la sélection d'un extrait de la base de donnée (la sélection géographique) et sélection des couches à importer

La finalisation de l'extraction est une procédure automatisée au cours de laquelle un bandeau de progression est affiché. Selon le « poids » de la requête et la charge du serveur le temps d'extraction peut varier de manière importante le logiciel peut sembler gelé, veuillez laissez le temps de chargement se compléter sous peine de corrompre votre projet.

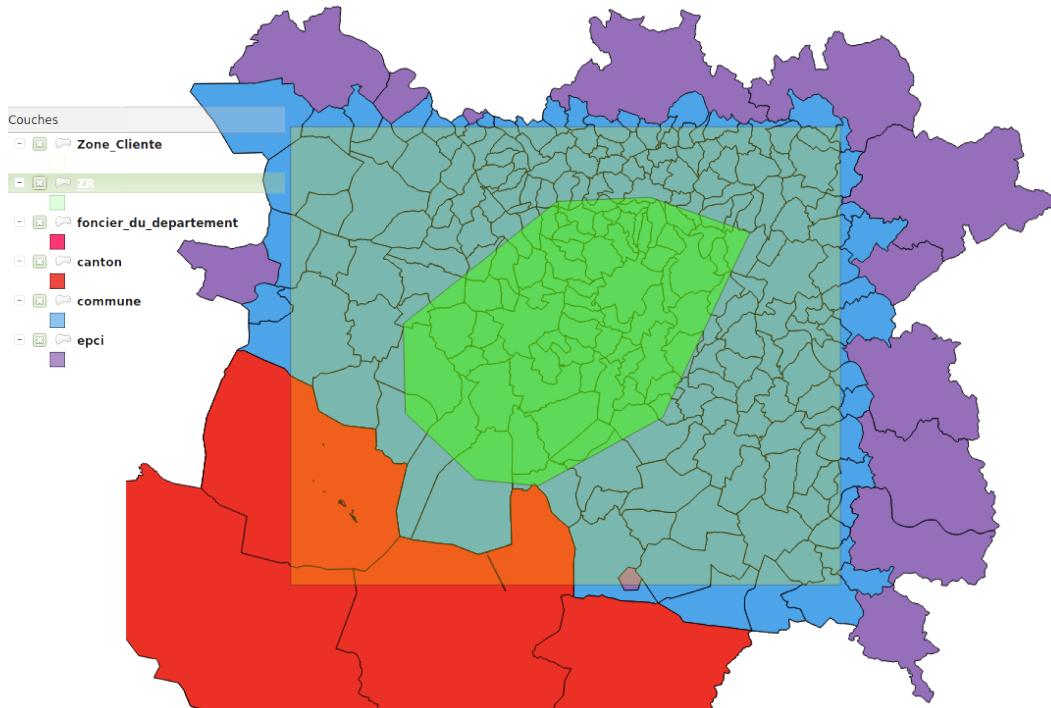
IMPORTANT :

L'étendue de la sélection géographique et le nombre de couches choisies déterminent la charge appliquée au serveur pour votre requête.

Il est important de bien restreindre votre sélection sous peine de temps de chargement important.

Il existe une quantité importante de données sur le serveur, une sélection inconsidérée n'aura aucun effet bénéfique sur votre projet et pénalisera les autres utilisateurs.

A ce stade d'avancement, le projet est finalement chargé et vous êtes prêt à démarrer l'édition.



*Illustration 48: Exemple de projet chargé et prêt à l'édition
b. Session d'édition des données*

Toutes les opérations d'édition sont autorisées dans votre projet incluant :

- Modification d'attributs d'objets préexistant
- Création d'objets
- Suppression d'objets
- Annulation d'action

Il est en revanche nécessaire impérativement de référencer vos objets modifiés au sein de zones de réconciliations pour pouvoir appliquer vos modifications sur le serveur.

- **Ce référencement peut avoir lieu a priori ou a posteriori.**
- **Les zones de réconciliation pourront être supprimées et redéfinies sans aucun dommage**
- **Les zones de réconciliation ne peuvent pas s'intersecter ! Un message vous informera de cette éventualité.**

Le référencement des objets se fait au travers de l'outil de dessin de zones de réconciliation disponible parmi les icônes du plugin (icône c **Illustration 40**). La boîte de dialogue (Illustration 49) propose d'indiquer un ou plusieurs motifs de réconciliation. Cliquer sur le bouton central (marqué **ZR**) pour activer l'outil de dessin. Dessiner un polygone englobant vos objets déjà modifiés ou définissant une zone au sein de laquelle tous les objets seront modifiés.

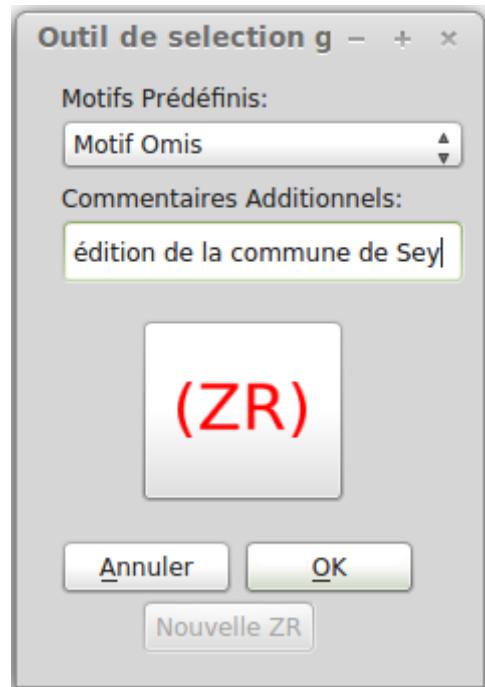


Illustration 49: Boîte de dialogue de création de zone de réconciliation

La suppression de ZR ne peut être réalisée qu'au travers du **module de suppression de ZR**, disponible parmi les icônes du plugin (**icône d Illustration 40**). Une boîte de dialogue permet de sélectionner les différentes ZR inscrites au projet et les mets en surbrillance pour permettre de les identifier sur la carte. Cochez les ZR à supprimer et appliquez (ou annulez) vos choix.



Illustration 50: Dialogue permettant la suppression de zones de réconciliation

c. Fin de session d'édition des données

Une fois vos éditions complétées et enregistrées vous pouvez fermer QGIS en toute sécurité. Votre projet est sauvegardé automatiquement.

d. Reprise d'un projet non complété

Vous pouvez reprendre une session d'édition **uniquement en suivant la procédure suivante**. Démarrez *QGIS*, cliquez sur le bouton de rechargement du plugin (**b Illustration 40**). Naviguez vers le répertoire de votre projet et chargez le fichier QGIS correspondant (fichier **XL_project.qgs**).

e. Réconciliation d'un projet avec le serveur

Une fois vos modifications finalisées et bien référencées au sein de zones de réconciliations (1 au minimum) vous pouvez cliquer sur le bouton de réconciliation (icône **e Illustration 40**).

Il s'agit d'un procédure automatisée qui se finalise par l'émission d'un rapport de réconciliation automatiquement affiché pour l'utilisateur dans son navigateur internet, et l'émission d'un email à l'administrateur du SIG.

Une fois complété, selon la réussite ou non de la synchronisation votre projet est marqué dans des états suivant :

- **COMPLET** – Votre projet est gelé, les modifications ont été inscrites au serveur et aucune nouvelle modification ne sera synchronisable. **Vous devez redéfinir un nouveau projet pour une nouvelle session d'édition.**
- **ECHOUÉ cause rejet base de donnée** – Le rapport de réconciliation vous indique la/les raisons de l'échec de réconciliation. Veuillez contacter votre administrateur pour résoudre le problème si vous ne le pouvez pas vous-même. Typiquement, une erreur de typage ou un champ non complété peuvent empêcher l'écriture. L'objet concerné est indiqué explicitement dans le rapport. Corrigez les objets correspondants et réitérer la réconciliation
- **ECHOUÉ cause conflit**– Le rapport de réconciliation vous indique la source du conflit et l'utilisateur avec lequel vous êtes en conflit (*cela peut être vous-même!*). Veuillez contacter l'utilisateur avec lequel vous êtes en conflit ainsi que votre administrateur pour résoudre le problème. Vous ne disposez pas des droits suffisants pour réaliser une résolution de conflit seul.

G.CONCLUSION

Les spécifications présentées dans ce document ont été mises en œuvre avec succès en structurant et implantant :

1. Un prototype de base de donnée servant un système de versionnement sécurisé et référencé de l'édition de l'information.
2. Un prototype de passerelle d'utilisation permettant un usage sécurisé des données centralisées.

Les principales fonctionnalités ont été testées et permettent la mise en concurrence d'utilisateurs lors de la modification des données partagées du serveur SIG du conseil Général des Landes :

1. La connexion à un serveur de base de donnée
 2. La découverte du contenu de la base de donnée
 3. La sélection d'un extrait de la base de données
 4. la constitution d'un projet par la migration de l'extrait vers un système de base de donnée locale
 5. La mise en place d'un système de référencement des modifications utilisateurs
 6. La synchronisation des données locales vers le serveur distant en gérant les différents états de synchronisation possibles.
 7. La production et la distribution automatique de rapports d'activités lisibles pour la validation et/ou la gestion des conflits.
3. A ce stade du projet, il ne s'agit néanmoins que d'un prototype qui nécessite d'être déployé sous supervision d'un ingénieur au sein d'une communauté restreinte de testeurs/utilisateurs pour valider son fonctionnement. Cette phase de test devra permettre de relever les éventuelles erreurs et incohérences dans l'exécution du programme mais également à suggérer d'éventuels développements et/ou modifications pour améliorer l'expérience à l'utilisation.

Enfin, des développements et suggestion d'altérations du projet ont été émises et devraient constituer les conditions de poursuite du développement de cet outil.

H.ANNEXES

Annexe I: Processus d'historisation de la BDUni

Annexe II: Graphe d'appel général du projet

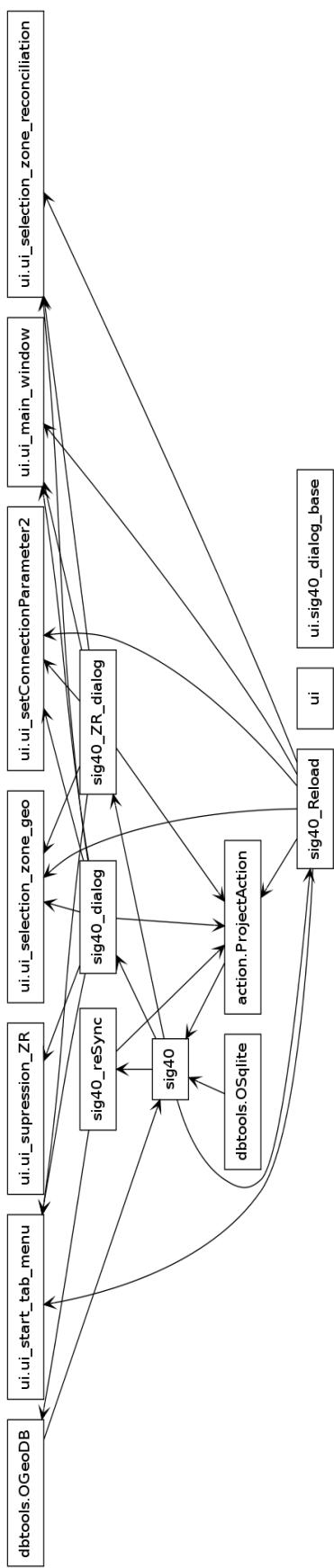


Illustration 51: Graphe d'appel général du projet sig40

Annexe III: Diagramme de classe UML du Projet

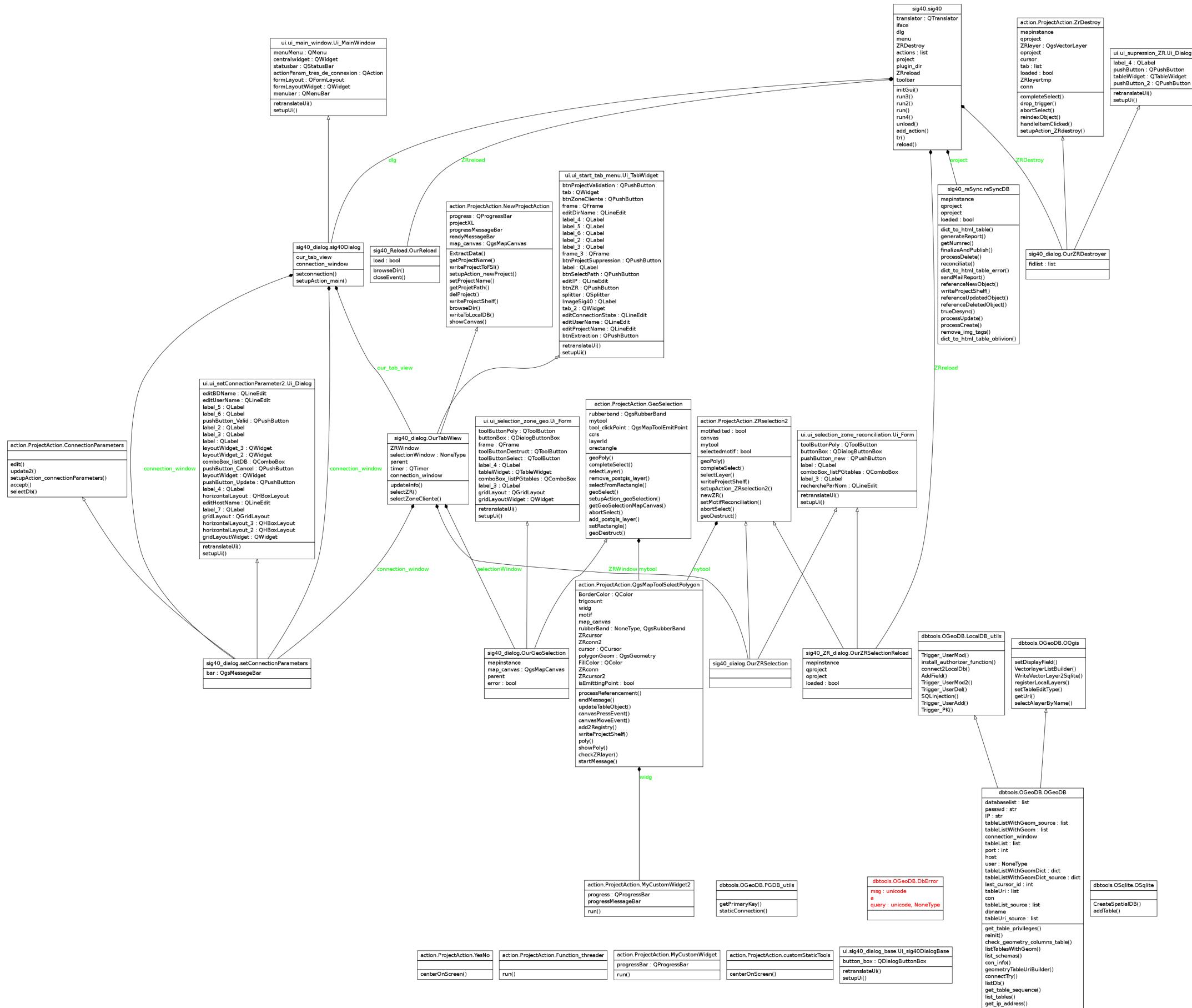


Illustration 52: Diagramme de classe UML du projet sig40 (incluant uniquement les paquetages développés, les bibliothèques importées ne sont pas représentées)

I. Bibliographie

Ressources Web

The PyQGIS Developer Cookbook (2014, en ligne):

http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/

QGIS API Documentation (2014, en ligne):

<http://www.qgis.org/api/>

The PyQt Reference Guide, (2014, en ligne):

<http://pyqt.sourceforge.net/Docs/PyQt4/>

PostgreSQL website, (2014, en ligne) :

<http://www.postgresql.org/>

Lorenzo Alberton WebPage on « extracting META info from PostgreSQL info.schema »,
(2014, en ligne) :

http://www.alberton.info/postgresql_meta_info.html#.VEe88niwdHb

Sqlite website, (2014, en ligne) :

<http://www.sqlite.org/index.html>

The Spatialite CookBook, (2014, en ligne) :

<http://www.gaia-gis.it/spatialite-2.4.0-4/spatialite-cookbook/index.html#desserts>

Shelve – Python Object Persistance, (2014, en ligne) :

<https://docs.python.org/2/library/shelve.html>

Psycopg – PostgreSQL database Adapter for Python, (2014, en ligne)

<http://pythonhosted.org//psycopg2/>

Sqilte3 – Database Adapter interface for SQLite databases, (2014, en ligne)

<https://docs.python.org/2/library/sqlite3.html>

The Dive Into Series, by Mark Pilgrim (2014, en ligne):

<http://www.diveinto.org/>

The Geographic Information Systems Stack Exchange (2014, en ligne)[ht:<http://gis.stackexchange.com/>](http://gis.stackexchange.com/)

Lien vers mon compte :

<http://gis.stackexchange.com/users/29984/peter-peterson>

Une liste de questions qui se sont posée lors du projet !

<http://gis.stackexchange.com/questions/109577/qgis-2-4-python-2-7-pyqt4-qtcore-qstringlist-import-failed>

<http://gis.stackexchange.com/questions/111125/pyqgis-start-stop-editing-with-data-provider-commiting-changes>

<http://gis.stackexchange.com/questions/109411/pyqgis-working-with-spatialite/109417#109417>

<http://gis.stackexchange.com/questions/104333/pyqgis-qgsmaptolemitpoint>

Ressources Documentaires

SHERMAN, Gary (2014). The PyQgis Programmer's Guide; *Locate Press LLC*, 196 pages

IGN, inconnu, 2013, Processus d'historisation de la BDUni et adaptabilité au RGFor, 35 pages