VTL Istat framework – technical analysis

Introduction

The VTL Istat framework is developed with the intention to support Istat in the validation and transformation process through the harmonization of the statistical rules using and the sharing of information inside and outside the statistical institute.

Scope of the document

The scope of the document is to describe the framework technically describing the architecture, the technologies used for the implementation and classes of the different building blocks.

The document is organised in the following section:

- Architecture - an overall view of all the blocks of the framework and their interactions
- Technologies – the technologies used in the development
- Building blocks - a description of the structure of the different building blocks, the classes used and their interactions
- Database component - a description of the databases used to support the functioning of the application
- Mock ups - shows the different screen shots that are part of the application which the complete description can be found in the user manual of the GUI

Architecture

The architecture of the VTL Istat framework is composed by different blocks as shown in Figure1.

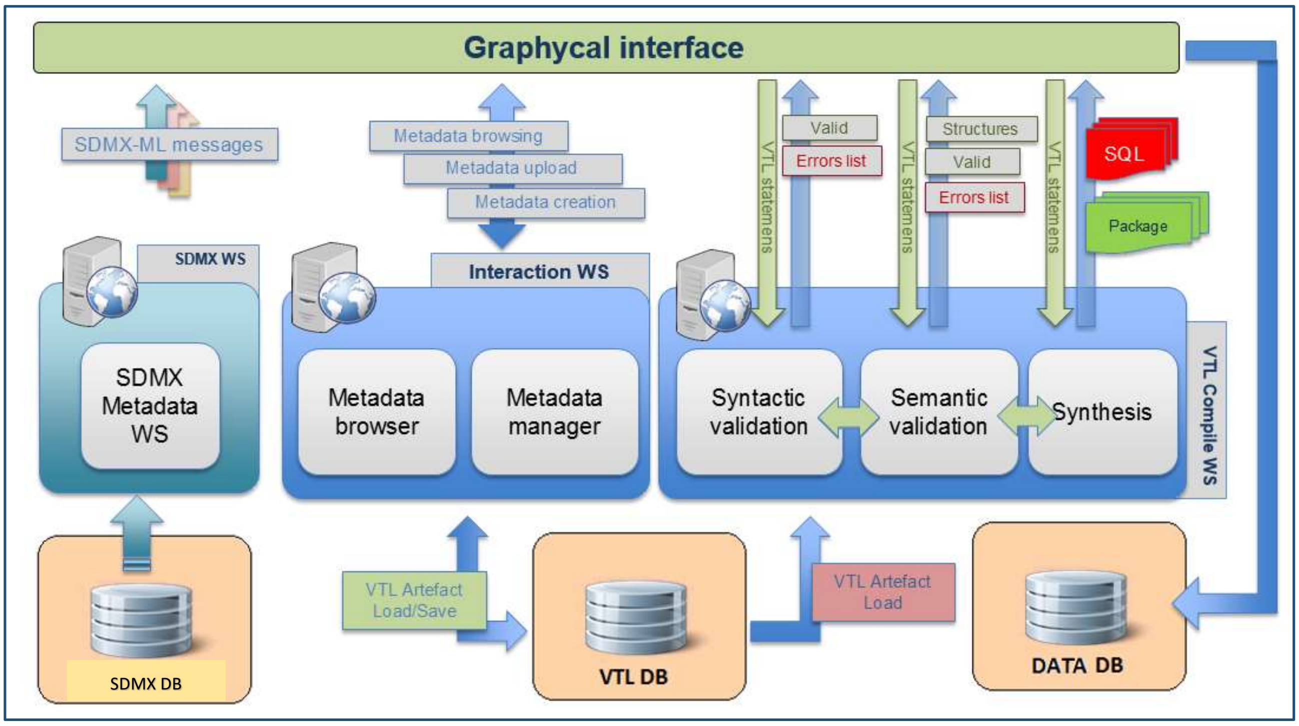


Figure 1 – High level architecture

VTL database (VTL DB): database used by the application. It contains the metadata on which rules are applied and the rules itself stored as *Transformations* and *TransformationScheme.* The database can only be ORACLE.

Database for data (DATA DB): this is not mandatory for the correct functioning of the application but in the architecture represents the place where data are stored. The DATA DB can be: Oracle, MySQL, PosgreSql, SQLServer.

SDMX Metadata WS – web service to get structural metadata. The requests and the responses of the WS with the GUI are in the SDMX standard, this allows to replace the SDMX WS with any other SDMX web service (e.g. SDMX global registry).

Interaction WS - web service interacting with the VTL DB. It retrieves metadata stored from the SDMX WS and metadata created by the GUI. The format used for the request and response is a XML Soap Message.

Ex.: Request to the web method

```
public bool TestDatabaseConnection() {…}
```

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tempuri.org/IService/TestDatabaseC
onnection</Action>
  </s:Header>
  <s:Body>
    <TestDatabaseConnection xmlns="http://tempuri.org/" />
  </s:Body>
</s:Envelope>
```

Web service answer

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <TestDatabaseConnectionResponse xmlns="http://tempuri.org/">
      <TestDatabaseConnectionResult>true</TestDatabaseConnectionResult>
    </TestDatabaseConnectionResponse>
  </s:Body>
</s:Envelope>
```

Methods exposed are:

- TestDatabaseConnection

- GetDataStructureComp
- GetValueDomainValues
- GetDataSets
- GetDataSetComp
- GetDataStructures
- GetValueDomains
- GetValueDomainSubset
- GetSubSetList
- GetTransformation
- GetTransformationList
- GetUserDefinedOperatorsList
- GetValueDomain_seq_id

- InsertTransformation
- RemoveTransformation
- InsertUserDefinedOperator
- InsertValueDomain
- InsertValueDomainDescribed
- RemoveValueDomain
- InsertValueDomainSubsetDescribed
- InsertValueDomainSubset

- RemoveValueDomainSubset
- InsertDataStructure
- InsertDataSet
- RemoveDataSet
- RemoveUserDefinedOperator

VTL Compile WS - web service in charge of the syntactical and semantical validation of VTL rules and consequently translation in SQL. The web service expose the following methods:

- SyntaxCheck
- OrderStatements
- SemanticCheck
- Translate
- Package

The VTL compile is based on Java 1.8 and it is integrated with Spring Boot 2.0.6 having the integrated configuration with Apache Tomcat. It is divided in the following modules:

- Vtl-Api: is composed by controllers, it provides to the client the available interacts and it is in charge to orchestrate the calls to the engine modules;
- Vtl-Parser: to parse the VTL rules using ANTLR (rif. 2.1.1), it is used by the Vtl-Intepreter;
- Vtl-Interpreter: interpreter engine, to interpret the commands received by the client using ANTLR and to generate a Data Structure coherent with the Vtl constraints;
- Vtl-Semantic: semantic engine, to semantically validate the VTL rules received using the Data Structure created by the interpreter engine;
- Vtl-Sqlbuilder: translator engine, collects all the results received by the interpreter and semantic engine to generate SQL instructions ready to be executed;
- Vtl-Common: includes all the VTL artefacts and other structures of types used by the other modules of the application;
- Vtl-Lexicon: contains all the services in charge to syntactically validate the VTL rules;
- Vtl-SdmxBuilder: contains all the services converting the VTL artefacts in SDMX artefact

Technologies

In this paragraph there will be listed the external tolls used to support the development of the framework.

ANTLR

ANTLR (Another Tool for Language Recognition) is a powerful parser generator, to read, elaborate, execute or translate structured texts or binary files. It is used to create languages, tools and framework.

ANTLR takes in input a grammar that specifies a formal language and generates a source code to recognize that specific language. The grammar must be expressed using the Extended Backus-Naur Form (EBNF).

ANTLR generate also lexer, parser, parser tree and combined lexicons-parser. Parsers can generate automatically analysis tree and abstract syntactical tree that can be further elaborated with parser tree. A peculiarity is that ANTLR provides a single coherent notation to specify lexer, parser and parser tree. Usually the parser converts the sequences of tokens that are part of the grammar in another structure called a abstract syntactic tree (AST). An AST is easily translated in a target language because contains implicitly added information due to the nature of its structure. The creation of the AST is the most important part of the language translation process.

The AST read a flow of statements and generates an error if the flow is not correct for the syntax defined by the grammar. If there are no syntactical errors, the defined action is simply to exit returning no messages. To use the language, actions can be attached to the elements of the grammar. These actions can be written in the programming language used to recognise the error and can be used to construct or control tables of symbols or to generate instructions in a target language in case of a compiler.

HyperSQL

HyperSQL (Hyper SQL Database) is a quick a little engine for relational database written in java. It offers tables in-memory and based on hard disk memory.

It includes tools like a web server, a command line and managing tools of a GUI (they can be executed as some applet). It can be executed on runtime Java. The standard conformity is the main characteristic of HyperSQL, it provides the access to the database by the user's application, inside an application server or as separate process. It can be executed completely in memory using fast memory elements dedicated instead of hard disk memory; it uses disk persistence in a flexible way, with a reliable recovery in the event of a crash. HyperSQL

HyperSQL, is the only open source relational database management system with a high performance dedicated lob storage system, suitable for gigabytes of lob data. It is the only relational database that can create and access large delimited files such as SQL tables. HSQLDB has three transaction control modes. Supports read committed and serializable isolation levels with table-level locking or multi-version concurrency control Supports read committed and serializable isolation levels with table-level lock or multi-version concurrency check (MVCC) or a combination of lock and MVCC. For data storage there are two types of main tables for storing long-lasting read-write data, if a transaction has been performed correctly, it is guaranteed that the data will survive the system failure and maintain their integrity. The default type MEMORY stores all data changes on the disk in the form of a SQL script.

When the engine is started, these commands are executed and data is reconstructed in the memory. Another type of table is CACHED, which allows more data to be stored also with lower performance. The HSQLDB engine loads data only partially and synchronizes it on the disk in transaction commits. However, the engine always loads all the lines affected during an update in memory. Other types of tables allow access to files with comma-separated values (CSV). These tables can participate, for example, in queries with JOIN and simplify the spreadsheet processing and the storage of data in no hard disk memory for reading and writing.

SPRING

Spring is a framework that can be used by any Java application. Spring is used to create high-performance, easily testable and reusable code. The main features of Spring Framework can be used in the development of any Java application. The Spring structure makes J2EE development easier to use and promotes good programming practices based on the programming model POJO.

Advantages of using Spring are:

- It Enables you to develop enterprise-class applications using POJO. The advantage of using POJO is that you do not need an EJB container product as an application server but you have the option of using only one servlet container like Tomcat.•
- Is organized in a modular way.•
- It uses some of the existing technologies such as different ORM frameworks, recording frameworks, JEE timers, Quartz and JDK and other technologies for visualization.•
- Test of an application written with Spring is simple because the environment-dependent code is moved into this framework. Furthermore, using POJO JavaBeanstyle, it becomes easier to use dependency injection for test data injection.•
- It provides a convenient API to translate specific technology exceptions (generated by JDBC, Hibernate or JDO, for example) into coherent and controlled exceptions.•
- IoC containers tend to be light, especially when compared to the EJB ones, for example. This is useful for developing and implementing applications on computers with limited memory and CPU resources.•
- It provides a consistent transaction management interface that can scale to a local transaction and scale global transactions.

Spring Boot is a rapid application development (RAD) solution that helps to develop stand-alone and production apps. The applications require a minimal configuration. It also presents a variety of non-functional functions common to large classes of projects. These include built-in servers, security, metrics, integrity checks and external configuration. The tool does not require any code generation and has no prerequisites for XML configuration.

When starting a project, Spring Boot has a Spring Application class that provides a convenient way to start a Spring application. In case of the application does not start, Failure Analyzer offers concrete actions to solve the problem in addition to simply displaying an error message. Application events are sent using the Spring Framework event publishing function, which ensures that an event published for listeners in a child context is also published for listeners in any context.

- Spring Boot allows users to share their configuration to work with the same application code in different environments. Developers can use property files, YAML files, environment variables, and command line arguments. Property values can be injected directly using the Value annotation, available via the Spring's Environment abstraction or associated with structured objects via Configuration properties. The software also uses a specialized property order that allows the values to be cancelled. Another feature is that it is suitable for developing web applications with tools to create an autonomous HTTP server through Tomcat, Jetty, Undertow or Netty. Spring provides support for all the most common Java data access frameworks: JDBC, iBatis / MyBatis, Hibernate, Java Data Objects (JDO), Java Persistence API (JPA), Oracle TopLink, Apache OJB and Apache Cayenne. For all these supported frameworks, Spring provides these features:
- Managing of resources
- Exception handling - translation of the exception related to the data access into a Spring data access hierarchy
- Participation in the transaction - transparent participation in ongoing transactions

- The disappearance of resources: recovery of database objects from connection pool wrappers
- Abstraction for the management of large binary objects (BLOB) and large characters (CLOB)

Spring is the only framework available in Java that offers data access environments managed outside a server or application container.

Apache TOMCAT

Apache Tomcat is an open source web server (in the form of a servlet container) developed by the Apache Software Foundation. Implement JavaServer Pages (JSP) and servlet specifications. It provides a software platform for running Web applications developed in Java language. Its standard distribution also includes the traditional web server features, which correspond to the Apache product.

Tomcat is distributed under the Apache License, and is written entirely in Java; it can, therefore, be run on any architecture on which a JVM is installed. Furthermore, it is not a service that fully implements the Java EE specification, as this specification, in addition to servlets and JSPs, supports many other technologies. So Tomcat cannot be considered an application server, even if it only partially supports some Java EE technologies (ie Servlet and JavaServer Pages), the developer can import others like JPA or other technologies always in Java EE environment. Tomcat can also be used as a servlet container for frameworks like Spring framework.

SdmxSource

SdmxSource is an openSource project that offers the functionality of generating and validating a file in standard SDMX. SdmxSource is an API standard common to all possible SDMX file generation implementations. SdmxSource guarantees quality, validity, the possibility to expand and to unify the file production process in the SDMX standard. SdmxSource is developed in java and c# and it offers solid documentation for the integration and the possible expansions that can be implemented.

Building blocks

Vtl Compiler

In this chapter are documented all the modules present in Vtl-Compiler and the main functions implemented

Description

In order to interpret and make the VTL language executable, VtlCompiler uses different modules, which operate independently to contribute to the result.

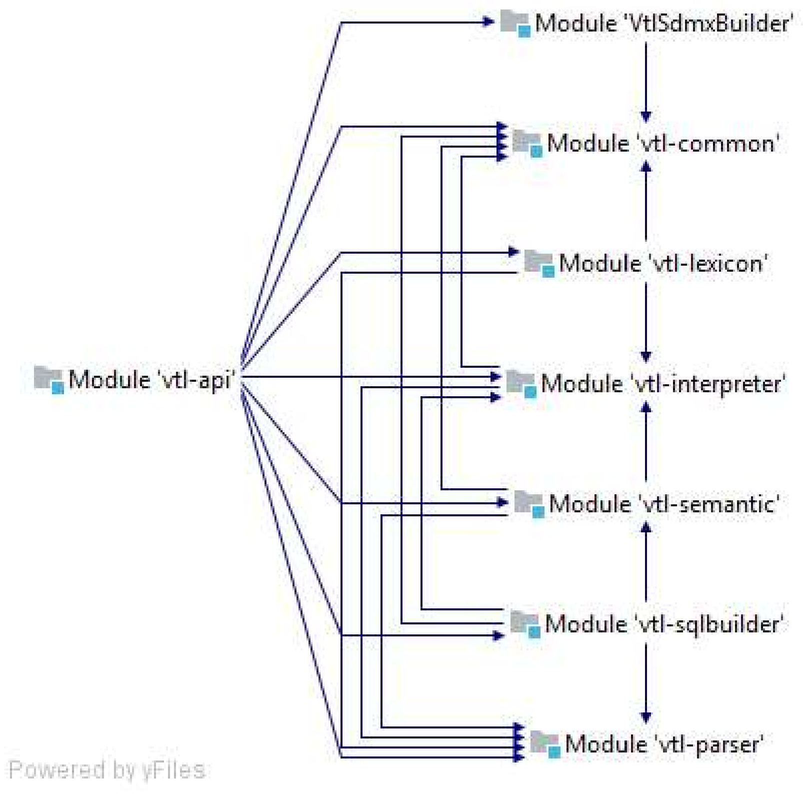


Figura 1 Moduli VtlCompiler

Vtl-Common module

The Vtl-Common module contains all the objects that model the solution that are common to all the modules. This module offers:

- Access to the database
- Utility services for objects that shape the final result

The Vtl-Common does not depend on any other module but acts as a dependency for all the modules of the project (with the exception of the Vtl-Parser)This module includes all the classes that represent VTL and define the data structure used by the other modules to interpret the entered commands and manage the results.

About 100 classes, collected within the “vtl” package, define the VTL language.

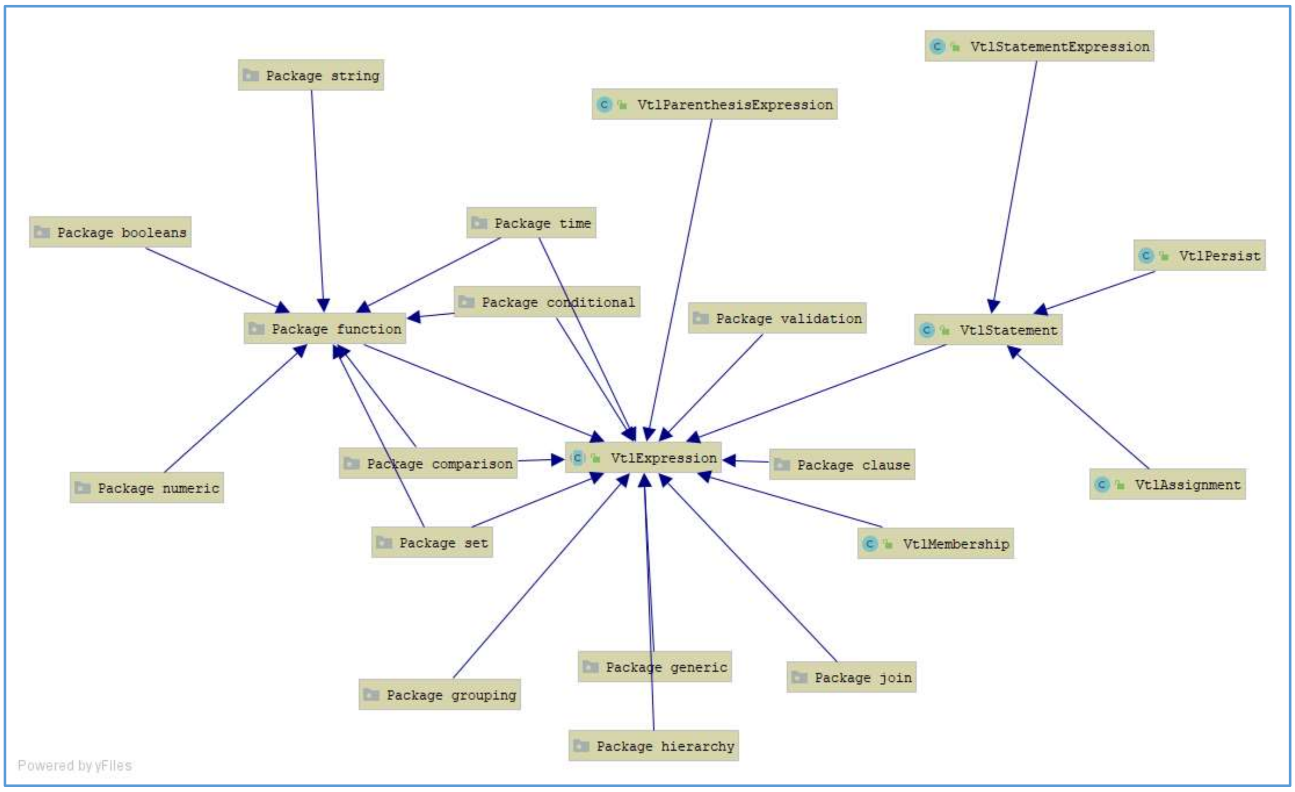


Figura 2 Model VtlCompiler

Dataset Model

Below the data model used by the application for the VTL structures. It is the application database that supports the solution that is in charge of making persistent the structures involved in the transformations and the configuration of the underlying data model.

The database used is Hyper SQL Database.

VtlDataset

The VtlDataset object represents the dataset. It is composed by a list of VtlComponent objects and a series of information needed for the various modules of the application to interpret the results.

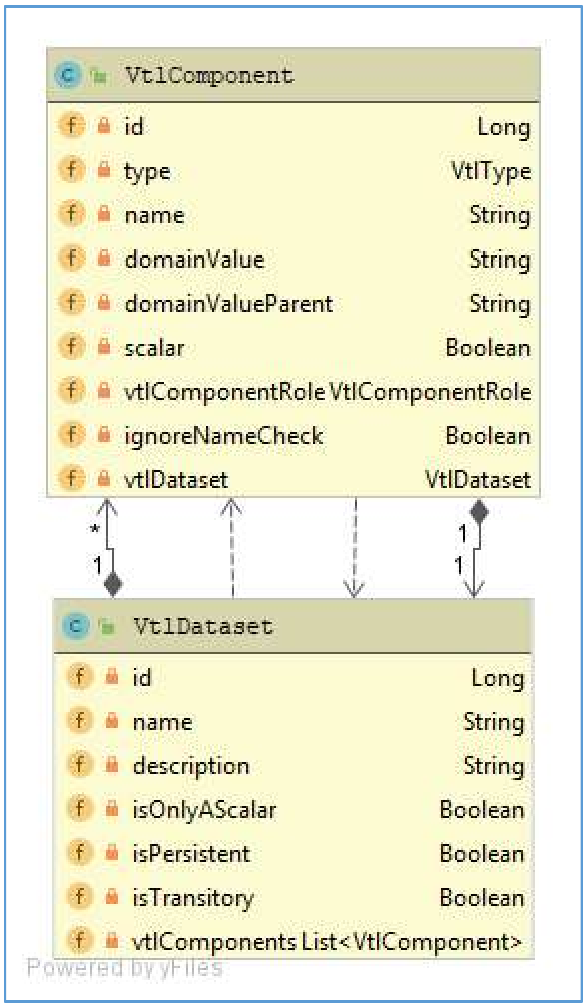


Figura 3 Rapresentation of a VtlDataset

The VtlDataset object is also represented in the HsqlDb via the following structure

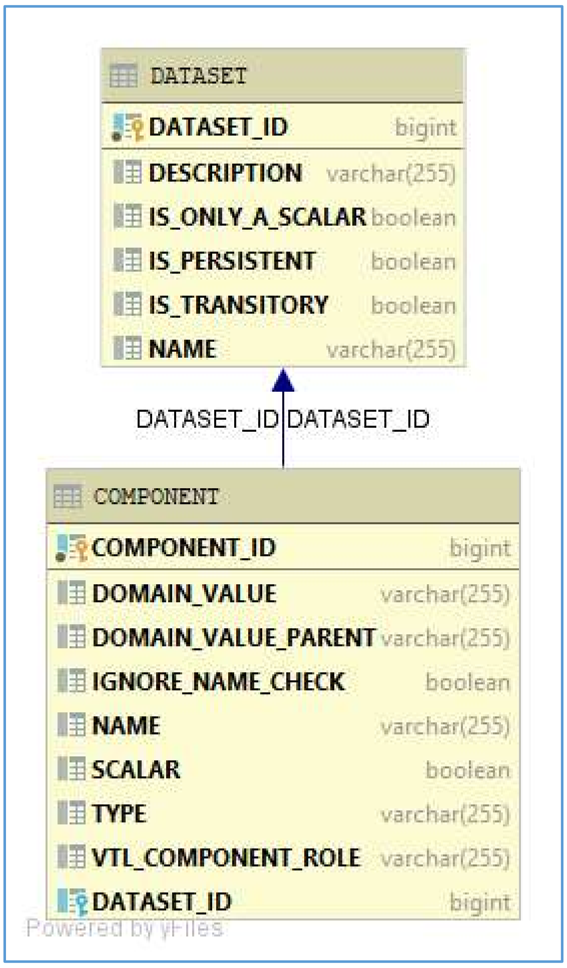


Figura 4 VtlDataset HsqlDB

VtlComponent

The VtlComponent object represents the component entity according to the VTL model. A name, a type, a role and a value domain mainly characterize a component.

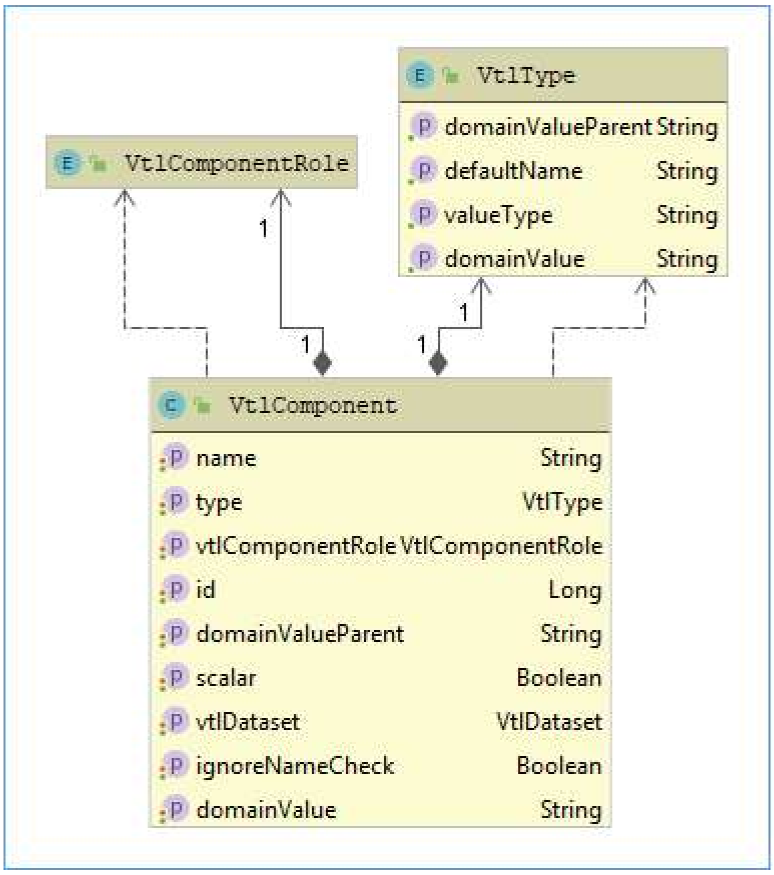
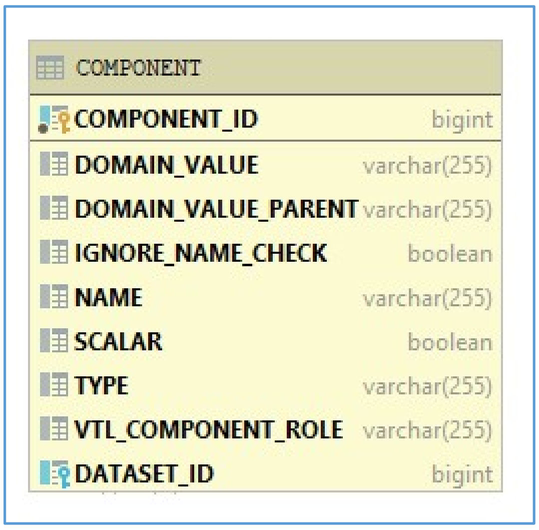


Figura 5 Representation of a VtlComponent

The object is represented also at database level with the following information:



COMPONENT	
COMPONENT_ID	bigint
DOMAIN_VALUE	varchar(255)
DOMAIN_VALUE_PARENT	varchar(255)
IGNORE_NAME_CHECK	boolean
NAME	varchar(255)
SCALAR	boolean
TYPE	varchar(255)
VTL_COMPONENT_ROLE	varchar(255)
DATASET_ID	bigint

Figura 6 VtlComponent HsqlDB

VtlType

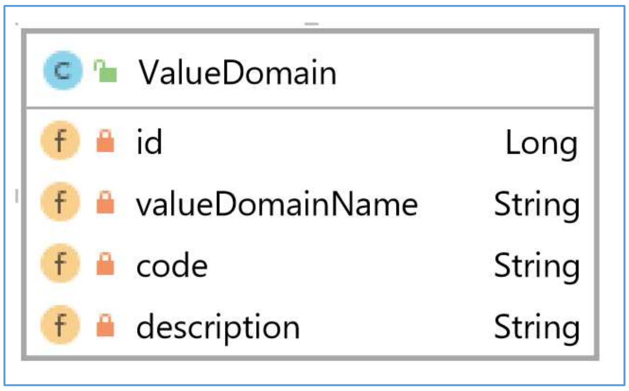
The VtlType object represents the type of the component in the VTL domain. An enumerated class includes all the types specified by the VTL Reference Manual.

VtlRole

The VtlRole class represents the role of the component in the VTL data model. The VtlRole is an enumerated class that offers the following values: Identifier, Measure, Attribute, Viral indicates a viral attribute component

VtlValueDomain

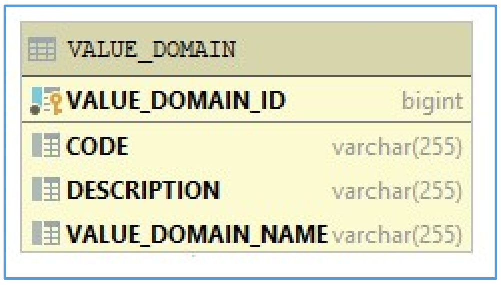
The object Value Domain is used to represent a VTL value domain and it is composed mainly by a name a code representing the VTL id of the value domain and a description.



ValueDomain	
id	Long
valueDomainName	String
code	String
description	String

Figura 7 Representation of Value Domain

The object is also represented at database level through the following database structure.



VALUE_DOMAIN	
VALUE_DOMAIN_ID	bigint
CODE	varchar(255)
DESCRIPTION	varchar(255)
VALUE_DOMAIN_NAME	varchar(255)

Figura 8 Value Domain HsqlDB

VtlUserFunction

The object VtlUserFunction represents the VTL functions defined by the user. It contains the name of the function, the type and the statements that are part of the function.

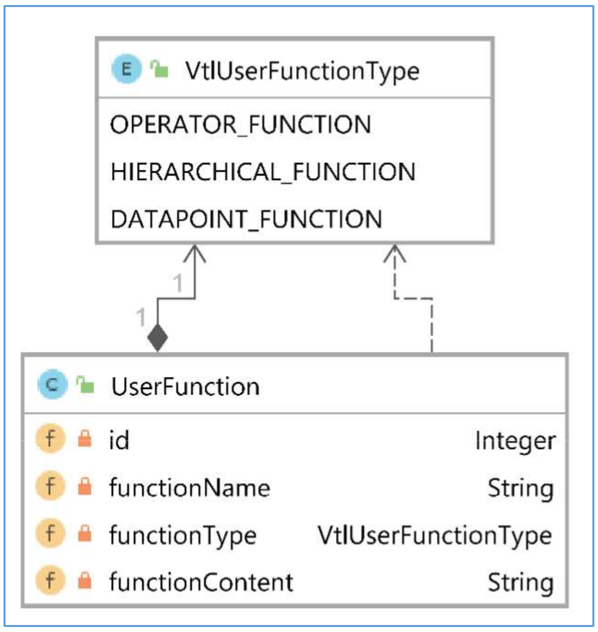
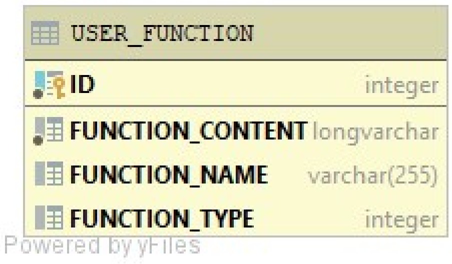


Figura 9 Representation of User Function

The object is represented at database level with the following information



USER_FUNCTION	
ID	integer
FUNCTION_CONTENT	longvarchar
FUNCTION_NAME	varchar(255)
FUNCTION_TYPE	integer

Powered by yFiles

Figura 10 User Function HsqlDB

Result Expression

The ResultExpression object represents the result of the semantic validator. It is composed of three main elements, a dataset, a component and an error message. Usually, only one of the three objects is set depending on the result obtained from the validation.

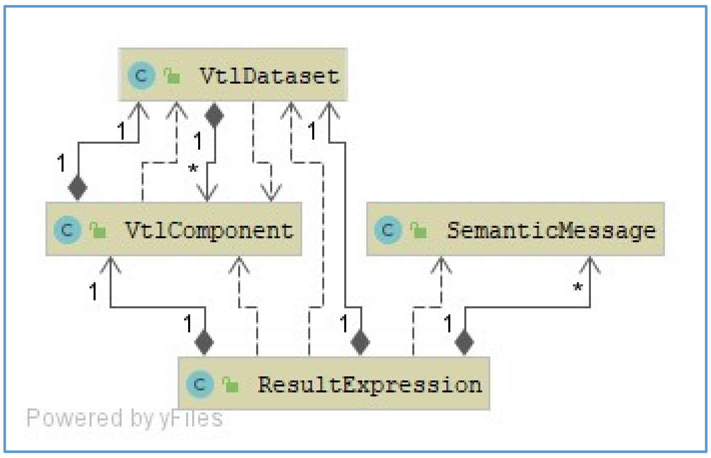


Figura 11 Representation of the Result Expression

SemanticMessage

The object SemanticMessage is created for the error messages deriving from the semantic validation. It contains all the information needed to create an error message.

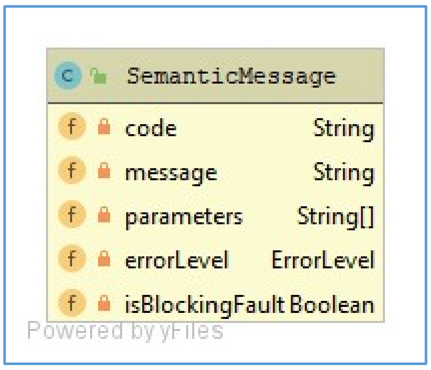


Figura 12 Representation of the Semantic Message

Vtl-Parser module

The Vtl-Parser module is in charge of, starting from the ANTLR grammar, to produce the needed classes for the navigation of the AST (Abstract Syntax Tree). In the module there is only the grammar that is elaborated by the Maven plugin of ANTLR. Classes are produced dynamically using the command “maven-install” of Vtl-Compiler project.

Vtl-Interpreter module

The VTL-Intepreter module represents the interpretative engine that is in charge to interpret the VTL language and reorganise it using a Java structure usable by the project’s engine. The VTL language is analysed through the use of the ALTLR Visitors. Visitors are ALTLR tools to navigate tokens that are activated by lessical structure.

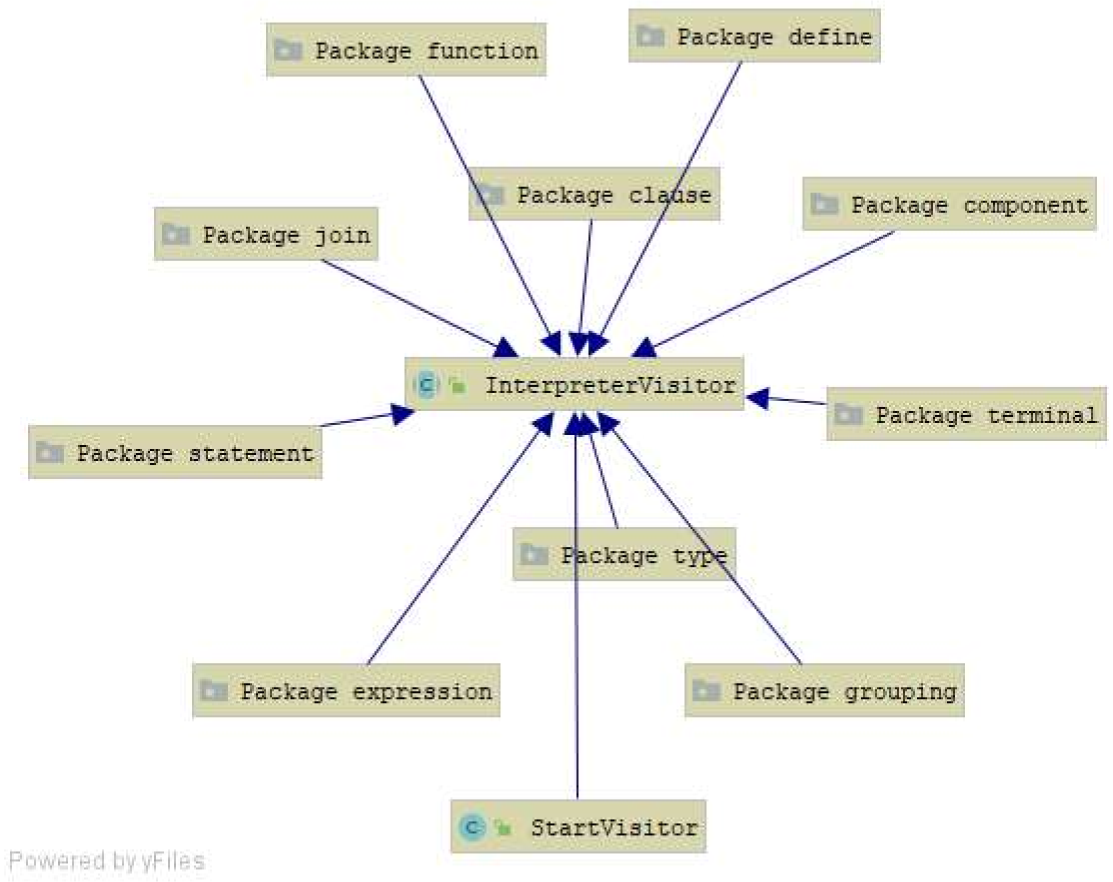


Figura 13 hyerarchy of the Visitor tree

Navigation through the Visitor is from left to right and results are taken b the first leaf met. Through the use of Visitor are not only analysed statements but also solved some ambiguities and at the end a java object, structured as a VTL artefact, is filled.

Given the statement DS_R := DS_1 + DS_2; the following AST is produced.

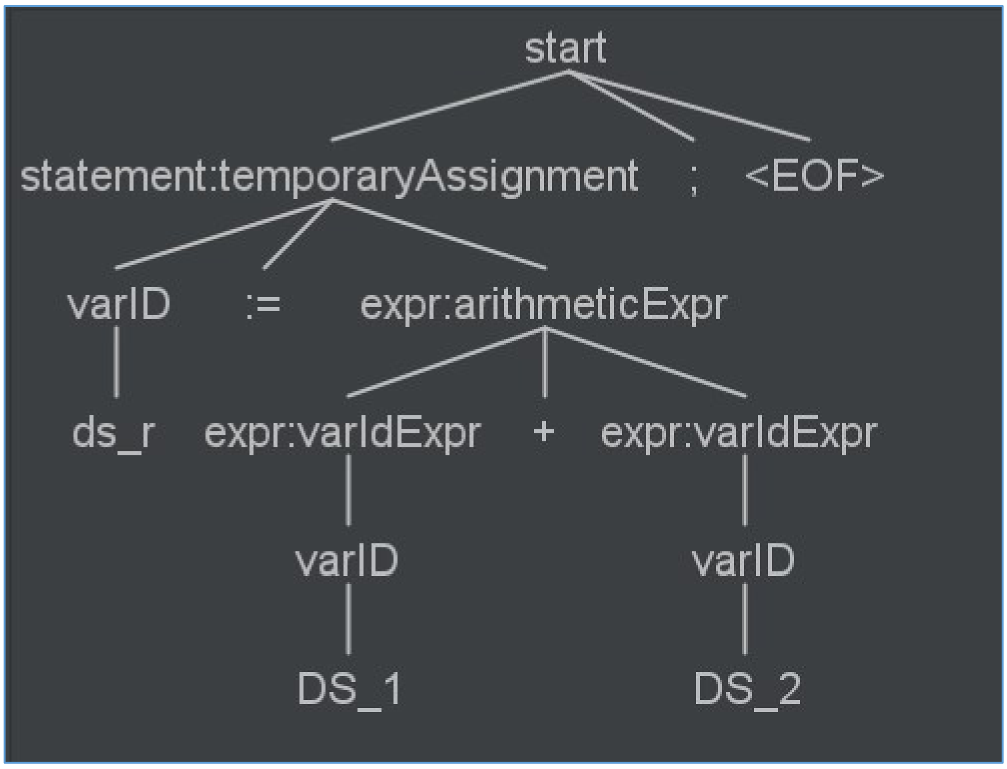


Figura 14 Example of a VTL AST

In the example showed in the picture above, five objects are generated to represent the VTL statement inserted:

- VtlVarId (DS_R) representing the resulting dataset;
- VtlVarId (DS_1) containing all the information representing the DS_1 dataset;
- VtlVarId (DS_2) containing all the information representing the DS_2 dataset;
- VtlMathBinary(+) linked to the object VtlVarId(DS_1) and VtlVarId(DS_2) representing the sum operator
- VtlTemporaryAssignment containing VtlVarId(DS_R) and VtlMathBinary representing the operation to create a temporary dataset.

The structure resulting, created by the union of different objects resulting by the navigation, is analysed, elaborated and enriched of the results of the other system engines (semantic and translation).

Classes describing the VTL language extended the same abstract class VtlExpression that provides a set of common methods for all the commands and it is used as mask for all the possible inserted command.

Vtl-Lexicon module

The Vtl-Lexicon module is in charge to perform the syntactical check and to reorder the VTL commands. The module is composed by two services called by the VTL-Api. The is linked to the Vtl Parser to use the navigation classes of the AST.

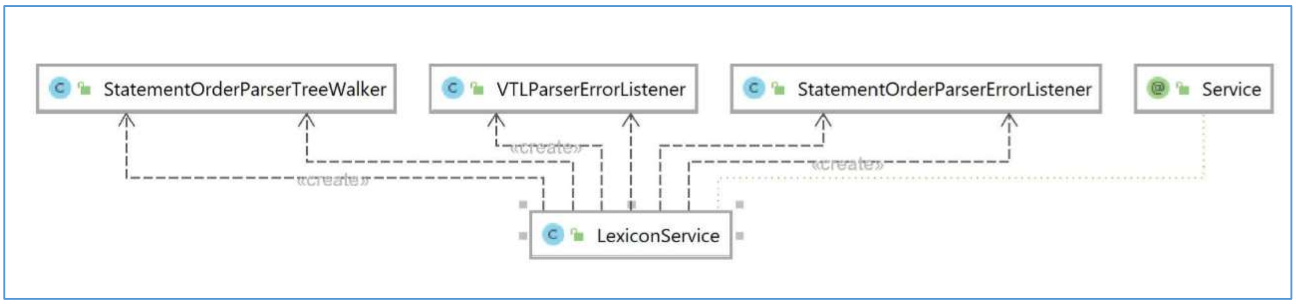


Figura 15 Services diagram with Lexicon

Vtl-Semantic module

The semantic engine is a VTL module that is in charge to semantically validate the VTL inserted statements. Through the structure of data created by the interpreting engine, every single statement is decomposed in its parts and all the partial results that are implicit in the execution of the VTL command are elaborated. Every single module of the semantic engine works independently by the others and it produces:

- The description of a dataset, maintaining all the information of components belonging to the dataset
- The description of a component containing all data related on the name, data type and role of the component.

Construction od the semantic validation tree

The SemanticFactory classes is in charge to orchestrate all the expressions with which a VTL statement is composed using the relative classes implementing the specific scenario of the semantic validation.

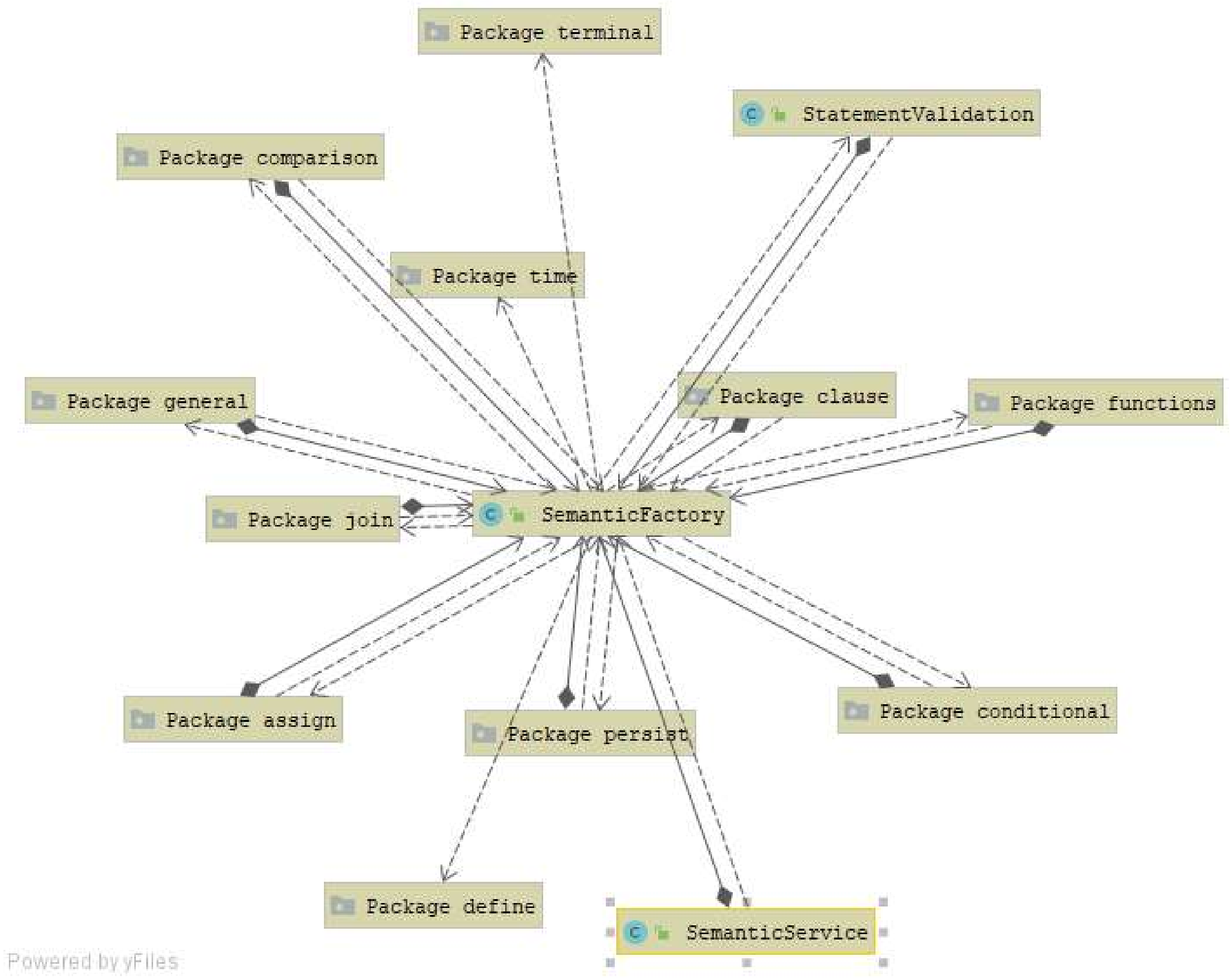


Figura 16 Hierarchy of the semantical validation tree

The SemanticFactory provides the final results of the validation or it is in charge to propagate the error provided by the other modules of the application.

The semantic module provides, to every element of the validation tree, three types of services: managing of the result, data transformation services and semantical data validation. Typically it is valuated the semantic of the operation to perform after the real data transformation, The final result is after returned to the single elements of the tree by FunctionResultService.

Services to manage the results

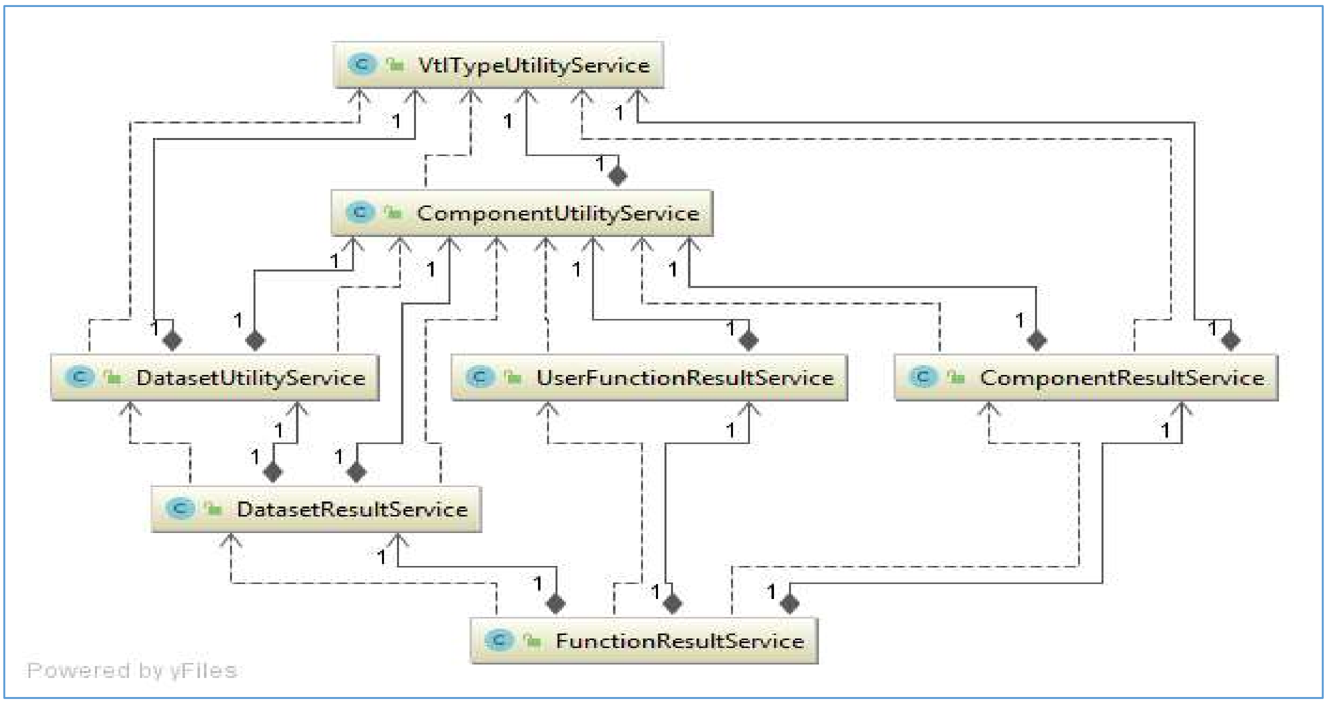


Figura 17 Diagram of translation and managing of the result services

In the module, there are the following services:

- **FunctionResultService** a service to recognize the scenario, call the validation service and returns the result.
- **DatasetResultService** the orchestrator for the semantical validation and dataset transformation
- **ComponentResultService** is a service in charge to orchestrate the semantical validation and component transformation
- **UserFunctionResultService** is a service in charge to orchestrate the semantical validation for the function defined by the user

Metadata transformation services

The data transformation is provided by three utility classes:

- **DatasetUtilityService** provide the functionalities related to the creation of the result at dataset level.
- **ComponentUtilityService** provide the functionalities related to the construction of the result at component level.
- **VtlTypeUtilityService** provide the functionalities related to the VTL data manipulation.

Validation services

The semantical validation does not data transformations but is in charge to verify that every operation is in the interaction scenarios allowed and it constructs an error message in case of a scenario not respected.

It is composed by the following services:

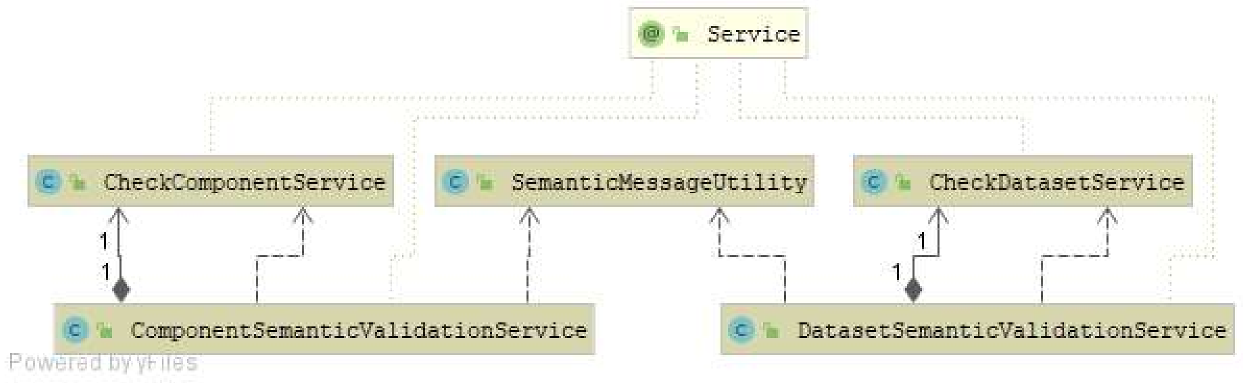


Figura 18 Diagam of validation services

- **DatasetSemanticValidationService** it offers all the possible validation in the possible scenarios of interaction between datasets and call services for the data check and retrieval of a possible semantic error. It takes in input a Validation Data object.
- **ComponentSemanticValidationService** it offers all the possible validation in the interaction scenarios between component and it calls the services for the data check providing an eventual semantical error. It takes in input a Validation Data object.
- **CheckDatasetService** it offers a set of methods dedicated to the dataset validation.
- **CheckComponentValidationService** it offers a set of methods dedicated to the component validation.
- **SemanticMessageUtilityService** it is a service in charge to produce error messages in case of the validation has a negative result, it is called by the "**ValidationService"**

Validation scenarios are constructed in the related "resultService" through the object ValidationData.
ValidationData contains all data that needs to be verified (depending on the scenario) and all checks that must to be performed.
Checks must be performed at Operator level and are configured in the Operator class and set automatically in the generation tree phase.

Vtl-Sqlbuilder module

The translation to SQL is solved by the Vtl-Sqlbuilder module that is in charge to interpret the VTL commands using the data structure created by the interpret engine. In addition, this module uses the powerful of the pattern Visitor. Every single java structure, resulting by the VTL- Intepreter module, is translated and return a partial result. Every single module of the Sqlbuilder work indipendently by the others.

Construction of the translation SQL tree

Similarly, to the semantic engine a translation tree is explored and constructed trough the TranslationFactory.

The TranslationFactory, receive the input by the interpreter and semantic engine and is in charge of:

- Create the translation elements to create the translation SQL tree
- Activate the specific translation services
- Propagate the translation result to the upper elements of the tree.

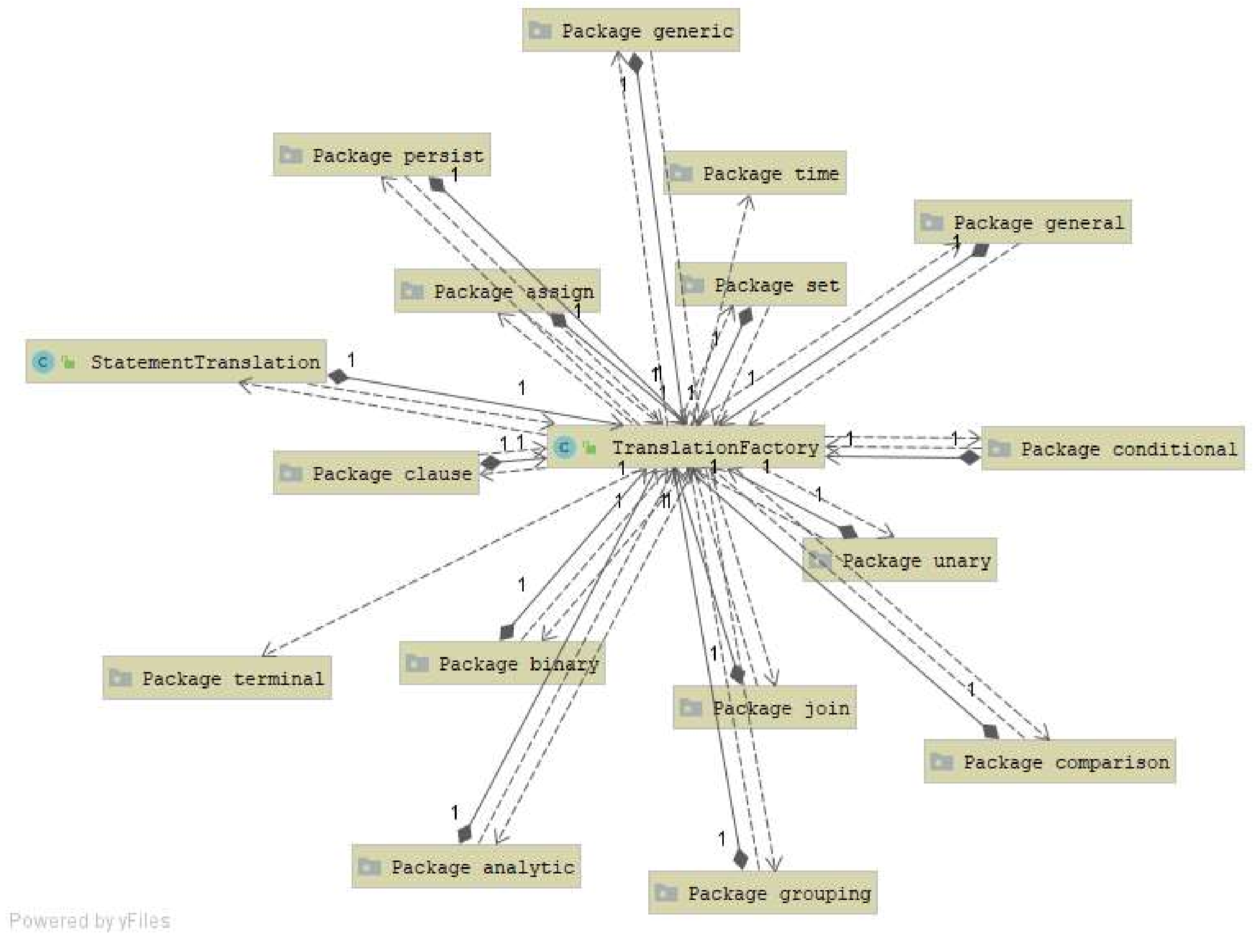


Figura 19 Hierarchical diagram of the translation tree

The SqlBuilder module generates a coherent SQL using objects representing in SQL a dataset or a component. These objects are then interpreted by specific transformation services that, once interpreted, become coherent SQL translation and specific of the SQL type required.

Mechanism to generate services

Services of the SqlBuilder module are produced in order to be expanded through new translation languages.

Are present four types or services SqlResult, SqlDataset, SqlComponent e SqlObject. All services uses the Java Interface and are assigned dynamically to the translation tree at the execution step, this allows the activation of specific services depending on the type of translation required.
For every service type has been used the inherited mechanism, the CommonSql service is extended in specific translation services (Oracle, MySQL, etc) extending the basic functionalities offered by the Common module and design the specific mechanism of SQL supported.

The specific services active by a parameter received during the translation request and assigned to the tree during the runtime translation.

Translation services

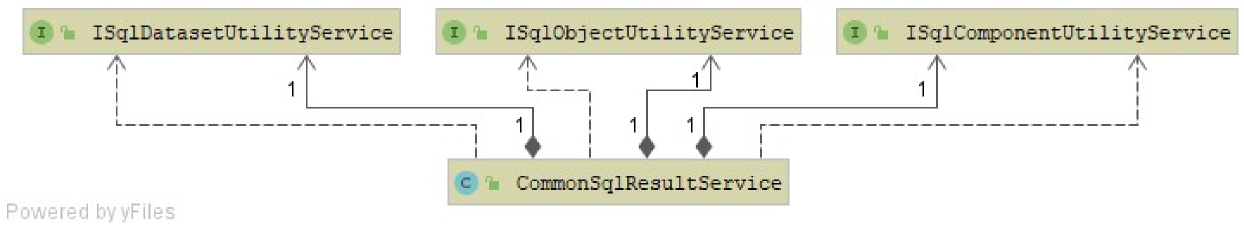


Figura 20 diagram of the services managing the SQL resultsl

The SQLBuilder module, offers the following translation services:

- **SqlResultService** the SQLResultService services are in charge to coordinate different translation services to construct the SQL instruction coherent with the VTL statements received. One part of the service contains all the elaborating behaviour of the translation common to all the sql (CommonSqlResultService) or distinguished by every type of SQL offered (OracleSqlResultService, MySqlResultService, PostgreSQLResultService, SqlServiceResultService).

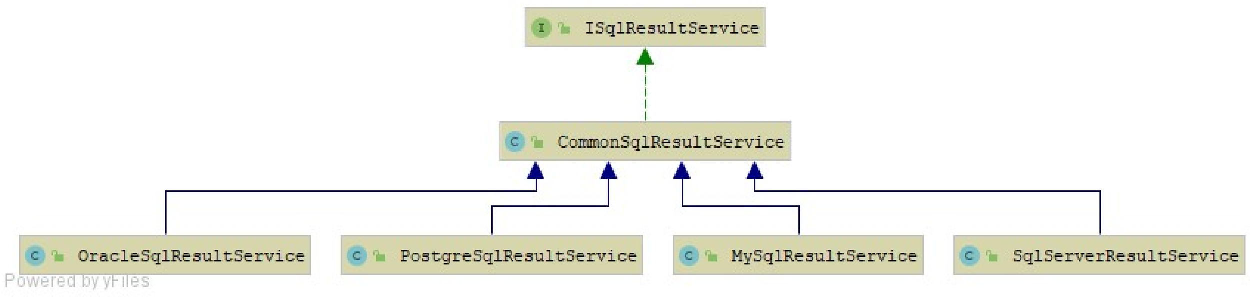


Figura 21 Diagram of services for the SQL results

- **SqlDatasetUtilityService** SqlDatasetUtilityService services are in charge to perform the needed translation to represent the SQL translation on datasets of tables involved in the VTL statements. These services usually work on the SqlDataset object, a complex object that offers all the needed information to link the dataset transformation in SQL transformations. The services is composed by a part that is common to all the SQL (CommonSqlDatasetUtilityService) and one for every type of SQL offered.

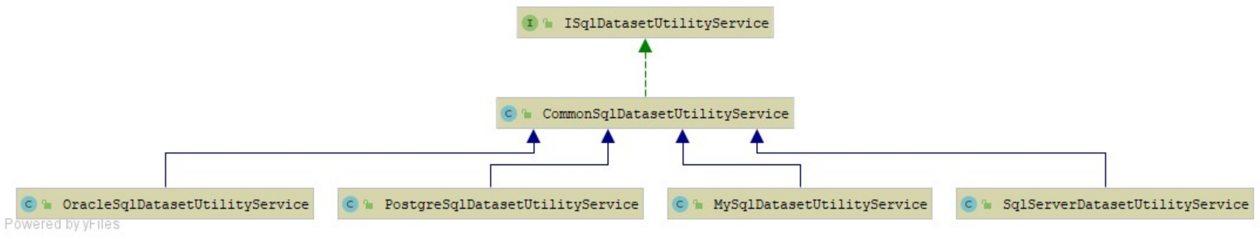

ISqlDatasetUtilityService
CommonSqlDatasetUtilityService
OracleSqlDatasetUtilityService
PostgreSqlDatasetUtilityService
MySqlDatasetUtilityService
SqlServerDatasetUtilityService
Powered by yFiles

Figura 22 diagram of Dataset translation services

- **SqlComponentUtilityService** SqlComponentUtilityService services perform all the needed transformation s to represent the SQL translation on field involved in the VTL commands. The service is composed by a part that is common to all the SQL (CommonSqlComponentUtilityService) and one for every SQL type.

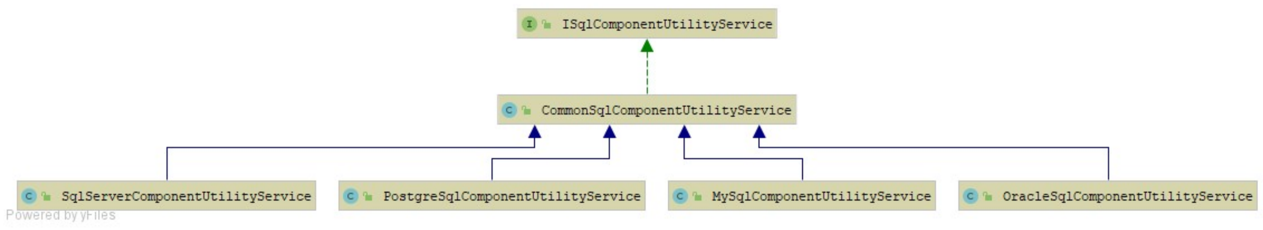

ISqlComponentUtilityService
CommonSqlComponentUtilityService
SqlServerComponentUtilityService
PostgreSqlComponentUtilityService
MySqlComponentUtilityService
OracleSqlComponentUtilityService
Powered by yFiles

Figura 23 Diagram or translation in SQL services

- **SqlObjectUtilityService** SqlObjectUtilityService interprets all the objects resulting by other utility services (SqlDataset e SqlComponent) and to construct the SQL translation. The service is composed by a part that is common to all the SQL(CommonSqlObjectUtilityService) and one for every type of SQL

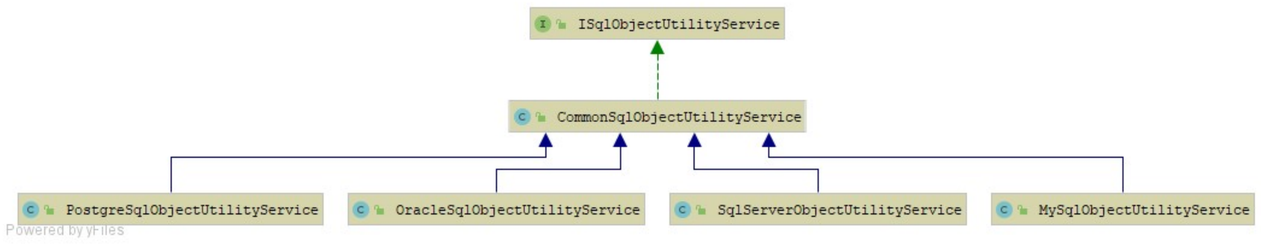

ISqlObjectUtilityService
CommonSqlObjectUtilityService
PostgreSqlObjectUtilityService
OracleSqlObjectUtilityService
SqlServerObjectUtilityService
MySqlObjectUtilityService
Powered by yFiles

Figura 24 diagram of SQL translation services

Translation in SQL language

To support the translation in SQL it has been produced a library of classes predefined. This library offers the advantage to be completely abstract by the logic on which query are constructed because it allows the access to a set of construct already predefined and easily usable internally to the java code. The SQL constructs created by the library is made using the ANSI SQL92 with some added by ANSI SQL99.

Default values

In the translation, the application uses three predefined values, two for the managing of the Boolean value and one for the VTL value domain.

- *application.default.value.true* represents the predefined value for the environment in which the value of Boolean is true. This is because some DB as if for example Oracle has non-implementation of Boolean type, this requires for every installation an own choice for this data type. Usually the types used are “T”, “TRUE”, “Y”, “YES”, “S”, “1”. The predefined setting is “TRUE”:
- *application.default.value.false* is the analogous to the previous one and represents the Boolean value *false*. The predefined value is “FALSE”

- *application.default.value.domain.code* is used in the select construct from the value domain tables and represents the name of the key column of these tables. The default value for this parameter is "ITEM_ID".

Persistent or non-persistent assignment

The VTL operators of Persistent assignment (<-) and Non-persistent assignment (:=) are translated in sql with the creation of a resulting table; this table will be permanent in case of permanent assignment while it will be dropped in the non-persistent assignment.

For every resulting table of a VTL command will be created also indexes for every identifier. This is not implicit in the VTL statement but it is useful in the managing of future query on the same table.

Examples of translation

Below are reported some examples of SQL translation with the description for the four dialect foreseen by the application.

Example 1

```
DS_r := rtrim(DS_x);
```

Structure of table DS_x

NOME	TIPO	RUOLO
ID_1	INTEGER	IDENTIFIER
ID_2	STRING	IDENTIFIER
ME_1	STRING	MEASURE
AT_1	STRING	VIRAL

Translation for: Oracle, MySql, PostgreSql

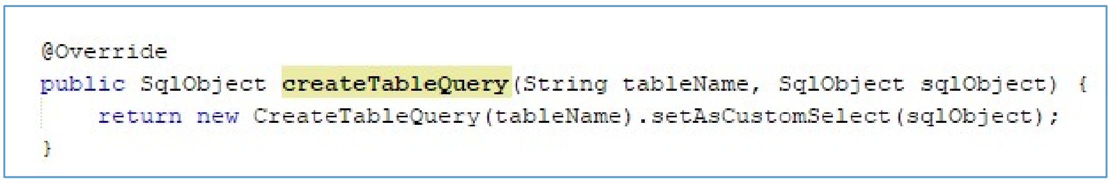
```
CREATE TABLE DS_r AS (
SELECT t0.ID_1 AS ID_1, t0.ID_2 AS ID_2, rtrim(t0.ME_1) AS ME_1, t0.AT_1 AS AT_1 FROM DS_x t0
);
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

Translation for: SqlServer

```
SELECT t1.* INTO DS_r FROM (
SELECT t0.ID_1 AS ID_1, t0.ID_2 AS ID_2, rtrim(t0.ME_1) AS ME_1, t0.AT_1 AS AT_1 FROM DS_x t0
) t1;
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

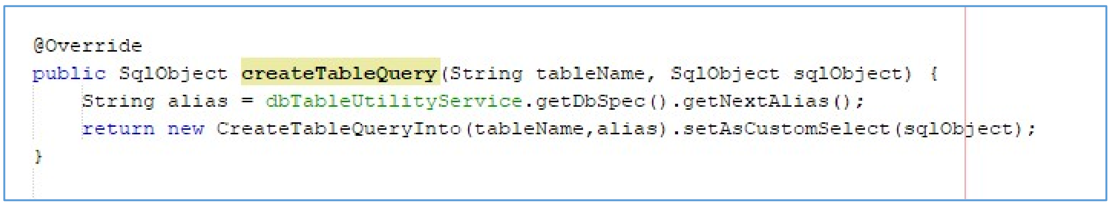
In this last example, the only difference is the use of "SELECT INTO" instead of "CREATE to create a table coming by a select.

CommonSqlObjectUtilityService



```java
@Override
public SqlObject createTableQuery(String tableName, SqlObject sqlObject) {
    return new CreateTableQuery(tableName).setAsCustomSelect(sqlObject);
}
```

SqlServerObjectUtilityService



```java
@Override
public SqlObject createTableQuery(String tableName, SqlObject sqlObject) {
    String alias = dbTableUtilityService.getDbSpec().getNextAlias();
    return new CreateTableQueryInto(tableName,alias).setAsCustomSelect(sqlObject);
}
```

In the class, CommonSqlObjectUtilityService there is the standard behaviour that is overloaded in every specialised class for a specific dialect. In our example Oracle, Mysql e Posgresql will use the method of class CommonSqlObjectUtilityService while Sqlserver will use the method of its specific class SqlServerObjectUtilityService. In both cases, CreateTableQuery and CreateTableQueryInto are two classes of the sqlBuilder library that return the SqlObject object, containing the instruction translated in sql.

Example 2

Below a more complex example.

```
DS_r := DS_1 [ unpivot ID_X, ME_X];
```

Structure of DS_1

NOME	TIPO	RUOLO
ID_1	INTEGER	IDENTIFIER
ID_2	STRING	IDENTIFIER
ME_1	STRING	MEASURE
AT_1	STRING	VIRAL

Translation to: MySql, PostgreSql

```sql
CREATE TABLE DS_r AS (
SELECT ID_1,ID_2,'ME_1' as ID_X,ME_1 as ME_X FROM DS_1
UNION
SELECT ID_1,ID_2,'ME_2' as ID_X,ME_2 as ME_X FROM DS_1
UNION
SELECT ID_1,ID_2,'ME_3' as ID_X,ME_3 as ME_X FROM DS_1
);
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

Translation to: Oracle

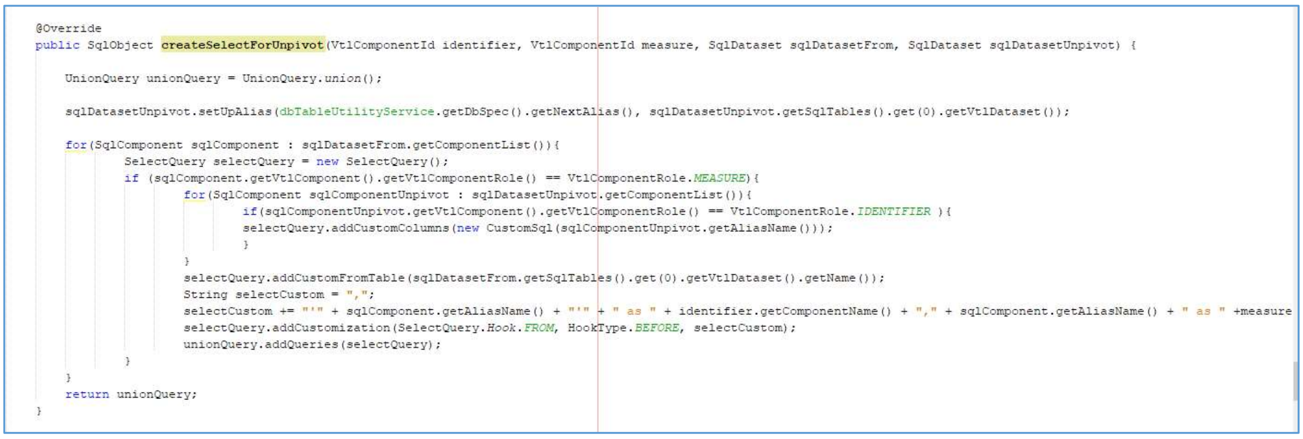
```sql
CREATE TABLE DS_r AS (
SELECT ID_1,ID_2,ID_X,ME_X FROM DS_1
  UNPIVOT (ME_X for ID_X in (ME_1 as 'ME_1',ME_2 as 'ME_2',ME_3 as 'ME_3') )
);
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

Translation to: SqlServer

```
SELECT t3.* INTO DS_r FROM (
SELECT ID_1,ID_2,ID_X,ME_X FROM DS_1
  UNPIVOT (ME_X for ID_X in (ME_1,ME_2,ME_3) ) t2
) t3;
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

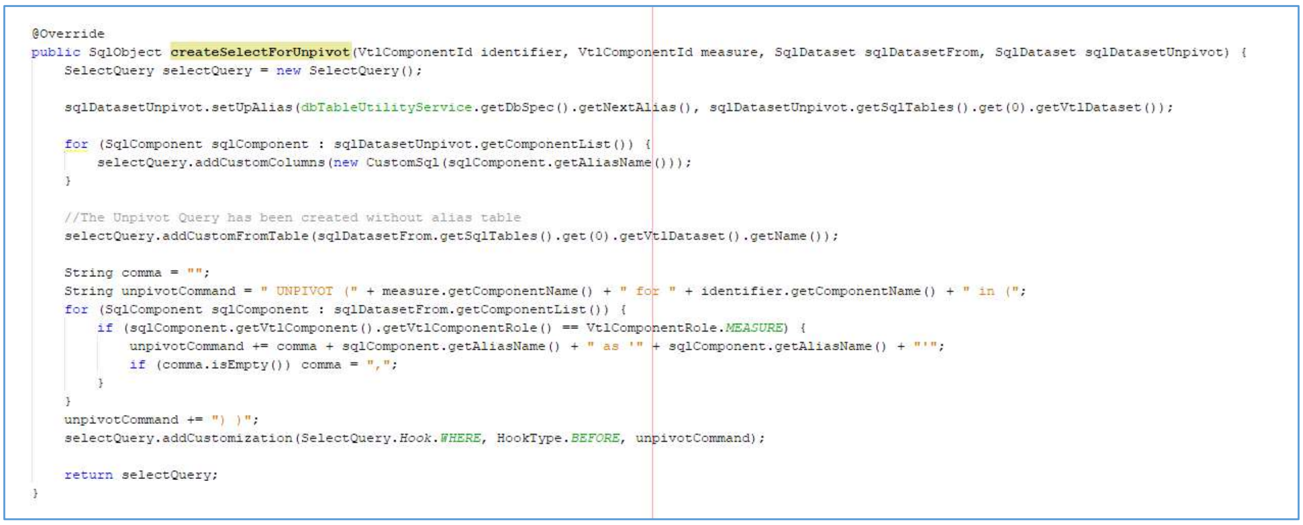
The Vtl "Unpivot" command is interpreted and translated into the three different modes: MySql and PostgreSql which do not have their own implementation of the unpivot functionality and for this reason they are implemented using the sql "Union" construct; for Oracle and SqlServer the specific construct of unpivot function is used. In the CommonSqlObjectUtilityService class the default translation valid for the two dialects MySql and PostgreSql is implemented, while a native function is used for Oracle and SqlServer classes.

CommonSqlObjectUtilityService



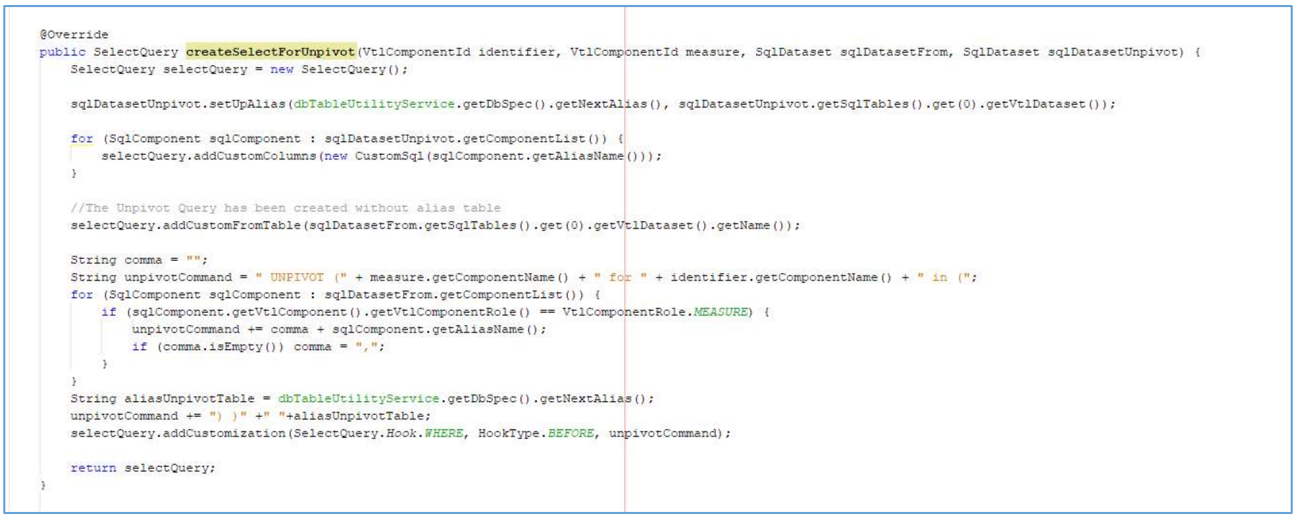
```java
@Override
public SqlObject createSelectForUnpivot(VtlComponentId identifier, VtlComponentId measure, SqlDataset sqlDatasetFrom, SqlDataset sqlDatasetUnpivot) {

    UnionQuery unionQuery = UnionQuery.union();

    sqlDatasetUnpivot.setUpAlias(dbTableUtilityService.getDbSpec().getNextAlias(), sqlDatasetUnpivot.getSqlTables().get(0).getVtlDataset());

    for(SqlComponent sqlComponent : sqlDatasetFrom.getComponentList()){
            SelectQuery selectQuery = new SelectQuery();
            if (sqlComponent.getVtlComponent().getVtlComponentRole() == VtlComponentRole.MEASURE){
                    for(SqlComponent sqlComponentUnpivot : sqlDatasetUnpivot.getComponentList()){
                            if(sqlComponentUnpivot.getVtlComponent().getVtlComponentRole() == VtlComponentRole.IDENTIFIER ){
                            selectQuery.addCustomColumns(new CustomSql(sqlComponentUnpivot.getAliasName()));
                            }
                    }
                    selectQuery.addCustomFromTable(sqlDatasetFrom.getSqlTables().get(0).getVtlDataset().getName());
                    String selectCustom = ",";
                    selectCustom += "'" + sqlComponent.getAliasName() + "'" + " as " + identifier.getComponentName() + "," + sqlComponent.getAliasName() + " as " +measure
                    selectQuery.addCustomization(SelectQuery.Hook.FROM, HookType.BEFORE, selectCustom);
                    unionQuery.addQueries(selectQuery);
            }
    }
    return unionQuery;
}
```

OracleSqlObjectUtilityService



```java
@Override
public SqlObject createSelectForUnpivot(VtlComponentId identifier, VtlComponentId measure, SqlDataset sqlDatasetFrom, SqlDataset sqlDatasetUnpivot) {
    SelectQuery selectQuery = new SelectQuery();

    sqlDatasetUnpivot.setUpAlias(dbTableUtilityService.getDbSpec().getNextAlias(), sqlDatasetUnpivot.getSqlTables().get(0).getVtlDataset());

    for (SqlComponent sqlComponent : sqlDatasetUnpivot.getComponentList()) {
        selectQuery.addCustomColumns(new CustomSql(sqlComponent.getAliasName()));
    }

    //The Unpivot Query has been created without alias table
    selectQuery.addCustomFromTable(sqlDatasetFrom.getSqlTables().get(0).getVtlDataset().getName());

    String comma = "";
    String unpivotCommand = " UNPIVOT (" + measure.getComponentName() + " for " + identifier.getComponentName() + " in (";
    for (SqlComponent sqlComponent : sqlDatasetFrom.getComponentList()) {
        if (sqlComponent.getVtlComponent().getVtlComponentRole() == VtlComponentRole.MEASURE) {
            unpivotCommand += comma + sqlComponent.getAliasName() + " as '" + sqlComponent.getAliasName() + "'";
            if (comma.isEmpty()) comma = ",";
        }
    }
    unpivotCommand += ") )";
    selectQuery.addCustomization(SelectQuery.Hook.WHERE, HookType.BEFORE, unpivotCommand);

    return selectQuery;
}
```

SqlServerObjectUtilityService



```java
@Override
public SelectQuery createSelectForUnpivot(VtlComponentId identifier, VtlComponentId measure, SqlDataset sqlDatasetFrom, SqlDataset sqlDatasetUnpivot) {
    SelectQuery selectQuery = new SelectQuery();

    sqlDatasetUnpivot.setUpAlias(dbTableUtilityService.getDbSpec().getNextAlias(), sqlDatasetUnpivot.getSqlTables().get(0).getVtlDataset());

    for (SqlComponent sqlComponent : sqlDatasetUnpivot.getComponentList()) {
        selectQuery.addCustomColumns(new CustomSql(sqlComponent.getAliasName()));
    }

    //The Unpivot Query has been created without alias table
    selectQuery.addCustomFromTable(sqlDatasetFrom.getSqlTables().get(0).getVtlDataset().getName());

    String comma = "";
    String unpivotCommand = " UNPIVOT (" + measure.getComponentName() + " for " + identifier.getComponentName() + " in (";
    for (SqlComponent sqlComponent : sqlDatasetFrom.getComponentList()) {
        if (sqlComponent.getVtlComponent().getVtlComponentRole() == VtlComponentRole.MEASURE) {
            unpivotCommand += comma + sqlComponent.getAliasName();
            if (comma.isEmpty()) comma = ",";
        }
    }
    String aliasUnpivotTable = dbTableUtilityService.getDbSpec().getNextAlias();
    unpivotCommand += ") )" +" "+aliasUnpivotTable;
    selectQuery.addCustomization(SelectQuery.Hook.WHERE, HookType.BEFORE, unpivotCommand);

    return selectQuery;
}
```

Example 3

As a third case we analyse the translation of a VTL command containing the use of a user function.

```
DS_r := sum_dataset_int(DS_x, 3);
```

Structure of DS_1

NOME	TIPO	RUOLO
ID_1	INTEGER	IDENTIFIER
ID_2	STRING	IDENTIFIER
ME_1	INTEGER	MEASURE
ME_2	INTEGER	MEASURE
AT_1	STRING	VIRAL

Structure of DS_2

NOME	TIPO	RUOLO
ID_1	INTEGER	IDENTIFIER
ID_2	STRING	IDENTIFIER
ME_1	INTEGER	MEASURE
ME_2	INTEGER	MEASURE
AT_1	STRING	VIRAL

Structure of user function: sum_dataset

```
define operator sum_dataset (x dataset, y dataset)  returns dataset is
   x + y
end operator;
```

Translation to: Oracle, PostgreSql

```
CREATE TABLE DS_r AS (
SELECT t2.ID_1 AS ID_1,t2.ID_2 AS ID_2,(t2.ME_1)+(t1.ME_1) AS ME_1,(t2.ME_2)+(t1.ME_2) AS ME_2,t2.AT_1 ||
':' || t1.AT_1 AS AT_1 FROM DS_1 t2
INNER JOIN DS_2 t1 ON (((t2.ID_1 = t1.ID_1) AND (t2.ID_2 = t1.ID_2)))
);
```

```
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

Translation to: SqlServer

```
SELECT t3.* INTO DS_r FROM (
SELECT t2.ID_1 AS ID_1,t2.ID_2 AS ID_2,(t2.ME_1)+(t1.ME_1) AS ME_1,(t2.ME_2)+(t1.ME_2) AS
ME_2,concat(concat(t2.AT_1,':'),t1.AT_1) AS AT_1 FROM DS_1 t2
INNER JOIN DS_2 t1 ON (((t2.ID_1 = t1.ID_1) AND (t2.ID_2 = t1.ID_2)))
) t3;
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

Translation to: MySql

```
CREATE TABLE DS_r AS (
SELECT t2.ID_1 AS ID_1,t2.ID_2 AS ID_2,(t2.ME_1)+(t1.ME_1) AS ME_1,(t2.ME_2)+(t1.ME_2) AS
ME_2,concat(concat(t2.AT_1,':'),t1.AT_1) AS AT_1 FROM DS_1 t2
INNER JOIN DS_2 t1 ON (((t2.ID_1 = t1.ID_1) AND (t2.ID_2 = t1.ID_2)))
);
CREATE INDEX DS_r_ID_1_IDX ON DS_r (ID_1);
CREATE INDEX DS_r_ID_2_IDX ON DS_r (ID_2);
```

In this example we have a difference on different dialects due to the use of the "concat" function rather than "||" to concatenate the values and the use of the "INTO" for SqlServer. The function, in order to be inserted and recognized within a VTL statement, it must be previously declared and saved using the path / userFunctionOperators api / createUserFunction, the use of the API is illustrated in the chapter "Communication API".

Vtl-Api module

Vtl Api is the module that exposes the features of the Vtl Compiler application externally. Structurally it is composed of the controllers that deal with implementing the rest interfaces, the configuration classes that define the configuration of the DBs and the main class of the application.

- VtlApiApplication is the main part of the application, in addition the profile linked to the spring.profiles.active parameter is managed here to specialize in SQL translation in the respective dialect

- DBHsqlDBServerConfiguration is the class that activates the BD HSQL server used internally by the application, it uses the following properties from the application.properties file:
    - *hsql.server.port*
    - *hsql.database.path*
    - *hsql.server.dbname*

- DBHsqlDBConfiguration configures access to the HSQL DB, uses the following properties from the application.properties file:
    - *spring.datasource.driver-class-name*
    - *spring.datasource.url*
    - *spring.datasource.username*
    - *spring.datasource.password*

- DBOracleConfiguration configures access to the Oracle DB containing the description of the VTL structures, it uses the following properties from the application.properties file:
    - *oracle.datasource.url*
    - *oracle.datasource.username*
    - *oracle.datasource.password*
    - *oracle.datasource.driver-class-name*

- EngineController implements the user interfaces of the path /engine

- UserFunctionController implements the user interfaces of the path /userFunctionOperators

- VtlDatasetController implements the user interfaces of the path /vtlDatasets

The APIs exposed by the three controllers are described in the "Communication API" chapter.

Application test

The Vtl Api module also contains all the test classes of the Vtl Compiler application, these are divided into two main groups: semantic and translate, dealing respectively with testing the functionality of the semantic engine and the translation engine towards the sql in four languages currently available.

Vtl-SdmxBuilder module

The Vtl-SdmxBuilder module takes care of transforming the structural metadata represented by the dataset object into a valid SDMX format. The module uses the open source libraries offered by SdmxSource to perform the transformation.

The module implements a series of interface services to allow the population of the different sections of the SDMX.

Loading services for SDMX

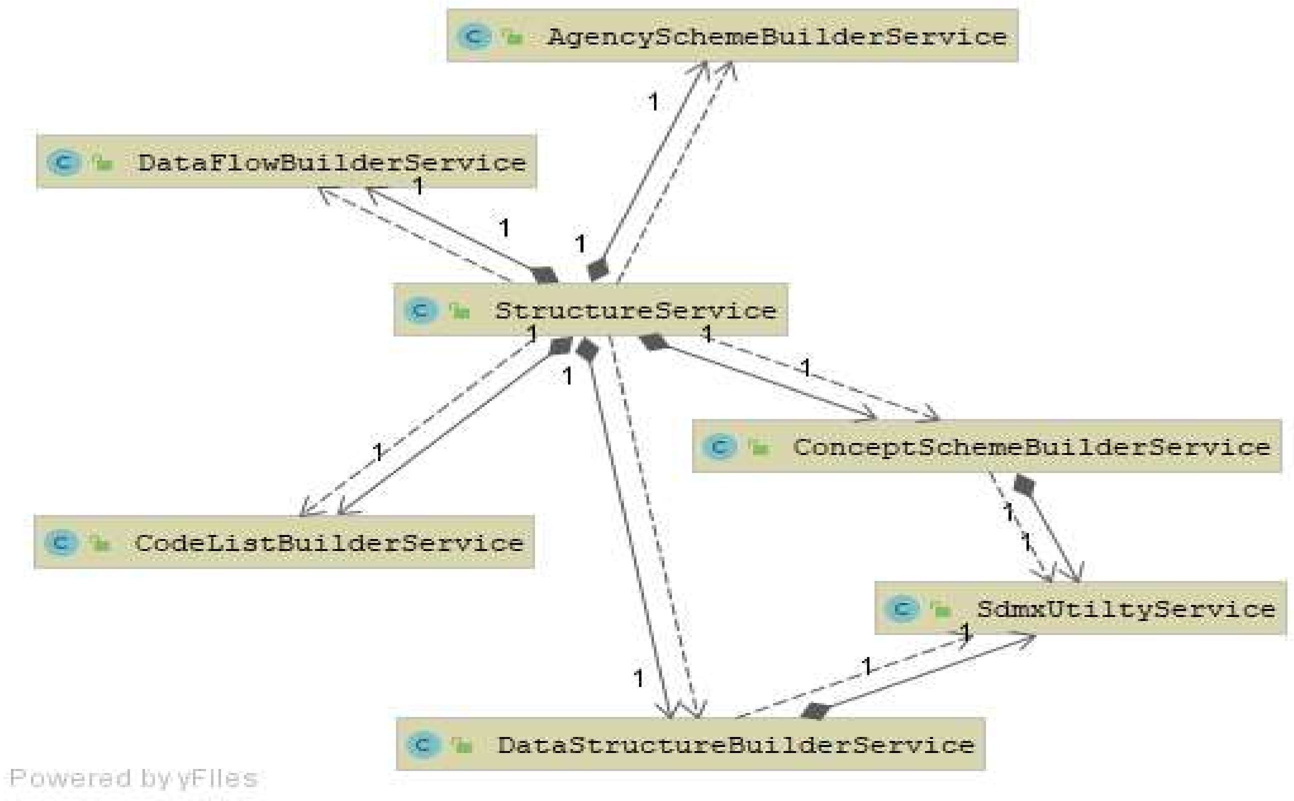


Figura 25 Diagramma dei servizi di trasformazione SDMX

The Vtl-SdmxBuilder module offers the following services:

- DataFlowBuilderService takes care of populating the DataFlow section in the Sdmx file
- CodeListBuilderService takes care of populating the CodeList section in the sdmx file
- ConceptSchemeBuilderService deals with populating the ConceptScheme section on sdmx
- SdmxUtilityService is the service that contains utility tools for transforming the SDMX file
- DataStructureBuilderService deals with the creation of the data structure
- AgencySchemeBuilderService is in charge of populating the agency scheme section of the SDMX file
- StructureService. is responsible for coordinating all other services and generating the XML file.

VTLEditor GUI

The VTLEditor infrastructure provide a graphical interface to allow the user to interact with the components of the infrastructure. It is a .NET Windows desktop application developed in C#.

It is composed by two main forms:

- VTL_Editor_PL.frmStart: this form is in charge of loading the common C# structure, the user settings and to check the connection to the web services and to the DB. It is runned as first and, if the checking phase is passed it will run the *frmMain*
    - VTL_Editor_PL.gui.frmStartMessageBoxError: this form is specialized to show error messages during the loading phase. It can be called only by frmStart
- VTL_Editor_PL.frmMain: this class is the main form of the VTLEditor graphical interface. It interacts with the user and allow him to manage all the other components.

The VTLEditor gui allow the user to:

- Edit VTL Statements
    - VTL_Editor_PL.gui.fastCodeTextBox: this object is an extension of a third part text box FastTestCodeBox aimed to create a text box specialized to support software development
    - VTL_Editor_PL.gui.FindForm: this form is used by fastCodeTextBox to insert the string that has to be searched in the text box
    - VTL_Editor_PL.gui.ReplaceForm: this form is used by fastCodeTextBox to insert the string that has to be replaced in the text box and the string that will replace it
    - VTL_Editor_PL.classes.tool.ShowCaretPositionEventArgs: this class contains the definition of the arguments in case of movement of the caret in the FastCodeTextBox
    - VTL_Editor_PL.gui.frmOpenTrasformation: this forms show the list of all the TransformationScheme already stored into the VTL-DB
    - VTL_Editor_PL.gui.frmSaveTrasformation: this form is in charge of asking the user the information to save a TransformationScheme
- Create structural metadata
    - VTL_Editor_PL.gui.frmArtefactManager: this module allows the user to create the user defined structural metadata
    - VTL_Editor_PL.gui.UserDefinedDataSetPanel: this user defined control allows the user to create a his own VTL dataset
    - VTL_Editor_PL.gui.UserDefinedDataStructurePanel: this user defined control allows the user to create a his own VTL datastructure
    - VTL_Editor_PL.gui.UserDefinedValueDomainPanel: this user defined control allows the user to create a his own VTL value domain
    - VTL_Editor_PL.gui.UserDefinedValueDomainSubsetPanel: this user defined control allows the user to create a his own VTL value domain subset
    - VTL_Editor_PL.gui.frmAddText: this form is in charge of allowing the user to insert the label of a new language during the creation of a user defined structural metadata
    - VTL_Editor_PL.gui.frmArtefactDescription: this form is used during the creation of a user defined structural metadata to allow the user to adding a localized description
- Import structural metadata
    - VTL_Editor_PL.gui.frmMetadataImport : this form allows the user to import structural metadata from a SDMX 2.1 web service.
- Browse already uploaded structural metadata

    - VTL_Editor_PL.frmMain: this form, also, allows the browsing of the already imported structural metadata
- Write user defined operator and ruleset
    - VTL_Editor_PL.gui.frmDefineDatapointRule: this user defined user control helps the user to write a user defined Data point ruleset
    - VTL_Editor_PL.gui.frmDefineHierarchicalRule: this user defined user control helps the user to write a user defined Hierarchical ruleset
    - VTL_Editor_PL.gui.frmDefineOperator: this user defined user control helps the user to write a user defined operator
    - VTL_Editor_PL.gui.frmSaveOperator: this form is in charge of asking the user if and how to save a user defined VTL artefact.
- Configure settings
    - VTL_Editor_PL.classes.config.MainApplicationSettings: it extends the ApplicationSettings class adding the LoadSettings and ReplaceSettings method to load and store changes to the user settings
    - VTL_Editor_PL.gui.frmSettings : this form is used to access and to modify the user settings
    - VTL_Editor_PL.gui.frmAddWebService: this form allows the user to insert a new web service. It is used by the frmSettings to add the information concerning the InteractionWS and the SDMX WS from which to get the structural metadata

Additional classed are aimed to support the workflow of the VTLEditor graphical interface

- VTL_Editor_PL.classes.common.BaseJSonMessage: this class contains the code to parse a Json Message and to return a c# object
- VTL_Editor_PL.classes.common.CommonItem: this class contains the constants and the object that are common in the application: ErrorManager and ApplicationSettings
- VTL_Editor_PL.classes.net.VTLInt_ServiceManager : this class in in charge of creating an object that contains all the information required to connect the application to the Interaction WS on the base of the user settings
- VTL_Editor_PL.classes.tool.ArtefactNodeInfo: this class is aimed to contain all the information of a VTL Artefact when it is attached ad Tag to a TreeViewNode
- VTL_Editor_PL.classes.tool.AutocompleteVocabulary: this class contains the method required to manager the list of the operators and tokens required by the fastCodeTextBox to implement the autocomplete mechanism
- VTL_Editor_PL.classes.tool.ErrorManager: the ErrorManager manages in generalized way all the error events in case of raised Exectpion
- VTL_Editor_PL.classes.tool.TreeViewElementFinder: it is a class implemented to find a TreeViewNode in a TreeView by the content of its text or node tag
- VTL_Editor_PL.gui.frmErrorDetails: this form is used by the ErrorManager to display the error of an Exception and its detail
- VTL_Editor_PL.gui.frmSelectValueDomain: this form, used in more than one feature of the VTLEditor, allows the user to select a VTL dataset , already present into the VTL-DB
- VTL_Editor_PL.gui.frmSQLViewer: this form is in charge of showing the SQL code as result of the translation process
- VTL_Editor_PL.gui.Switch: this user defined control is used in more than one form in the VTLEditor interface to throw the event needed to switch from the description of an artefact to its code and vice-versa.

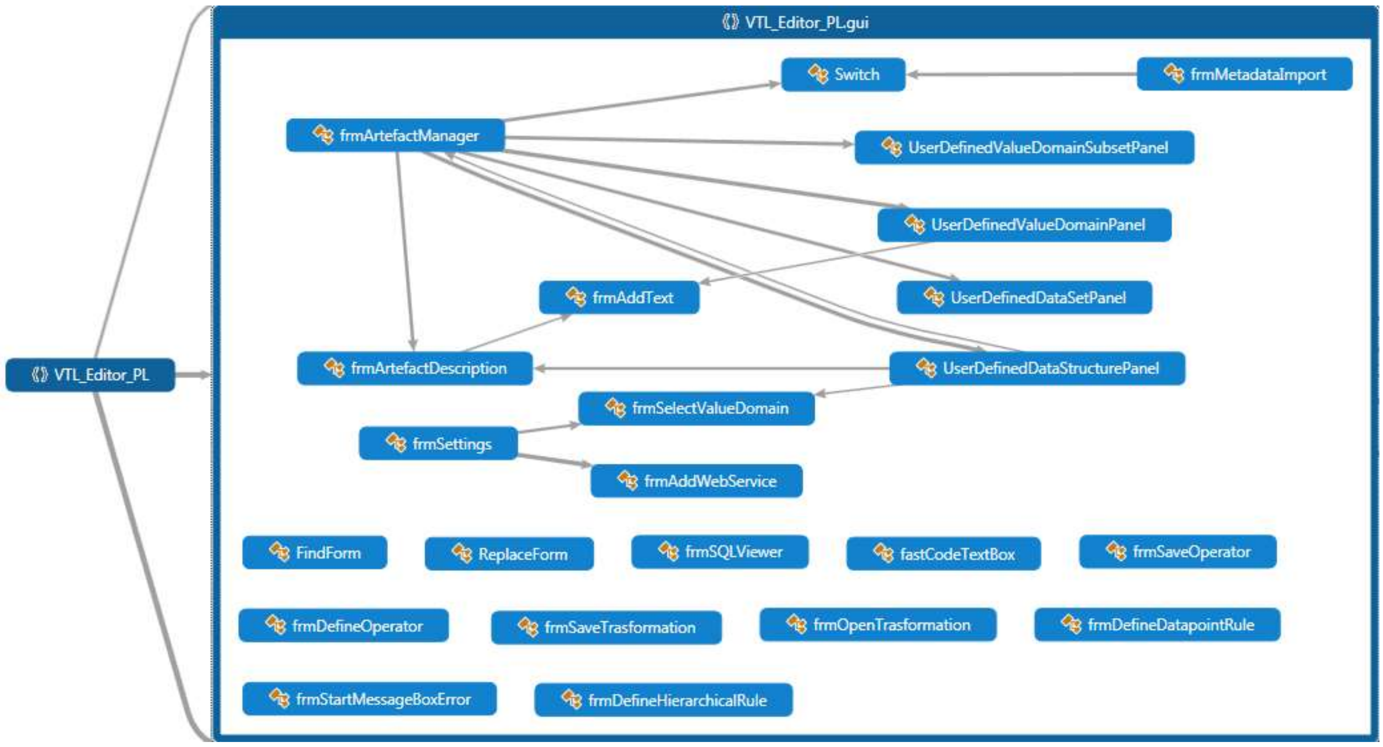


Figura 26 - Graphic objects of the VTLEditor

Some specific features of the graphical interface have been designed and developed in separated projects, because they can be reused by other components of the infrastructure or because they may be subject to changes due to the new version of the EBNF.

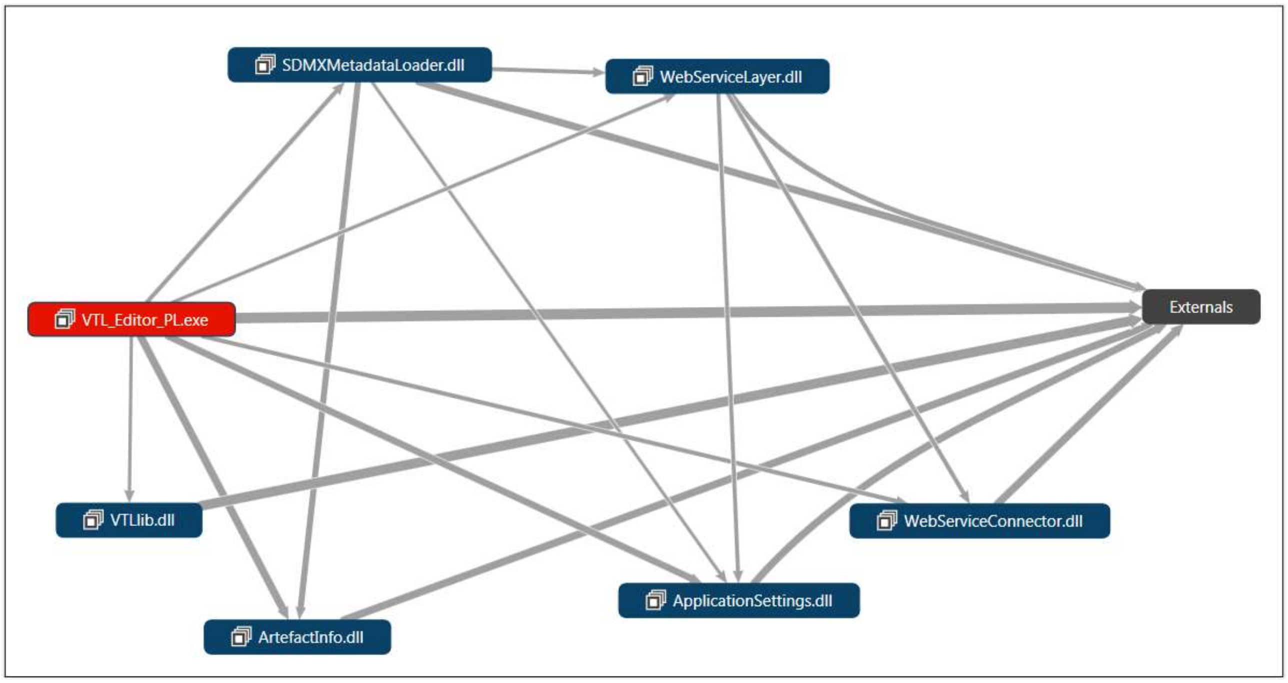


Figura 27 - Additional shared classes

The specific class VTL_Lib contains all the information used by the GUI to show the information of the VTL operators in the operator browser

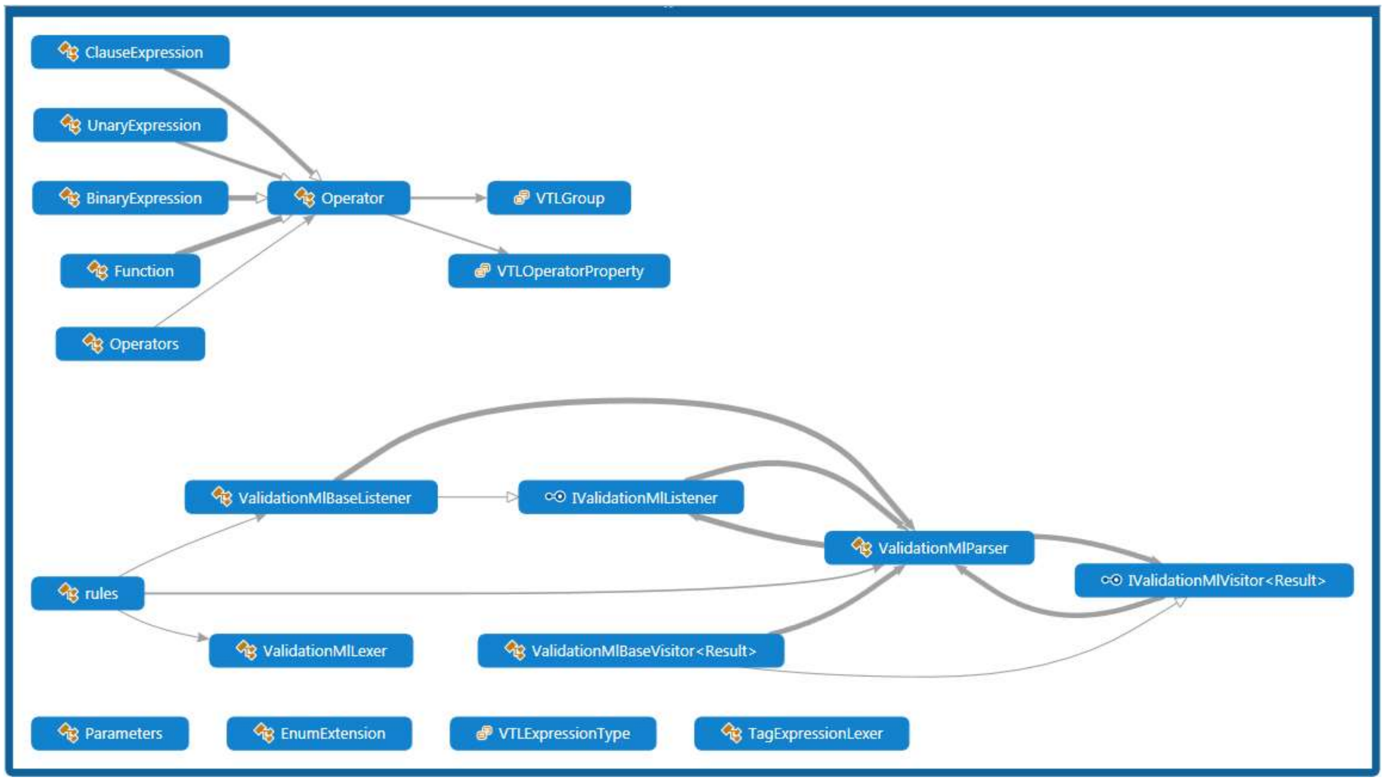


Figura 28 - VTL_Lib

It is composed by:

- VTLlib.Operator: Define the VTL operator
- VTLlib.Operators: Define the list of VTL operators
- VTLlib.DML: Defines the VTL expressions
- VTLlib.Rules: Defined the VTL rules

In order to implement the autocomplete system a specific XML file was developed due to the complexity and the ambiguities of the EBNF.

```
<AutoCompleteVocabulary>

  <operators>
            <operator ID="sum">
                     <method>sum(^)</method>
                     <description>sum(operand)</description>
                     <help>The operator returns the sum of the input values.</help>
            </operator>
           <operator ID="rtrim">
                     <method>rtrim(^)</method>
                     <description>rtrim(operand)</description>
                     <help>Removes trailing or/and leading whitespace from a string.
                          </help>
            </operator>
             […]

      </operators>
        <keywords>
              <key>aggr</key>
              <key>aggregates</key>
               <key>all</key>
               <key>all_measures</key>
                  […]
        </keywords>
 <declarations>
          <declaration>if ^ then thenOperand else elseOperand</declaration>
  </declarations>
[…]

</AutoCompleteVocabulary>
```

Figura 29 - xmlAutoComplete.xml

Shared DLL:

ArtefactInfo: this set of classes represent the translation in a C# structure of the VTL artefacts

- ArtefactInfo.model.BaseArtefactInfo: the base class that contains the common info of all the VTL artefacts
- ArtefactInfo.model.BaseComponentInfo: the base class that contains the common info of all the VTL components
- ArtefactInfo.model.BaseUserDefinedOperator: the base class that contains the common info of all the VTL operators
- ArtefactInfo.model.CommonConstant: it defines some constants used in the ArtefactInfo set of classes
- ArtefactInfo.model.DataSetInfo: it extends the BaseArtefactInfo to contain the additional info concerning the VTL Dataset
- ArtefactInfo.model.DataStructureComponentInfo: It extends the BaseArtefactInfo to contain the additional info concerning the VTL DataStructure
- ArtefactInfo.model.LocalizedValue: this class contains the structure of the VTL Localized Value
- ArtefactInfo.model.MainteinableSDMXArtefact: it represents a generic SDMX maintainable artefact
- ArtefactInfo.model.TimeInfo: it is a class created to manage the information contained into a SDMX Time Dimension

- ArtefactInfo.model.ValueDomainInfo: it extends the BaseArtefactInfo to contain the additional info concerning the VTL Value Domain
- ArtefactInfo.model.ValueDomainInfoInDataStructure: it contains all the info to link a Value Domain to a Data Structure
- ArtefactInfo.model.ValueDomainSubsetInfo: it extends the BaseArtefactInfo to contain the additional info concerning the VTL Value Domain Subset
- ArtefactInfo.VTL.VTL_Model: it defines the VTL DataType

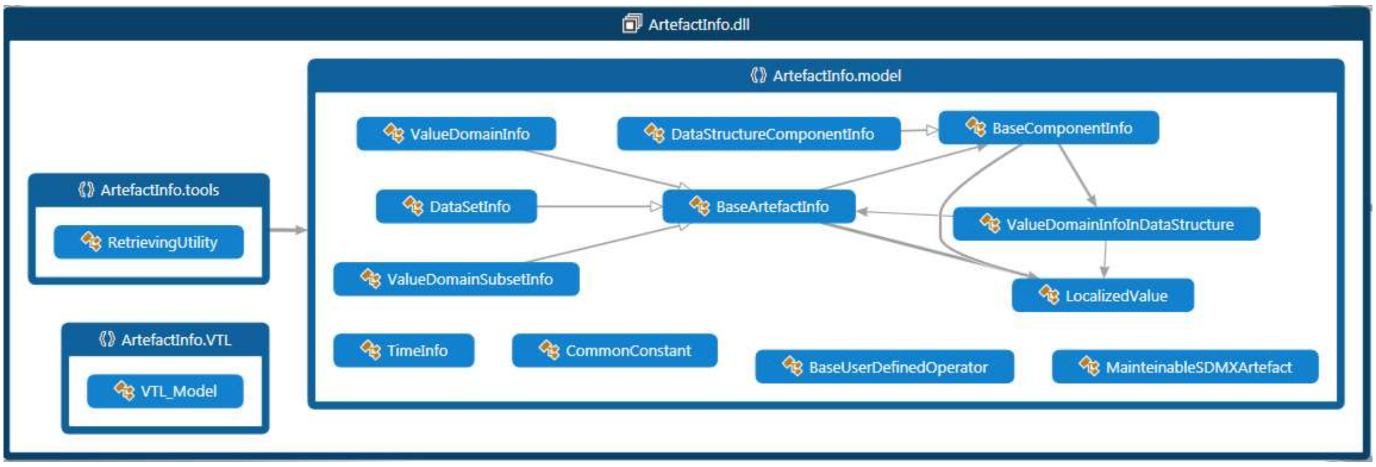


Figura 30 - ArtefactInfo

ApplicationSettings: this set of classes is in charge to represent the configuration file and to manage it loading, saving and updating it.

- ApplicationSettings.classes.common.CommonConst: it defines some constants used in the ApplicationSettings set of classes
- ApplicationSettings.classes.services.common.CommonNamespace: it defines as constant string the SDMX Namespace used into the project
- ApplicationSettings.classes.services.ApplicationSettings: it is the mail class of the package it set of classes
- ApplicationSettings.classes.utility.StringCryptography: it implements algorithm to crypt or decrypt a string.

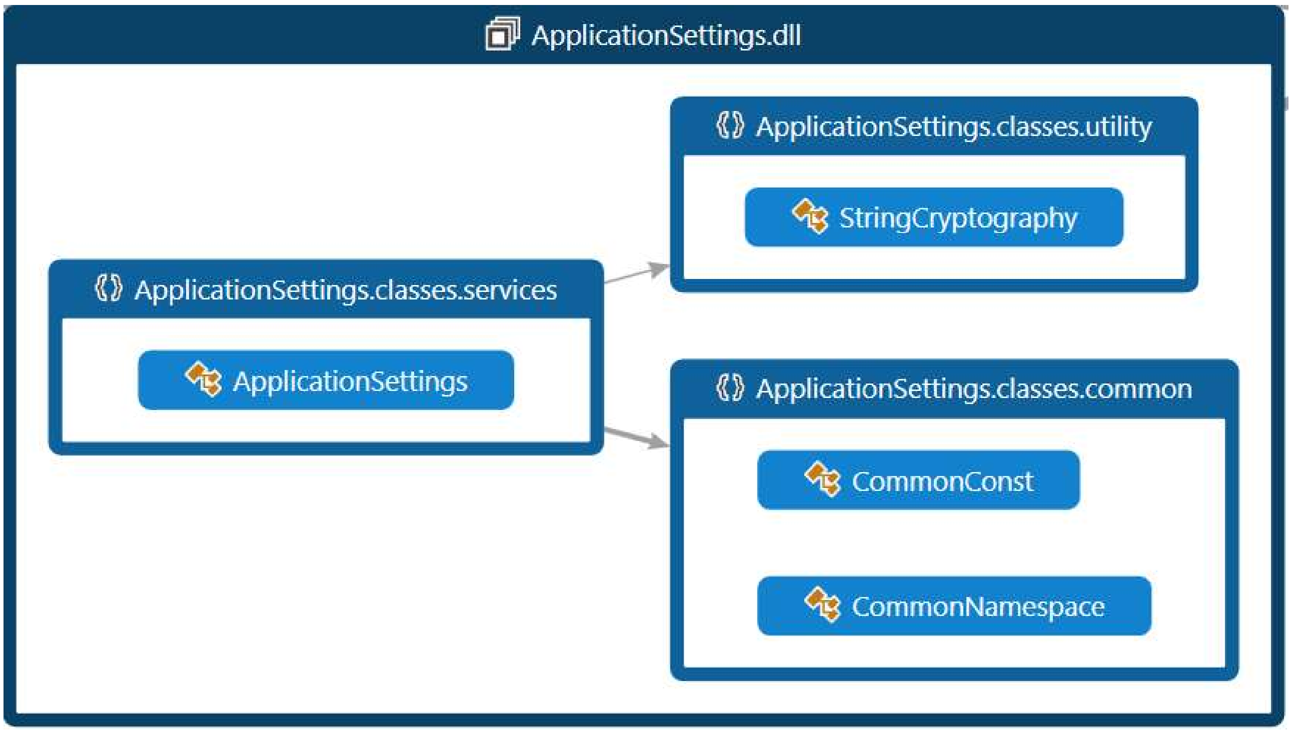

ApplicationSettings.dll
ApplicationSettings.classes.utility
StringCryptography
ApplicationSettings.classes.services
ApplicationSettings
ApplicationSettings.classes.common
CommonConst
CommonNamespace

Figura 31 - ApplicationSettings

SDMXMetadataLoader: this set of classes is in charge of parsing of a SDMX file to get DSD, Codelist and Dataflows from a SDMX webservice

- SDMXMetadataLoader.service.CommonNamespace: it defines as constant string the SDMX Namespace used into the project
- SDMXMetadataLoader.service.MetadataLoader: gets the structural metadata (Codelist, DSD and Dataflow) from a SDMX webservice 2.1
- SDMXMetadataLoader.service.MetadataRetriever: this class is in charge of parsing the structure SDMX XML files that contain the Codelist, DSD or dataflow

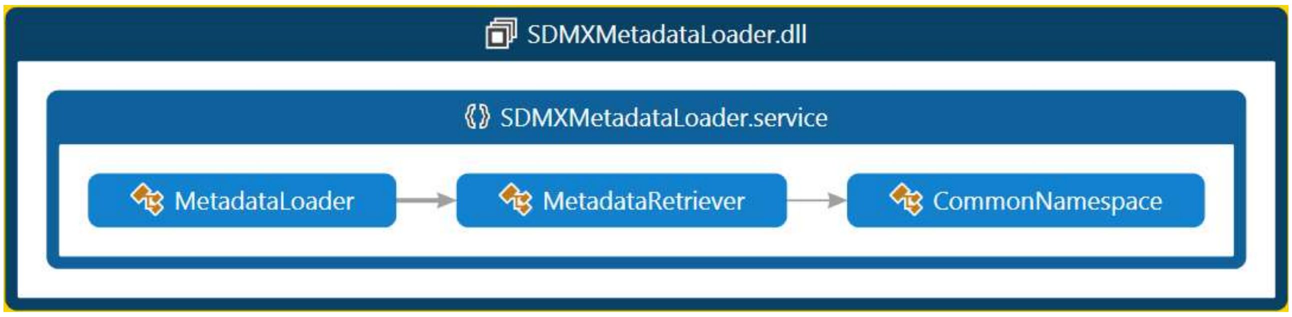


Figura 32 - SDMXMetadataLoader

WebServiceConnector: this set of classes is in charge of management of the calls to a soap or rest webservice

- WebServiceConnector.ResponseUtil: this class contains some useful method to manage a resulting message
- WebServiceConnector.RestWebConnector: this connector manages a Rest call to a web service
- WebServiceConnector.WebServiceClient: it implements the method to send a request to a web service and to get a result
- WebServiceConnector.WebServiceClientStreaming: it implements the method to send a request to a web service and to get the resulting stream in a synchronous and asynchronous way

- WebServiceConnector.WebServiceConstants: this class contains the constants of the WebServiceConnector.dll
- WebServiceConnector.WsConfigurationSettings: this class contains all the information required to perform a web service connection to a specific WS
- WebServiceConnector.WSDLSettings: this information contains all the information to access a WSDL file

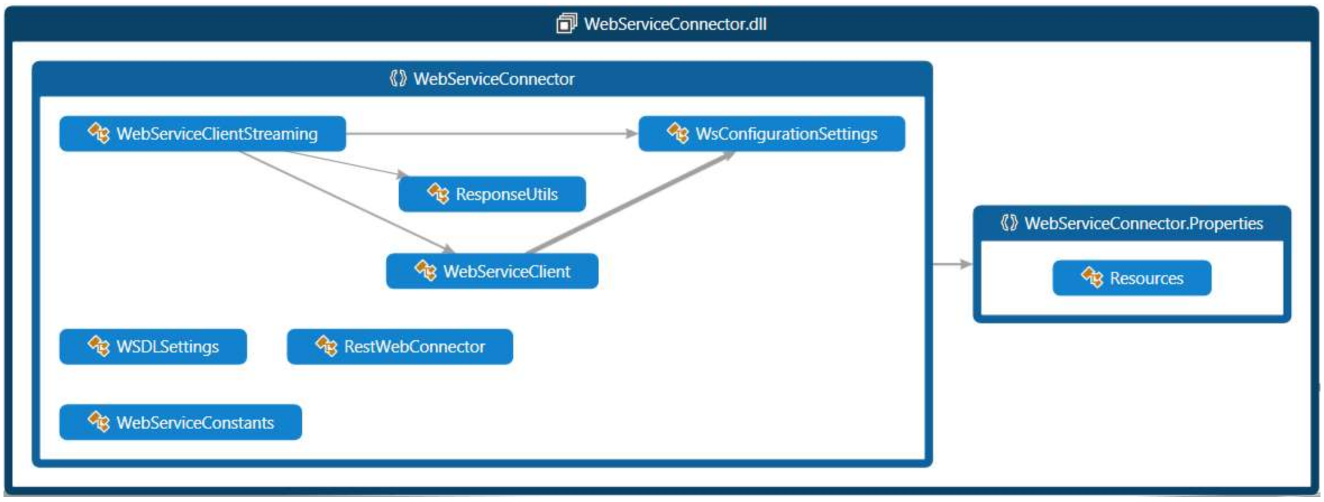


Figura 33 - WebServiceConnector

WebServiceLayer: this set of classes is aimed to create a layer to manage all the information concerning a web service connection

- WebServiceLayer.classes.services.Net.WebServiceLayer: this class contains all the information needed to create a connection to a Web service

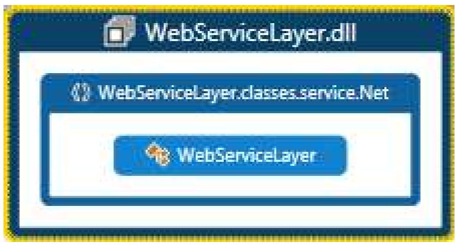
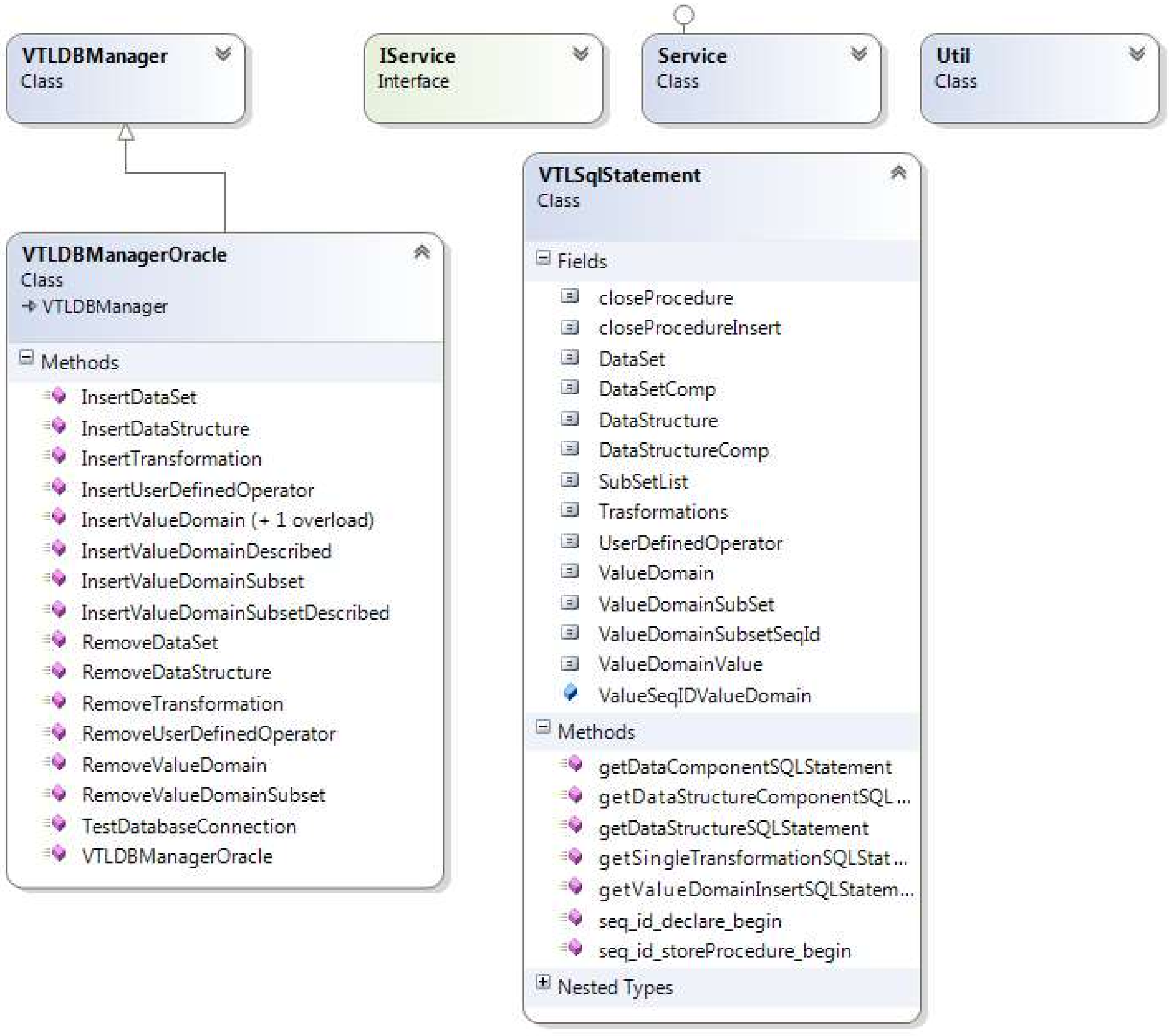


Figura 34 - WebServiceLayer

Interaction WS

VTLInt_WS : this application represents the Interaction Web Service that is in charge of create a middle layer between the user and the VTL-DB.

- VTLInt_WS: it is the front-end of the Interaction Web Service
- VTLInt_WS.VTLDB.Util: a set of utilities used into the web service
- VTLInt_WS.VTLDB.VTLDBManager: the main class to interact with the VTL-DB
- VTLInt_WS.VTLDB.VTLDBManagerOracle: it inherits the VTBDBManager class to be used specifically with Oracle
- VTLInt_WS.VTLDB.VTLSqlStatement: the SQL statement used to interact with the VTL-DB

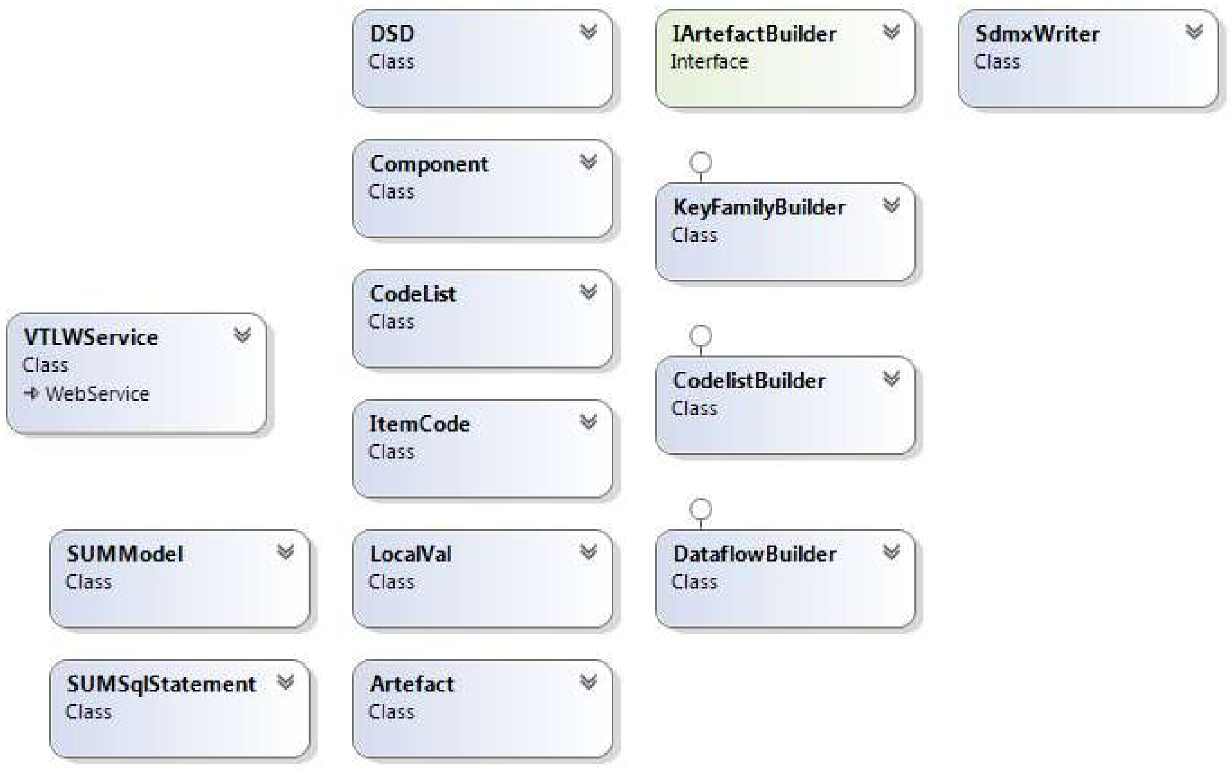


SUM_WS

IstatVTLWebService: it implements the web method GetDataStructure, GetCodelist and GetDataflow to get structural metadata from the Istat's Unitary Metadata System and return them in an SDMX format

- IstatVTLWebService.SUMService: the main component of the web service
- IstatVTLWebService.Model.SUMModel: it contains the classes to get the structural metadata from the SUM database

- IstatVTLWebService.SUMSqlStatement: the SQL statements created to access the SUM DB
- IstatVTLWebService.Builder.IArtefactBuilder: It is the main class that contains all the code to create a generic SDMX artefact
- IstatVTLWebService.Builder.CodelistBuilder: it extends the IArtetafactBuilder to create a Codelist
- IstatVTLWebService.Builder.DataflowBuilder: it extends the IArtetafactBuilder to create a Dataflow
- IstatVTLWebService.Builder.KeyFamilyBuilder: it extends the IArtetafactBuilder to create a DSD
- IstatVTLWebService.Writer.SdmxWriter: it implements the methods required to write a SMDX structure file
- IstatVTLWebService.Artefact: it contain all the information of a generic SDMX artefact
- IstatVTLWebService.CodeList: it contains the information of a SDMX codelist
- IstatVTLWebService.Component: it contains the information of a SDMX component
- IstatVTLWebService.DSD: it contains the information of a SDMX DSD
- IstatVTLWebService.ItemCode: it contains the information of a SDMX item



ISTAT_DB_DAL: It implements all the method to access the Unitary Metadata System database

- DataProvider: it implements all the method to create a connection to the SUM's.
- ProviderType: it containes an enum to specify the db type