On the fly

Technical Guide

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type of Document** | Technical Guide | | | | |
| **Reference:** |  | | | | |
| **Issue:** |  | **Revision:** |  | **Status:** | Draft |
| **Created by:** | Gino Mascotti | | | **Date:** | 23/12/2014 |
| **Updated by:** | Gino Mascotti | | |  | 09/06/2015 |
| **Approved by:** |  | | | | |

SUMMARY

[**Project description** 3](#_Toc408584668)

[Architecture 3](#_Toc408584669)

[View Level 3](#_Toc408584670)

[Control Level 4](#_Toc408584671)

[Model Level 5](#_Toc408584672)

[Structural cycle 6](#_Toc408584673)

[Implementations Management 7](#_Toc408584674)

[SDMX Common API 7](#_Toc408584675)

[Model 8](#_Toc408584676)

[View 8](#_Toc408584677)

[OnTheFly 8](#_Toc408584678)

[SOAPSdmx 10](#_Toc408584679)

[RestSdmw 11](#_Toc408584680)

[Utils 12](#_Toc408584681)

[Control 13](#_Toc408584682)

[FlyEngine 13](#_Toc408584683)

[FlyController 20](#_Toc408584684)

[FlyLog 24](#_Toc408584685)

[Model 25](#_Toc408584686)

[FLYMapping 25](#_Toc408584687)

[FlyDotStat\_Implementation 26](#_Toc408584688)

[FlySDDSLoader\_Implementation 28](#_Toc408584689)

[Test Client 29](#_Toc408584690)

[FlyCallWS 29](#_Toc408584691)

[Pubblicazione 30](#_Toc408584692)

# **Project description**

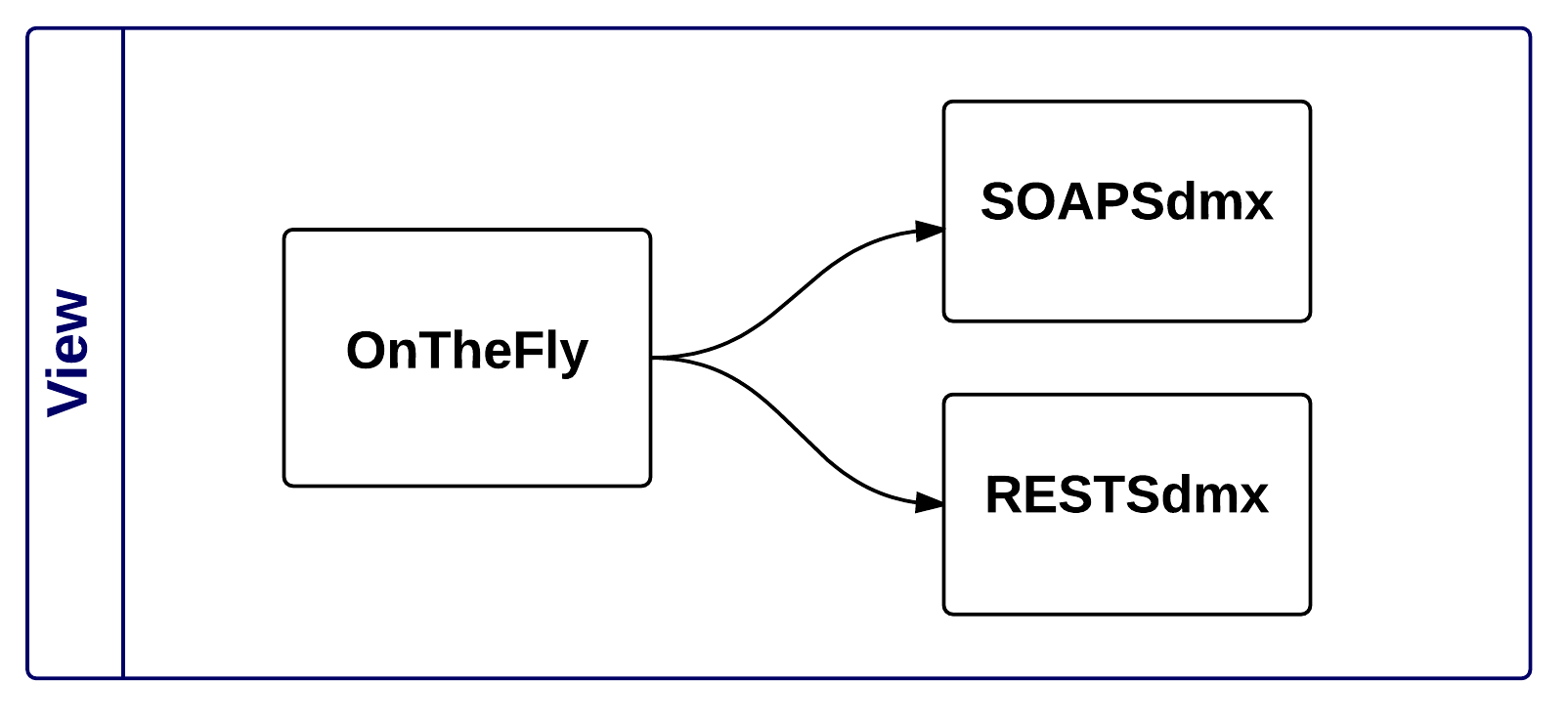
The project OnTheFly WebServices was created to execute queries SDMX on different types of databases.  
Is not necessary to have the database structured according to a specific model, but some stored procedures are enough to perform all queries and create all required structures "On the Fly".

## Architecture

The project OnTheFly respects the architecture of three-tier model

Model-View-Control + (Implementations)

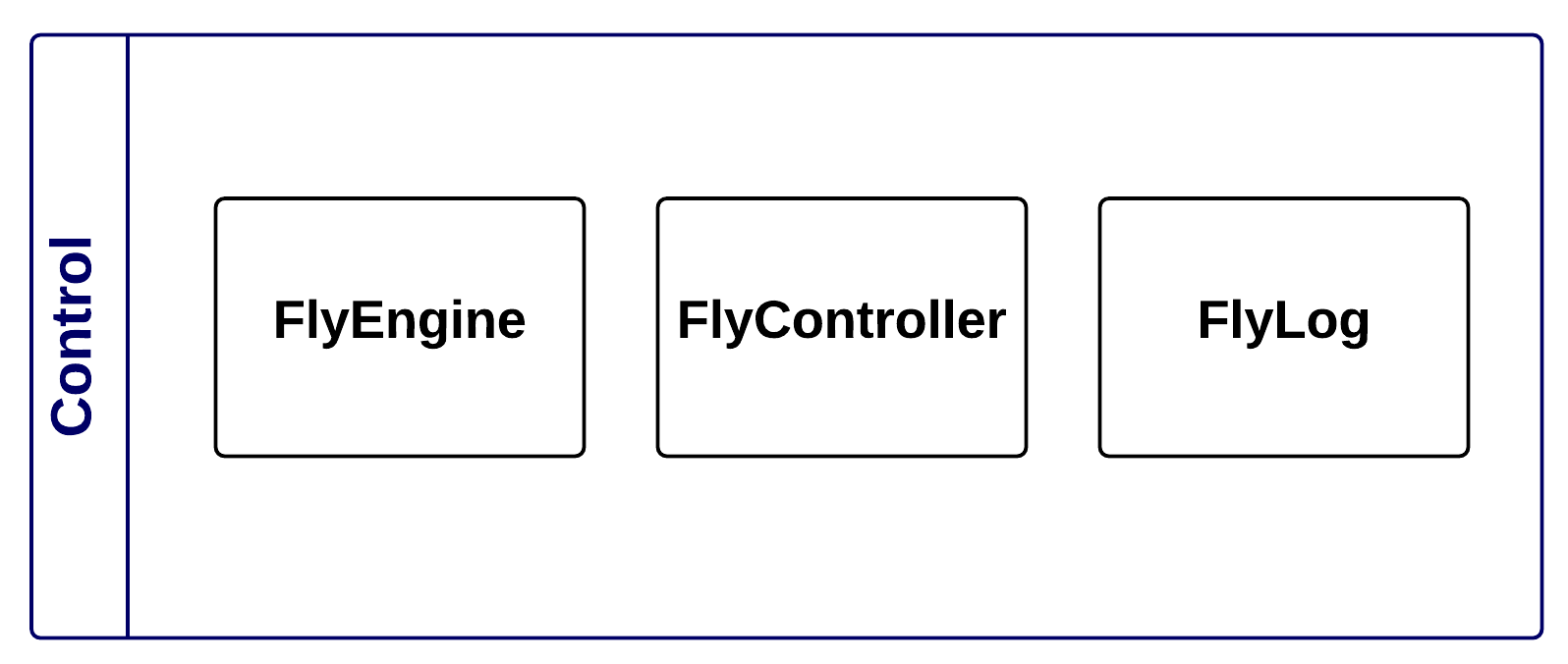
### View Level



**The View level** is represented by the projects that are accessible to the end user:

* OnTheFly
* RestSdmx
* SoapSdmx

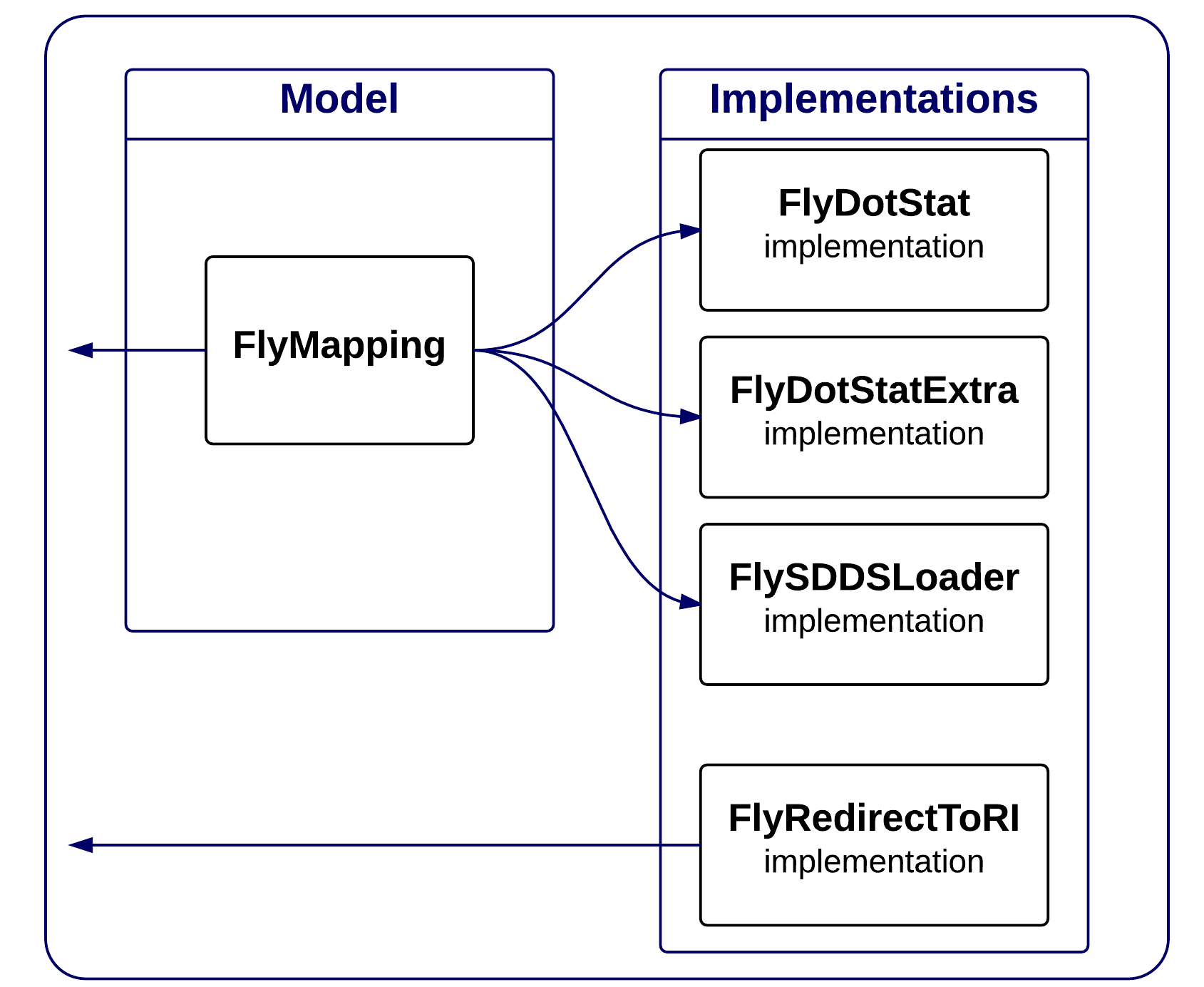
### Control Level



**The Control level** is represented by all the projects used to parse the requests arising from the View level, request data to the Model level . It requests data to the Model level and processes data by creating the structures, the standards SDMX objects and provide a response to the View level

* FlyEngine
* FlyController
* FlyLog

### Model Level



**The Model level** is the project delegated to direct the Control level requests to the Implementations level, and it provides all the necessary interfaces for creating new implementations

* FlyMapping

**The Implementations level is represented by all the projects used to create database queries by calling the correct Stored Procedure, each Implementations project has basic interfaces exposed by the Model level**

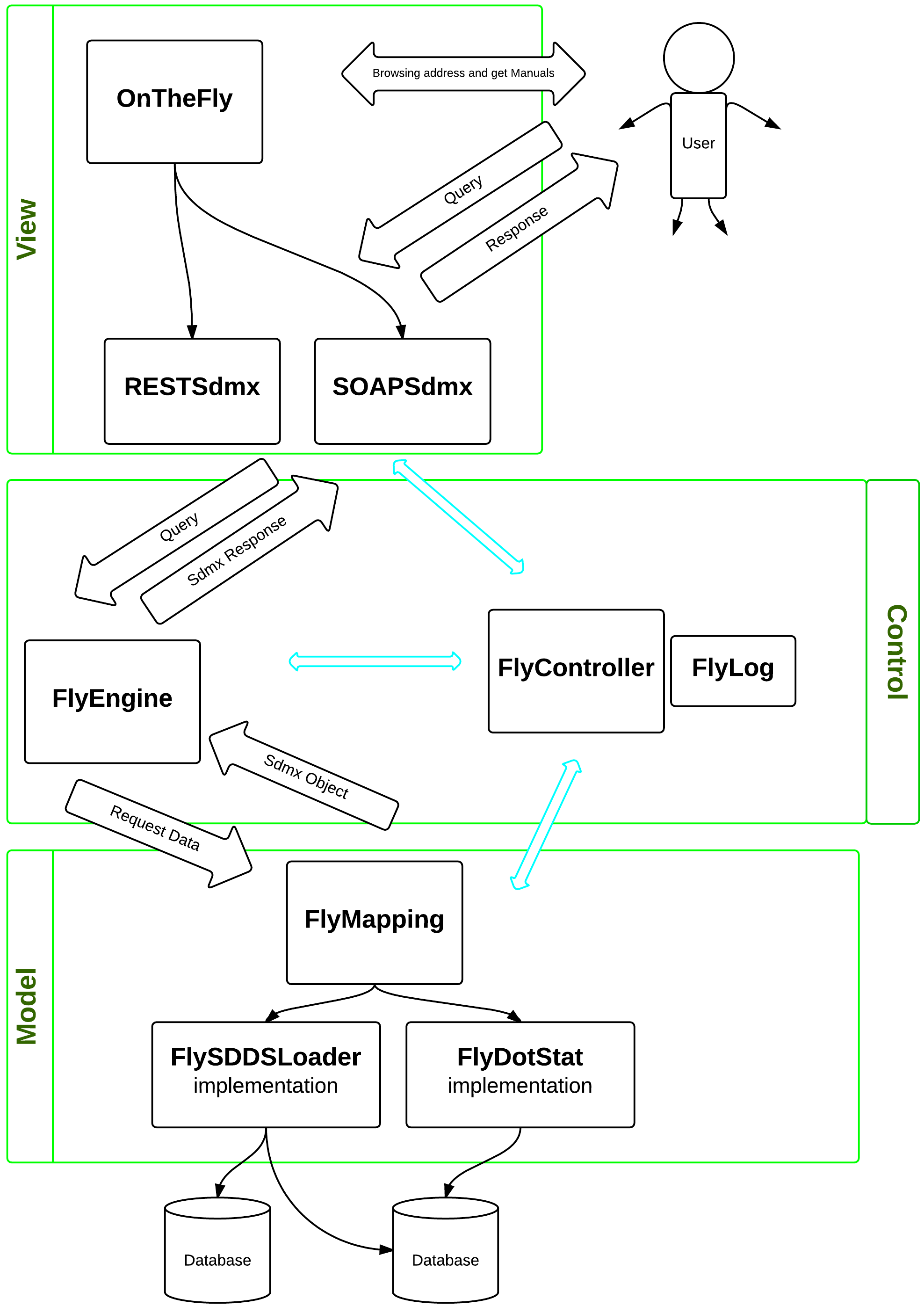
* FlyDotStat\_implementation
* FlyDotStatExtra\_implementation
* FlySDDSLoader\_implementation

È stato aggiunto tra le implementazioni un modulo completamente diverso dagli altri

* FlyRedirectToRI\_implementation

Questo non implementa le funzionlità di FlyMapping ma viene richiamato direttamente dal View Level.

## Structural cycle



In this diagram you can see how the system performs the steps between levels:  
  
The end user is able to see the homepage of the service where the entrypoint of the application is visible, and he can downloads the manuals and the test client.  
  
The FlyMapping initialize the correct implementation and will invoke the correct method for retrieving data and then pass it as SdmxObject to FlyEngine..  
  
The FlyEngine will handle the query and make the request to the correct FlyMapping.  
  
The FlyMapping initialize the correct implementation and will invoke the correct method for retrieving data and then pass it as SdmxObject to the FlyEngine.  
  
The FlyEngine receives data from the Model level and creates the correct answer to pass to the two WCF webServices, SOAPSdmx and RESTSdmx, and will direct the user to the respective protocols..

## Implementations Management

It's possible to create new project implementations if the user wants to retrieve data in a different way, by other sources, and then extract the data in a different way from the current OnTheFly project.For creating a new implementation you must follow these steps:

1. Create a new project
2. Modify the Switch in FlyEngine to direct the program to the new implementation FlyEngine.Model.MappingConfiguration
3. Develop the classes deriving from the implementation of the FlyMapping interfaces

## SDMX Common API

The OnTheFly project uses CommonAPI of SDMX in most of its projects.

The CommonApi are mainly used for:

* Use the objects SDMX, constants, and enumerators
* Creating new objects from objects SDMX
* Make the query Parsing
* Create the answers and write them directly into a stream Rest or Soap :
* ISdmxObject for Metadata
* IDataWriterEngine for Data

# Model

This chapter will explain the features of all projects divided by level.  
All projects have been created with .NET technology  
VisualStudio 2012  
.NET Framework 4  
The C # language

## View

This is the level of interfacing with the user.

### OnTheFly

This is the starting point of the whole application. this project has two Asp.Net functions:

1. Show to the user a Welcome Page where he can see the addresses of etrypoint application and he can download manuals and test client.

In the Default.aspx page it’s possible to change the welcome page

1. Create the services with Windows Comunication Foundation tecnology

The most important thing in this project is:

the WebConfig that is called from IIS during installation. In this file are specified:

* The uri schemeSettings that must have the option DontUnescapePathDotsAndSlashes.
* The system.webServer security that should allow DoubleEscaping
* The routing settings
* The system.serviceModel that must have aspNetCompatibilityEnabled

There are other settings that are used for a correct application working

The Global.asax, creates the WSDL in the way described below and is responsible for routing requests, translation of addresses with double points and of addresses that end up with wsdl

WSDL Build

In the Global.asax of the OnTheFly project, there is the method that deals with the building of the WSDL.

This file takes advantage of the two files at the project level

* OTFSdmx20.wsdl
* OTFSdmx21.wsdl

In these two files there are all the information to successfully create the services and descriptions of the EntryPoint with their input parameters and return values.  
This file also provides information about the scheme, definition and namespace  
  
In the folders

* Sdmvv20
* Sdmxv21

It contains all the xsd file for the query validation.

In the Global.asax method then are created the routing tables of the whole application

* RouteTable.Routes.Add(new ServiceRoute("SoapSdmx20", new SoapServiceHostFactory(typeof(IOnTheFly\_SOAP\_SDMX\_v20), "OTFSdmx20.wsdl"), typeof(OnTheFly\_SOAP\_SDMX\_v20)));
* RouteTable.Routes.Add(new ServiceRoute("SoapSdmx21", new SoapServiceHostFactory(typeof(IOnTheFly\_SOAP\_SDMX\_v21), "OTFSdmx21.wsdl"), typeof(OnTheFly\_SOAP\_SDMX\_v21)));
* RouteTable.Routes.Add(new ServiceRoute("rest/data", new SdmxRestServiceHostFactory(typeof(IDataResource)), typeof(DataResource)));
* RouteTable.Routes.Add(new ServiceRoute("rest", new SdmxRestServiceHostFactory(typeof(IStructureResource)), typeof(StructureResource)));

the other two projects SoapSdmx and RESTSdmx are also initialized.

### SOAPSdmx

This project was created as a simple library of classes, it will be loaded as a WCF Web Service.

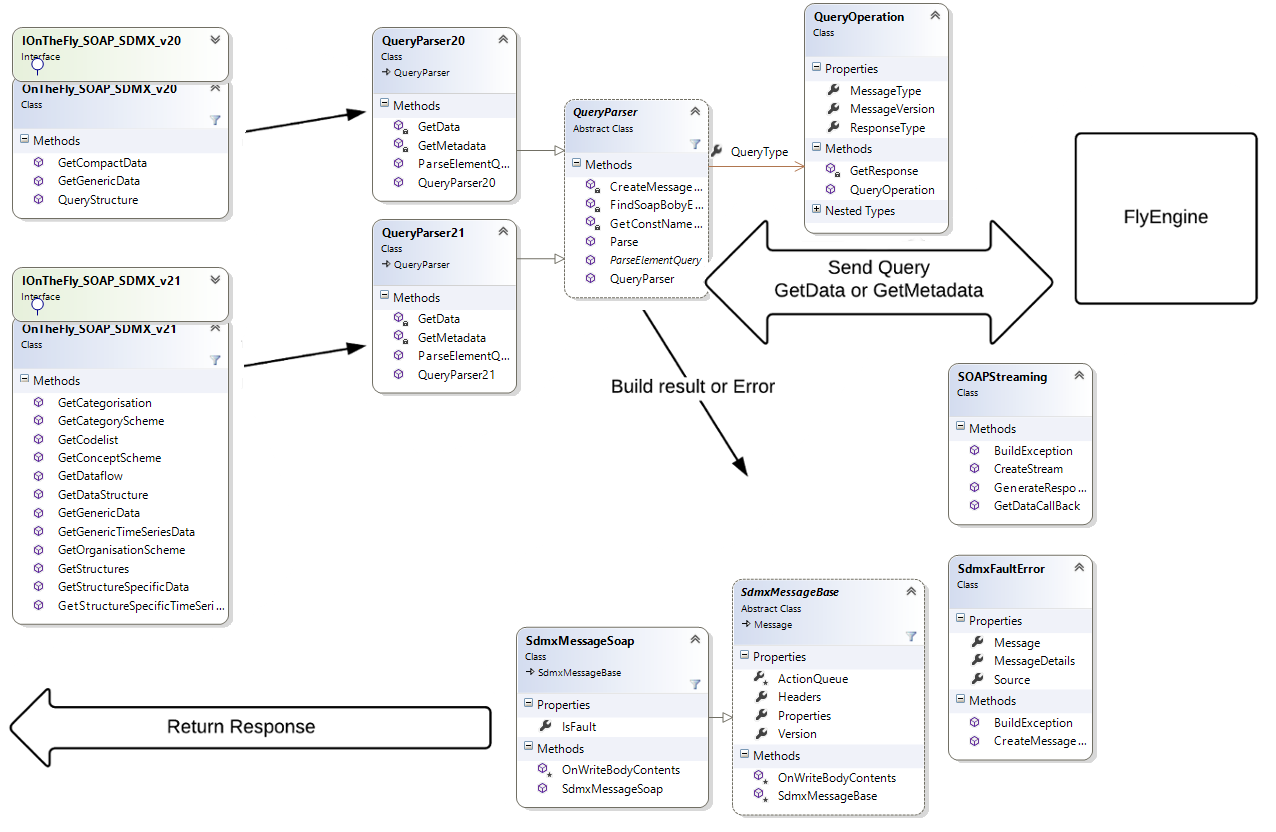
Like all the webservices, it's composed by interfaces (ServiceContract) that define the entry point that will be used by a class. This class extends the same interface that will serve as the main class of service (ServiceBehavior)

In this project there are two interfaces

* IOnTheFly\_SOAP\_SDMX\_v20
* IOnTheFly\_SOAP\_SDMX\_v21

One for SDMX version 2.0 and the other for SDMX version 2.1

The query is processed and then are created structures that indicate the version and type, if are requested also data or metadata. In the case of Metadata is made a pre-parsing of the query to check the correctness or the existence within the project of the required functionality.  
  
At this point everything is passed to FlyEngine that respond with an object that indicates the response or the error that return to the user. This object is encapsulated in a SOAP message and is turned over to the user.



### RestSdmw

This project was created as a simple library of classes, it will be loaded as a WCF Web Service.

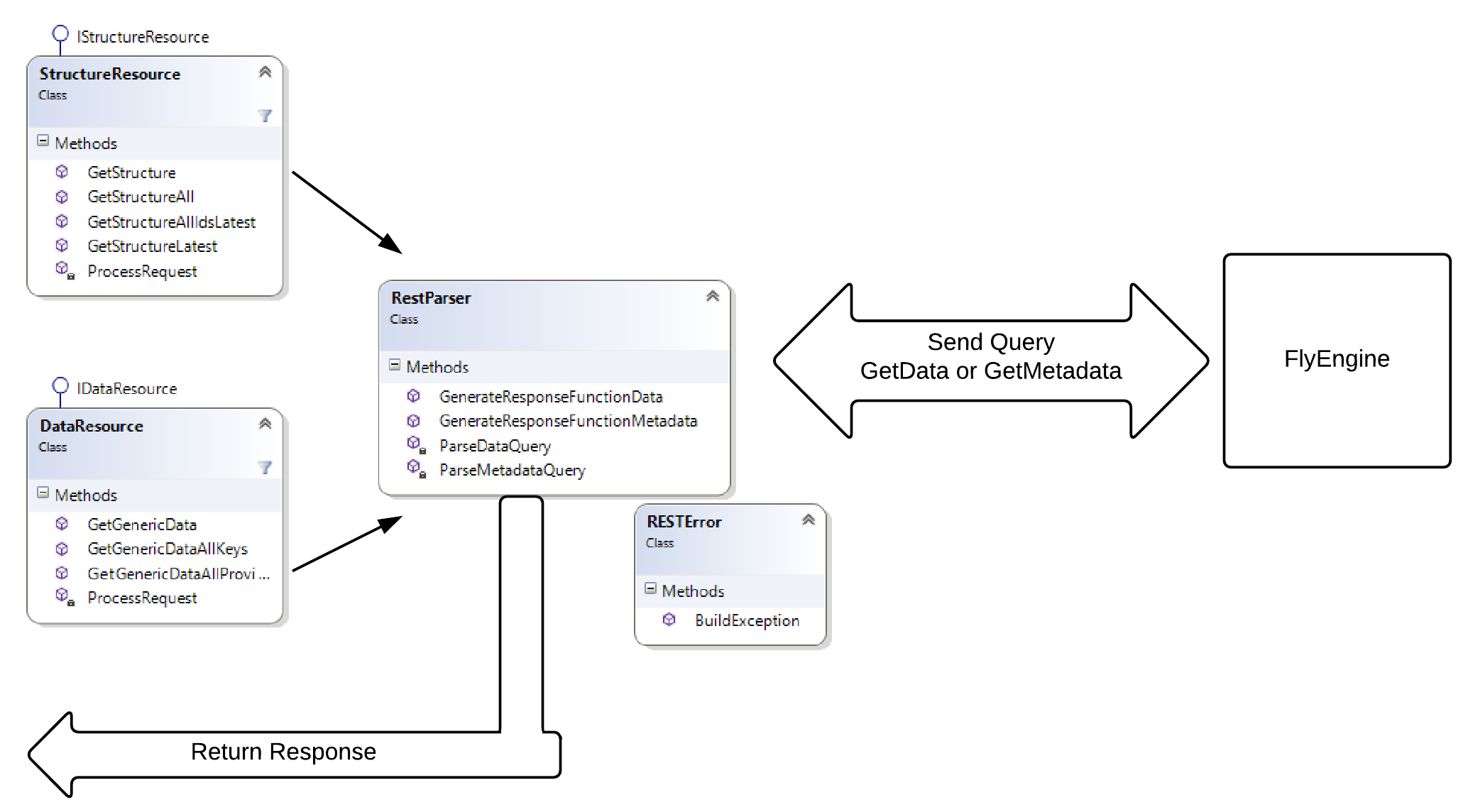
Like all the webservices, it's composed by interfaces (ServiceContract) ) that define the entry point that will be used by a class. This class extends the same interface that will serve as the main class of service (ServiceBehavior)

In this project there are two interfaces

* IStructureResource
* IDataResource

One for the Metadata and the other for Data

The query is processed and then are created structures that indicate the version and type, if are requested also data or metadata. In the case of Metadata is made a pre-parsing of the query to check the correctness or the existence within the project of the required functionality.  
  
At this point everything is passed to FlyEngine that respond with an object that indicates the response or the error that return to the user. This object is encapsulated in a REST message and is turned over to the user.



### Utils

This project was created only for the purpose of storing the files necessary for the operation of the entire application. This project is imported as a reference in the other and it moves the files inside the bin of the other projects.  
  
Contains:

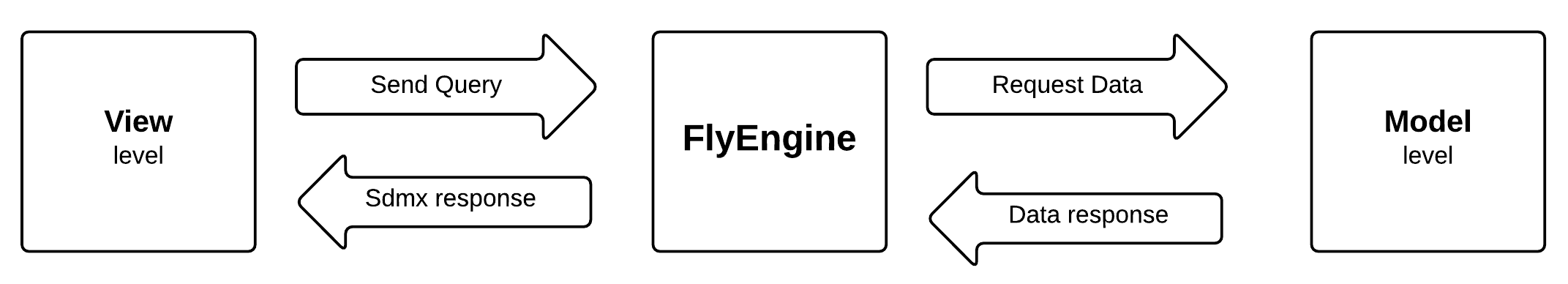
* folder doc, with all the documentation files
* folder Help, with the release of the document Documentation.chm
* folder Libswith all CommonApi libraries used throughout the application
* folders Queries, four folders with all the templates used for the queries in TestClient
* project SandCastle “On The Fly Documentation.shfbproj” for documentation

## Control

This level contains all projects that have the logic of construction and data recovery proceedings to the purpose of the application. These projects are the main part of the application because contain the methods based on the operation of the whole cycle structural application.

### FlyEngine

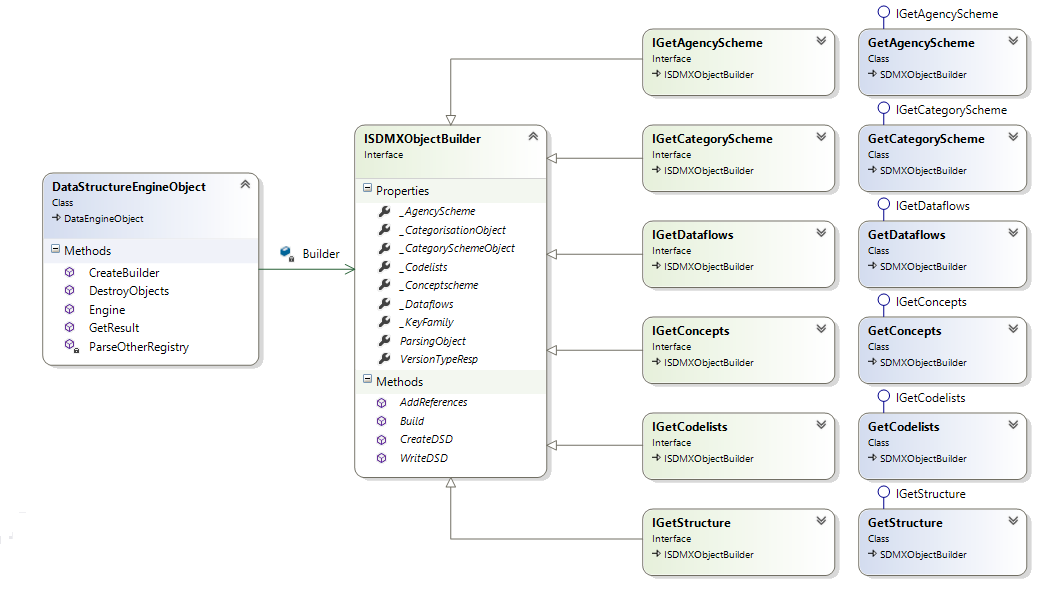
This project is the central engine of the whole application



As you can see from this project has the main features:

* Queries Parsing
* Data and Metadata request
* Creating SDMX structures of response

Metadata



The query gets to DataStructureEngineObject, and this processes it with CommonApi to check its accuracy and verify that there are no yntax errors. Then builds the two key objects for retrieval of all informations required

* ParsingObject
* VersionTypeResponse

These objects are then transported throughout the application and contain informations about the request of the query without parsing every time.

Is then created the *MetadataBuilder* correct

* GetAgencyScheme
* GetCategoryScheme
* GetCodelists
* GetConcepts
* GetDataflows
* GetStructure

These are all artifacts buildable from OnTheFly, if a request comes from a different artifact, the program generates an exception of type NotImplemented.

All these objects are encapsulated in a ISDMXObjectBuilder class and they populate the correct property for the both metadata required and then references.

The ISDMXObjectBuilder class will construct the SDMX object which will be returned on the basis of all the properties that have been populated.

RetreivalReferences

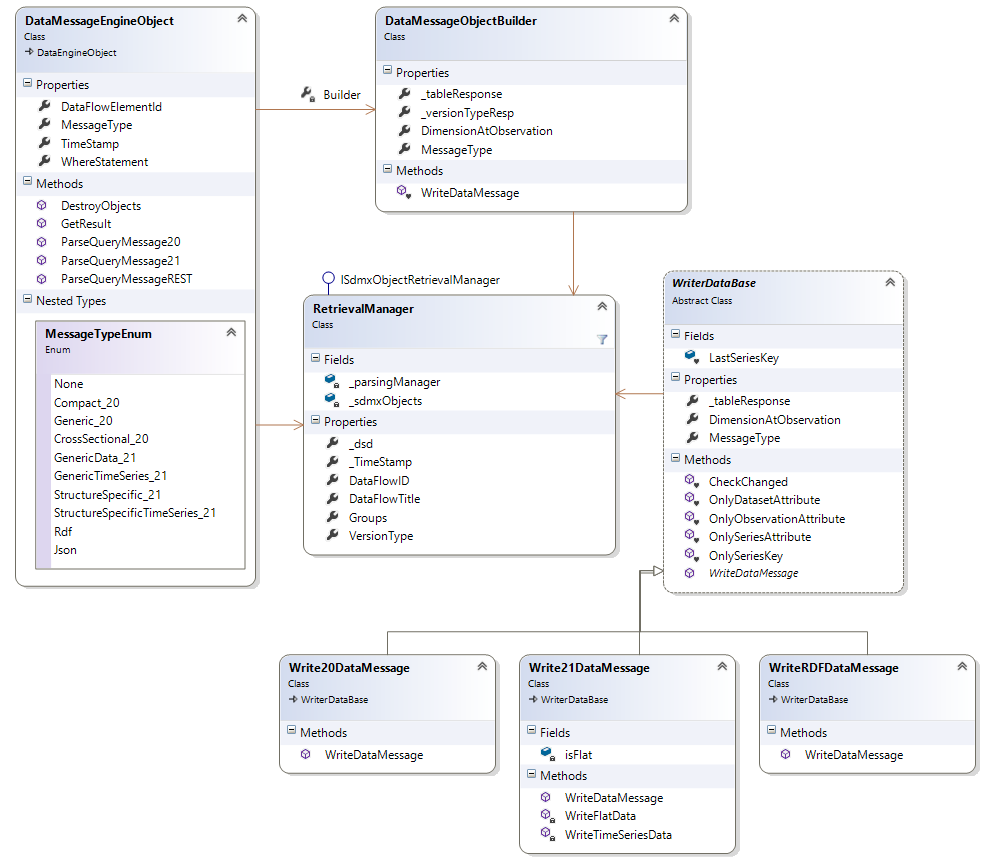
Each *MetadataBuilder* contains the information in ParsingObject, and its reference has been requested for the artifact.

The types of reference from characterized by SDMX version

|  |  |
| --- | --- |
| Sdmx v 2.0 |  |
| ResolveReference = False | Nothing References |
| ResolveReference = True | |  |  | | --- | --- | | Artefact | References | | CategoryScheme | Categorisations + Dataflows | | Dataflow | Categorisations + CategorySchemes + Dsd | | Dsd | ConceptScheme + Codelist | | ConceptScheme | Codelist | | Codelist | ConceptScheme | |

|  |  |
| --- | --- |
| Sdmx v 2.1 |  |
| None | Nothing References |
| Parents | |  |  | | --- | --- | | Artefact | References | | Categorisation | Nothing | | CategoryScheme | Categorisations | | Dataflow | Categorisations | | Dsd | Dataflow | | ConceptScheme | Dsd | | Codelist | Dsd | |
| ParentsAndSibling | |  |  | | --- | --- | | Artefact | References | | Categorisation | Nothing | | CategoryScheme | Categorisations + Dataflow | | Dataflow | Categorisations + CategoryScheme | | Dsd | Dataflow | | ConceptScheme | Dsd | | Codelist | Dsd | |
| Children | |  |  | | --- | --- | | Artefact | References | | Categorisation | CategoryScheme + Dataflow | | CategoryScheme | Nothing | | Dataflow | Dsd | | Dsd | ConceptScheme + Codelist | | ConceptScheme | Nothing | | Codelist | Nothing | |
| Descentands | |  |  | | --- | --- | | Artefact | References | | Categorisation | CategoryScheme + Dataflow + Dsd + ConceptScheme + Codelist | | CategoryScheme | Nothing | | Dataflow | Dsd + ConceptScheme + Codelist | | Dsd | ConceptScheme + Codelist | | ConceptScheme | Nothing | | Codelist | Nothing | |
| All | |  |  | | --- | --- | | Artefact | References | | Categorisation | CategoryScheme + Dataflow + Dsd + ConceptScheme + Codelist | | CategoryScheme | Categorisations + CategorySchemes + Dataflow | | Dataflow | Categorisations + CategoryScheme + Dsd + ConceptScheme + Codelist | | Dsd | Dataflow + ConceptScheme + Codelist | | ConceptScheme | Dsd + Codelist | | Codelist | Dsd + ConceptScheme | |

Data



The query object gets into DataMessageEngineObject that processes the query objetc with CommonApi to check it and verify that there are no Sintax Errors. It builds the key for data processing into RetreivalManager.

In the next step, will be called the methods of model level to retrieve data from database. The data won't be worked directly but the Model level will return a data pointer and it will write into a stream. This way will avoid OutOfMemory problems.

Will be instanced the right class method WriteDataBase that will write data to the communication channel. These Writers use their internal class IDataWriterEngine of CommonApi.

In this version are implemented only TimeSeries requests as

* Sdmx v2.0
* GetCompactData
* GetGenericData
* Sdmx v2.1
* GetGenericTimeSeriesData (only with ObservationAtObservation=TIME\_PERIOD)
* GetStructureSpecificTimeSeriesData (only with ObservationAtObservation=TIME\_PERIOD)
* GetGenericData (only with ObservationAtObservation=TIME\_PERIOD)
* GetStructureSpecificData (only with ObservationAtObservation=TIME\_PERIOD)

RetreivalManager

The RetreivalManager is the main object that parses data queries. It derives from the ISdmxObjectRetrievalManager of CommonApi and implements all the methods.

This object creates a special DSD from the query parsing to creates the response flow of data. The DSD is not visible in response. For optimize the processing was decided not to take all the items of the codelist but only the codelist.

MappingConfiguration

The MappingConfiguration is the static class responsible for direct requests between different implementations of Model level.  
  
The MappingConfiguration could be also be placed in FlyMapping but this is not possible because all implementations derive from FlyMapping and there would be a circular reference.  
  
The class consists of three functions

* MetadataFactory
* DataFactory
* GroupFactory

**MetadataFactory** is the function responsible for sorting all requests for metadata. This takes as a return value interface implementation IMetadataFactory, thus forcing implementer to build all the necessary methods to the return of all the metadata and their references  
  
**DataFactory** is the function responsible for sorting all requests for data. This takes as a return value of a class implementation IDataMessageManager that contains all the methods for creating delegates and pointers to the data tables to create a response stream.  
  
**GroupFactory** is the function responsible for sorting all requests to recover all the groups members to a data set.

### FlyController

This is the project responsible for providing all the projects of the system classes and methods of model objects and common use.

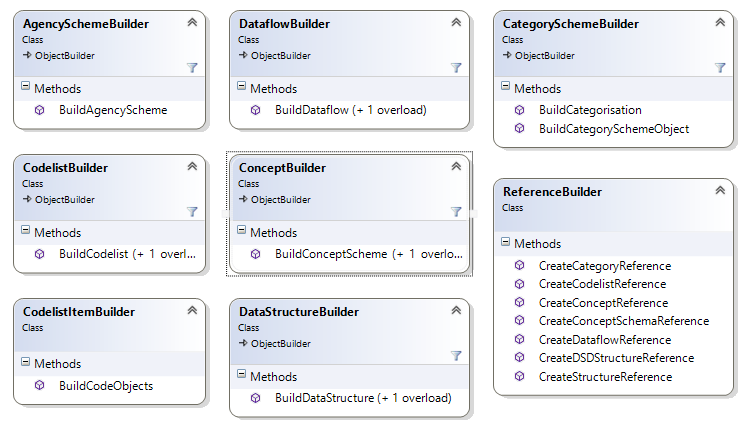
Configuration

The class FlyConfiguration reads once the application configuration file "ServiceConfiguration.xml" and sets the class’s static fields.  
  
  
  
The "OnTheFly UserGuide" contains informations about the parameters configuration.

To change the application version is necessary to modify the implementation of the OnTheFlyVersion property

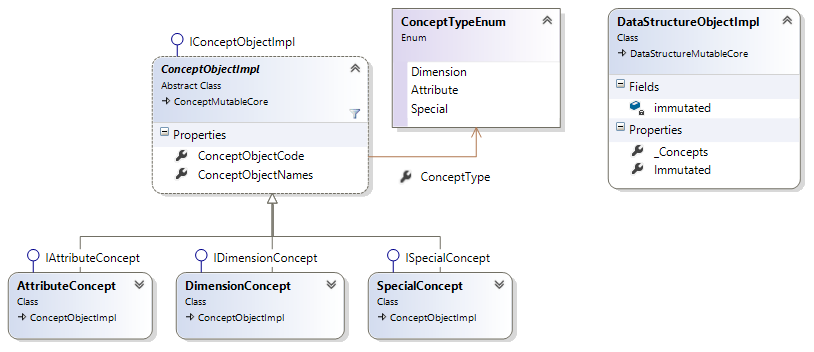
Builders

In namespace FlyController.Builder there are all the objects that transform metadata, that were parsed by FlyEngine, in SDMX objects that can be processed in the object ISDMXObject



Metadata Objects

In specific cases, it was necessary to implement objects deriving from CommonApi by extending these objects with properties necessary to the proper working of the operation



* This image shows how the objects were extended, adding to the list of datastructure only ConceptObjectImpl.ConceptMutableCore
* DataStructureMutableCore

In the ConceptMutableCore was added a types differentiation. Each concept type implements its specific class with their own properties.

The implements properties useful to define the attribute type. The attribute type can be FLAG, OBS\_VALUE or integrated methods that derive from DimensionReference when AttributeAttachmentLevelType is a DimensionGroup type .

The DimensionConcept detects the dimension. The dimension can be "dimension of frequency" or "dimension of Time". The DimensionConcept renames the concepts if the concept name deriving from database is different from SDMX standard..

SpecialConcept, are not real concepts, but are created for building Codelists Constrain also different for types of requests:

* CL\_TIME\_PERIOD request of codelist constrain CL\_TIME\_PERIOD with AgengcyID = MA

Return to the start date and the end date of the dataset required

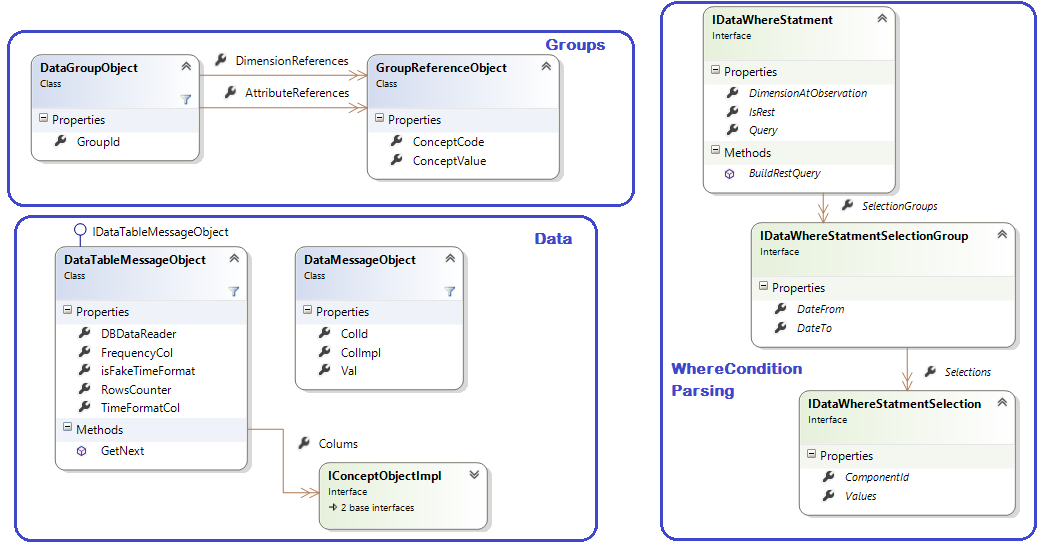
* CL\_COUNT request of codelist constrain CL\_COUNT with AgengcyID = MA

Returns the number of observations for a given dataset

* CL\_CONTRAINED request of codelist constrain

If prompted for a Codelist Constrain not only from dataflow but also from other CodeItem other codelist

Data Objects



The image shows the three groups of basic objects for data recovery.

All these objects are shared between the FlyEngine, and FlyMapping implementations

In Groups there are classes for the recovery of the groups to be included in DataMessage

In Data there are the classes that are used by FlyEngine to scroll the table of data that derives from the FlyMapping implementations.

In WhereCondition Parsing there are classes that transform the query parsing in the WhereCondition to be passed at StoreProcedure that takes data from the Database

Error Management

In namespace FlyController.Model.Error there are objects that manage exceptions throughout the application. Errors can be of two types: derived from CommonApi and derived from the system.

The chapter "Error Management" of "OnTheFly UserGuide" contains informations about the composition of the errors".

To generate an exception and create a custom error message is necessary to write the following statement anywhere in the source code::

throw new SdmxException(this, FlyExceptionObject.FlyExceptionTypeEnum.InternalError, ex);

The signature of SdmxException object is composed by:

1. Object that fired the error
2. FlyExceptionType
3. Optional Exception

The Object that fired the error will compose the Source of the returning message error.

Usually is used the this command, but if it can't be used (for example if is used in a static method) is used Typeof(StaticMethodName).

The FlyExceptionType is the enumerator that establishes the description, the error type and code.

To add a new type of error is necessary to add the new type also in the File management - error descriptions that's in the file ConfigurationXml\ErrorDescription.xml.

More informations about the file editing are contained in the manual "OnTheFly UserGuide".

The Optional Exception is an optional field and its description composes the MessageDetail.

An object that generates an SdmxException can overwrite an SdmxException previously generated by another object of lower level.

To Solve the the problem of overwriting errors is necessary use this statement:

try

{

}

catch (SdmxException) { throw; }

catch (Exception ex)

{

throw new SdmxException(this, FlyExceptionObject.FlyExceptionTypeEnum.CreateImmutable, ex);

}

### FlyLog

FlyLog is a project that concerns only the log writing of whole application.. The Flylog reference is in all the other projects.  
To writing a Log is necessary write:

OnTheFlyLog.FlyLog.WriteLog(this, FlyLog.LogTypeEnum.Error, "text log”);

The signature of static method named FlyLog.WriteLog is composed by:

1. Object that fired the error
2. LogType
3. Text of the log
4. Optional parameter for the completion of StringBuilder

The Object that fired the error composes the Source of the returning error message.

Usually is used the this command, but if it can't be used (for example if is used in a static method) is used Typeof(StaticMethodName).

The LogType is an enumerator composed of:

* Error Only Error Log
* Warning = Important Log and Error,
* All = Debug Mode Log Level All Possible Log

SdmxException don’t need to write a log because the initialization of Exception writes a log of the Error type.

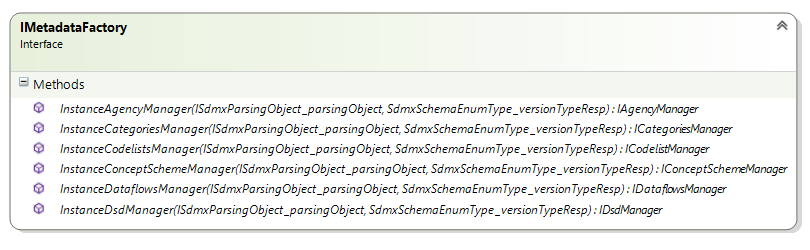
The text of the log can be written also in the string.Format style by inserting all the additional optional parameters.

## Model

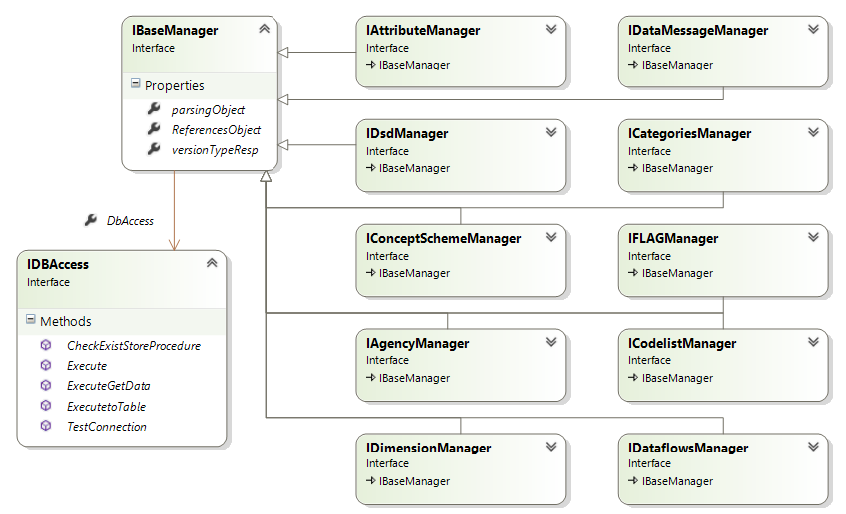
The level Model is composed by the project FlyMapping that is only a draft Interfaces that are made available to all implementation projects.

### FLYMapping

The project FlyMapping is a filter between the FlyEngine and implementations.  
It provides all the interfaces for retrieving metadata from any source.  
The first step is implement the IMetadataFactory class



This image shows that all interfaces Metadata are recalled.  
  
These functions will be passed parameters, ISdmxParsingObject and SdmxSchemaEnumType explained in chapter View-> FlyEngine-> Metadata



All the interfaces are derived from a base interface IBaseManager.

IBaseManager contains in addition to ISdmxParsingObject and SdmxSchemaEnumType even ReferencesObject, in this object are stored all artifacts already taken to avoid recall several times.

For each interface is necessary to implement two methods:

The first method recalls the artifact in question,

*eg. For IConceptSchemeManager there is a method GetConceptScheme*

*This will recall all ConceptScheme or, if there is a MaintenableId in ISdmxParsingObject object, only the one with that code.*

The second method will recall the artifact as a reference, so depending on other artifacts found in the subject ReferencesObject:

eg. *For IConceptSchemeManager there is a method GetConceptSchemeReferences*

*If in ReferencesObject there is a DSD in ReferencesObject,It won’t take all ConceptScheme but only those belonging to that DSD.*

### FlyDotStat\_Implementation

The implementation FlyDotStat is called in version 1.0 of the application.

This version uses a single database .STAT to retrieve data and metadata.

here is shown below how the metadata and data are taken and the implementations that have been used

* **AgencyScheme**

The information about AgencyScheme and OrganisationScheme are taken from the Configuration file. More informations about the application configuration are containedinto the manual "OnTheFly\_UserGuide".

To write this artifact was used the interface named: IAgencyManager .

* **Category and Categorisation**

This metadata can be retrieved by taking information from another database that will have a table named "ContentTree". This table contains information about the themes and the categories to use. If there are more than one themes, is possible to set the name of the theme to use in the parameter CategoryName that is in CategorySetting.

To write this artifact was used the interface named: ICategoriesManager.

* **Dataflows**

The dataflow is retrieved through the store procedures GetDatasets.  
These will be taken as the key to building all other artifacts.

To write this artifact was used the interface named: IDataflowManager.

* **DataStructures**

Te DSD are created on the fly and so each dataset has only one DSD. The code is created with the code of the dataflow. The configuration parameter DsdFormat description will be used for the description of the Dataflow.  
For each DSD there is always only one ConceptScheme and its concepts are taken in the way described below

To write this artifact was used the interface named: IDsdManager.

* **ConceptScheme**

Also the ConceptScheme are created on the fly, one for each dsd. The code is composed by the code of Dataflow with the configuration parameter ConceptSchemeFormat description that will be used for the description of DSD

The concepts will be writed:

For dimensions using StoreProcedure: GetDimensions

For attributes will be read from a file that is in ConfigurationXml\ AttributeConcepts.xml

Will be added another attribute FLAG if you set the configuration parameter: ConceptObservationFlag

Will be added automatically the OBS\_VALUE primary measures

To write this artifact was used the interfaces named: IDimensionManager, IAttributeManager, IFLAGManager and IConceptSchemeManager.

* **Codelist**

For the codelist concepts of type dimension, is possible to write constrianed codelist (related Dataflow) with StoredProcedure GetDimensionCodelistConstrain, or unconstrained codelist going to call the same StoredProcedure for all Dataflow found or send the system in error by setting the configuration parameter CodelistWhitoutConstrain to false.

The system will take the Attribute concept codelist from the file located in ConfigurationXml\ AttributeConcepts.xml

If the frequency dimension is not found, it is added in any case to the DSD, because the frequency dimension is mandatory, and its codelist is taken by the file ConfigurationXml\FrequencyCodelist.xml.

To write this artifact was used the interface named: ICodelistManager.

* **Special Codelist**

Il progetto OnTheFly come per il web service standard RI elabora anche codelist speciali che si dividono in 3 tipologie

* *CL\_TIME\_PERIOD*: restituisce la data iniziale e quella finale del Dataset
* *CL\_COUNT*: restituisce il numero di osservazioni presenti per il Dataset
* *ConstrainMember:* è una richiesta di codelist constrainata non solo a al Dataflow ma anche ad altri CodeItem di altre codelist

Queste sono le uniche informazioni che non venono recuperate attraverso StoreProcedure ma vanno direttamente ad interrogare il database

Queste sono implementate solo per le chiamate specifiche in Soap per Sdmx v2.0

### FlyDotStatExtra\_Implementation

In questa versione viene preso in considerazione un solo Database che è .Stat compatibile per il recupero dei metadati e dei dati ma sono state costruite delle StoreProcedure appositamente per il recupero di queste informazioni.

Questo database sarà chiamato DDB

* **AgencyScheme**

Verrà rigirata l’esecuzione alle classi di FlyDotStat\_Implementation

* **Category and Categorisation**

Verrà rigirata l’esecuzione alle classi di FlyDotStat\_Implementation

* **Dataflows**

Verrà rigirata l’esecuzione alle classi di FlyDotStat\_Implementation

* **DataStructures**

Le DSD vengono create on the fly, per ogni dataset ci sarà una sola DSD il cui codice sarà creato con il nome del dataflow come specificato nel parametro di configurazione DsdFormat Per la descrizione sarà utilizzata quella del Dataflow. Per ogni DSD ci sarà sempre un solo ConceptScheme I cui concetti sono presi attraverso 2 StoreProcedure: una per le dimensioni e l’altra per gli attributi

To write this artifact was used the interface named: IDsdManager.

* **ConceptScheme**

Also the ConceptScheme are created on the fly, one for each dsd. The code is composed by the code of Dataflow with the configuration parameter ConceptSchemeFormat description that will be used for the description of DSD

The concepts will be writed:

For dimensions using StoreProcedure: GetDimensions

For attributes or using StoreProcedure:GetAttributes

Will be added another attribute FLAG if you set the configuration parameter: ConceptObservationFlag

Will be added automatically the OBS\_VALUE primary measures

To write this artifact was used the interfaces named: IDimensionManager, IAttributeManager, IFLAGManager and IConceptSchemeManager.

* **Codelist**

Per le codelist il giro è uguale a quello dell’implementazione FlyDotStat\_implementation tranne che per il fatto che in questa versione abbiamo a disposizione tutte le diverse StoreProcedure che nel DB .Sata mancano:

* GetDimensionCodelistNOConstrain

Per le codelist delle dimensioni NON constrainate al dataflow

* GetDimensionCodelistConstrain

Per le codelist delle dimensioni Constrainate al dataflow

* GetAttributeCodelistNOConstrain

Per le codelist degli attributi NON constrainate al dataflow

* GetAttributeCodelistConstrain

Per le codelist degli attributi Constrainate al dataflow

To write this artifact was used the interface named: ICodelistManager.

* **Special Codelist**

Verrà rigirata l’esecuzione alle classi di FlyDotStat\_Implementation

### FlySDDSLoader\_Implementation

The implementation FlySddsLoader is called in version 2.0.

In this version is used database for data, and another for Metadata with the exception that reported later. To upgrade to OnTheFly 2.0 version is necessary to enter the connectionstring in the second database configuration parameter MsConnectionString

Per quanto riguarda il database Mastore

* **AgencyScheme**

It will be called the methods used for the FlyDotStat implementation

* **Category and Categorisation**

It will be called the methods used for the FlyDotStat implementation

* **Dataflows**

Questo artefatto verrà preso con delle query dirette al database Mastore

To write this artifact was used the interface named: IDataflowManager.

* **DataStructures**

The dsd are recovered effettuando delle interrogazioni dirette nel DB Mastore

* **ConceptScheme**

The ConceptScheme are recovered effettuando delle interrogazioni dirette nel DB Mastore

* **Codelist**

Le Codelist non Constrainate verranno prese effettuando delle interrogazioni dirette nel DB Mastore. The Constrain Codelist will be retrieved in the same way as di FlyDotStat\_Implementation

* **Special Codelist**

This metadata will be retrieved in the same way as the di FlyDotStat\_Implementation

### FlyRedirectToRI\_Implementation

Se viene utilizzata l’implementazione FlySDDSLoader\_implementation, questa implementazione si occupa essenzialmente di capire se è arrivata una richiesta dati o metadati constrainati, questi possono risiedere sul DDB settato nei settings o se in un altro Database. In quest’ultimo caso la richiesta viene rigirata ad un WebService RI Standard i cui indirizzi ed endpoint sono configurati

# Test Client

The project TestOnTheFlyClient was created to:

* Testing the functionality of OnTheFly WebServices
* Making data Queries
* Testing the accuracy of the query

To not have influences of conduct by SDMX CommonApi was chosen not to use them in this project. This project presented in the main file "TestFlyQueryCreation" as a creator and interpreter of XML files.

The other project files are usercontrol used to test all features of the services

* Sdmx20
* Sdmx21
* Rest
* RDF
* DSPL
* JSON

The main interface has fields that help to interrogate WebServices OnTheFly even without knowing the input query that are passed. It uses the template xml listed with field replaceable.

Top right of the main screen, there is a button that opens the window to test the query. This have text fields where the user can enter the query to be passed to all the EntryPoint of OnTheFly WebServices also to test the accuracy of the query itself and the answer.

More informations about the behavior use are referred in the documentation file "OnTheFlyClient.docx"

### FlyCallWS

FlyCallWS is a dll that is separated from TestOnTheFlyClient. This dll handles calls to the WebService and manages streaming answers to avoid to generate an OutOfMemory Exception into the application. The FlyCallWS is also involved in receiving the response stream.

The returning reply is saved in complete mode (including headers and body envelope for calls soap) in a file on the filesystem. If the reply exceeds the total numbers of bytes configured, it will truncated. If the file is in xml format, it will be closed and parsed.

# PUBLISH

In OnTheFly Solution is possible to find the project FlySetup

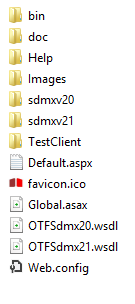
This is a Console Application that will automatically build

* OnTheFly project that builds all other projects attached to the functioning of the web part and WCF web services
* TestOnTheFlyClient that builds also FlyCallWs dll

Windows Console will appear where you can see the development of the Build project in progress and the outcome thereof.

This processing is not limited to build projects but also moves all of its files in the correct directory so it’s simple to copy and paste it into a directory under IIS.

It will create a folder "Release" at the root level of the entire solution. Once finished processing is possible to control the outcome, checking the files in this folder.



The image shows the root level and is possible to see the necessary files to start the service and web page

* Default.aspx
* OTFSdmx20.wsdl
* OTFSdmx20.wsdl

These are the files for the start and for the configuration of the site to be imported in IIS

* Global.asax
* Web.config

Here are shown below all the folders

* Bin: it contains all the compiled dll, the SDMX CommonAPI and configuration files of the project
* Doc: it contains all the documentation files
* Help: it contains the file Documentation.chm
* Sdmx20 and Sdmx21: theycontain all files XML and XSD, for the validation of query input services
* TestClient: contains the folder containing the realease of the project TestOnTheFlyClient, and the same in zip format under the name of "xcopy".