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:** |  | | |  |  |
| **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**

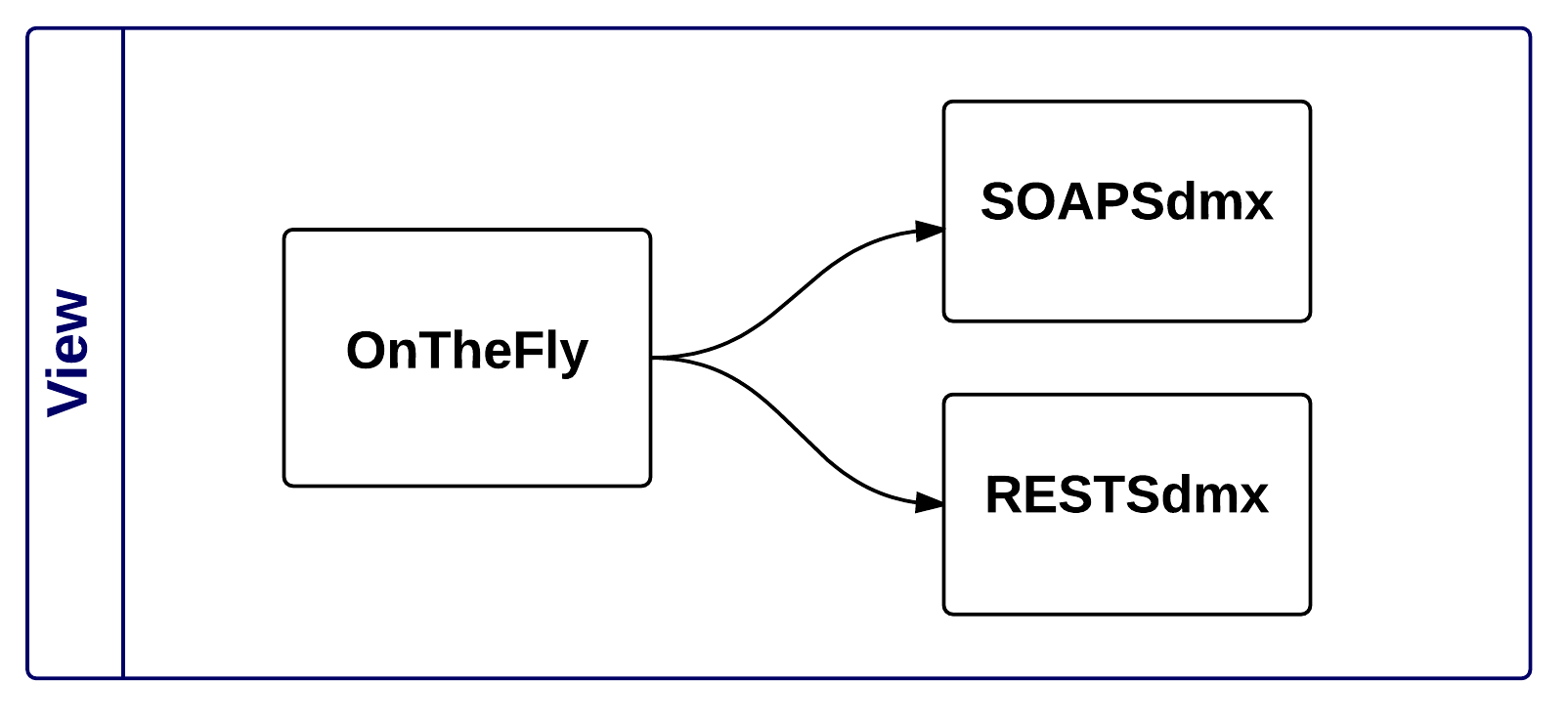
The project OnTheFly WebServices was created to execute queries SDMX to different types of databases.  
Is no longer necessary to have the database structured according to a specific model but simply the presence of some store procedures to perform all queries and create "On the Fly" all required structures.

## Architecture

The project OnTheFly respects the architecture of three-tier model

Model-View-Control + (Implementations)

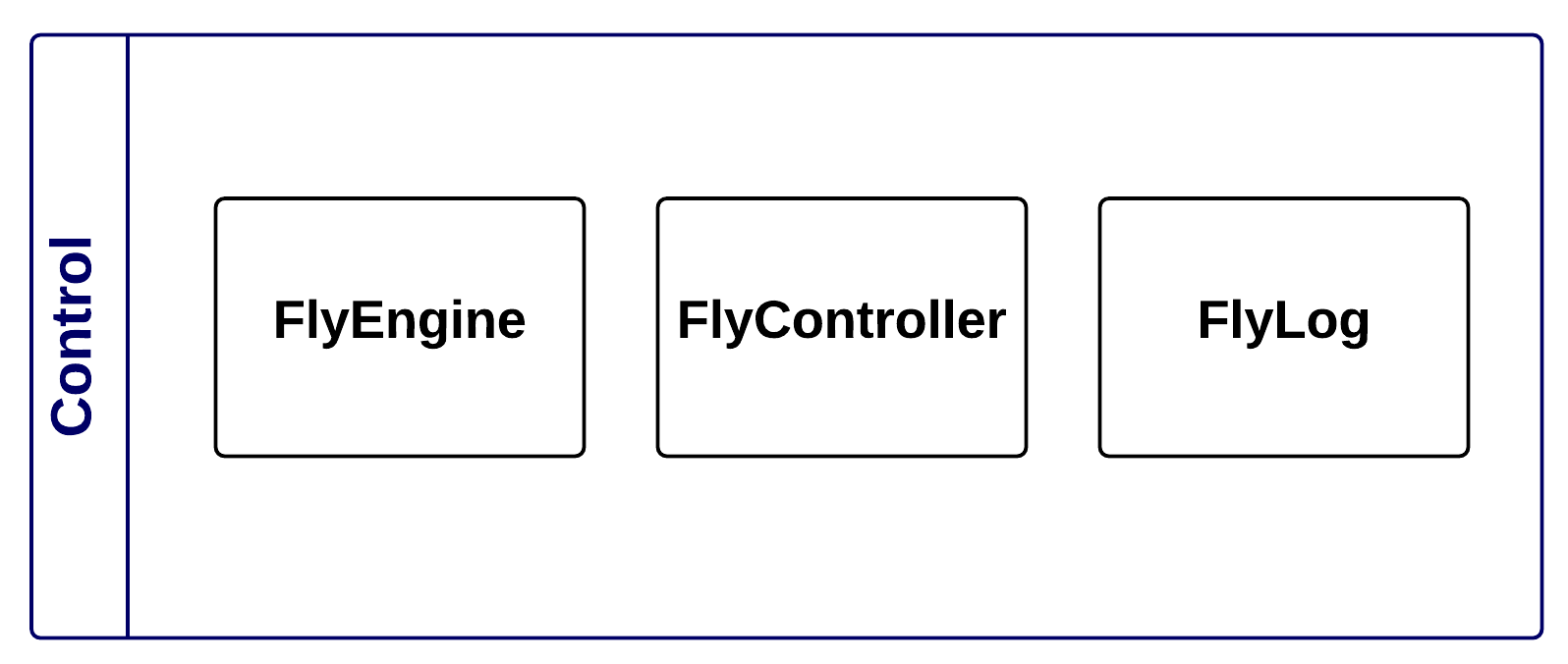
### View Level



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

* OnTheFly
* RestSdmx
* SoapSdmx

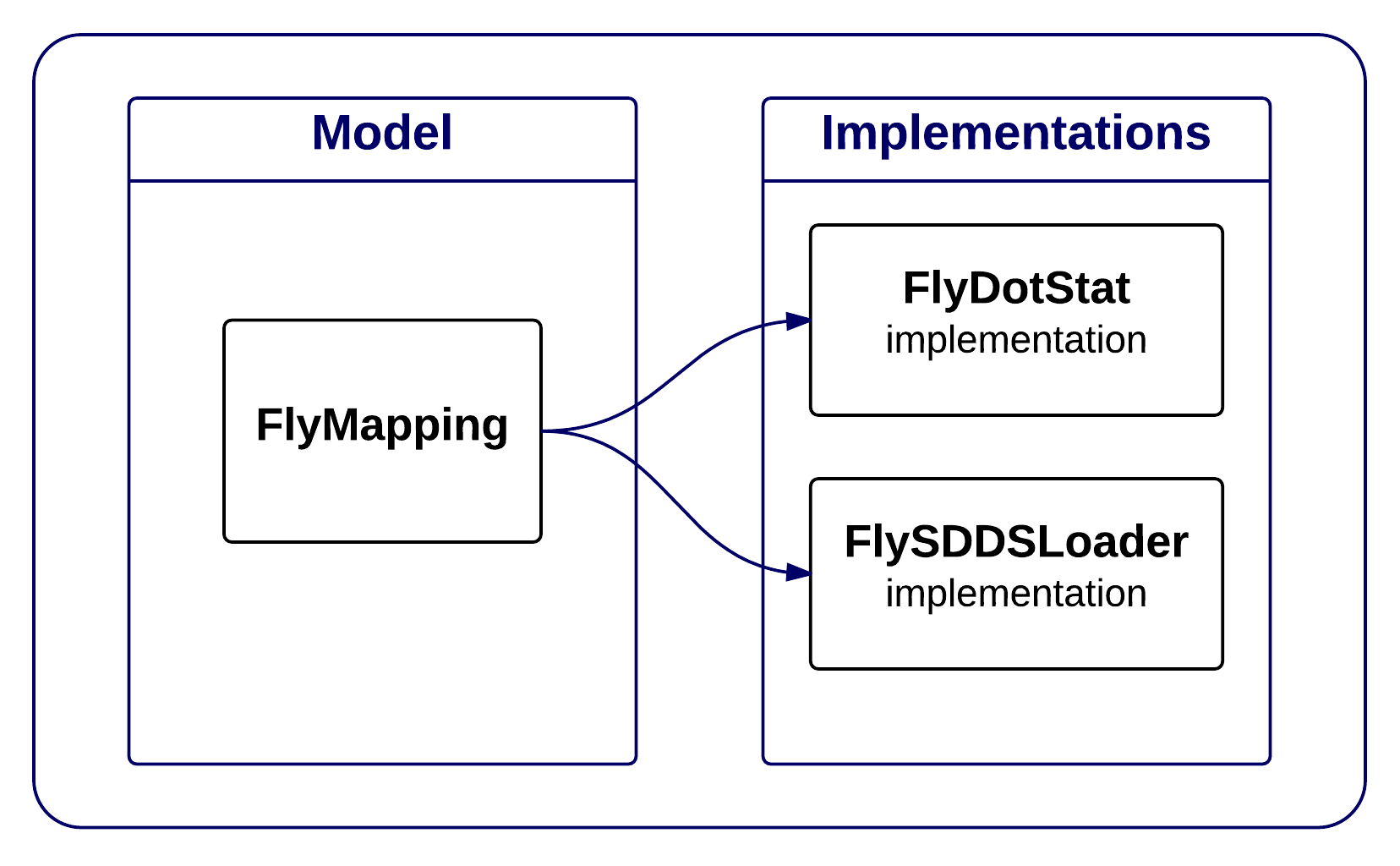
### Control Level



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

* FlyEngine
* FlyController
* FlyLog

### Model Level



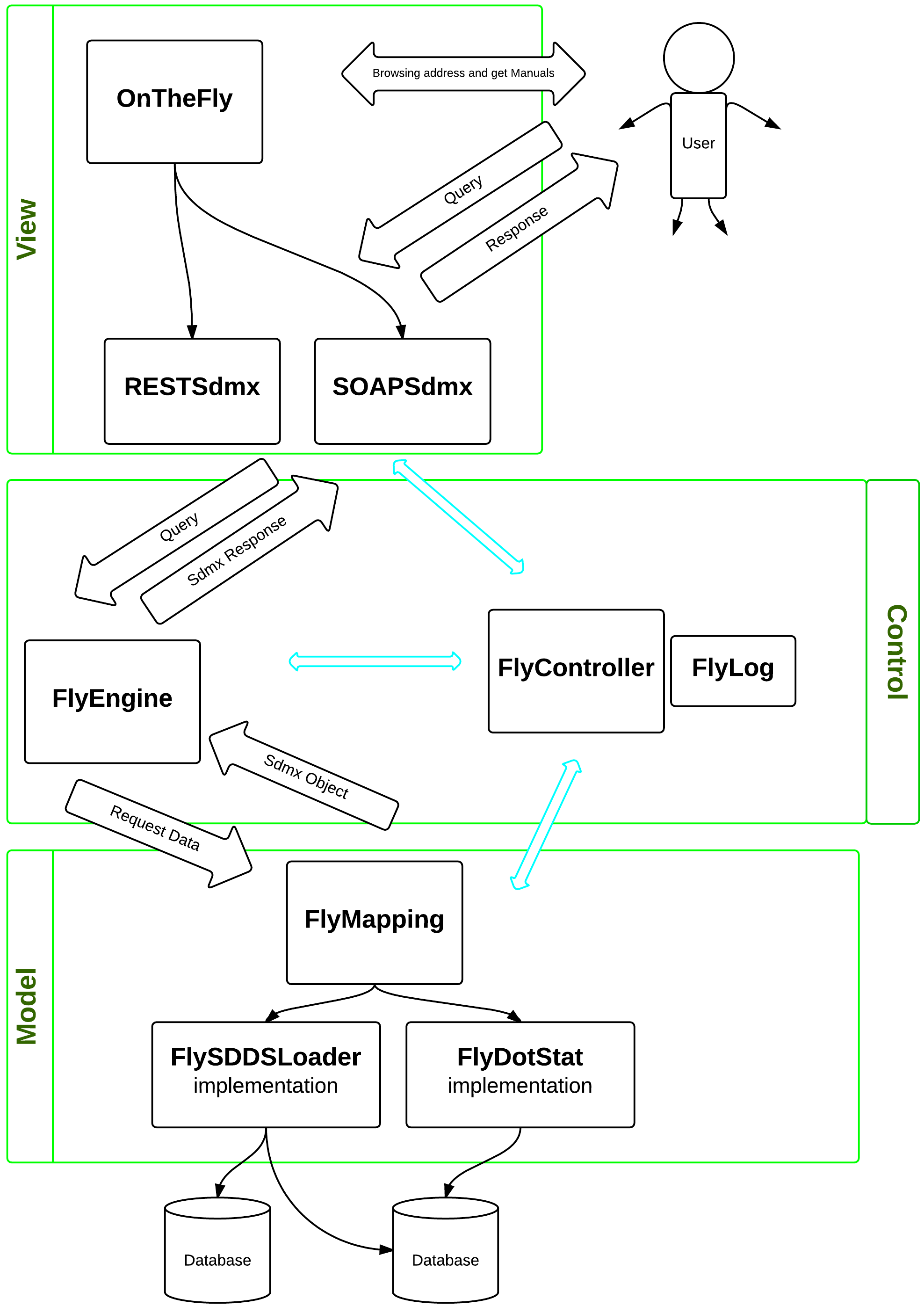
**The Model level**, is the project commissioned to direct requests of the control level projects at the level Implementations level and provide all the necessary interfaces to the creation of new implementations

* FlyMapping

**The Implementations level,** is represented by all the projects used to make queries to the database by calling the correct StoreProcedure, each Implementations project have basic interfaces exposed by the Model level

* FlyDotStat\_implementation
* FlySDDSLoader\_implementation

## Structural cycle



In this diagram you can see how the system performs the steps between levels:  
  
The end user can see the homepage of the service where you can see the entrypoint of the application, and you can download the manuals and test client.  
  
The project OnTheFly, creates two WCF webServices: SOAPSdmx and RESTSdmx receiving queries and call the right entrypoint project FlyEngine.  
  
The FlyEngine will handle parse 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 FlyEngine.  
  
The FlyEngine received data from the Model level, they seemed content and creates the correct answer you will pass to the two WCF webServices SOAPSdmx and RESTSdmx that send the user in the respective protocols.

## Implementations Management

You can create new implementations to the project if you decide you want to retrieve the data in a different way or other sources and then take the data in a different way from what is currently the 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 almost all of its projects.

The CommonApi have been primarily 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 or Rest or Soap :
* ISdmxObject for Metadata
* IDataWriterEngine for Data

# Model

This chapter will explain all the features of all the projects divided by level.  
All projects have been created in .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 is a project that has two functions Asp.Net

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

In the Default.aspx page, you can change the welcome page

1. Make services with Windows Comunication Foundation tecnology

Most important in this project is:

the WebConfig, that goes called from IIS during installation. This file 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

And other settings that are used for application to work correctly

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

WSDL Build

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

This takes advantage of the two files at the project level

* OTFSdmx20.wsdl
* OTFSdmx21.wsdl

In these two files are present all the information to created successfully of services and descriptions of the EntryPoint with their input parameters and returns.  
Also provides information about the scheme, definition and namespace  
  
In the folders

* Sdmvv20
* Sdmxv21

Contains all the xsd file for validation of the query.

Are then created in the method of the Global.asax 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)));

they are also initialized the other 2 projects SoapSdmx and RESTSdmx.

### SOAPSdmx

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

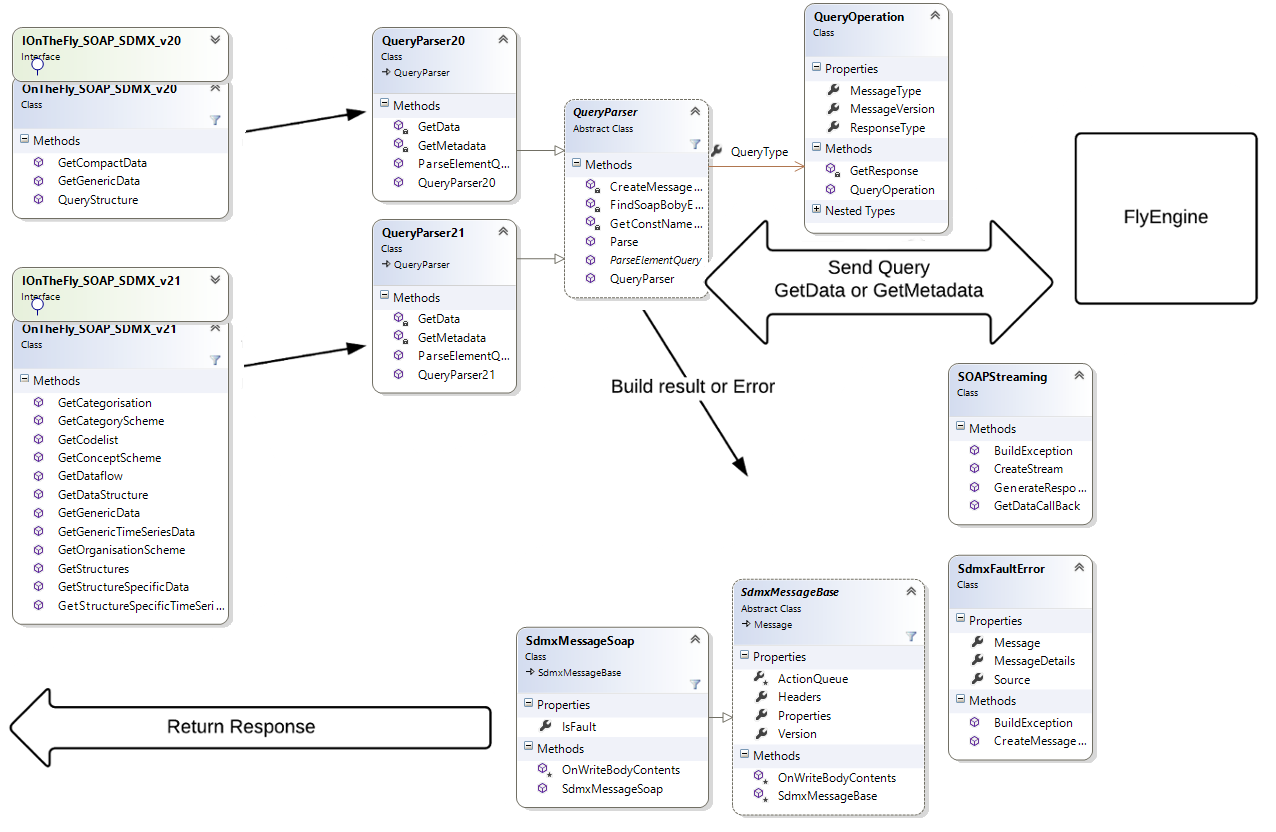
As with all webservice consists of interfaces (ServiceContract) that define the entry point used then the class that extends the same interface that serves as the main class of service (ServiceBehavior)

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

After arriving, the query is processed and are created structures that indicate the version and type, if requested 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 to 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, will be loaded as a WCF Web Service.

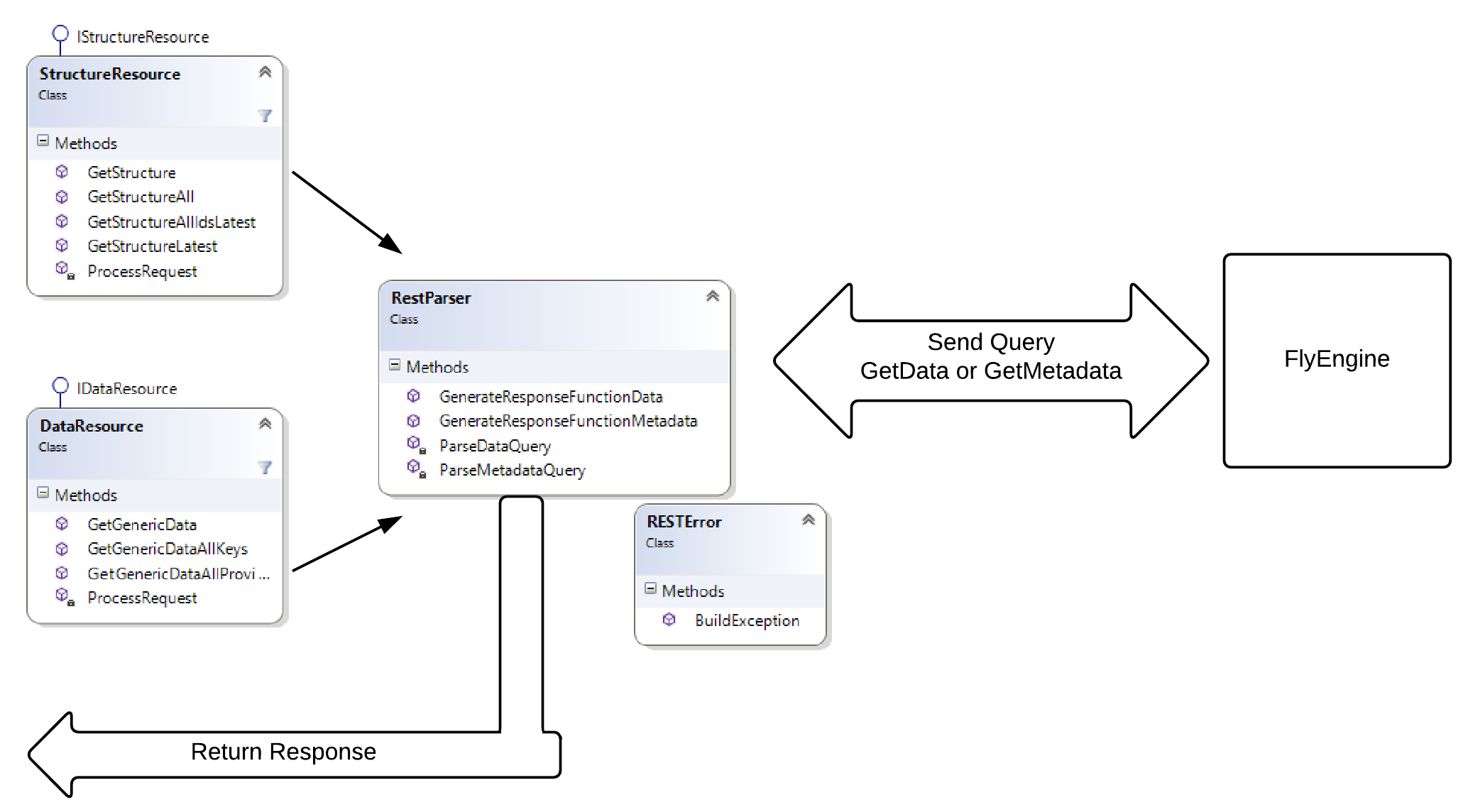
As with all webservice consists of interfaces (ServiceContract) that define the entry point used then the class that extends the same interface that serves as the main class of service (ServiceBehavior)

In this project there are two interfaces

* IStructureResource
* IDataResource

One for the Metadata and the other for Data

After arriving, the query is processed and are created structures that indicate the version and type, if requested 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 to 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 and it is this project that is imported as a reference in the other, moves the files inside the bin of the other projects.  
  
Contains:

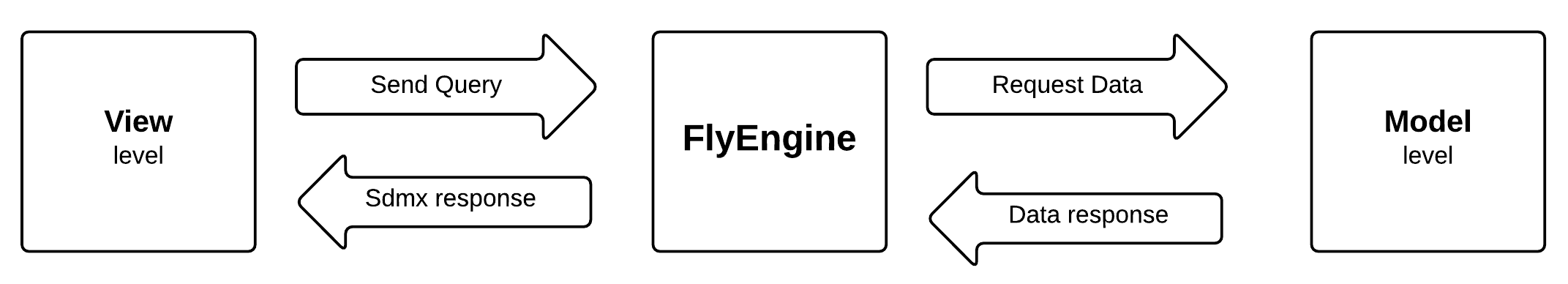
* folder doc, with all the documentation files
* folder Help, with the release of the document Documentation.chm
* folder Libswith all CommonApi libraries used throughout the application
* folders Queries, four folders with all the templates used for the queries in TestClient
* project SandCastle “On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

## Control

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

### FlyEngine

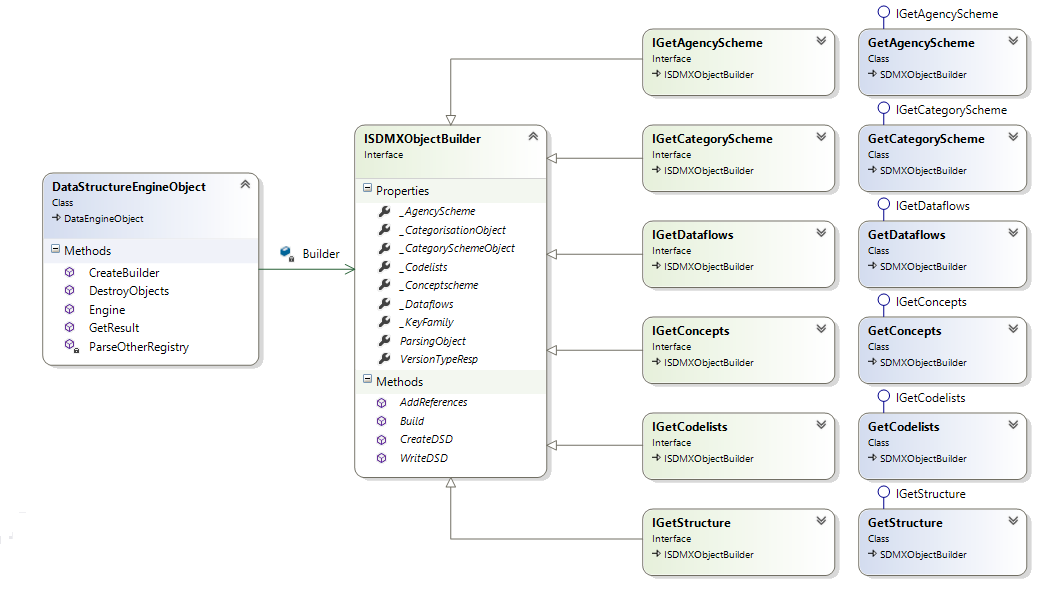
This project is the central engine of the whole application



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, this processes it with CommonApi to check its accuracy and verify that there are no SintaxError. Then build the two key objects for retrieval of all information required

* ParsingObject
* VersionTypeResponse

These objects are then transported throughout the application and contain information on the request of the query without having to parse 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 an artifact different the program generates an exception of type NotImplemented.

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

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

RetreivalReferences

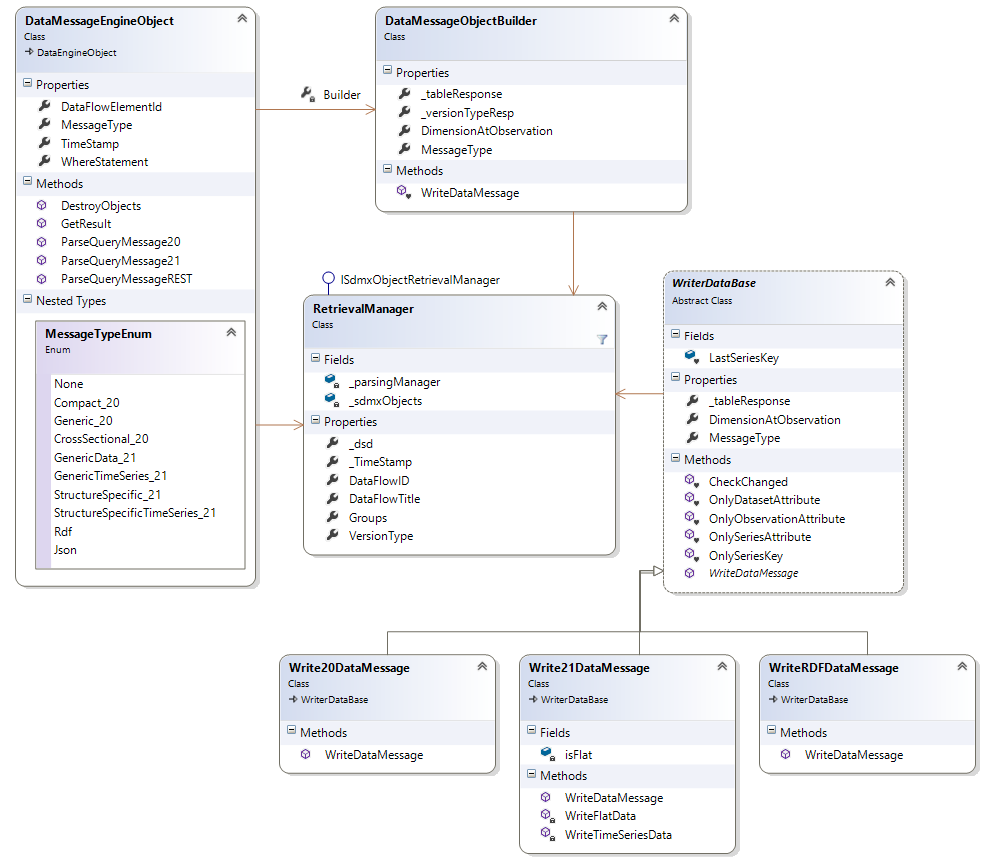
Each *MetadataBuilder* created brings with it the information in ParsingObject, to which reference has been requested for the artifact.

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 arrive into DataMessageEngineObject, this processes it with CommonApi to check its correttenza and verify that there are no SintaxError. Build the key for processing of data into RetreivalManager.

Will be called the methods of model level, to retrieve data from the database. The data will not be taken, and passed all thrown to the writer, but the Model level will return a pointer to the same data that will write into a reply stream. Thus avoid OutOfMemory problems.

Will be instanced the correct class method WriteDataBase who will write data to the communication channel. These Writer use their internal class IDataWriterEngine of CommonApi.

Are currently implemented only TimeSeries requests as

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

RetreivalManager

The RetreivalManager is the main object to parse the query requests data. This comes from the ISdmxObjectRetrievalManager of CommonApi and it implements all the methods.

This object will create a special Dsd just to parse the query and create 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.

For problems with CommonApi, if a request comes from data in FLAT mode, will be added to DSD a concept DimensionAtObservation, order to properly process the data being written.

MappingConfiguration

The MappingConfiguration is the static class responsible for direct requests between different implemetazioni of Model level.  
  
Logically could be also be placed in FlyMapping but this is not possible because all imlpementazioni 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 delgati and pointers to the data tables to create a response stream.  
  
**GroupFactory** is the function responsible for sorting all requests for recovering all the groups members to a set of data.

### FlyController

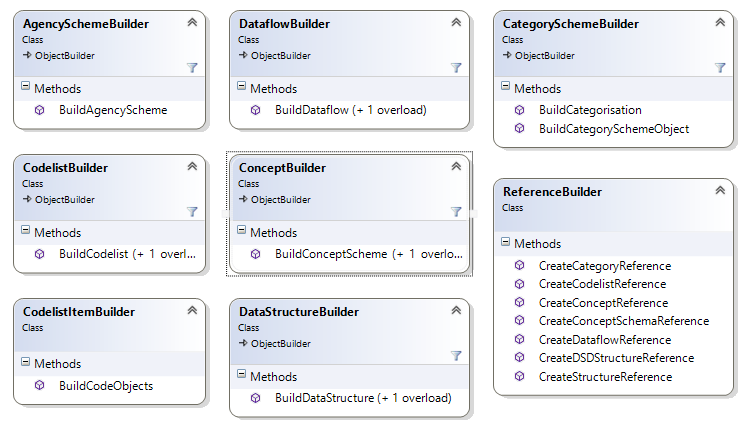
This is the project responsible for providing all the projects of the system classes and methods of model objects and common use.  
  
We will see now some of the most important.

Configuration

The class FlyConfiguration is responsible to read (once) the application configuration file "ServiceConfiguration.xml"  
  
Creates and populates all property configuration and makes it available in a static to all methods of the application thus avoiding every time rereading  
  
For information on what and how to use the configuration parameters refer to the "OnTheFly UserGuide".  
  
This file is also responsible to change the version of the application so if you want to change the version you need to change the implementation of OnTheFlyVersion property

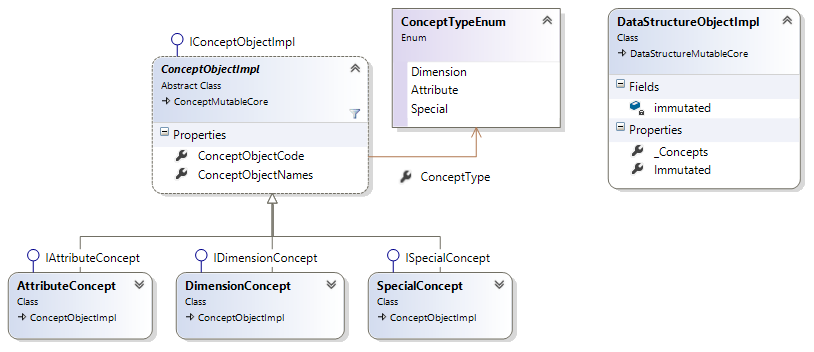
Builders

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



Metadata Objects

In some occasions it was necessary to implement objects that dervano CommonApi by extending them with properties that serve the operation of the application



In the image we can see how they were extended objects

* ConceptMutableCore
* DataStructureMutableCore

Adding to the list of datastructure only ConceptObjectImpl.

For ConceptMutableCore, instead, we added a clear differentiation of types and each type of concept implements its specific class with their own properties.

AttributeConcept implements properties that understand whether it is an attribute of type FLAG or type OBS\_VALUE or integrated methods that derive from DimensionReference when AttributeAttachmentLevelType is type of DimensionGroup .

DimensionConcept identify the dimension and differ in "dimension of frequency" or "dimension of Time" and have an internal logic to rename these concepts when their names taken from the database are not like 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

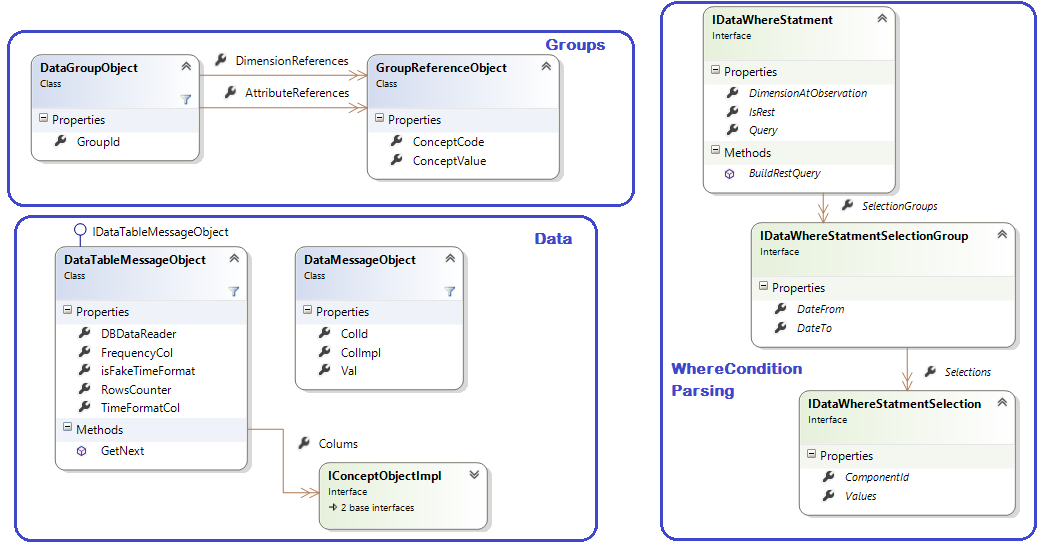
* CL\_COUNT request of codelist constrain CL\_COUNT with AgengcyID = MA

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



In the image you can see 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 comes from the FlyMapping implementations.

In WhereCondition Parsing there are classes that deal with transforming the parsing of the query in the WhereCondition to be passed at StoreProcedure with task of taking 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.

For more information on the composition of the errors read chapter Error Management in the manual "OnTheFly UserGuide".

To generate an exception and create your own error message simply write anywhere in the source code the following statement:

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

The signature of SdmxException object is composed of:

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

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

You will normally use this if this is not possible, for example because you are inside a static method, you use Typeof(StaticMethodName).

The FlyExceptionType is the enumerator establishing the description, the error type and code. If you do not want to use one those present, you need to add a new one, and be careful to also add in File management error descriptions found in ConfigurationXml\ErrorDescription.xml. For more information on how to edit this file read the manual "OnTheFly UserGuide".

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

There might be the problem of overwriting errors: an object of lower level that already unleashes a SdmxException and if we instantiate another SmxException we will overwrite the exception of first. To solve the problem of overwriting errors and to exit the exception actually unleashed by the object that triggered it, you need to either use this statement:

try

{

}

catch (SdmxException) { throw; }

catch (Exception ex)

{

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

}

### FlyLog

FlyLog is a project to write only the log of the whole application. Indeed its reference we can find in all other projects.  
To write a Log simply write:

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

The signature ofstatic method named FlyLog.WriteLog is composed of:

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 will compose the Source of the error message the return.

You will normally use this if this is not possible, for example because you are inside a static method, you use 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

For SdmxException do not need to write a log, because the initialization of Exception, will write a log of the Error type.

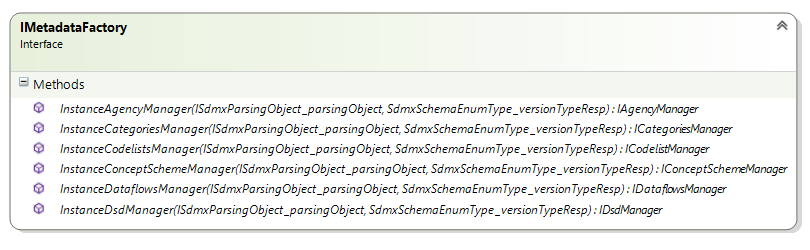
The text of the log that can be written also in style string.Format inserting below all additional optional parameters.

## Model

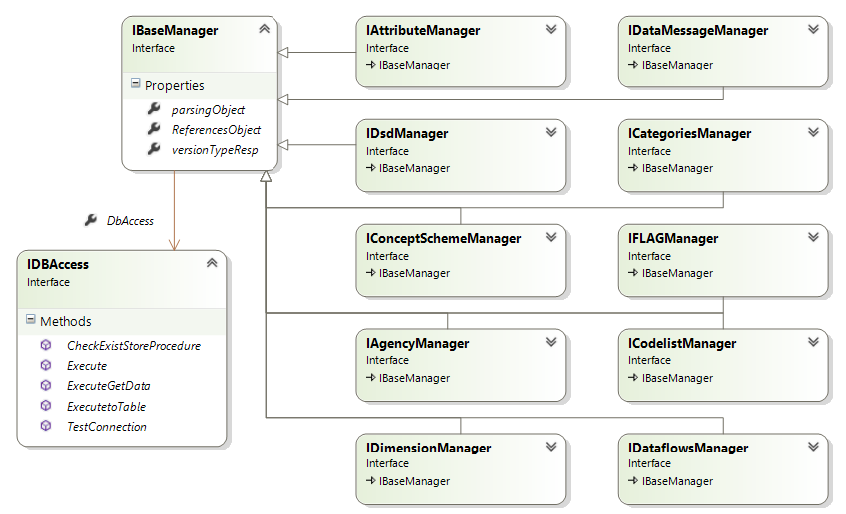
The level Model is composed of the project FlyMapping which is only a draft Interfaces that are made available to all implementation projects.

### FLYMapping

The project FlyMapping, acts as a filter between the FlyEngine and implementations.  
It provides all the interfaces for retrieving metadata from any source.  
The first thing to do is implement the IMetadataFactory class



As you can see will be recalled all interfaces Metadata.  
  
These functions will be passed parameters, ISdmxParsingObject and SdmxSchemaEnumType explained in chapter View-> FlyEngine-> Metadata



As we can see 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 you need to implement two methods:

The first recalls that 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.*

Second the 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, not will take all ConceptScheme but only those belonging to that dsd.*

### FlyDotStat\_Implementation

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

In this version uses a single database is to retrieve data and metadata. Now let's see in detail how the metadata and data are taken e queli implementazioni sono state usate

* **AgencyScheme**

The information about AgencyScheme and OrganisationScheme, are taken from the Configuration file. For more information on how you configure the application to refer the manual "OnTheFly\_UserGuide".

To derive this artifact was used interface named: IAgencyManager .

* **Category and Categorisation**

This metadata can be retrieved by taking information from another database that will have a table "ContentTree". This table contains information about the themes and the masts to be used. In the CategoryName in configuration parameter CategorySetting, you can enter the name in the theme to use if they are more than one.

Alternatively you can use a StoreProcedure that return directly the necessary information to the population of these metadata.

To derive this artifact was used 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 derive this artifact was used interface named: IDataflowManager.

* **DataStructures**

The dsd are created on the fly for giving the premise that for each dataset there is one and only one dsd. The code is created with the code of the dataflow and the configuration parameter DsdFormat description will be used for the description of the Dataflow.  
For each Dsd is always only one ConceptScheme whose concepts will be taken as described below

To derive this artifact was used interface named: IDsdManager.

* **ConceptScheme**

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

The concepts will be so derived:

For dimensions using StoreProcedure: GetDimensions

For attributes or using StoreProcedure:GetAttributes or these will be read from a file that is in ConfigurationXml\ AttributeConcepts.xml

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

Will be added automatically the OBS\_VALUE primary measures

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

* **Codelist**

For codelist concepts of type dimension, we can derive codelist or constrain (related Dataflow) with StoreProcedure: GetDimensionCodelistConstrain or not constrain going to call the same StoreProcedure for all Dataflow found or send the system in error by setting the configuration parameter CodelistWhitoutConstrain to false.

For codelist concepts of type attribute, we can derive codelist constrain (related Dataflow) with StoreProcedure: GetAttributeCodelistConstrain and not constrain the StoreProcedure: GetAttributeCodelistNOConstrain oppure il sistema andra a prendere le codelist dal file che si trova in ConfigurationXml\ AttributeConcepts.xml

For codelist concepts of frequency can not find the concept associated with the dataflow but since this is required to build the dsd, in this case we're going to take the codelist in the file that is located in ConfigurationXml\FrequencyCodelist.xml. In the case the concept is the codelist will be taken as any other Dimension

To derive this artifact was used interface named: ICodelistManager.

### 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 I report later. To upgrade to OnTheFly 2.0 version you must enter the connectionstring in the second database configuration parameter MsConnectionString

* **AgencyScheme**

Will be called the methods used for the FlyDotStat implementation

* **Category and Categorisation**

Will be called the methods used for the FlyDotStat implementation

* **Dataflows**

The dataflow is retrieved through StoreProcedure: GetDataflows

To derive this artifact was used interface named: IDataflowManager.

* **DataStructures**

The dsd are recovered through StoreProcedure: MSGetDSD

To derive this artifact was used interface named: IDsdManager.

* **ConceptScheme**

The ConceptScheme are recovered through StoreProcedure: MSGetConceptScheme

To derive this artifact was used interface named: IConceptSchemeManager.

* **Codelist**

The codelist not constrinate are recovered through StoreProcedure:

MSGetCodelist

To derive this artifact was used interface named: ICodelistManager.

For codelist constrainate instead it all use the same methods for FlyDotStat Implementation

# Test Client

The project TestOnTheFlyClient was created to:

* To test the functionality of OnTheFly WebServices
* To make data Query
* To test the correctness 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 (only for the part of the processing of the request, does not the result)

The main interface has fields that help to interrogate WebServices OnTheFly even without knowing the input query that are passed. This 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 you can enter the query to be passed to all the EntryPoint of OnTheFly WebServices also to test the correctness of the query itself and the answer.

To know how to use and behavior refer to the documentation file "OnTheFlyClient.docx"

### FlyCallWS

It's been moved to a separate dll from TestOnTheFlyClient, a dll that handles calls to the WebService and manage streaming answers for not to generate into application an OutOfMemory Exception.

This dll besides creating classes to perform properly calls to WCF services and manage all passwords and network proxy is also involved in receiving the response stream.

The answer you get is saved in complete mode (including headers and body envelope for calls soap) in a file on the filesystem. The answer is truncated if the response exceeds a tot number of bytes configured and the file is closed and parsed in case it is an xml file.

# Pubblicazione

In OnTheFly Solution you can find the project FlySetup

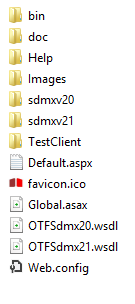
This is a Console Application that will automatically build

* OnTheFly project that build all other projects attached to the functioning of the web part and WCF web services
* TestOnTheFlyClient that build 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 you can simply copy and paste into a directory under IIS.

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



As you can see from the root level, there will be the necessary files to start the service and web page

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

And the files for the start and behavior, for the configuration of the site to be imported in IIS

* Global.asax
* Web.config

Then we find all folder

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