TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Demur Nodia

166797IVSM

SDMX Type Provider for F#

Masters's Thesis

Supervisor: Juhan-Peep Ernits

PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Demur Nodia

166797IVSM

SDMX tüübitekitaja programmeerimiskeelele F#

Magistritöö

Juhendaja: Juhan-Peep Ernits

PhD

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Demur Nodia

06.01.2019

Abstract

The importance of making good quality statistical data easily accessible and usable is ever increasing. Statistical databases are sources of huge amount of data that can yield valuable insights when used appropriately. SDMX standard was devised by large data providing organizations including e.g. European Central Bank to give standardized access to statistical data. A recent version of the standard, version 2.1, includes a restful web service by providing a data schema alongside the actual information [1].

On the other hand the F# programming language has a powerful feature called type provider that enables generation of types at both design time and run time. Several type providers have been implemented in the past. FSharp.Data is a library that is a collection of frequently used type providers. Among these is a World Bank type provider that can be used to query custom World Bank data API.

Within the current work we set out to develop an SDMX type provider that could be used to access all SDMX standard version 2.1 compliant statistical data REST APIs. The thesis document details the design decisions taken in the development process. The result is validated by accessing World Bank and European Central Bank data via different APIs and comparing the results.

Keywords:

F#, TypeProvider, functional programming, SDMX, data services

Annotatsioon

Hea kvaliteediga statistiliste andmete kerge kättesaadavuse ja kasutatavuse olulisus üha kasvab. Statistika andmebaasid sisaldavad suures koguses andmeid, mis võivad korrektsel ja asjakohasel kasutamisel pakkuda suurt lisaväärtust. Euroopa Keskpank koos teiste suurte statistilisi andmeid pakkuvate organisatsioonidega töötas välja standardi SDMX, mis on mõeldud statistiliste andmete ligipääsu ühtlustamiseks. Hiljutine versioon, versiooninumbriga 2.1, sisaldab ka veebiteenuse REST liidest, võimaldades ligipääsu lisaks andmetele ka andmete struktuurile [1].

Teisest küljest toetab programmeerimiskeel F# andmetüüpide tekitamist nii programmi arendamise ajal kui ka programmi töö ajal. Seda omadust nimetatakse tüübitekitajaks (ingl. k. type provider). F# keele jaoks on varasemalt arendatud mitmeid tüübitekitajaid, millest paljud tihemini kasutatavad on koondatud teeki FSharp.Data. Nende hulgas on tüübitekitaja, mis võimaldab pärida andmeid Maailmapanga statistikaandmebaasist.

Käesolevas töös töötame välja ja arendame valmis tüübitekitaja, mis võimaldab pöörduda kõigi SDMX standardi versiooni 2.1 ja REST liidesega statistiliste andmebaaside poole. Töös on detailselt dokumenteeritud disainiotsused ja arendusprotsess. Töö tulemust valideeritakse Maailmapanga ja Euroopa Keskpanga andmete abil pärides neid üle erinevate liideste ning tulemusi võrreldes.

Keywords:

F#, tüübitekitaja, funktsionaalne programmeerimine, SDMX, andmeteenused

Contents

1 Introduction

TypeProvider is a powerful feature of the F# programming language. Type provider makes it possible to provide types, properties, and methods for use in programming language at design time and run time, which enables to benefit from the modern static type system while working with data schemas, that are defined externally from the language, such as databases, web services or any other data providers exposing information about their schema.

There currently already exist multiple type provider implementations which already work well in practice. Some of them are general-purpose ones like CSV [3] or JSON [4] type providers from FSharpData [5]. they are very beneficial because they can be used with a wide variety of data sources, they can work with local files as well as files served from external sources via web. For example, JsonProvider [4] can be used with any kind of JSON [6] format compatible data, the only requirement is to initially provide a schema, based on which the type provider can infer types and provide them on the fly. This means that the user will get suggestions after typing a dot(.) while writing F# code in the editor and if you mistype a column name for example code will not compile.

Because of some web data providers may have more complex URLs, they also accept some parameters which could be acquired from different endpoints, it is becoming more complex to use general-purpose type providers for such cases. For example to consume data from the WorldBank APIs one could use XmlProvider or JsonProvider but this requires some domain specific knowledge. In comparison there is more specific type provider WorldBank Provider [7] which does not require any knowledge from the user about the API endpoints or their schema, the details are hidden behind the type provider implementation and the user gets all the data into F# type system ready for exploring.

Listing 6 contains an example of a using the WorldBank type provider. As you can see it is possible to navigate through data using dots and even more advantage comes from the autocomplete suggestions of types provided via the IDE (or editor).

In 2001 several large data producing organizations such as e.g. European Central Bank came together to devise a standard way to access data. The consortium came up with the Statistical Data and Metadata eXchange (SDMX [1]) standard that has by now matured and provides a way of publishing data on the web using sdmx-rest [8]. The standard specifies a way of exposing schemas, metadata and an approach for querying

statistical multidimensional data through a RESTful API. In short, there is all the necessary information a type provider might need in order to infer types and then allow the user to query the data. The aim of the current thesis is to build an SDMX type provider which would be generalized across all the SDMXdata providers. We will discuss the current status and amount of data currently available adhering to the SDMX standard, also the importance and benefits of having such type provider.

There are multiple areas to study that are in the scope of the current thesis. The two major ones are F# type providers and the SDMX standard itself. For type providers, it is important to know how they work, what difficulties or limitations exist and what kind of data is necessary to create a new type provider.

For the SDMX standard important parts to know are how it exposes data, what are available formats, how required information can be acquired. Also, identify some real-life examples check how well they work and if they follow to the standard.

The following documentation is described in the Guidelines for the use of Web Services Section 7 [9]. More specifically part 4: "SDMX RESTful API" is the one used in creating the SDMX type provider.

The SDMX standards are developed publicly by the SDMX Technical Standards Working Group [10] this is another good source of information about the technical details and examples of how things work and how services should be used correctly.

Currently, SDMX is implemented by multiple organizations. Some of them are listed by the working group [11]. Determining whether the data sources not listed by the working group adhere to the standard is a separate task.

All the tools and technologies used in the scope of the current thesis are open source. The resulting library developed in the context of the current thesis will also be open source, open for contributions, with a steady baseline and potential to make it possible for other people reproduce, maintain or improve.

Many new SDMX standard based data sources have emerged recently. The standard is used by many organizations and a large amount of data is available through sdmx-rest services. Thus the standard is a good fit for type provider approach and creates a higher level of abstraction for F# users to work with such data.

It is possible to achieve even better results to make the data available and easy to use not only for programmers but also for data journalists and readers. The Gamma [12]

project is an ongoing project, it is a great tool for open data storytelling. The approach lets anyone modify the queries, aggregate and the visualize data. Creating pivot data services for The Gamma tool becomes much easier using SDMX type provider.

On the other hand Statistics Estonia [13] adopted a new statistical database software [13] called .Stat (DotStat) in 2018. This gives the possibility to use a new database for implementing pivot data service for The Gamma project. After finishing the work data journalists and storytellers will have a possibility to use real updated data directly from Statistics Estonia and have all the features from The Gamma project. Finally, the whole database is delivered in a simple usable form to F# programmers, data journalists, and readers.

For the validation of the SDMX type provider implementation results it will be used against real SDMX endpoints, for example against the WorldBank, European Central Bank databases and validate if different data flows can be retrieved and used to build charts. In the case of WorldBank, it will be easy to validate the results by providing the same data that is available in the WorldBank type provider documentation [7]. In general the provider should be generic enough to work with any standard-compliant SDMX data sources.

2 Related work and background

2.1 Background

In order to implement an SDMX type provider it is necessary to understand how type providers work and how they are meant to be implemented and how SDMX is meant to be used. Thus the main questions which need to be answered are: what is SDMX, F# type provider and how those two fit together. In addition we will touch upon potential use cases of the contribution, i.e. answer the questions why it is a good idea to spend time on developing the type provider, what benefits and advantages can it have to different segments of users, focussing mainly on developers and data journalists.

In the SDMX standard we mostly focus on sdmx-rest specification details, which will cover

- web services
- URL schemes
- API versions
- supported features
- response formats
- internationalization.

The main reason for giving a priority in this direction is that the information acquired will be essential to be able to fetch the metadata and data in a reliable way. The knowledge will give ideas about type provider design possibilities and functionality. We will be able to specify the minimum requirements to achieve viable functionality and suggest feature ideas for the future roadmap of further development. Besides learning the standard it is important to know about the existing, real-life SDMX implementations. How widely are these used between organizations and what is the approximate amount of data available? Evaluation of compatibility to the standard will give a solid understanding of implementation challenges and possibilities to achieve simplicity of the design. SDMX is openly accessible to everyone and organizations publishing the data to the world. People have already tried to create simplified ways to access such data. The final part includes exploring already

available tools and libraries created for accessing SDMX data through the web APIs. Learning the ways how people use and what their software offers can be useful for reusing the approach and also will be good for comparison with TypeProvider approach to state advantages or disadvantages between them.

The second major part is to research F# and the concept of type providers. We will discuss the question: why is it a good idea to create generic TypeProvider for all SDMX data sources? To answer this some related research materials will be discussed to explain what are the underlying principles of type providers and what is possible to achieve at this point. The literature contains a considerable amount of contributions towards the goal: articles discussing possibilities and benefits of integrating big scale external information-rich services into programming language type system. More specifically focus here is on the development time when the IDE (or editor) triggers compiler to keep the connection to an external data source and generate types on-demand while typing. There are many type provider implementations available which give an important knowledge in practice to quickly glance what is already available and what is possible to achieve. For example open source project FSharp.Data [14] has a number of type providers available for use, it is essential to use existing knowledge and experience while working on new type provider implementation.

An additional section is to show possible future continuations and provide ideas about usage scenarios. One way is to chain an SDMX type provider with The Gamma project, which would act as the end point of the whole toolchain and deliver data from SDMX restful API through F# Type Provider and the Gamma service to data journalists. They can use all those tools on a high level, via a web browser. This enables data journalists to create stories reaching out to actual data without needing much technical knowledge. Readers have a simple interface to explore the data right in the browser.

2.2 Publishing statistical data on the web

Statistical data is provided by a myriad of organizations and good quality data is key to evidence-based governance. Thus, in addition to making sure that the data collected represents the reality in a true way, it is also important to make the data easy to access, process and visualize. In governance, science, business and many other domains statistical data analysis gives answers to important questions. Ways of publishing such data on the

web are described in the research paper [15]. There are two main tools discussed in the research and both of them use the popular SDMX standard to represent multidimensional data using RDF. There are many organizations already using the SDMX approach and have real data publicly available. The number of publishers and data will grow over time. That's why it is important to have a standardized way to access data to enable consumers of the data to build tools and focus on meaningful parts of usages rather than figuring out ways of just fetching correct data and useful format.

We will discuss SDMX standard in more detail in the following section and emphasize the relevant parts in the context of the requirements of type providers.

2.3 SDMX

SDMX, which stands for Statistical Data and Metadata eXchange is an international initiative that aims at standardizing and modernizing the mechanisms and processes for the exchange of statistical data and metadata among international organizations [16].

SDMX is initiated and sponsored by seven institutions: the Bank for International Settlements (BIS), the European Central Bank (ECB), Eurostat (the statistical office of the European Union), the International Monetary Fund (IMF), the Organisation for Economic Co-operation and Development (OECD), the United Nations Statistics Division (UNSD), and the World Bank [16].

There have been several version releases since the approval of the first version (1.0) in 2005. Currently the latest version is 2.1, issued in May 2011. In 2013, SDMX was published by the International Organization for Standardization (ISO) as International Standard (IS) 17369 [17]. Since then several organizations implemented and showed how it works in practice, we will specify some of those organizations in the following part of this section.

The SDMX website provides comprehensive guidelines on the official website, not all the parts are relevant for the scope of this thesis. The main focus will be on version 2.1. All parts which describe web service specifications and message structures is a point of interest. sdmx.org provides special introduction page [18] which contains SDMX 2.1 User Guide [19]. Schema and Samples from SDMX version 2.1. We will mention some of the key points from User Guide and then go more in details in Guidelines for the use of Web Services section 7 [9]. SDMX has documented major version changes in a separate

document [20]. For example, we can read that Support for the RESTful interface has been added to the web services specification in version 2.1.

2.3.1 Background

SDMX comes out of the world of official statistics. "Official statistics" are the data which is collected and disseminated by a set of governmental and international organizations to provide the factual basis for making policy and supporting research.

In general, SDMX supports improved business processes for any statistical organization as well as the harmonization and standardization of statistical metadata. Documentation includes:

- technical standards (including the Information Model)
- statistical guidelines
- an IT architecture and tools

From this huge information, we can focus on relevant parts throughout this thesis. Technical Specifications [21] provides a full introduction to the technical side of the standard and implementation details. This includes multiple stages, In this case, the main points are in Section 7 – Web Services Guidelines [9] which describes ways to publish statistical data and metadata on the web in the machine-readable format. For standardizing metadata and data output special SDMX-ML specification was created. It documents the schema and structure of the message. The XML format is used to describe data structure, reference metadata.

Web applications traditionally expose their functionality through application programming interfaces(APIs), SDMX is not an attempt to invent new way of sharing data, it uses already well-established protocols(SOAP and REST), but in order to make more complex data accessible and usable it provides SDMX-ML, a standardized XML format for exchanging data and structural metadata within the statistical realm.

Besides REST specification SDMX also provides SOAP web services which were supported even in earlier versions. For describing web services in a machine-readable way WSDL (Web Services Description Language) and WADL (Web Application Description Language) is used.

2.3.2 Domains

This chapter will characterize current users of SDMX according to the User Guide [19]. It has become very widely used in the world of official statistics, so much so that it is difficult to form a comprehensive list of users. There is now a grown interest in setting up a global registry so that all SDMX data and metadata sources can be easily found: https://registry.sdmx.org. There are two most common uses:

- Use SDMX as a reporting and collection format within the central banking community and among the statistical agencies in Europe.
- Dissemination of statistical data from websites

Domain usage is quite broad if you glance over an existing database collection you will find very few domains not covered in total. Some of the major ones are listed below:

- Census and Demography
- Education
- Financial and Monetary Indicators
- Economic Indicators
- National Accounts
- Labour
- Food and Agriculture including fisheries
- Epidemiology
- Transport
- Data Quality
- Development Indicators

2.3.3 Provider implementations

There are two major groups of SDMX provider implementations first one is a group of direct implementations of the standard without using any framework [22]. Few from such examples are:

- ECB (European Central Bank) [2].

- EUROSTAT (Statistical Office of the European Union) [23].
- WorldBank [24].

Another group is .Stat Suite [25] use case. .Stat Suite is an SDMX based modular Open Source platform already used by more than 10 organizations [26]. Few from such examples are:

- Statistics Estonia [27].
- UKData Service [28].
- OECD (Organization for Economic Co-operation and Development) [29].
- Australian Bureau of Statistics [30].

2.3.4 Use Case

To see the whole picture lets review Web Data Dissemination System. The whole picture is shown on Figure1. In our case, Data Web Service is the only part we will have direct contact and everything behind this is hidden in as implementation details. Approximate sequence of setting up the system is the following: (1)Based on the Structural metadata database tables and relations are created. (2)load data into database from SDMX dataset files. (3) Data discovery system helps to identify what kind of data is available. (4) Start building a query which will be used to retrieve the data. (5)(6) Data and related metadata are returned in a structured format which is possible to use for further processing or visualizations(7).

2.3.5 Structural Metadata

It is important to understand structural metadata since the users of the API need this knowledge in order to be able to use the service. The information acquired in this part is necessary to build the data query. There are two main keywords in this context: Dataflow and Dimensions. Dataflows is considered as a single set of data collection, can be relatively compared to a database table. For example, Currency exchange rates can be a single Dataflow. Once you have chosen a dataflow next is to get dimension information about it. In most cases, there are more than one dimensions. Dimension information is represented

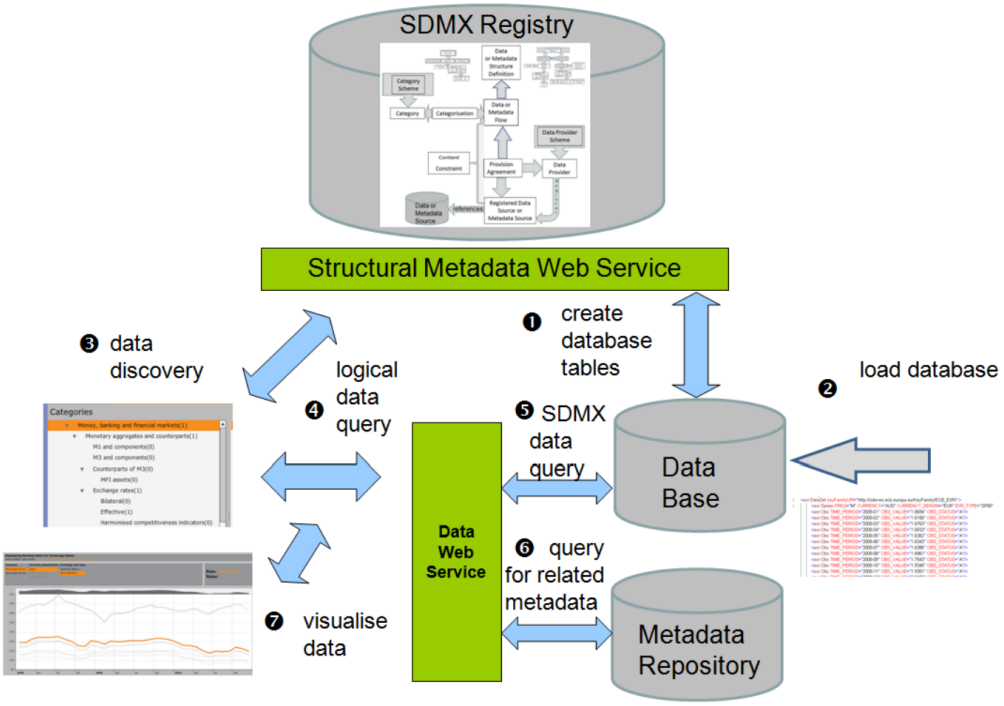

Figure 1. Process flow of an SDMX Web Data Dissemination System

as a DataStructure Component, each Datastrcture has a relation to Concepts which is more detailed and human-readable description.

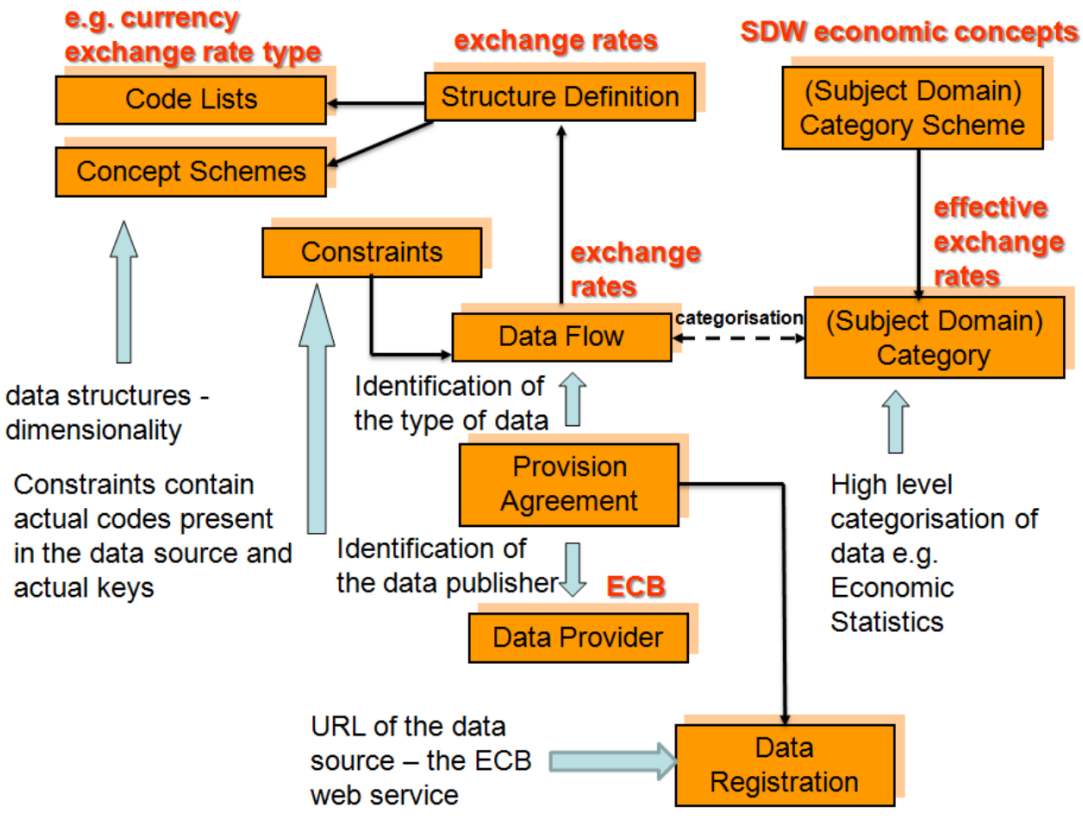

Figure 2. Structural and Provisioning Metadata Used in the Scenario

To query data according to the standard rules using REST API we need to build a query string in which major part is the sequence of dimensions separated by dots(.). We will provide examples metadata and data query is flowing sections.

2.3.6 Tools

Before actually diving into SDMX REST API and start fetching all the data using that, there is one more interesting area to research. The growing availability of software tools which support some of the aspects of SDMX, most of them is open source. This gives a chance to observe and learn how people are using the standard. Based on research about already existing client libraries and tools, I could discover several open source projects which can work with SDMX API endpoints. One of them is SMDX Helper Tool [31] implemented in Java as a cross-platform desktop application. It is very easy to run and is useful for trying different providers, currently, it supports 19 different providers. This tool was also useful to discover some of the providers and test them right away using user interface. UI looks simple example is shown on Figure3. 3 main components: Dataflows, Dimensions, and Dimension Values(Codelists) are displayed as a table and filtered depending on each other. Queries sent in the background is logged in a bottom section which is useful to check what are the endpoints and the parameters o the query.

SDMX Helper is a good tool to use as a desktop client to explore different provider metadata. It is not possible to reuse as a component while building other applications. In contrast to F# TypeProvider data fetched from service is just another structure which can be used as any other kind of structure and sent to a visualizer or other part of the software you are building.

Another open source tool is an object-oriented library sdmx-rest4js [32]. Javascript implementation is reusable and embeddable to other applications. It represents a client library for SDMX REST AP and is simple to install using the node package manager. sdmx-rest4js allows to create metadata and data queries [33], as a result, you get simple javascript object and traditional ways to manipulate. Example usage of how to query data using sdmx-rest4js is shown in Listing 1.

When using sdmx-res4js library similarly to other object-oriented ones, you as a user of the library need to be aware of some keywords which in this particular case are flow and key values. To define this in terms of SDMX flow is a Dataflow and EXR would be

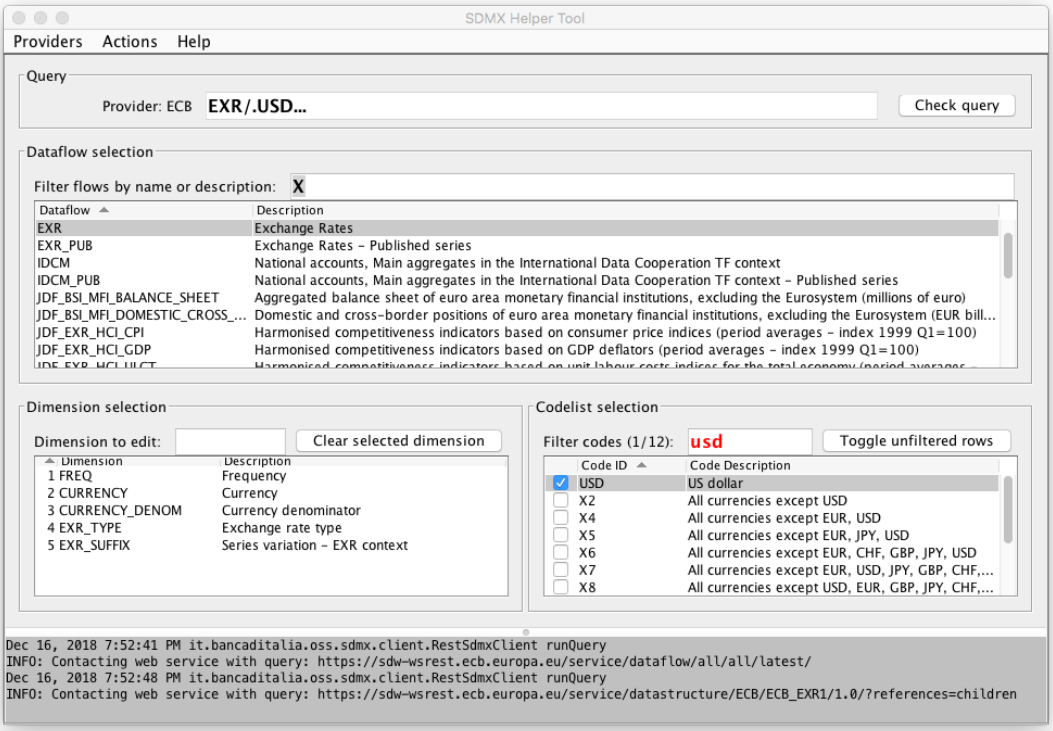

Figure 3. Sdmx Helper tool

```
var sdmxrest = require('sdmx-rest');
var query = {flow: 'EXR', key: 'A.CHF.EUR.SP00.A'};

sdmxrest.request(query, 'ECB')
  .then(function(data) {console.log(data)})
  .catch(function(error){console.log("something went wrong:
  ↪  " + error)});
```

Listing 1. Javascript: sdmx-rest4js usage example

Exchange Rates and key which is a combination of dimension values delimited by a dot(.). In this example, there are 5 dimensions: Frequency, Currency, Currency denominator, Exchange rate type, Series variation - EXR context.

Value of the key 'A.CHF.EUR.SP00.A' represents ordered dimension values separated by a dot, that would be: Annual, Swiss franc, Euro, Spot, Average. This is an example of one specific Dataflow of one specific data provider, obviously, user needs a separate way of looking up those flows and dimension values in order be able to query the data. Here we can state an advantage of TypeProvide in case of SDMX services since type provider can generate types on demand you don't need to search ways of looking up dataflow names

or dimension values. After typing a dot(.) IDE(or editor) will provide suggestions of all available values and filters while typing, this means you can create SDMX query in one go and makes data exploration much easier.

2.3.7 SDMX RESTful API

The current section is strongly referenced to materials from SDMX standards: section 7 [9]. We will discuss most of the key technical details which are necessary to know for retrieving metadata and data from provider endpoints.

In REST terminology any specific information is known as a "Resource". In SDMX such specific resources are called code lists, concept schemes, Dataflows, dimensions etc. It is possible to identify each of this resource by unique global URI. Even though REST supports resource manipulation (POST, PUT, DELETE), SDMX covers only data retrieval part(HTTP GET). It is possible to use HTTP Content Negotiation and the HTTP Accept request header to request one from multiple available SDMX-ML representations.

Responsible party for writing REST API Specifications is SDMX Technical writing group [10]. They announce a period of public reviews before every release which gives every interested party possibility to participate. The very compact and informative page is provided on a single cheat sheet [34].

There are two kind of queries supported first is structural metadata queries which gives information about all the resources like dimensions, code lists, concept schemes, etc. Second is a data query which needs some information from previous queries in order to query desired data.

Both kinds of queries have the same starting point URL, named WsEntryPoint. For structural metadata, queries it is possible to request to resolve references. For example, when querying data structure definitions we can also request related concepts and code lists used in data structure definitions. To do this we can use query parameter $?references = children$.

SDMX RESTful web services offer two modes of operation:

- Data retrieval, where users know the data they want to retrieve (e.g.: daily exchange rates of the Japanese yen against the euro).
- Data discovery, where, using a metadata-driven approach, users need to discover the data exposed by the web service.

Parameter	Type	Description
agencyID	String SDMX common: NCNameIDType	The agency maintaining the artefact to be returned
resourceID	String SDMX common: IDType	The id of the artefact to be returned
version	String SDMX common:VersionType	The version of the artefact to be returned

Table 1. Parameters are used for identifying resources

Keyword	Scope	Description
all	agencyID	Returns artefacts maintained by any maintenance agency
all	resourceID	Returns all resources of the type defined by the resource parameter
all	version	Returns all versions of the resource
latest	version	Returns the latest version in production of the resource

Table 2. Keywords used for resource group of resource identifiers

Structural metadata query looks like this:

URI: $/resource/agencyID/resourceID/version/itemID?queryStringParameters$

Resource in our case is one of the following: Datastructure, Conceptscheme, Codelist, Dataflow. For the full list of resources check cheat sheet [34] It has 3 main parameters for identifying resources details in Table 1. $agencyID$, $resourceID$, $version$. In case we don't know specific values for those parameters it is possible to use some keywords which also default parameters in case you don't provide one at all Table 2. For example, a valid query using default keywords would look like that:

URI: $/resource/all/all/latest/all?queryStringParameters$

This works well when queering dataflows when we don't know what kind of Dataflows provider supports we could run a query like that:

URI: $/dataflow/all/all/latest/$

Sdmx Technical writing group wiki pages [33] has provided as a tutorial guideline for people who are intending to use SDMX RESTful APIs.

Data queries for which resource is $data$ and $flowRef$ is Dataflow reference ID (e.g. WDI). key is the most important part of the data query identifying the data requested. $providerRef$ (eg ECB, WB.) is an optional parameter by default keyword all is used.

$queryStringParameters$ are optional query parameters which allow additional filterings like exact period or specific data update time.

$/resource/flowRef/key/providerRef?queryStringParameters$

For short summary in order to get desired data there are at least three HTTP calls required.

- structural metadata information: Dataflows
- structural metadata information: Datastructures(Dimensions, Concept Schemes, Code Lists)
- data query

In the upcoming sections, we will discuss each of these calls separately based on specific examples. I chose one direct SDMX implementation from WorldBank and will follow process step by step until we get to the final part where we get desired data. Lets set the goal to query Annual Agricultural land (sq. km) for Germany. The entry point for WorldBank SDMX service is:

$https://api.worldbank.org/v2/sdmx/rest/$

We will eliminate this part from URLs since it is the same for all the endpoints and use only resource identifiers which varies, for purpose of keeping lines short.

2.3.8 Dataflows

To query all available dataflows is the very first step, this gives Dataflow identifiers which is required for next the step of getting structural metadata information.

Dataflows: $dataflow/all/all/latest/$

XML Response contains message structure containing header and actual structure we requested, for keeping result simple and compact some not important parts are eliminated from Listing 2.

There are few information we can see in Listing 2. Two dataflows are available

- SDG
- World Development Indicators

There are just several information we need to extract, note that some of the tags i.e: Name are available in different languages.

```xml
<Structure>
  <Header>...</Header>
  <Structures>
    <Dataflows>
      <Dataflow id="SDG" agencyID="UNSD" version="1.0"
       ↪  isFinal="false">
        <Name xml:lang="en" >SDG</Name>
        <Structure>
          <Ref id="SDG" version="0.4" agencyID="UNSD" />
        </Structure>
      </Dataflow>
      <Dataflow id="WDI" agencyID="WB" version="1.0"
       ↪  isFinal="true">
          <Name xml:lang="en">World Development
           ↪  Indicators</Name>
          <Structure>
            <Ref id="WDI" version="1.0" agencyID="WB" />
          </Structure>
      </Dataflow>
    </Dataflows>
  </Structures>
</Structure>
```

Listing 2. XML response: WorldBank dataflows

```fsharp
let xd = XDocument.Parse(dataflowsXml)
let rootElement = xd.Root
let headerElement = rootElement.Element(xmes "Header")
let structuresElements = rootElement.Element(xmes
 ↪  "Structures")
let dataflowsEelements = structuresElements.Element(xstr
 ↪  "Dataflows").Elements(xstr "Dataflow")
let dataflows =
    [ for dataflowsEelement in dataflowsEelements do
        let structureElement =
         ↪  dataflowsEelement.Element(xstr "Structure")
        let refElement = structureElement.Element(xn "Ref")
        let dataflowDisplayName =
         ↪  dataflowsEelement.Element(xcom
         ↪  "Name").Value.Trim()
        let dataflowId = refElement.Attribute(xn
         ↪  "id").Value
        let dataflowAgencyId = refElement.Attribute(xn
         ↪  "agencyID").Value
        let dataflowVersion = refElement.Attribute(xn
         ↪  "version").Value
        yield {
            Id = dataflowId
            Name = dataflowDisplayName
            AgencyID = dataflowAgencyId
            Version = dataflowVersion
        }
    ]
```

Listing 3. F# Parse dataflows

- Name: Structure>Structures>Dataflows>Dataflow>Name
- Id: Structure>Structures>Dataflows>Dataflow>Structure>Ref:id
- AgencyID: Structure>Structures>Dataflows>Dataflow>Structure>Ref:agencyID
- Version: Structure>Structures>Dataflows>Dataflow>Structure>Ref:version

Name Element: Structure>Structures>Dataflows>Dataflow>Name Containing names in different languages. Ref Element: Structure>Structures>Dataflows>Dataflow>Structure>Ref Containing: id, agencyID and version.

Following Code Listing shows WorldBank Example of how to acquire this information.

2.3.9 Datastructures

Next step is to get metadata structure information, there are several ways to fo this. one is to query all of them separately and second is to use data structure references=children parameter to resolve feature which helps to query all relevant information at once in a single request. We are going to use id, agencyId and version information acquired in previous step. Let's say we choose World Development Indicators as a dataflow to proceed according to this the final URL will look like the following:

Datastructures: $datastructure/WB/WDI/1.0/?references = children$

Response example in Listing 4. The example contains only important components and reference ids for keeping it as compact as possible and at the same time maintain the relations logic to be visible. Response contains Header with general information about provider and Structures with $Codelist$, $Concepts$, $Datastructures$. The sequence of extracting information start from looking into $Datastructures$ which contains Dimensions without names or descriptions it has just id and references to ConceptIdentity an Enumeration. Concept scheme provides human-readable names and descriptions for each dimension and relations is linked by Dimension>ConceptIdentity>Ref:id [35]. This means that using that id it is possible to look up Concepts and extract Name and Description information in proffered language. Next we look up for Codelists using reference id Dimension>LocalRepresentation>Enumeration>Ref:id [36]. These are values for frequency dimension, so we can extract human-readable name in proffered language: in English $Annual$ and related id A. Other values can be $Monthly : M$, $Quarterly : Q$ and so on as many as provider outputs in response.

2.3.10 Data

After extracting all the ids and information we can construct query for data. Lets remember what we were looking for: Annual Agricultural land (sq. km) for Germany. Those are combination of all the dimensions provided in datastructure query. We need to take ids of dimensions and sort it by the position attribute provided to each of them. In this case data key would look like $A.AG\_LND\_AGRI\_K2.DEU$ since id for Annual is A, for Agricultural land (sq. km) - $AG\_LND\_AGRI\_K2$ and for Germany - DEU. Final data URL will be following:

Data: $data/WDI/A.AG\_LND\_AGRI\_K2.DEU/$

```xml
<Structure>
  <Header>...</Header>
  <Structures>
    <Codelists>
      <Codelist id="CL_FREQ_WDI">
        <Name xml:lang="en">Frequency code list</Name>
        <Code id="A">
          <Name xml:lang="en">Annual</Name>
        </Code>
      </Codelist>
    </Codelists>
    <Concepts>
      <ConceptScheme id="WDI_CONCEPT">
        <Name xml:lang="en"> Default Scheme</Name>
        <Concept id="FREQ">
          <Name xml:lang="en" >Frequency</Name>
          <Description xml:lang="en">
            Indicates rate of recurrence...
          </Description>
        </Concept>
      </ConceptScheme>
    </Concepts>
    <Datastructures>
      <DataStructure  id="WDI">
        <DataStructureComponents>
          <DimensionList>
            <Dimension id="FREQ" position="1">
              <ConceptIdentity> <Ref id="FREQ" />
              ↪ </ConceptIdentity>
              <LocalRepresentation>
                <Enumeration> <Ref id="CL_FREQ_WDI"/>
                ↪ </Enumeration>
              </LocalRepresentation>
            </Dimension>
          </DimensionList>
        </DataStructureComponents>
      </DataStructure>
    </Datastructures>
  </Structures>
</Structure>
```

Listing 4. XML response: WorldBank datastructures

```xml
<GenericData>
  <Header>...</Header>
  <DataSet action="Append" structureRef="WB_WDI_1_0">
    <Series>
      <SeriesKey>
        <Value id="REF_AREA" value="DEU" />
        <Value id="SERIES" value="AG_LND_AGRI_K2" />
        <Value id="FREQ" value="A" />
      </SeriesKey>
      <Obs>
        <ObsDimension id="TIME_PERIOD" value="1964" />
        <ObsValue value="194580" />
        <Attributes>
          <Value id="UNIT_MULT" value="0" />
        </Attributes>
      </Obs>
    </Series>
  </DataSet>
</GenericData>
```

Listing 5. XML response: WorldBank data

Data response is outputted as a Series of observations containing ObsDimension and ObsValue which we can extract into a sequence of tuples or any other data structure.

2.4 F# TypeProvider

The second major part of the research is related to the TypeProvider. We will talk about this feature in upcoming sections and relate a few pieces of research which leads provides lots of examples and use cases prving the benefits TypeProviders. Feature was introduced in F# 3.0 and it is defined to be an extension to the compiler that uses code generation while type checking. To create both new types and new program code that makes use of these types, with the aim of gaining the benefits of a static type system and the associated tools while working with data sources external to the language ecosystem. Tools such as IDE(or editors) gain access to provided types incrementally, as the type providers are added to a program by a developer.

F# community has a range of type providers implemented which can work with external data sources, like databases, some of the web APIs, File System and etc. FSharp.Data [14] is a Library for Data access and contains several TypeProviders which is possible to use while working with data. Researching this library and checking approaches is one important part of the thesis since SDMX TypeProvider is strongly related to data with external schema and experience of FSharp.Data is valuable to reuse and try to build new TypeProvider on top of it.

2.4.1 Strongly-Typed Language Support for Internet Scale Information Sources

The world is experiencing an enormous growth of information available over the web. Data is published using web APIs and it is already uncommon to see services which do not expose data and provide access through the open web API. On the other hand, there are programming services as components which consume, manipulate and reuse data in larger systems. Nowadays it is rear to discuss a system which does not have at least one external service integration. Despite this growth, there are few strongly typed programming languages which can integrate such kind of external information sources as a language component directly into the type system.

The size and number of information spaces are growing rapidly, with respect to both data and metadata[37].

There are three main techniques used in traditionally statically typed languages while connecting to external services.

- hand-written static libraries,

```
#r "../../../bin/lib/net45/FSharp.Data.dll"
open FSharp.Data

let data = WorldBankData.GetDataContext()

data
  .Countries.``United Kingdom``
  .Indicators.``Gross capital formation (% of GDP)``
|> Seq.maxBy fst
```

Listing 6. WorldBank usage example

- generated static libraries,
- dynamically-typed information representation.

None of those approaches scale with large metadata sizes: e.g. hundreds of thousands of different types, schemas are read eagerly and they are not well connected with exploratory programming with such big scale.

Dynamically-typed bridging mechanisms discard the benefits of strongly-typed programming[37]. Using this type of dynamic representation it is possible to achieve big scales, but it also discards benefits strongly-typed programming.

Functional programming language F# makes this challenge easier to overcome, thanks to the TypeProviders feature. It is a compile-time component that, given optional static parameters identifying an external information space and a way of accessing that information space, provides two things to the host F# compiler/tooling[37]. We can see existing WorldBank Provider example from FSharp.Data in reference section $Example : WorldBank$[37]. Screenshots provided demonstrates how easy it is to explore types and data provided by the service. We can also see the code of usage in Listing 6.

Using this feature F# has a scalable architecture for the direct integration of stable external information spaces as strongly-typed components. Since types are inferred on on-demand compiler can work with extremely large external data sources in a scalable way. SDMX TypeProvider is another example of integrating external services just much larger scopes since every standard complaint data service will be possible to explore the same way, this will include WorldBank and many other SDMX-REST service providers.

Design-time Assistance

There are interesting of the benefits which TypePovider provides and makes the development process more enjoyable and fast, in a sense that all the information you need about the external source is available right in the editor.

- Interactive type checking during development (“red squigglies”)
- Provision of context-sensitive declaration lists (“auto-completion”)
- Type-directed information on gestures such as mouse-hover (“quick info”)
- Type and name-directed help systems (“F1 help”)
- Name-directed, type-directed or type-safe refactorings

This is known as a design-time experience for strongly typed languages. All new typed languages should be in this kind of tooling experience in mind.

Current programming language challenge is to integrate internet-scale information services directly into programming languages. This will increase programmer productivity, performance, application robustness and application maintainability.

2.4.2 How TypeProviders work

There are two phases of running type provider design time and runtime. While design-time type inference is performed. For this necessary schema information is requested. Everything happens in a lazy way depends on usage of the type provider. IDE(or editor) performs the main operations to trigger the process of compilation while writing code. For example when initializing type provider or while typing .(dot) after a specific type.

Runtime phase comes in action when we send the type provider usage code to F# interactive for evaluation. This may re-use some parts which were already done during design time but in addition fetch more data and deliver F# data structures to the user for further usage.

TypeProvider is loaded by the compiler and executed at compile-time. When referencing a DLL into the source code and writing usage code, editor triggers compiler to perform the type compilation. A type provider builds information about types and makes them available to the compiler [38]. This is done in a lazy way so provider does not need to provide types for entire information space. For building types initial information context(schema) is required. There are 3 main possibilities of failure TypeProvider.

- *Type provider failure*. TypeProvider is implemented in F# and it can fail for several reasons, for example if there is no internet to fetch the schema, or access to the protocol is restricted. But if type provides succeeds it will generate valid F# code without compilation errors.

- *Runtime failure*. When the outside information space changes while running the type provider, for example some of the metadata information is removed, this may result in runtime error.

- *Recompilation failure*. When the outside information space changes. For example some enumerations are removed this will fail to recompile the same code, this can be looked as negative aspect but there is a clear benefit that helps to detect the change at earlier stages during recompilation.

Given the general definitions of the failures, we can now look at how those failures would happen based on SDMX type provide an example. Let's assume we have a working SDMX type provider implementation.

- If there is no internet when using type provider for the first time. It is not able to obtain any information about schema so no types will be generated. Since there will be a caching support, it is possible to generate types offline for those ones which has schema already cached.

- If a dataflow or some code list, for example country, or currency name is renamed. type provider will still continue to work(there will be no runtime failure) because for communicating with service providers APIs IDs of the code lists are used and it will still work unless ID's are not also changed. But in case of some code list removal there will be runtime failure since type provide will use already removed or depreciated id while communicating with service providers.

- If dataflow or some code list is renamed or removed, code which used to work before will no longer compile because types will also adapt alongside to the new schema and old type names will remain obsolete.

```python
olympics = pd.read_csv("olympics.csv")

olympics[olympics["Games"] == "Rio (2016)"]
        .groupby("Athlete")
        .agg({"Gold" : sum})
        .sort_values(by = "Gold", ascending = False)
        .head(8)
```

Listing 7. Python, pandas access data

2.4.3 Data exploration through Dot-driven Development

Importance of data and ways to read represent or interact with it is becoming more and more important. One of the goals while working in this direction is to achieve a simple and intuitive way of interacting with data, which will be easy to learn and use for normal people without any special technical background. In this paper design of data exploration language is described and shown advantages to other approaches by examples. The way of using type providers in new language simplifies the complexity of the language and makes it easy to understand and straightforward to write for exploring data.

To see the difference and advantages of this language lets discuss the example query, first using popular python library pandas [39]. The Listing 7 shows a pandas script which finds top 8 athletes by the number of gold medals they won in Rio 2016, and it looks simple and short but requires the user to understand python and should also be aware of the pandas library, how it works.

We can see that [olympics["Games"] == "Rio (2016)"] is used for filtering data and for aggregation we pass "Gold" : sum dictionary, here it is not possible to get the benefit of autocompleting and even using group_by, agg, sort_values, requires knowledge of how to use it, what parameters it expects, you may need to check the documentation for that.

The same query for the same data using the language presented in the paper will look like in Listing 8. Which is the same size as previous but you can see a few advantages.

After typing each dot you get auto-completion which not only suggests the features of the language but can see the data and provide 'Athlete' or 'Gold' as a suggestion, this means that you don't need to know much about the data you are expiring as you build the query. And if you mistype Athlete error hint will correct about the issue, this means that data is actually built in types which helps to eliminate typing issues as well. On the other

```
olympics
  .'filter data'.'Games is'.'Rio (2016)'.then
  .'group data'.'by Athlete'.'sum Gold'.then
  .'sort data'.'by Gold descending'.then
  .'paging'.take(8)
```

Listing 8. The Gamma query

hand in case of pandas you are required to know about the 'Athlete' keyword exactly as it is and the only way to get this information is by looking it up separately using documentation or looking into the data source using different tools.

3 Design and implementation of SDMX TypeProvider

The implementation of the SDMX TypeProvider was difficult because the TypeProvider SDK has had architectural and structural changes between releases. Some of the tutorials or snippets from books are outdated and does not work anymore. This work is a contribution to providing up to date step by step approach of how is it possible to achieve results as we have in SDMX TypeProvider so that important parts of the code snippets and ideas to considered are explained.

3.1 Development Environment

The recommended way to set up the development environment is Windows Operating system and Visual Studio software, reasons for recommending this setup is because of simplified installation and debugging possibilities. The simplest way of setting up debugging TypeProviders is possible using Visual Studio $Debug/Attachtoprocess...$ option. Although F# works cross-platform, it is possible to set up the environment on Linux or MacOS. Installation instructions are provided by fsharp.org [40]. During the work of this thesis, both environments have been used. It is possible to confirm that both approaches work and gives the possibility to use or create something real.

3.2 Implementation details

There are several ways to get started, one is to build everything from scratch which is not recommended because this is not a trivial task and there already exists some basic boilerplate code which simplifies the job a lot. Microsoft page provides a guideline to get starting creating type provider [41]. By the recommendation, another way is to use F# TypeProvider SDK [42]. $ProvidedTypes.fs$ provided by the SDK gives an API for creating TypeProviers, additionally Documentation and samples are provided on type provider creation.

For the example of thesis third way was chosen, to contribute existing opensource library FSharp.Data [14]. FSharp.Data is an open source project run by the community it has already implemented TypeProviders for accessing the data. SDMX TypeProvider would have many similarities to WorldBank TypeProvider, more specifically it could fetch the same data, with two differences: using SDMX endpoints with slightly changed usage and as a benefit, it would be more generic. Which means you could use same SDMX TypeProvider for other SDMX endpoints. There was a suggestion made to the community using public github.com issue with a brief summary and initial design [43]. Reference to the idea suggested is to create an SDMX TypeProvider which can query same data as WorldBank TypeProvider does, with the difference that it will use the SDMX APIs, as a benefit provider should be generic enough to work with other data sources like European Central Bank.

There was positive feedback provided from the community member on the issue, which agreed that SDMX TypeProvider would be a good fit to FSharp.Data. This gives another validation of the idea that contributing the FSharp.Data library could be beneficial. Achieving all the community standards and getting more feedback it will be possible to make a pull request to the main FSharp.Data source.

For a better understanding of how would, SDMX TypeProvider fit to FSharp.Data library we can look at Figure 4. It shows some major components and connection between them. The provider will communicate to different service providers using SDMX REST API which will be configurable using static parameters while initializing a TypeProvider. Data provided by the service providers will be based on SDMX-ML standard.

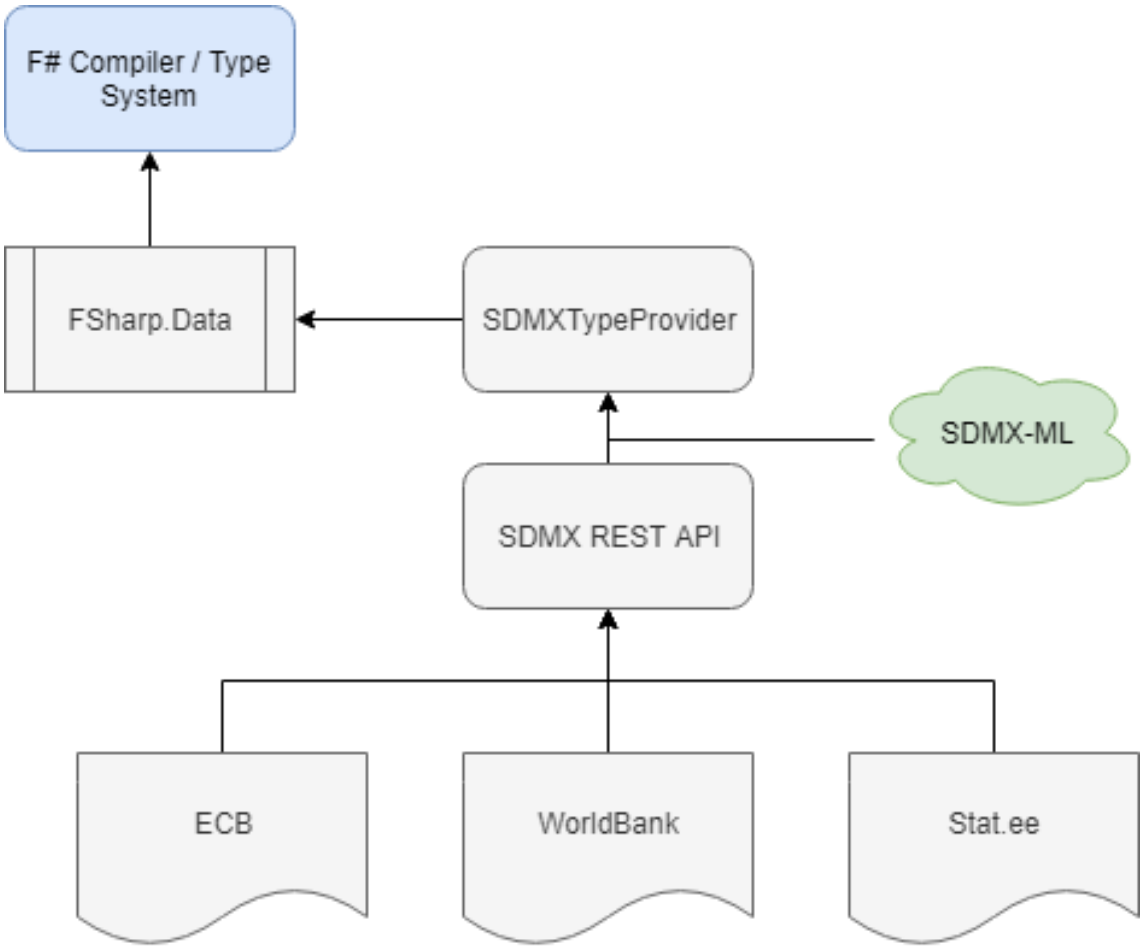

Figure 4. SDMXTypeProvider with FSharp.Data

3.3 Project Structure

There are two modes of running TypeProvider: DesignTime and RunTime. This means that we should be able to understand difference and way these two modes connect each other, in order to be able to configure the project and provide a working solution for a specific scenario. Code for those two modes is split into two files(SdmxRuntime.fs and SdmxProvider.fs) and each of those files compiles separately. Design time is responsible to trigger compiler and generate types on the fly while the user interacts with editor or IDE. For example, if we reference a TypeProvider in our script, design time code will already start running first by checking and validating usage correctness and then generating types depending on whether they are set to be delayed or not. As soon as user will execute the code run time part of the project will execute. Design time requires some of the run time components and code to be available. That's project compilation file order should be strictly set. Run time file should be before the design time file like it is in Listing 9.

Responsibilities are split this way, run time code is responsible for interaction with

```
1 <Compile Include="..\Sdmx\SdmxRuntime.fs" />
2 <Compile Include="..\Sdmx\SdmxProvider.fs" />
```

Listing 9. FSharp.Data.DesignTime.proj snippet

external data sources and extracting schema into F# data structures. It handles caching and HTTP communications. The processing which is required to do in order to prepare schema and data. The design time code is responsible for generating ProvidedTypes, constructors, methods, and properties, setting the XmlDocs to members for better user experience. For doing all this it is required to have schema information available since everything is based on the schema, so every time during design time there is a need for details about schema run time code should be able to provide this information.

3.4 Testing TypeProvider

There are some difficulties while testing TypeProviders while developing. The recommended way is to have two separate instances of IDE or editor. One can be open all the time for implementation of the provider. Another one is just for referencing compiled DLL and testing the usage. The reason for such configuration is because of the issue of locking DLL. When test instance process has the reference process open and at the same time the next compilation of the TypeProvider fails with the following error.

> Error MSB3021 Unable to copy file "..\FSharp.Data.DesignTime\bin\Debug\net45\FSharp.Data.DesignTime.dll" to "..\..\bin\typeproviders\fsharp41\net45\FSharp.Data.DesignTime.dll". The process cannot access the file '..\..\bin\typeproviders\fsharp41\net45\FSharp.Data.DesignTime.dll' because it is being used by another process.

There is no way of avoiding this issue other than killing processes which reference the DLL file. A simple way is to close the test instance of IDE or editor, compile new DLL and then re-open test version again. This makes the development process a bit slow because there is extra manual work. Build time of the FSharp.Data project is roughly 50-130 seconds depending on the development machine computational power. One very handy tip is to set different sound command outputs depending on build status, nice article demonstrating this feature [44].

In order to use compiled TypeProvider, using a test version of IDE or editor we should

```
#r @"../../../../bin/lib/net45/FSharp.Data.dll"
open FSharp.Data
```

Listing 10. Reference FSharp.Data.dll

reference the relative or absolute path to initial lines and then open FSharp.Data namespace Listing 10. This gives access to all available base types in the DLL file.

3.5 Type Structure Design

Designing type structure of the type provider is important for achieving a good level of usability. You can always make a type provider which works but is difficult to use. It is more challenging to achieve a simple type structure so it is intuitive and easy to use. This can be a process of trying different versions of the design in practice and evaluate usage simplicity. This process includes constant testing and validation of each small step. The general guideline for design is simple. Keep type hierarchy as simple as possible and use type provider infer as much of types as possible. At the same time maintain the high scalability. In this way, user gets more out of the type system as protection and easier to use the tool.

While working TypeProvider its good to make decisions about when to use different components. Choose whether members should be static or not and do they need to be delayed or not. Delaying members means that subtypes and members for the particular type will not be generated until the user will type a dot(.). This is important because of a few reasons. First one is for achieving better scalability and user experience while working on large data structures. For example, if you do not make dimension types delayed this will result in the following scenario. As soon as the provider will initialize, it will start downloading all the dimensions for all the data flows. This might work in case of having 1 dataflow and 3 dimensions. But on a larger scale which SDMX offers in practice, the approach will fail. Another benefit of correctly delaying members is independent failures. For example, if some particular branch of the type structure is broken, because of this external data source has an issue or it has been changed. The provider will still work for other types which has a stable schema available. For example, if SDMX provider has an issue for particular dataflow or dimension this only fails relevant type in TypeProvider but

other data flows and dimension types still will working.

To summarize work required to do for implementing SDMX type provider we can see the following bullet points.

- Design type usage
- Implement Run Time
  - Implement Connection
  - Define Record Types based on SDMX structures
  - Caching
- Provider Design Time
  - Define static parameters and possible configurations
  - Choose nested type structure and members: properties, methods

In the case of SDMX, we have a case of multidimensional data. When making a query we need to provide a set of dimension value IDs to the query function. Chaining dimensions as it is in WorldBank type provider might lead to too long nested type structures since some of the data flows have more than 10 dimensions. For this reason, initial design will have a possibility to generate separate types per each dataflow and per each dimension. Dimension types will provide properties as values with all available and useful information required for the data query. Currently, two properties are important

- Position - for sorting keys in the query string
- Dimension ID - for building . separated data query

Listing [11] shows an example of how a user will be able to use TypeProvider for exploring and querying the data. Line 1 is the initialization of the type provider. Type WDI as a ProvidedType can be instantiated by providing dimensions as arguments, since each dimension has information about the position, the order of the arguments does not matter and query function will sort the values accordingly. WDI exposes access to the Dimensions fo that dataflow, and by following dimensions we can reach the dimension values such as Frequency: Annual. After constructing and providing all the arguments to

```
type WB =
 ↪  SdmxDataProvider<"https://api.worldbank.org/v2/sdmx/rest">
type WDI = WB.``World Development Indicators``

let data =
 WDI(WDI.Frequency.Annual_A,
     WDI.``Reference Area``.``United Kingdom_GBR``,
     WDI.Series.``Gross capital formation (% of
      ↪  GDP)_NE_GDI_TOTL_ZS``)
```

Listing 11. SDMX TypeProvider design

the WDI type we can execute the file. Note that until that time there is no need to execute code and design time of the type provider was triggering compiler to provide all necessary types. After executing we get a result object in data. Which by itself contains information about the data and actual enumerated values.

There are two stages to complete, first is to make a working version of TypeProvider which fetches the same data which is available in WorldBank TypeProvider documentation [7]. While doing this we should keep following SDMX standard guidelines and see if the same provider will also work for European Central Bank data. This approach will make it simple to compare existing provider to the new implementation and make the validation process easier.

Challenge is to first make a provider which works for simple cases on smaller scales and then incrementally try how it works on large scales and fix issues discovered. At the same time, it is important to always keep in mind to maintain simple usage.

3.6 Implementation

Implementation starts by creating two files $SdxmRuntime.fs$ and $SdmxProvider.fs$. Since we already have initial type structure defined we can start building it. For this, we need to implement run time code to prepare all the data required for generating types.

Run time part of the implementations is in $SdxmRuntime.fs$ file. It contains three main classes

- $ServiceConnection$ - used for communicating with external services and extracting schema from SDMX-ML format,

```
ProvidedTypeDefinition(asm, ns,
                       "SdmxDataProvider",
                       None,
                       hideObjectMethods = true,
                       nonNullable = true)

let parameters =
  [ ProvidedStaticParameter("WsEntryPoint", typeof<string>,
    ↪  default)
    ProvidedStaticParameter("Asynchronous", typeof<bool>, false)
    ProvidedStaticParameter("Language", typeof<string>, "en")]
```

Listing 12. SDMX Type Provider initial Definitions

- $DataFlowObject$ Object reperenting Dataflows, which also will get access to the data,
- $DimensionObject$ Object reperenting Dataflows.

Design time implementation starts from defining a base type and deciding which static parameters we need to provide. The base type is a starting point used for accessing the whole inherited type structure. Name of the base type is $SdmxDataProvider$ with one required static parameter $WsEntryPoint$. This is an HTTP/HTTPS URL later used for accessing the SDMX provider endpoints. Initially, two additional optional parameters are available: $Asynchronous$ like all other TypeProviders in FSharp.Data and $Language$ in case of multiple languages are available from data provider it is possible to force the preferred one. Snippets from the implementation of base type definition and static parameters are provided in Listing 12.

The next step is to attach other types to the base type. In the case of SDMX, we need to browse as through the Dataflows and choose one of them. This makes it logical to provide a list of data flows after user types dot(WB.). For this, we need to provide run time implementation, which accepts $WsEntryPoint$ argument and fetches Dataflows according to the SDMX standard definitions. All the Dataflows are constructed into F# record types[16] and provided to the design time component for usage. Listing 13 shows how Dataflow types and constructors are defined.

For each dataflow separate type is generated which by itself can provide access to the connected Dimensions and their values. At the same time, dataflow type has a possibility to provide access to the data by calling a constructor which accepts the number of parameters

```
ProvidedTypeDefinition(dataflowName, Some typeof<DataFlowObject>,
                       hideObjectMethods = true, nonNullable = true)

ProvidedConstructor(
    parameters = [
        for dimension in connection.GetDimensions(agencyId, dataflowId) do
            yield ProvidedParameter(dimension.Name, typeof<DimensionObject>)
    ],
    invokeCode = ( fun args ->
        let folder = fun state e -> <@@ (%%e:DimensionObject)::%%state @@>
        let dims = List.fold folder <@@ []:List<DimensionObject> @@> args
        <@@
            DataFlowObject(wsEntryPoint, dataId, %%dims)
        @@>
    )
)
```

Listing 13. SDMX TypeProvider Dataflows Type and Data Constructor

based on dimensions. Design decision which was made here is that dataflow Type is self-containing and can provide ways of requesting data and at the same time help in exploring and fetching every single dimension. Figure 5 shows accessing dataflows from visual studio.

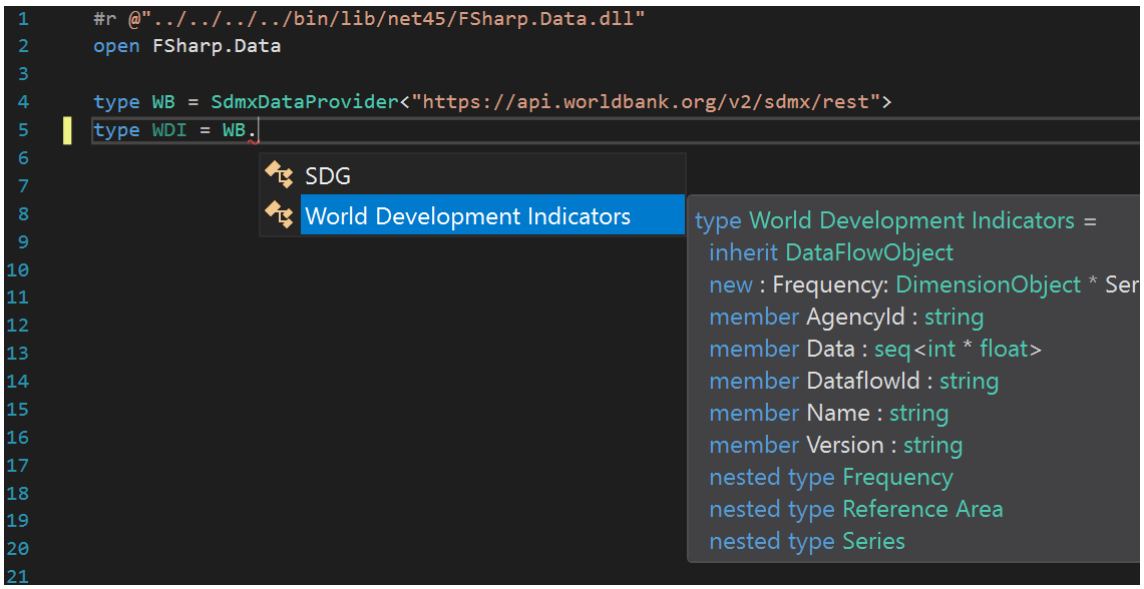

Figure 5. Access Datafows using SDMXTypeProvider

```
[for dimension in connection.GetDimensions(agencyId, dataflowId) do
    if dataflowId = dimension.DataStructutpreId then
    let dimensionTypeDefinition =
        ProvidedTypeDefinition(dimension.Name,
            Some typeof<DimensionObject>,
            hideObjectMethods = true,
            nonNullable = true)
    let dimensionId = dimension.Id
    for dimensionValue in dimension.Values do
        let dimensionValueId = dimensionValue.Id
        let dimensionValueProperty =
            ProvidedProperty(dimensionValue.Name,
                typeof<DimensionObject>,
                isStatic=true,
                getterCode = fun _ ->
                    <@@
                        DimensionObject(wsEntryPoint,
                        agencyId,
                        dataflowId, dimensionId,
                        dimensionValueId)
                    @@>
            )
        dimensionTypeDefinition.AddMember
        dimensionValueProperty
    dimensionTypeDefinition.AddXmlDoc(dimension.Description)
    yield dimensionTypeDefinition]
```

Listing 14. SDMX WorldBank Type Provider Dimension types and dimension value properties

According to SDMX, each Dataflow leads so the set of dimension. To translate this into TypeProvider case, after choosing a Dataflow we need to get a list of dimensions and their values with descriptions. For this, we can define nested types by attaching members to each dataflow type. $AddMembersDelayed$ is a method by $ProvidedTypeDefinition$ which accepts an anonymous function as an argument. This function should return a list of new types configured as described above. Implementation of the function body is provided in Listing 14.

After implementing run time and building the project DLL is ready to use. We can check out usage example image in Figure 6.

Note here that each dimension value has $_1D$ appended in type name, this is made for the following reason. In case we try to attach properties with the same name TypeProvider SDK raises an error and does not display any properties at all. In Figure 7 is shown error

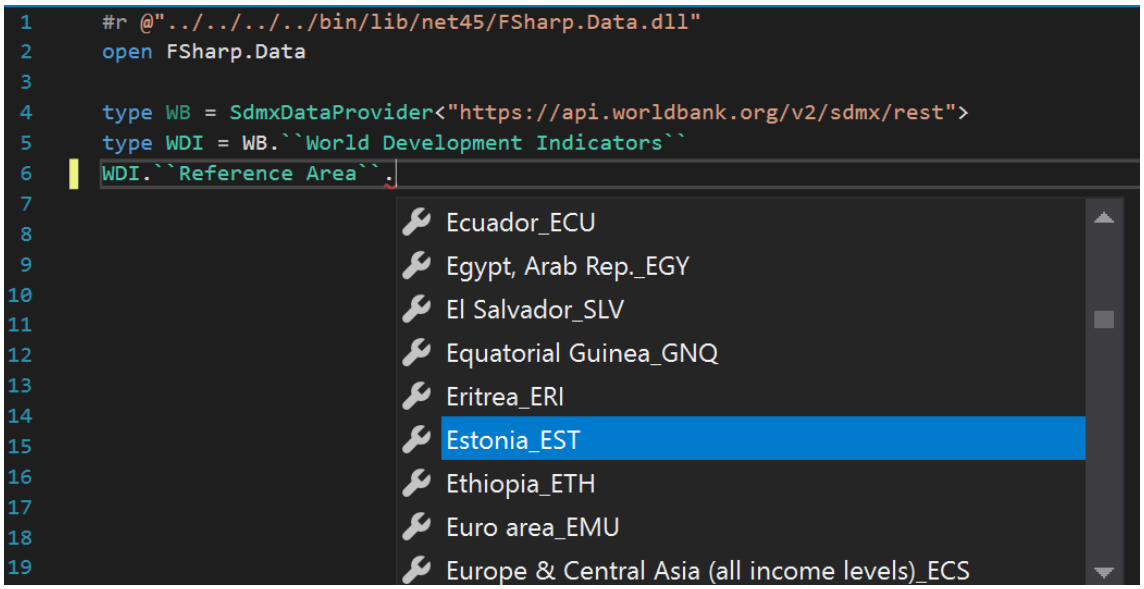

Figure 6. Access Dimensions using SDMXTypeProvider through the Dataflow Type

The type provider 'ProviderImplementation.SdmxProvider' reported an error: GetMethodImpl. not support overloads, name = 'get_Serbian dinar', methods - '[|Method get_Serbian dinar; Method get_Serbian dinar|]', callstack = '"

Figure 7. Property error message

message raised when exploring ECB Exchange rates Currency dimension. There are some of the properties repeated i.e. $Serbiandinar$ duplication shown in Listing 15. We cannot just generate types for all of them. For that reason, some kind of uniqueness is required. Appending $_ID$ fixed the uniqueness issue but made a type name a bit longer. ID appended is used in data query URL on the later stage.

```
<Code id="CSD">
  <Name xml:lang="en">Serbian dinar</Name>
</Code>

<Code id="RSD">
  <Name xml:lang="en">Serbian dinar</Name>
</Code>
```

Listing 15. ECB Currency Codelists snippet: Name duplication.

```
type internal DataflowRecord =
    { Id : string
      Name : string
      DataId : string
      AgencyID : string
      Version : string
      Header : HeaderRecord}

type internal DimensionRecord =
    { Name : string
        Description : string
        DataStructureId : string
        AgencyId : string
        Id : string
        EnumerationId: string
        Position: string
        Values : DimensionValueRecord seq
        Header : HeaderRecord}
```

Listing 16. SDMX TypeProvider Metadata Records

3.7 Records Types and Caching

For having an informative object representation in F# type system during run time it is handy to have record types defined. Listing 16 shows F# representation of SDMX responses in the two most important data structures Dataflows and Dimensions. It will be possible to check Header or Agency information separately in Listing 11 by accessing data(Dataflow) or Dimension object properties. This kind of design leaves options for easier structure changes in the future in case there are optimization issues or some new properties are required to add.

Caching responses from SDMX APIs is possible to do because SDMX-ML Schema is stable and is not meant to change frequently over time. FSharp.Data is using file based caching strategy for both Design time and Run time connections, there are cases when same endpoints are requested twice, cached results will be reused fine in such situations. More spesifically every time before requesting data froma specific URL first the local cache is checked if it was already requested before during runtime or design time. If there is no cached data only in this case new HTTP request will be sent and results will be stored in the cache for futue use.

```
type WB =
 ↪  SdmxDataProvider<"https://api.worldbank.org/v2/sdmx/rest">
type WDI = WB.``World Development Indicators``

let data =
 WDI(WDI.Frequency.Annual_A,
     WDI.``Reference Area``.``United Kingdom_GBR``,
     WDI.Series.``Agricultural land (sq.
      ↪  km)_AG_LND_AGRI_K2``)
```

Listing 17. SDMX TypeProvider design

4 Usage Scenarios and validation

4.1 WorldBank

At this point it is possible to use SDMX type provider for fetching exactly the same data, which is presented in WorldBank [7] documentation from FSharp.Data. Listing 11 described in design section, is the working implementation. We can change the values of dimensions in order to get different data, for example, to get annual agricultural land for United Kinkdom we can update the dimension value of the series as in Listing 17.

```
1  #r @"../../../../bin/lib/net45/FSharp.Data.dll"
2  #load
    ↪  @"../../../../packages/test/FSharp.Charting/FSharp.Charting.fsx"
3  open FSharp.Data
4  open FSharp.Charting
5
6  type ECB =
    ↪  SdmxDataProvider<"http://a-sdw-wsrest.ecb.int/service">
7  type EXR = ECB.``Exchange Rates``
8  let ecbData = EXR(EXR.Frequency.Annual_A,
9                    EXR.Currency.``US dollar_USD``,
10                   EXR.``Currency denominator``.Euro_EUR,
11                   EXR.``Exchange rate type``.Spot_SP00,
12                   EXR.``Series variation - EXR
                      ↪  context``.Average_A)
13 ecbData.Data |> Chart.Line
```

Listing 18. SDMX TypeProvider ECB Exchange Rates

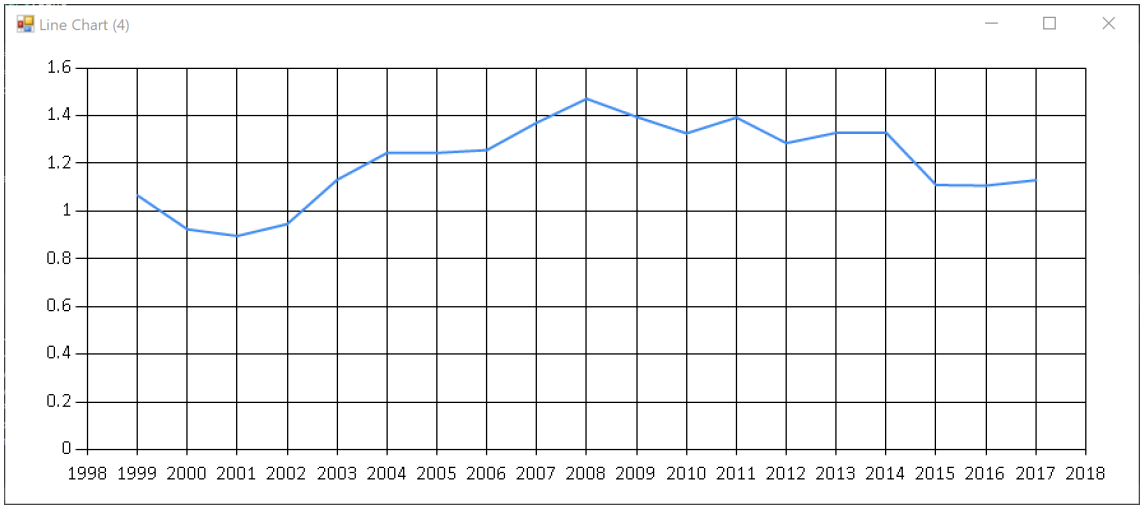

Figure 8. ECB Exchange Rates data plot

4.2 European Central Bank

To validate that approach works for other service providers as well we will try to use SDMX type provider against ECB. Let's say we want to query Exchange Rates data. For that we instantiate type provider as on line 6 in Listing 18. Then navigate to Exchange Rates dataflow and begin providing dimension value arguments to the constructor. Once we have provided all the arguments we can pass the data to the Charting library to see the results visually in Figure 8.

For the second example, we can see more complex use case. In case the user wants to

explore the data referenced in e.g. the ECB Statistics Bulletin, it is necessary to establish which data flows and combinations of dimensions are used. The ECB provides a tool called Statistical Data Warehouse [45]. Where it is possible for the users to explore the data in an interactive way.

Let us take a look at an example. Let us assume that we are interested in the data provided in Figure 2.3.9 in the ECB Statistical Bulletin [46]. Instead of proportional change, we would like to take a look at absolute changes in the types of loans. The data table 2.3.7 in the bulletin has quarterly resolution. We can use the ECB Statistical Data Warehouse to create the chart [47].

The resulting chart is given in Figure 9.

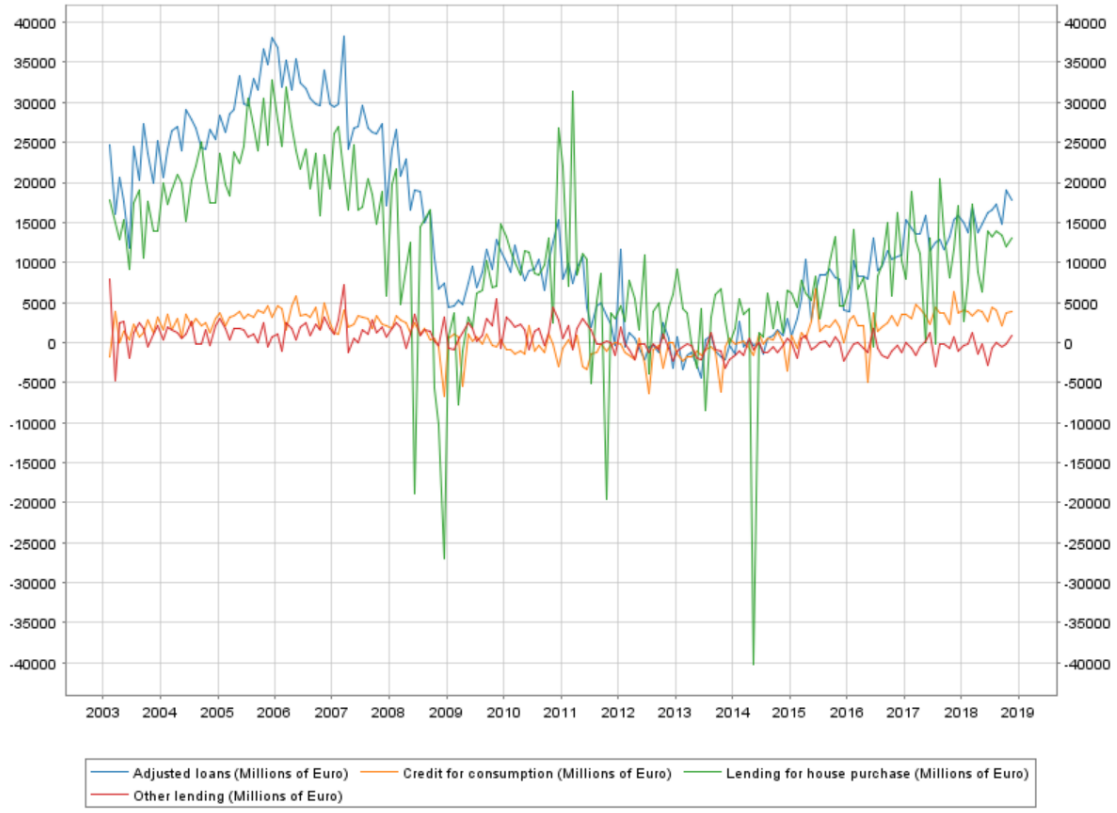

Figure 9. ECB Statistics Bulletin: Banking and investment funds

On the other hand, we can use the following code and the SDMX TypeProvider to achieve a very similar result: Listing 19.

As a reult we get a chart provided by FSharp.Charting in Figure 10.

```
#r @"../../../bin/lib/net45/FSharp.Data.dll"
#load
 ↪  @"../../../packages/test/FSharp.Charting/FSharp.Charting.fsx"
open FSharp.Data
open FSharp.Charting

type BSI = ECB.``Balance Sheet Items``

let getLoanData creditDimension = BSI(
   BSI.Frequency.Monthly_M,
   BSI.``Counterpart area``.
     ``Euro area (changing composition)_U2``,
   BSI.``Adjustment indicator``.
     ``Working day and seasonally adjusted_Y``,
   BSI.``BS reference sector breakdown``.
     ``Monetary and Financial Institutions (MFIs)_U``,
   creditDimension,
   BSI.``Original maturity``.Total_A,
   BSI.``Data type``.``Financial transactions (flows)_4``,
   BSI.``Reference area``.
     ``Euro area (changing composition)_U2``,
   BSI.``BS counterpart sector``.``Households and non-profit
    ↪  institutions serving households (S.14 and S.15)_2250``,
   BSI.``Currency of transaction``.
     ``All currencies combined_Z01``,
   BSI.``Balance sheet suffix``.Euro_E)

let data = [ getLoanData BSI.``Balance sheet item``.
                           ``Adjusted loans_A20T`` ;
             getLoanData BSI.``Balance sheet item``.
                           ``Credit for consumption_A21``;
             getLoanData BSI.``Balance sheet item``.
                           ``Lending for house purchase_A22``;
             getLoanData BSI.``Balance sheet item``.
                           ``Other lending_A23``]

// alternative:
let data' = [ BSI("M.U2.Y.U.A20T.A.4.U2.2250.Z01.E");
              BSI("M.U2.Y.U.A21.A.4.U2.2250.Z01.E");
              BSI("M.U2.Y.U.A22.A.4.U2.2250.Z01.E");
              BSI("M.U2.Y.U.A23.A.4.U2.2250.Z01.E") ]

data |> List.map Chart.Line
     |> Chart.Combine
     |> Chart.Show
```

Listing 19. SDMX TypeProvider ECB Bulletin Example Usage

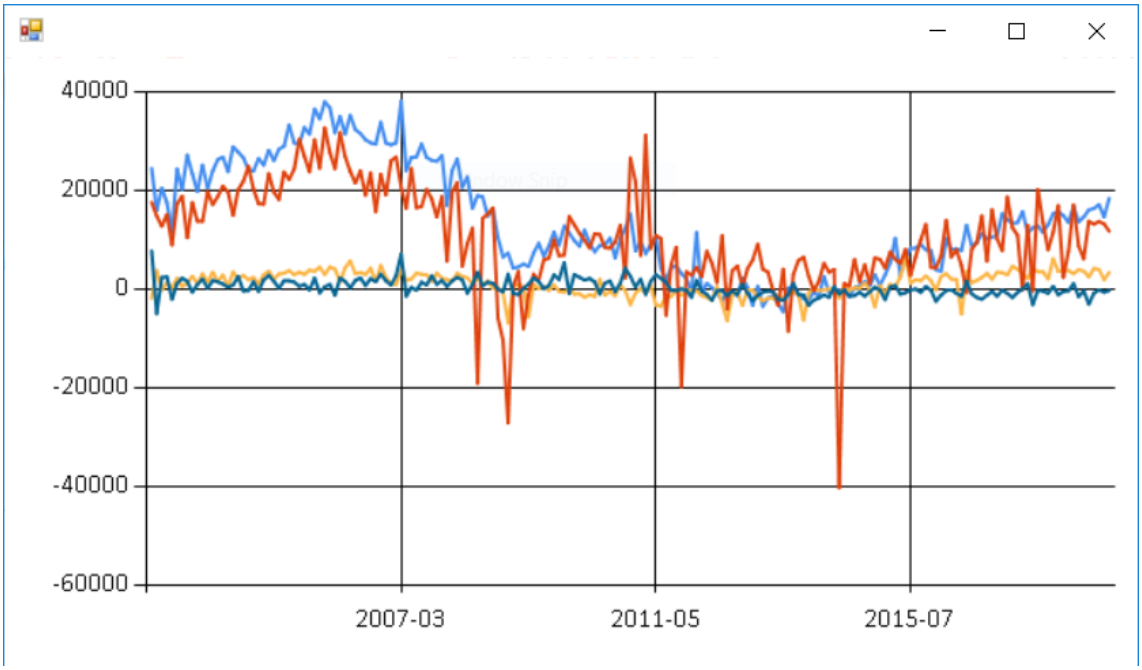

Figure 10. SDMX Examle chart of ECB Statistics Bulletin: Banking and investment funds

4.3 Future Work

A possible follow up of the current work is to create an SDMX data service for the Gamma project [12]. This is now much more easily achievable using the SDMX type provider. Since the Gamma is a great tool for delivering data to the data journalists and readers, it will become much more usable, since SDMX data service will open access to the larger scale of statistical information. This can be done using only single data service implementation which will use SDMX type provider by itself.

Some parts of the SDMX standard are not currently fully supported by the current implementation. For example, it is not possible to apply additional parameter filtering like periods, users will have to do the filtering locally. The paging support of the SDMX standard is currently not supported. Some of the meta information, e.g. headers and labels, are not fully extracted from metadata and provided. The remaining work is not conceptually complicated and is achievable by incrementally without changing the core functionality. The more important and difficult part is to test the type provider usage against more data providers. This is a huge task and is not achievable to do by one person. That is why it is important to involve the community and potential users in order to get more examples and suggestion how type provider usage API can be improved.

5 Summary

At this point, the initial SDMX type provider is implemented and tested with two different service providers, World Bank, and European Central Bank. The results allow us to conclude that the basic scenario of using the SDMX type provider to access SDMX data providers works at the proof of concept level. There are a few different directions of work which will improve the stability and check for support of various features within the standard. Checking the SDMX type provider against different service providers and data flows may identify some of the issues which are not known currently. Fixing these issues will improve the provider and make it more reliable. The current version does not support wildcards, this means that it is not possible to query data for all the dimensions or choose several dimensions(all countries or several countries). The .Stat suite already has a wide group of data providers including the Statistics Estonia, stat.ee [13]. Providing support for such endpoints will noticeably increase the amount of data access to which is supported by the SDMX type provider. The source code of the implementation is publicly available on github.com [48]. It includes example usage script in the examples folder.

References

[1] SDMX. Statistical data and metadata exchange. The official site for the SDMX community. [Online]. Available: https://sdmx.org/

[2] E. C. Bank. European central bank. [Online]. Available: https://sdw-wsrest.ecb.europa.eu

[3] CsvProvider. Csvprovider. [Online]. Available: http://fsharp.github.io/FSharp.Data/library/CsvProvider.html

[4] JsonProvider. Jsonprovider. [Online]. Available: http://fsharp.github.io/FSharp.Data/library/JsonProvider.html

[5] FSharp.Data. Fsharp.data. [Online]. Available: https://github.com/fsharp/FSharp.Data

[6] JSON. Json. [Online]. Available: http://json.org/

[7] FSharp.Data. Worldbank provider. [Online]. Available: http://fsharp.github.io/FSharp.Data/library/WorldBank.html

[8] S. T. W. Group. Sdmx rest. [Online]. Available: https://github.com/sdmx-twg/sdmx-rest

[9] sdmx. (2013) Guidelines for the use of web services section 7. [Online]. Available: https://sdmx.org/wp-content/uploads/SDMX_2-1-1-SECTION_07_WebServicesGuidelines_2013-04.pdf

[10] SDMX. Sdmx technical standards working group. [Online]. Available: https://github.com/sdmx-twg

[11] sdmx twg. Where is sdmx used. [Online]. Available: https://github.com/sdmx-twg/sdmx-rest/wiki/Where-is-it-used

[12] T. Petricek. The gamma project. [Online]. Available: https://thegamma.net/

[13] S. Estonia. Statistics estonia. [Online]. Available: https://www.stat.ee/database

[14] FSharp.Data. Fsharp.data docs. [Online]. Available: http://fsharp.github.io/FSharp.Data/

[15] P. E. R. Salas, M. Martin, F. M. Da Mota, S. Auer, K. Breitman, and M. A. Casanova, "Publishing statistical data on the web," in *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*. IEEE, 2012, pp. 285–292.

[16] sdmx.org. What is sdmx. [Online]. Available: https://sdmx.org/?page_id=3425

[17] ——. Sdmx versions. [Online]. Available: https://sdmx.org/?page_id=2555/

[18] SDMX. Introducing sdmx. [Online]. Available: https://sdmx.org/?page_id=1119

[19] sdmx.org. Sdmx user guide. [Online]. Available: https://sdmx.org/wp-content/uploads/SDMX_2-1_User_Guide_draft_0-1.pdf

[20] ——. Sdmx major version changes. [Online]. Available: https://sdmx.org/wp-content/uploads/SDMX_2-1_Major_Changes.pdf

[21] ——. Sdmx technical specifications. [Online]. Available: https://sdmx.org/?page_id=5008

[22] SDMX. Sdmx implementations by organisation. [Online]. Available: https://sdmx.org/?page_id=4713

[23] EUROSTAT. Statistical office of the european union. [Online]. Available: https://ec.europa.eu/eurostat/web/sdmx-infospace/sdmx-projects/dsd-availability

[24] WorldBank. Worldbank sdmx api queries. [Online]. Available: https://datahelpdesk.worldbank.org/knowledgebase/articles/1886701-sdmx-api-queries

[25] S. I. S. C. Community. [Online]. Available: https://siscc.org/what-we-do/products/

[26] ——. [Online]. Available: https://siscc.org/who-we-are/members/

[27] S. of Estonia. Sdmx database. Http://andmebaas.stat.ee/Index.aspx. [Online]. Available: http://andmebaas.stat.ee

[28] U. Service. Ukdata service. [Online]. Available: http://stats.ukdataservice.ac.uk/

[29] OECD. Organisation for economic co-operation and development. [Online]. Available: https://stats.oecd.org/

[30] A. B. of Statistics. [Online]. Available: http://stat.data.abs.gov.au

[31] A. Mattiocco. Sdmx helper tool. [Online]. Available: https://github.com/amattioc/SDMX

[32] X. Sosnovsky. sdmx-rest4js. [Online]. Available: https://github.com/sosna/sdmx-rest4js

[33] ——. Sdmx rest wiki. [Online]. Available: https://github.com/sosna/sdmx-rest4js/wiki

[34] S. TWG. Sdmx cheat sheet. [Online]. Available: https://github.com/sdmx-twg/sdmx-rest/blob/master/v2_1/ws/rest/docs/rest_cheat_sheet.pdf

[35] Metadatatechnology. Conceptscheme. [Online]. Available: https://metadatatechnology.com/about-sdmx.php#conceptscheme

[36] Metadatatechnology.com. Codelist. [Online]. Available: https://metadatatechnology.com/about-sdmx.php#codelist

[37] D. Syme, K. Battocchi, K. Takeda, D. Malayeri, J. Fisher, J. Hu, T. Liu, B. McNamara, D. Quirk, M. Taveggia *et al.*, "Strongly-typed language support for internet-scale information sources," *Technical Report MSR-TR-2012–101, Microsoft Research*, 2012.

[38] T. Petricek, D. Syme, and Z. Bray, "In the age of web: Typed functional-first programming revisited," *arXiv preprint arXiv:1512.01896*, 2015.

[39] *pandas*, https://pandas.pydata.org/.

[40] fsharp.org. Setup fsharp on linux and macos. [Online]. Available: https://fsharp.org/use/linux

[41] Microsoft. Tutorial: Create a type provider. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/fsharp/tutorials/type-providers/creating-a-type-provider

[42] Fsprojects. fsprojects. [Online]. Available: https://github.com/fsprojects/FSharp.TypeProviders.SDK/

[43] D. Nodia. (2018) Fsharp.data issue. [Online]. Available: https://github.com/fsharp/FSharp.Data/issues/1203

[44] dailydotnettips.com. Visual studio tip. [Online]. Available: https://dailydotnettips.com/did-you-know-you-can-play-sound-when-build-succeeded-or-failed-in-visual-studio/

[45] ECB. Statistics bulletin. [Online]. Available: https://sdw.ecb.europa.eu/reports.do?node=1000005

[46] ——. Monetary statistics. [Online]. Available: https://sdw.ecb.europa.eu/reports.do?node=10000030

[47] ——. Monetary chart. [Online]. Available: https://sdw.ecb.europa.eu/browseChart.do?df=true&ec=&dc=&oc=&pb=&rc=&DATASET=0&removeItem=&removedItemList=&mergeFilter=&activeTab=BSI&showHide=&BS_ITEM.14=A20T&BS_ITEM.14=A21&BS_ITEM.14=A22&BS_ITEM.14=A23&DATA_TYPE.14=4&BS_COUNT_SECTOR.14=2250&MAX_DOWNLOAD_SERIES=500&SERIES_MAX_NUM=50&node=bbn30&legendRef=reference&SERIES_KEY=117.BSI.M.U2.Y.U.A20T.A.4.U2.2250.Z01.E&SERIES_KEY=117.BSI.M.U2.Y.U.A21.A.4.U2.2250.Z01.E&SERIES_KEY=117.BSI.M.U2.Y.U.A22.A.4.U2.2250.Z01.E&SERIES_KEY=117.BSI.M.U2.Y.U.A23.A.4.U2.2250.Z01.E

[48] D. Nodia. Sdmx typeprovider implementation. [Online]. Available: https://github.com/demonno/FSharp.Data/tree/sdmx-types