Using Web Services from statistical packages for remote access to SDMX data sources

Gianpaolo Lopez
Attilio Mattiocco
Diana Nicoletti

Bank of Italy

{ gianpaolo.lopez, attilio.mattiocco, diana.nicoletti } @bancaditalia.it

Introduction

The Bank of Italy uses data from a number of different sources for its institutional and research activities. Managing different data formats and communication protocols means costly IT activities that often lack flexibility and are not reusable. In this paper we will describe a Proof of Technology that we have conducted for showing how the use of established standards such as SDMX and Web Services allows direct access to an SDMX provider site (like the European Central Bank’ Statistical Data Warehouse) from within statistical packages running in a local environment. Also, the design considerations that we made during the study will be explained in light of the current and future IT Environment of the Economic Research Area of the Bank of Italy.

The goal of the Proof Of Technology is to explore, from various standpoints, the possible patterns that allow leveraging of web service technology in statistical packages. As a starting point we decided to focus on two well known software tools: Matlab\cite{1} and R\cite{2}. This choice is based on various features: the two packages are very popular in the statistical and economic community as well as in the Bank of Italy, they are designed to be modular and extensible and, as we’ll see in the coming chapters, they already provide functions (out-of-the-box or in external modules) that can be used as the building blocks for integration with web services. Furthermore, they embody two main software philosophies: R is free, Open Source software that relies on the community for support and evolution, while Matlab is proprietary software developed and supported by the Mathworks Company. We will analyze the different possibilities that these tools offer for consuming web service and for processing SDMX-ML data. Particularly, we will focus on accessing
the ECB/SDW Public Web Services[^1] that are available in SOAP and REST flavor and provide data according to the SDMX 2.0 standard.

Currently the Bank of Italy receives data of interest to its statistical and economic research from multiple sources. Fig. 1 gives an overview of the main processes that involve data in the current IT environment. For the most part, data coming from institutional providers (ECB, OECD, BIS...) is gathered using a “PUSH” paradigm and replicated in the internal databases. Researchers and automated processes access data locally by means of specialized drivers that in turn depend on the tool and the various data formats used.

The picture just described could change in the next few months due to technological evolution. Externally, many organizations are providing data access by SDMX web services while internally all the statistical data organization and processes will be reshaped by a wide project that will introduce many architectural concepts from a Service Oriented Architecture[^2].
In Fig. 2 we tried to give an idea of how this new context could modify the current processes related to statistical data.

![Diagram](image.png)

In this new picture the end users would be able to access data from within their statistical packages by directly querying the data providers using a “PULL” paradigm and the Data Warehouse of the Bank of Italy would be populated exactly the same way. Of course this is only true for the data providers that allow this paradigm, while a “PUSH” method should be maintained for the others. The local Data Warehouse would then be accessible by internal users as if it were just one of the available data providers or, to put it another way, the users would not notice any difference between local and external data sources.

In this scenario maintaining local copies of data that is always available from outside sources would no longer be needed, resulting in many benefits in cost and efficiency. Of course in some cases and for particular reasons (e.g. performance, critical data availability) data replicas would also be retained, but this would be an exception.

The added value of SDMX in this new scenario is given by its broadness:

1. It provides a common data model that frees from the burden of adapting to the data models adopted by the various data providers.
2. It provides ready to use XML bindings for the data model, allowing web services developers and consumers to focus on the technical aspects and exchange well known and standardized message structures.

3. It provides some prescriptive constraints and some best practices for implementing web services, reducing the need of consumers to adapt to the different implementations

4. It provides a structural metadata model that allows consumers to search the data that is needed before pointing directly to it.

It should be clear, at this point, how the Proof of Technology that we’re going to describe is just a piece of a bigger picture.
Strategies for Web Services integration in statistical packages

There are two basic approaches that can be followed when implementing web services in R and Matlab (but this is true for most statistical packages). The first is based on using only functions that are offered by the tool itself or by its plug-ins, in various flavors and levels of abstraction. In the following pages we'll refer to this as the “native” approach.

The second way is focused on the use of an intermediate layer that is based on a technology that hides all the details about the web services technology (including XML) and thus simplifies development. Of course in this case there will also be the need to integrate this technology in the statistical tools, and to create custom components for transforming (marshalling and unmarshalling) data coming from the intermediate technology into native data formats. We’ll refer to this possibility as the “middle layer” approach.

In the following chapters we’ll describe most of the possibilities that R and Matlab offer for implementing native integration with web services and for processing the SDMX-ML parameters and results. On the other hand, where the middle layer approach is concerned, we’ll focus on the Java technology for its recognized maturity when dealing with web services (we’ll use the JAX-WS technology) and for its characteristic of platform independence and easy reusability in different contexts. Furthermore, both R and Matlab integrate powerfully with Java and this, as we’ll see later, makes it quite straightforward to implement the façade that will handle the middle layer API.

Even if we are going to describe all the possible choices we are not going to implement all of them anyway. To keep things simple we decided to explore two scenarios for both statistical packages and in particular:

For MATLAB

1. Implement a REST web services client following the native approach
2. Implement a SOAP client in JAVA, using the JAX-WS technology and a native façade for use from statistical packages

For R

1. Implement a SOAP web services client following the native approach
2. Implement a SOAP client in JAVA, using the JAX-WS technology and a native façade for use from statistical packages (same as Matlab)

The rest of this paper aims to be a very practical description of our experiences during the Proof of Technology and of the lessons that we learned from it.
The “Native” approach

Developing a Matlab client
Matlab has a strong and varied native support for web services. There are basically three ways to implement a web service client with the functions that this software provides out-of-the-box:

1. Function `createClassFromWSDL`. It is a very high level and powerful function that automatically generates a client that can then be used to consume the web service. It must pass an URL input pointing to the web service WSDL and it then creates for the user a set of convenience Matlab classes that can then be used for querying the service. Unfortunately this function has some limitations that do not allow its use with complex WSDLs, particularly with the ones that refer to namespaces other than the targetNamespace of the WSDL document.

2. SOAP functions: `createSoapMessage`, `callSoapService`, `parseSoapResponse`. They represent a lower level interface and expect the user to have a basic knowledge of the SOAP protocol as well as of the web service that he’s going to consume. They provide greater flexibility than the `createClassFromWSDL`.

3. Function `urlRead`. It is not explicitly targeted to web services, it’s a web read function that lets the user perform HTTP GET operations. It is very simple and useful for accessing REST web services.

As anticipated, with regards to Matlab we decided to use the REST web services API that are provided by the ECB Statistical Data Warehouse. This choice has of course simplified our operations removing, on one hand the need to create SDMX-ML Query Messages, and on the other hand the need to handle the SOAP protocol complexity. Moreover Matlab provides a very powerful function for parsing an XML file into a DOM structure: `xmlread`. This way, firing a REST call from Matlab and getting a DOM tree shaped result is as easy as writing the following code:

```matlab
function dom = run(query)
    xml = urlread(query);
    IS = org.xml.sax.InputSource();
    IS.setCharacterStream(java.io.StringReader(xml));
    dom = xmlread(IS);
end
```
Once the DOM tree has been built, the following code shows how it can be navigated for accessing, for example, the names of the dimensions in a Key Family Structure:

```matlab
function dimensions = getDimensionList(dom)
    allDimensions = dom.getElementsByTagName('structure:Dimension');
    length = allDimensions.getLength;
    if length > 0
        dimensions = cell(1, length);
        fprintf('Found %d dimensions for the Key Family.
', length);
        for k = 0:length-1
            thisDim = allDimensions.item(k);
            attributes = thisDim.getElementsByTagName;
            for i = 0:attributes.getLength-1
                thisAttr = attributes.item(i);
                name = char(thisAttr.getName);
                value = char(thisAttr.getValue);
                if strcmp('conceptRef', name)
                    fprintf('Dimension %d: %s
', k+1, value);
                    dimensions{k+1} = value;
                end
            end % for i
        end % for k
    else % if
        dimensions = cell(0);
    end % function
```

The drawback of using a DOM based parsing is probably the loss of performance and the risk of exceeding available memory when dealing with large files. We'll return to this in the chapter on performance.

In Fig. 5 you can see a typical use case developed during the PoT: the main web services client class (`ECBWSCli`ent) is created and it is the starting point of all interactions with the service. In the snapshot a data query is fired passing as input arguments a Time Series ID (as it is gathered from the SDW). The client (delegating a set of internal classes that can be seen in the top left panel) retrieves the Key Family structure and from this the list of dimensions, then it maps each dimension to its value in the ID, creates the REST query and retrieves the data requested. The data is then plotted with a Matlab function provided by the `timeseries` class.
Fig. 3
Developing an R client

There are three R packages that show functions for building a generic web services client and, as with Matlab, they too provide different levels of abstraction:

1. Package **RCurl**. It provides an interface to basic HTTP and Web functions by means of the well known **libcurl** library which it represents as a façade inside R. The basic function **curlPerform** can be used to fire a query to a web service (REST or SOAP) and provides great configuration flexibility, at the cost of handling all the details of the SOAP protocol and of the XML payload.

2. Package **SSOAP**. It is based on RCurl and provides higher level functions for handling the SOAP protocol and WSDL standard. One of the functions, **processWSDL** reads and converts a WSDL file into a set of R functions that can be used to consume the corresponding web services. Unfortunately this function does not have either general or complete support since it is still evolving. Anyway SSOAP also provides a set of lower level functions (.SOAP, toSOAP, parseSOAP, etc…) that are useful in hiding the complexity of building and processing SOAP messages.

3. Package **XML**. It provides general functions for processing XML files with different technologies (SAX, DOM, XPATH etc…)

In the native implementation for R we decided to consume SOAP web services using the RCurl package. This particular choice was made because of its more granular configurability that allowed us to setup Kerberos authentication for the http proxy (a thing that as of now we would still not be able to do with the functions provided by SSOAP). Of course this approach has forced us to manually implement all the code for handling the SOAP interactions and for processing the payloads. This even means the necessity of implementing functions to build the SDMX Query messages and to parse SDMX Responses.

As an example, once the SOAP envelope containing the SDMX Query message has been built, the code for calling the curlPerform function is shown in fig.6.

```r
hea = basicTextGatherer()
res = basicTextGatherer()
# set options for proxy, Kerberos authentication
opts=curlOptions(proxy=proxy_name,proxyuserpwd=":",proxyauth=4,header=FALSE)
res$reset()
hea$reset()

# SOAP action (e.g., sa="http://webservices.sdw.ecb/GetGenericData")
sa=paste(xml_namespace,"/",WS_name, sep="")
```
Once the service has been called, the payload of the response is the SDMX-ML message corresponding to the query made. The message has to be processed using the XML functions that R provides. As in the Matlab section, in fig. 7 we show an example of code for getting the names of the dimensions in a Key Family Structure.

```r
# set the new namespace for the incoming message
ns = c(x="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/structure",
y="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message")

Dims = list()
# get conceptRef
Dims = xpathSApply(doc_xml, "/x:Dimension", xmlGetAttr,"conceptRef", namespaces=ns)
```

Fig. 8 shows a snapshot of the same use case that has been shown in Matlab. The snapshot has been taken inside an Eclipse plug-in for R: STATET [10]
The “Middle Layer” approach

Developing a Java client

As we mentioned earlier, this approach relies on the development of a generic layer that acts as a proxy and provides a set of APIs that can be called inside the various statistical packages. The packages have to convert the results coming from the middle layer into their native data structures, so a little coding is required on the native side as well. Of course the complexity of the code required in the packages will depend largely on how well the package is integrated with the middle layer technology. Nevertheless, a well-crafted design of the data structures that traverse the layers can simplify this to a great extent. As mentioned previously, the technology selected for implementing the Middle Layer is Java. Some reasons for this choice are:

- Java supports web services with a huge number of mature technologies. The developer can choose many different approaches for building a web services client. Among them, one of the most efficient is using JAX-WS. With this technology creating a web services client from a WSDL is straightforward.
- Java is platform independent, thus the code developed can be reused in different contexts (windows user desktops, UNIX servers).
- Java components can easily be wrapped inside JEE components, which means the same code can be reused locally in the statistical packages as well as in centralized Enterprise Applications.
- R and Matlab provide powerful integration with Java, and we will see in detail in the forthcoming chapters how this simplifies the façade code.

Fig.9 shows a snapshot of some of the classes and packages that have been automatically created by the `wsimport` [11] command. As it can be observed, the main classes are created in the `ecb.sdw.webservices` package, while a number of additional packages named `org.sdmx.resources.sdmxml.schemas.v2_0.*` have been created for the JAXB data bindings with the SDMX Data Model.
At this point the code for creating a query and sending it over the web is just a matter of correctly using the generated classes. In the following boxes we highlight the code used to build a query, send it to the endpoint and process the results.

**Build a query with the name of the Key Family:**

```java
public static KeyFamilyWhereType getKeyFamilyQuery(String keyFamily){
    // create a where clause for the key family retrieval
    KeyFamilyWhereType wt = new KeyFamilyWhereType();
    if(keyFamily != null && !keyFamily.isEmpty()){ // set appropriate value
        wt.setKeyFamily(keyFamily);
    }
    return wt;
}
```
Submit the query:

```java
//get service
SDMXQueryService service = new SDMXQueryService();
// get port from service
SDMXQuery queryPort = service.getSDMXQueryPort();
// build query
KeyFamilyWhereType kfwt = SDMXQueryBuilder.getKeyFamilyQuery(keyFamily);
// fire query to endpoint
MessageStructureTypeWrapper results = queryPort.getKeyFamily(kfwt);
```

Process results and store data in appropriate structures.

```java
public static List<String> getDimensionList(MessageStructureTypeWrapper message){
    List<String> dimensions = new ArrayList<String>();
    List<KeyFamilyType> kfList = message.getStructure().getKeyFamilies().getKeyFamily();
    for (Iterator<
        KeyFamilyType> iter = kfList.iterator(); iter.hasNext();) {
        KeyFamilyType kft = (KeyFamilyType)iter.next();
        List<String> ksNames = kft.getName();
        for (Iterator<String> iterator = ksNames.iterator(); iterator.hasNext();) {
            String name = (String) iterator.next();
        }
        List<DimensionType> dims = kft.getComponents().getDimension();
        for (Iterator<DimensionType> iterator = dims.iterator(); iterator.hasNext();) {
            DimensionType dim = (DimensionType) iterator.next();
            dimensions.add(dim.getConceptRef());
        }
    }
    TimeDimensionType time = kft.getComponents().getTimeDimension();
    return dimensions;
}
```

Clearly the JAX-WS technology frees the developer from the need to handle SOAP structures or complex XML file parsing or building. The same results can be obtained using many different approaches in Java, at different abstraction levels. The struggle between supporters of highly automated tools that speed up development and “hand written” code that gives you full control is ongoing, and we’re not going to take a position. In the end, it always depends on the context and its aims.

**Java integration in Matlab**

Java integration in Matlab is provided out-of-the-box and is user-friendly. In the following box there is part of the code developed for un-marshalling a timeseries coming from the Java layer (in a `PortableTimeSeries` object that is basically a container of Array Lists) to a Matlab `timeseries` object.
As you can see, the conversion of Java objects is very simple (provided they have been properly designed). From a user perspective, consuming from the web service using native classes or Java classes just means the results need to be converted using a simple call to a custom converter like this one. This means very few lines of native code.

It is worth noting that, if the web calls have to pass through an HTTP Proxy, then the integration with Java will suffer from a bug that makes it impossible for Matlab to honor the proxy settings when dealing with a virtual machine. This behavior is being investigated by Mathworks and a workaround for it is to run in a non-graphical session (option \texttt{-nodesktop} at startup).

**Java integration in R**

To integrate Java classes in R we used the \texttt{rJava}\cite{rJava} package. It gives a low level set of APIs for creating and using Java objects. The functions provided by this package are a little less user-friendly than in Matlab, with “JNI style” signatures, but they are powerful and make it possible to write very few lines of code to obtain the desired results. In the next box we show the code that transforms a Java \texttt{PortableTimeSeries} object in a R \texttt{ts}.

```r
s = .jcast(rList[[i]], new.class = "it/bankitalia/reni/sia/sdmxclient/util/RTimeSeries", check = TRUE);
name = .jcall(s,"Ljava/lang/String;","getName", evalString = TRUE);
freq = .jcall(s,"Ljava/lang/String;","getFrequency", evalString = TRUE);
dimensions = .jcall(s,"[Ljava/lang/String;","getDimensionsArray", evalArray = TRUE, evalString = TRUE);
attributes = .jcall(s,"[Ljava/lang/String;","getAttributesArray", evalArray = TRUE, evalString = TRUE);
timeSlots = .jcall(s,"[Ljava/lang/String;","getTimeSlotsArray", evalArray = TRUE, evalString = TRUE);
observationsJ = .jcall(s,"[Ljava/lang/Double;","getObservationsArray", evalArray = TRUE, evalString = TRUE);
observations = array();
for(k in 1:numOfObs){
  observations[k] = observationsJ[k];
}
```

\texttt{ts} = \texttt{timeseries(observations, timeSlots)};
% set Attributes in DataInfo.UserData
\texttt{ts.DataInfo.UserData} = \texttt{cArrayAttrs};
% set Dimensions in ts.UserData
\texttt{set(ts, 'UserData', cArrayDims)};
\texttt{ts.timeinfo.startdate} = \texttt{cArrayTimeSlots(1)};
Some considerations about performance

Performance data that we summarize in the next table are just preliminary results in a test environment; at the moment local access is more efficient as compared to the remote one via SDMX web services; optimization efforts should be therefore performed in this area.

The results are related to a single query for Generic Data on the EXR Dataflow, with monthly frequency for a period from 2000 to 2010 and all currencies. This means about 40 time series with 130 observations each.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R “native”</td>
<td>6 secs</td>
</tr>
<tr>
<td>Matlab “native”</td>
<td>12 secs</td>
</tr>
<tr>
<td>Java</td>
<td>8 secs</td>
</tr>
<tr>
<td>Matlab + Java</td>
<td>9 secs</td>
</tr>
<tr>
<td>R + Java</td>
<td>15 secs</td>
</tr>
</tbody>
</table>

All measurements seem coherent with the fact that the Java middle layer implies an overhead that is due to the communication between the two layers and data marshalling. The only surprising measurement is the one related to the native Matlab implementation, but in this case the DOM parser is very likely to be responsible for the performance decrease.
Final Considerations

The experience gained from this Proof of Technology has shown quite clearly that, regardless of the type of approach that is chosen, an integration of SDMX Web Services into Matlab and R is unquestionably feasible. Also, the tools that were used showed powerful integration capabilities both for the native approach and for the mediated one. Which approach is best is something that is very likely to depend on the context and its aims. Nevertheless, let us list some pros and cons for each of them.

Native integration

Benefits:

- No need to know programming languages other than the package specific ones. Users of statistical tools often know the programming language of the package they use, but they are rarely programmers.
- Following on from the previous point, this approach makes it easier for users to be trained by IT staff so they can create their own web services clients independently.
- No infrastructure is needed. This makes it easier to implement, deploy and maintain all the artifacts.

Drawbacks:

- Limited reuse. All the code has to be developed in the statistical package language, so it cannot be reused with other tools. This means many different implementations of the same functions with high development and maintenance costs.
- Support for advanced web services features is not assured.

Middle Layer integration

Benefits:

- The majority of development is in the Middle Layer with greater benefits for code reuse. Only a small portion of the code will be needed on the native side. Code reuse, as usual, reduces the costs of development and maintenance.
- Developing the Middle Layer using a general purpose language, like JAVA, enables the user to leverage with greater flexibility regarding the choice of tools, configuration and technologies.
- The components that are written in a general purpose language can easily be reused outside statistical packages. As an example, consider an Enterprise Application (possibly part of a Data Warehouse) that feeds data into local databases. The same code, correctly componentized, will be used as the Middle Layer for statistical tools.
Drawbacks:

- Users that wish to develop their own drivers will have to learn a second programming language. This will probably mean that the IT Department will be heavily involved.
- Development and Deployment is more complex since there are at least two components to be managed.

It is clear that all the approaches show strengths and weaknesses, thus the choice for the most efficient approach strongly depends on the scenario. In a context in which the focus is on the statistical tool, with the need to develop many clients for different data sources, the native approach is likely to be the correct one. On the other hand, if the requirement is to build a well known set of clients for stable data sources and make the drivers available for different statistical tools, a middle layer approach can be a worthwhile investment.

One final consideration can be taken into account regarding the different implementations of the SDMX standard. It is beyond the scope of this paper to deal with multiple data sources, but some informal testing made with data providers other than the ECB (like the OECD.STAT) showed slight differences in the interpretation of the standard that, even if not relevant in itself, created many difficulties when attempting to reuse code and develop general purpose components. Furthermore, the non prescriptive nature of some best practices contained in the “Web Services Guidelines” can lead to a proliferation of very different implementations that can expose consumers to strong customization efforts. It could be useful to analyze the pros and cons of transforming some best practices into prescriptive rules.

Next Steps

There were a number of things left out because they were considered beyond the scope of this PoT. It is a list that we hope to soon examine. Among them:

- Investigate the impact of Security requirements especially when user authentication is required, as it happens for instance in the ESCB version of ECB Statistical Data Warehouse
- Extend PoT to more statistical packages (STATA, SAS, FAME, Modeleasy…)
- Extend PoT to new Data Providers (OECD, IMF, ISTAT…)
- SDMX 2.1 support and new features (e.g. list of data flows, list of timeseries in a dataflow…)
- Extension to SDMX “Navigation/Browsing”
Acknowledgements

Many thanks to Demetrio Condello (Bank of Italy), Francesca Perino (Mathworks) and Duncan Temple Lang (University of California) for their help throughout this PoT. Demetrio helped us with the many configuration issues that we had to overcome during the PoT. Duncan and Francesca shared with us their precious knowledge and experience with R and Matlab.

References