Blaise and Xml: Experiences Of An Outsourcing Project[1]

Leif Bochis Madsen, Statistics Denmark

Abstract

Xml is a standard for the representation of tree structures with decorated nodes and today it has become a common for exchanging information in a platform-independent way.

Because of their hierarchical nature Blaise data and meta data are quite easy to convert into xml, however, various demands may influence the way this should be done – among them conformance to standards requirements and, of course, the actual application.

Statistics Denmark has recently been through a process of outsourcing telephone as well as personal interviewing for the Labour Force Survey while post processing of forms including coding still should be carried out in-office.

Among the tasks was definition of a data exchange format using xml for the exchange of questionnaire data and development of tools for automatic generation of e.g. xml schemas to support validation of xml documents and for conversion between Blaise and xml.

The paper summarizes some experiences and challenges from this project.

1. Introduction

Early in 2006 it was decided that the interviewing for the Danish Labour Force Survey should be outsourced from January 2007.

During the spring and summer of 2006 Statistics Denmark prepared for a competitive tendering and in the autumn a contract was signed with a consortium of organizations on the interviewer market (hereafter 'The Interview Consortium').

Meanwhile, the development of the various artefacts needed to support the process began in order to be ready for a pilot study in November 2006 and for production in January 2007.

[1] The opinions and assessments stated in this article are those of the author and do not necessarily reflect opinions and assessments of Statistics Denmark.

1.1 Requirements

Essential for the architecture was the decision to exchange data in xml format. This was a demand published in the material for the competitive tendering, and the process of defining an exchange format started in the early summer of 2006.

Post processing of forms should be carried out in-office and it was decided to reuse the already developed Blaise instruments for this purpose. In order to save time it was then an obvious choice to use the previously used Blaise Interview instrument as the basis of developing an interview test instrument and as the metadata base on which we could automatically generate proper xml definitions. Experimental work had already been done concerning the representation of Blaise data in xml, but these experiments had never been used for practical work. Now, these efforts showed to be a firm base for the contemporary needs.

The software that Statistics Denmark needed to develop in order to support the new data capture process thus consisted of the following elements:

- A Cameleon script for the generation of an xml schema
- A VB program for changing the character set (as Cameleon is not capable of writing data in UTF-8)
- A VB program for extracting data into xml format (for testing the application)
- A set of further VB-programs /Cameleon scripts for converting the metadata into xml
- A C# program for loading xml data into Blaise
- A java script for changing the xml into a format suitable for reading into Blaise (splitting the data into different versions and changing some inapplicable codes into applicable ones)
- A Maniplus program for managing the process

The C# program and the java script were quite new artefacts while the rest all could be written as modifications of previous artefacts.

Further requirements for the project comprised an effort to conform to standards for data exchange set up by the Danish Ministry of Science, Technology and Innovation – the so-called OIO standards.

These standards are not mandatory for a “closed shop” data exchange, but reuse of elements already defined by other organizations and a general policy of Statistics Denmark to conform to the standards implies the use anyway.

We shall briefly describe the purpose and characteristics of these standards.

Also, the Blaise 4.8 xml export was yet to come, so we had to invent this wheel, too. Through a case study we shall take a look at some of the differences and similarities between Blaise 4.8 xml format and the xml exchange format for this project.

1.2 OIO – Public Information Online

The purpose of the Danish Government OIO (Danish acronym for "Public Information Online") initiative is to improve standardization and data exchange.

As stated on the project homepage:

"*The vision is that the Danish OIOXML project will:*

- *Improve the exchange of data both internal in the public sector and between the public and private sector.*
- *Improve the processing of data, and make easy access to already collected data and the re-use of these.*
- *Make it easier to implement E-services*

To realise this vision two initiatives are initiated: The Infostructurebase and the standardising work "[2]

The Infostructurebase comprises among other things a set of definitions of public information in the shape of xml schemas. It is an obligation for a public organization responsible for some public information to maintain the xml schemas in the Infostructurebase and to provide the information available as xml data conforming to these definitions. Organizations comprise government agencies, municipal authorities, private organizations etc.

Among the standardization efforts are a restriction of how xml schemas should be designed:

"*XML schema definitions*

- *Write the tag names in English; the names should be indicative of the purpose. Danish descriptions should be schema definitions as alternative definitions.*
- *Standardise the naming of central elements. Those are e.g. primarily the central registers, in which it is important to establish the naming of central elements such as First name, Last name, Address, as these definitions are used across the boundaries of public authorities. Whether or not it is possible to use PersonName and other standards of XML.org should be investigated.*
- *Write the tag names with the following notation: "UpperCamelCase", i.e. initial capital letter, the rest lower case. Compounds such as Community Number should be written as "CommunityNumber".*

[2] From the OIO website

- *Fundamental attributes should be established in order to be able to make external references to the relevant registers.*

- *UTF-8 is to be used as the standard character set.*

- *If there are homogeneous constructions in more places, these should be made as independent definitions.*

- *Elements are to be documented in schema definitions, so that every field is well defined.*

- *Make strong data types. Ex.: For a date the data type DATE should be used in schema definitions.*"[3]

As mentioned in the introduction it is not mandatory to conform to this standard for peer-to-peer systems like the one developed for the exchange of data between Statistics Denmark and the Interview Consortium. It is, however, a general policy of Statistics Denmark to do it so far it is convenient, and some of the information used for the LFS actually stems from sources belonging to this scheme.

Therefore, as it was not possible to ignore the OIO we had to cope with it.

2. Architecture of the LFS data capture system

The capture of data for the new LFS now comprises the following work steps:

Each quarter a revision of the LFS questionnaire is carried out, for the most resulting in a revised version of the questionnaire.

The design of the questionnaire is carried out in the LFS section at Statistics DK and implemented in Blaise for testing purposes. At the same time a post editing tool is developed from the same Blaise source.

A description of the questionnaire along with an xml schema defining the interchange format is mailed to the Consortium for their implementation.

Respondent data are sent to the Consortium once a month in xml format.

Daily, an xml file is uploaded to Statistics Denmark containing the completed interviews from the previous day along with non-response and expired interviews.

The data from this xml file is loaded into a Blaise database for post editing.

2.1 Design of the questionnaire

Although it should not be used for actual interviewing, a questionnaire for each version of the survey is implemented as a Blaise instrument for the purpose of testing the filters.

[3] Ibid.

By the use of prepare directives it has been quite easy to augment the source code of the interview instrument with source code for the data editing instrument.

As roughly 99.9 percent of the interview instrument may be reused for the data editing instrument and the interview instrument might comprise some 95 percent of the data editing instrument this approach has proven efficient.

Thus, by the end of the design and test phase, everything is prepared for receiving the data from the Interview Consortium.

2.2 Interchange format

From the Blaise interview instrument it is possible to extract the metadata by means of Cameleon or the Blaise API. Therefore, we developed a Cameleon script that could automatically create an xml schema to be used by the Interview Consortium for validation before sending the data. Also, some other documentation of the questionnaire is produced by Cameleon scripts and some VB programs using the Blaise API. This documentation is then sent to the Interview Consortium for their preparation of the questionnaire.

2.3 Overall architecture of the Data Capture Process

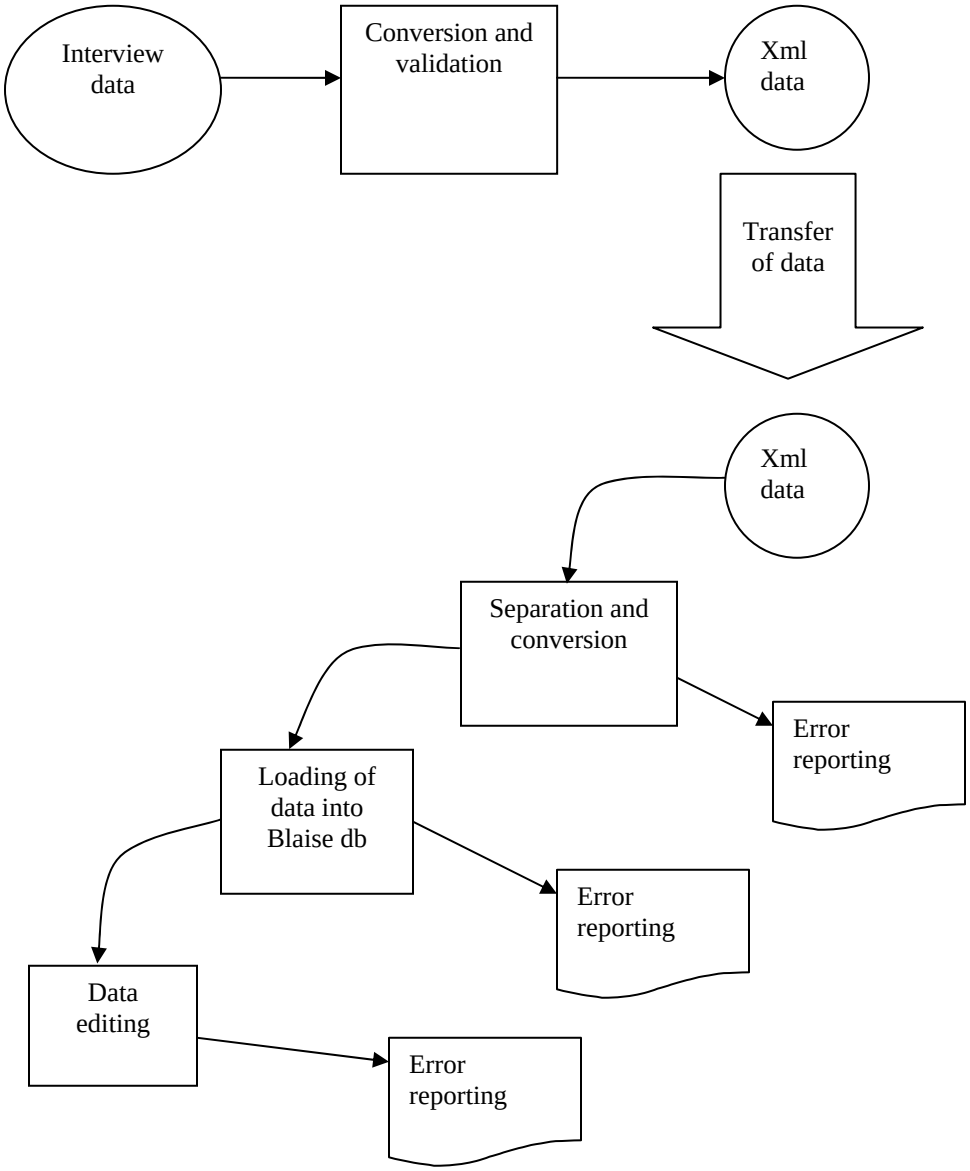

Fig. 1: Overall architecture of the data capturing system for the LFS

The Interview Consortium collects data from interviewers – from Telephone as well as Personal interviewing – on a daily basis. These data are converted into an xml document and validated towards the xml schema. The data are validated separately for each active survey, i.e. in the beginning of each quarter some data may belong to the stratum of the previous quarter. Also, a pilot study with interviews in yet another questionnaire may

possibly be part of the data stream. These data are combined into one single document ready for transfer (fig. 2).[4]

```
<Forms>
        <Form_Lfs2007q2> … </Form_Lfs2007q2>
        <Form_Lfs2007q2> … </Form_Lfs2007q2>
        …
        <Form_Lfs2007q3> … </Form_Lfs2007q3>
        <Form_Lfs2007q3> … </Form_Lfs2007q3>
        …
        <Form_MyPilotStudy> … </Form_MyPilotStudy>
        …
</Forms>
```

Fig. 2: Xml document

2.4 Processing

After receiving the xml document some processing is carried out:

1. It is split up into separate xml documents – one for each actual survey.
2. Some data in the xml document is converted into a format suitable for reading into a Blaise database.
3. The data are actually read into a Blaise database (Interview format).
4. The data are automatically coded and converted to another Blaise database (Editing format)
5. Dirty forms are edited (primarily coding of Occupation, Business and Education)

From each of these steps certain automatic or semiautomatic processing is carried out. The automation is heavily based on strict naming conventions for xml elements and file names.

2.4.1 Separation of xml document

On receipt of the xml document the first step is reading the document and separating into portions for each survey. This process is carried out by a java script that reads the data as a DOM document and writes the data into separate xml documents based on the

[4] Eventually, the data were split into two documents – one for telephone interviews and another for personal interviews, due to the different handling of the two kinds of interviews by the Interview Consortium and different deadlines for completion of interview strata dependent on the interview mode. Telephone interviews must be completed by one and a half week after the period questioned, personal interviews may be completed until four weeks after. Either of the xml documents, however, may contain forms from more than one version of the questionnaire.

Form_XXX-elements. The resulting xml documents are all named as the input file with the name of the data model appended.

LfsData_2007-07-26.xml →
LfsData_2007-07-26_ Lfs2007q2.xml + LfsData_2007-07-26_ Lfs2007q3.xml + …

From this process an error may be reported if the xml document is not well formed, or if the resulting documents cannot be validated towards the respective schemas.

2.4.2 Conversion of unsuitable values

Certain fields of the data model (stratum data) allow enumerated values with spaces. This is perfectly legal in xml and also according to the OIO xml definition.

As an example, in OIO xml the marital status is defined like this:[5]

```
<schema
   targetNamespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/09/19/"
   elementFormDefault="qualified" version="1.1">
<element name="MaritalStatusCode" type="cpr:MaritalStatusCodeType">
        <annotation>
                <documentation>
                Identifies the maritial status and whether the person is alive or dead.
                </documentation>
        </annotation>
</element>
<simpleType name="MaritalStatusCodeType">
        <restriction base="string">
                <enumeration value="married"/>
                <enumeration value="divorced"/>
                <enumeration value="widow"/>
                <enumeration value="registered partnership"/>
                <enumeration value="abolition of registrered partnership"/>
                <enumeration value="longest living partner"/>
                <enumeration value="deceased"/>
                <enumeration value="unmarried"/>
        </restriction>
</simpleType>
</schema>
```

Fig. 3: OIO definition of Marital Status

An example data field therefore could comprise the following value:

```
<MaritalStatus>longest living partner</MaritalStatus>
```

However, these values are not well suited for reading directly into a Blaise enumerated field, so these field values are changed in advance. This is also carried out by the same java script as mentioned above.

[5] From: The OIO Infostructurebase

As an impact of our efforts to conform to the OIO standard we implemented an XML language in the Blaise data model in order to represent references to the OIO.

Until now only two fields of the LFS data model are affected, so it's still manageable. It should be possible, though, to write a Cameleon script that could automatically generate a java script for carrying out this conversion based on the defined Blaise data model including XML language texts.

2.4.2 Conversion into Blaise

Conversion of the xml document into a Blaise database is carried out by a program written in C# and using the Blaise API. The program traverses the xml document and for each value the proper field is imputed, for example, the following data:

```
<MyBlock><MyField>MyValue</MyField></MyBlock>
```

should be assigned as:

```
MyDB.Field("MyBlock.MyField").Text = MyValue;
```

From this process error reporting comprises illegal field values as well as illegal (unknown) field names etc.

2.4.4 Automatic coding

The process of automatic coding is a well-established process in the LFS post processing in Statistics Denmark. The error reporting from this process mainly is about unsuccessful coding and may be used to refine the automatic coding process or to produce feed-back to interviewers about ambiguous descriptions etc.

2.4.5 Data editing

The data editing instrument is primarily developed for carrying out coding of Occupation, Business and Education.

However, this editing tool also showed to be the place systematic errors in routing or data conversion from the side of the Interview Consortium has been discovered.

3. Overall experience of the process

First of all, the project has been a success in the sense that we have managed to outsource interviewing for a large survey and still get data that are comparable to what we would have gained from continuing the interviewing in-house.

Obviously, there is a risk of loosing the control of the process when outsourcing a core process like interviewing in the data capture process. Assessment of the quality of data is therefore a crucial part of such a project.

After six months of operation the daily exchange of data works all right, though there have been examples of misinterpretations detected – especially in connection with set up of new versions of questionnaire.

One of the most important challenges is that an xml schema may assure correct format of the data, but not the integrity (rules). Some routing errors have thus been detected when reading the xml document into the Blaise database for editing.

Considerations for improving the control of data therefore imply means for combining the validation against the xml schema with other validations.

Some research is now being carried out in this field.

4. Case study: The differences and similarities between Blaise 4.8 xml and Statistics Denmark exchange formats

Noticeable, the xml data files generated by Blaise 4.8 and the Statistics Denmark exchange format are very equal in structure – probably due to the fact that Blaise data are hierarchical by nature and therefore by same nature should be converted into xml while retaining this hierarchical structure.

The first difference that appears to the viewer is that Blaise 4.8 xml uses a document element <Dataset> containing one <Datarecord> element for each form in the dataset. The similar elements in the Statistics Denmark exchange format are <Forms> and <Form_DATAMODELNAME>, respectively.

The reason for naming the record element with the name of the data model is due to a requirement that one single xml document should be delivered from the supplier, possibly containing forms from (at least) two consecutive surveys (or pilot studies). Thus, this naming model allows the supplier to deliver one xml document with e.g. contents like this:

```
<Forms>
        <Form_Lfs2007q2> … </Form_Lfs2007q2>
        <Form_Lfs2007q2> … </Form_Lfs2007q2>
        …
        <Form_Lfs2007q3> … </Form_Lfs2007q3>
        <Form_Lfs2007q3> … </Form_Lfs2007q3>
        …
</Forms>
```

When receiving this document it's quite easy to split it into a number of xml documents each containing forms of only one data model for further processing.

When looking at the interior of the form/datarecord the elements are named in both formats by the field or block instances:

```
<MyBlockInstance>
        <MyFirstField> … </MyFirstField>
        <MySecondField> … </MySecondField>
        …
</MyBlockInstance>
```

The actual values, however, are supplied slightly differently. In the Blaise 4.8 xml format the values are supplied as simple text between the name element tags, for example:

```
<Age>48</Age>
<Gender>Male</Gender>
<CivilStatus>Married</CivilStatus>
```

While the Statistics Denmark exchange format would supply these values like this:

```
<Age><value>48</value></Age>
<Gender><value>Male</value></Gender>
<MaritalStatus><value>Married</value></MaritalStatus>
```

Why? Well, look at how non response values are supplied. In the Blaise 4.8 format non response values are supplied through element attributes.

The Statistics Denmark exchange format, however, conforms to the recommendation from the OIO committee that element attributes should be avoided whenever possible.

```
<Age><nonResponse>EMPTY</nonResponse></Age>
<Gender><nonResponse>DONTKNOW</nonResponse></Gender>
<MaritalStatus><nonResponse>REFUSAL</nonResponse></MaritalStatus>
```

Set values are rather equally filled:

```
<MySetField blaiseContainerType=”set”>
        <choice>blue</choice>
        <choice>green</choice>
        <choice>yellow</choice>
</MySetField>
```

compared to:

```
<MySetField>
        <value>blue</value>
        <value>green</value>
        <value>yellow</value>
</MySetField>
```

Notice, that when reading these data into Blaise the information that the field is of type set is not needed. Why? Because, when reading data into a Blaise database the Blaise datamodel is available and can tell the xml reader that this field is a set field.

Similarly, non response values are supplied just like other kinds of fields, respectively:

<MySetField xsi:nil=”nil” blaiseNonResponseType=”DONTKNOW” />

<MySetField>
<nonResponse>DONTKNOW<nonResponse>
</MySetField>

Array fields have different layouts in the two formats:

<MyArrayField blaiseContainerType=”array”>
<MyArrayFieldItem blaiseIndex=”1”>
myvalue1
</MyArrayFieldItem>
<MyArrayFieldItem blaiseIndex=”2”>
myvalue2
</MyArrayFieldItem>
…
</MyArrayField>

Compared to the somewhat simpler:

<MyArrayField><value>myvalue1</value></MyArrayField>
<MyArrayField><value>myvalue2</value></MyArrayField>

Just like in the case of set types when reading data into a Blaise database you know from the data model that this field is an array and the proper index may be supplied automatically.

You may still supply empty array field, e.g.:

<MyArrayField><value>myvalue1</value></MyArrayField>
<MyArrayField/>
<MyArrayField><value>myvalue3</value></MyArrayField>

Or:

<MyArrayField><value>myvalue1</value></MyArrayField>
<MyArrayField><nonResponse>EMPTY</nonResponse></MyArrayField>
<MyArrayField><value>myvalue3</value></MyArrayField>

References

OIO website, http://www.oio.dk [last viewed July 27, 2007]

OIO Infostructurebase, http://isb.oio.dk/info [last viewed July 27, 2007]