Validation of Survey Data in Xml

Leif Bochis Madsen, Statistics Denmark[1]

Abstract

In a number of surveys conducted by Statistics Denmark all or part of the data collection is carried out outside the office and delivered in some kind of electronic format.

Increasingly, xml is used as format of delivery and this offers some opportunities to push validation of the data towards the supplier. For example, an Xml Schema may be used by the data supplier in order to assure the correct format before submitting the data.

However, other xml technologies are available which provide different means of validating the data in even more sophisticated ways.

The paper will discuss some technologies available and the possible role of Blaise.

Introduction

In a paper written for the IBUC/2007[2] an outsourcing project for the Danish Labour Force Survey (LFS) is described. The requirements and architecture of this project is fully described in this paper and I shall only summarize briefly.

Due to legal causes Statistics Denmark could not require the use of specific software – i.e. Blaise – by the organizations offering tenders for this project. Therefore - among other aspects – the outsourcing project involved specification of a data exchange format and efforts made to check validity of the data transferred from the organization responsible for conducting interviews to Statistics Denmark. The format chosen was xml and the work carried out in Statistics Denmark comprised construction of software capable of validating the xml data and importing the data into Blaise[3].

The validation was split into two parts. The first part consisted of checking the conformance to an xml schema and was the responsibility of the supplier of data. The second part was implemented as a Blaise data editing instrument and carried out inside Statistics Denmark.

After three years of operation some problems in this setup have been identified and ideas of improving the data exchange have occurred.

The given conditions are:

1. Data must be exchanged in xml format.
2. It is not possible to require external data providers to use non-standardized software (i.e. Blaise)

[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.

[2] See [LBM07]

[3] This project was carried out before the Blaise Xml format was published in Blaise 4.8. A lot of work could have been avoided if the project had been carried out one year later, but that's life!

Problems

Different survey software offers different sets of properties and possibilities. For example, in Blaise it is easy to describe rules comprising a computation of dates – e.g., the day six months earlier than today – and using this computed date in a condition – e.g., comparing this day with the respondent's start of job – and this way decide which questions to ask next. This kind of computation was not possible to carry out in the software used by the interviewing organization. Following this, some forms that were correctly filled in the interview software turned out to be faultý in the Blaise editing instrument.

Another issue is the treatment of warnings. It will not make sense in post-editing to examine – or re-examine – all warnings that occurred and may have been suppressed under interviewing. However, a statistical treatment of the remaining warning conditions may be relevant in order to identify, for example, weak parts of the questionnaire. This statistical treatment is not straight-forward to carry out using Blaise tools.

Also, during the three years of operating the LFS a number of misunderstandings concerning routing etc. has been discovered. It is an assumption that there is a potential for improved documentation of the survey and better communication between the supplier and receiver of interview data[4]. This requires a media that may be used for exchange of metadata.

Eventually, there is a wish to push more validation towards the supplier in order to discover possible discrepancies at an earlier stage.

Overall, the project has emphasized the need for platform independent tools for validation and formal description of routing, checks and warnings.

Tools for Validation

In a thesis[5] written at the IT University of Copenhagen, a study of different tools for validation of xml data has been carried out using the LFS outsourcing project as a case study.

The study examines a number of available tools like Xml Schema, RELAX NG[6] and Schematron[7], that all support some kind of 'co-occurrence constraints'.[8] The two most comprehensive of the tools are Xml Schema and Schematron and they will both be briefly discussed here.

Both are so-called schema languages, i.e. languages that describe an xml document type and can be used in order to test whether a particular xml document conforms to its definition.

Xml Schema is a grammar-based language which means that a grammar describes the structure and contents of a document. Validation implies that errors are reported if the document does not conform to the grammar. From version 1.1 – released autumn 2009 – Xml Schema incorporates so-called assertions that may describe co-occurrence constraints. However, it lacks some features that are widely used in Blaise rules like conditional computations, modularity and the distinction between errors and warnings.[9]

[4] [LBM09], p. 46.
[5] This paragraph is based entirely on [LBM09].
[6] RELAX NG home page, URL: http://www.relaxng.org
[7] Schematron. A language for making assertions on patterns found in xml documents, URL: http://www.schematron.com/
[8] The term 'co-occurrence constraints' here denotes the set of routing, check and warning constraints in Blaise terminology, i.e. the rules.
[9] Ibid., p. 30-32.

Schematron is a rule-based language which means that it is used to describe the rules that a particular xml document must conform to. Schematron supports validation of almost all types of routing and integrity constraints. However, it is not possible to change the contents of a document so the possibility to define default values and/or computed values is not present. Also, using the rules-based approach it is possible but not practical to validate structure and content.

Both of the languages use Xml Path (XPath) as query language for validation of co-occurrence constraints which suggests that the possible validations are fairly equal. To a certain extent they are, but in Xml Schema there are a number of limitations that make it almost impossible to use a modular approach. For example, the XPath expressions are only allowed to query downwards in the hierarchy and it is not possible to declare local variables to hold intermediate results. In Schematron there is no limitation on the axes used in XPath expressions and it is possible to declare local variables that may be used to represent, for example, parameters and auxfields as they are known in Blaise.[10]

Compared to Blaise, the grammar-based approach (Xml Schema) is well suited for validating the structures, i.e. fields, types and blocks while the rules-based approach (Schematron) is better suited for validating the rules.[11]

Finally, in order to produce reports Schematron provides the best possibilities. The generation of human-readable error messages to be used in a production environment is fully supported.[12]

A possible setup

In [LBM09] it is recommended to divide the validation process into three levels:

1. Well-formedness (syntax) is the first level of validation and is equally determined for any xml document. Checking the well-formedness – and character set – of a document is a basic task and requires no knowledge on the purpose of the document and its contents.
2. Structure (semantics) is the next level of validation and verifies that a document conforms to a given schema. Violation of the structure is a sign of errors in the generation of the xml document.
3. Constraints (pragmatics) may be violated because of inconsistencies in the collected data. For example, caused by a routing error in an instrument used for interviewing or caused by an interviewer ignoring a warning from the instrument and carrying on interviewing, thus creating an inconsistent form. [LBM09, p. 47]

The three levels imply three different strategies – and may be supported by different tools. The first level is supported by any xml-aware system; the second may be supported by an Xml Schema definition and the third possibly by a Schematron definition.

The next step is the question of how to achieve this scheme. As it is also concluded:

> "None of the languages, however, are particularly user-friendly. (..) (And) none of them compares to the expressiveness of the Blaise language. Schema languages in xml syntax

[10] The languages are compared in ibid., p. 42-44. The conclusions about the use of XPath in Xml Schema 1.1 also refer to later studies, because there was no implementation available at the time of writing the thesis.
[11] [LBM09] suggests using RELAX NG as the language for validation of structure and content. Partly because RELAX NG and Schematron are standardized in the same suite as complementary tools, partly because RELAX NG provides a representational (human readable) language as well as its xml equivalent.
[12] Ibid., p. 43-44.

> are in their wordiness and unreadability userunfriendly by definition while xml is still crucial for exchanging the schemas across platforms." [Ibid., p.48]

Therefore an obvious solution would be to generate these definitions from a Blaise source.

From Blaise 4.8 generation of an Xml Schema definition is part of the Blaise package which makes this part simple, but even earlier versions made it possible to generate an Xml Schema definition with the help of Cameleon.[13]

Unfortunately, Cameleon is not capable of handling the rules of a data model why it is necessary to use the Blaise API in order to generate a Schematron definition comprising the rules of the data model.

Schematron examples

Looking at a few examples may clarify the job. The first example shows a Schematron definition that reports the number of records in a Blaise xml document:

```
<rule context="/Database">
  <let name="AntalRecords" value="count(./Datarecord)"/>
  <report test="true()">
    Validating <value-of select="$AntalRecords"/> Datarecord-elements.
  </report>
</rule>
```

The **rule** element contains a set of computations, assertions and reports limited by the context which could refer to a block in a Blaise data model, but in this case refers to the document element of a Blaise xml document.

The **let** element declares a local variable identified by its name and a value retrieved using an XPath expression. The let element may be used to represent parameters, auxfields and locals in a block.

The **report** element is used to write a message to an output channel (e.g. a file) if the test expression yields **true** (which it always does in the example above). There is a quite similar **assert** element that will output a message if the test yields **false**. Asserts should be well known to Blaise users as it is the normal behaviour of checks and signals.

Let us look at another example[14]:

```
<rule context="//Job">
  <let name="currAge"
    value=”ancestor::Datarecord/Person[1]/PersonAge”/>
  <assert test=
    "(year-from-date(current-date()) - (number(./YearOfJobStart))
      lt number($currAge)">
    You cannot have started your job before you were born!
  </assert>
</rule>
```

This rule applies to all blocks named Job (i.e. the block Job in all the datarecords in the dataset) , it defines a local variable which retrieves its value from a list of persons in the household. An assert element is used to check that the respondent has not been longer in his current job than his actual age allows.

[13] Examples are mentioned in [LBM07] and [LBM09], appendix B.
[14] From [LBM09], p. 28-29

The validation above may be retrieved from the similar rules in a Blaise data model:

```
FIELDS
   YearOfJobStart "When did you start in your current job?" : 1900..2010
AUXFIELDS
   currAge : 0..120
RULES
   currAge := Person[1].PersonAge
   YearOfJobStart.ASK
   CHECK
      (YEAR(SYSDATE) – YearOfJobStart) < currAge
         "You cannot have started your job before you were born!"
```

Future work

As the small examples above suggest the logic expressed in Schematron is not very far from the logic expressed in Blaise. Therefore, it appears practical to convert Blaise rules into a Schematron definition. Because Schematron is a standardized language, it should be possible to use this solution in order to push further validation towards the supplier of data leading to cleaner data when transferred. Also, Schematron may be used to produce reports on the number of suppressed warnings and thus fulfils most of requirements from the introduction.

Important is the ability to automatically generate Schematron definitions from the Blaise metadata. There still remains some work in this field.

References:

[LBM07]
Leif Bochis Madsen: Blaise and Xml: Experiences Of An Outsourcing Project, in: IBUC 2007 11th International Blaise Users Conference, URL: http://www.blaiseusers.org/2007/papers/Z4%20-%20Blaise%20and%20XML.pdf

[LBM09]
Leif Bochis Madsen: Definition of Validation Rules for Xml Data for Electronic Questionnaires, Unpublished thesis from the IT University of Copenhagen, 2009, temporarily available at URL: http://www.itu.dk/people/lbm/Validation_Rules_Thesis.pdf