Generating Blaise from DDI

Gerrit de Bolster, Statistics Netherlands
July 28, 2013

1. Introduction

Within the Blaise community there is much interest for the Data Documentation Initiative (DDI). Already several efforts have been made to connect Blaise with DDI. Based upon suggestions from ABS and being in the possession of a CAWI instrument generator a working Proof Of Concept (POC) was created to generate operational CAWI instruments from a DDI 3.1 instance. The structure of the DDI 3.1 instances created within this (POC) are mainly based on samples available from the DDI community. Furthermore a DDI 3.1 instance structure was created to define Blaise datamodels (even including a nested Block structure and arrays) including data views and the data itself. A Maniplus was developed to generate the Blaise datamodel from this DDI instance. This paper will give a small introduction to DDI and explain in more detail the structure of the DDI instances and how they link to the Blaise language.

2. What is DDI?

On the web page of the DDI Alliance the following definition is found:
*"The **Data Documentation Initiative (DDI)** is an effort to create an international standard for describing data from the social, behavioural, and economic sciences."*

From Wikipedia, the free encyclopedia:
*"The **Data Documentation Initiative (DDI)** is an international project to create a standard for information describing statistical and social science data (i.e., metadata). Begun in 1995, the effort brings together data professionals from around the world to develop the standard. The DDI specification, written in XML, provides a format for content, exchange, and preservation of information. Version 3.1 of the DDI standard was released in 2009. DDI fills a need related to the challenge of storing and distributing social science metadata, due to the ubiquity of proprietary file formats and no international standard for the design of codebooks."*

Started as a standard for metadata for all kind of documents since version 3 the concept of a data collection instrument was introduced. This makes DDI very interesting to use as a standard in a data collection architecture where Blaise is used as the tool. It also supports concepts that can be used as a repository including codebooks. The physical implementation of DDI is a set of XML-schemas.

3. Why using DDI?

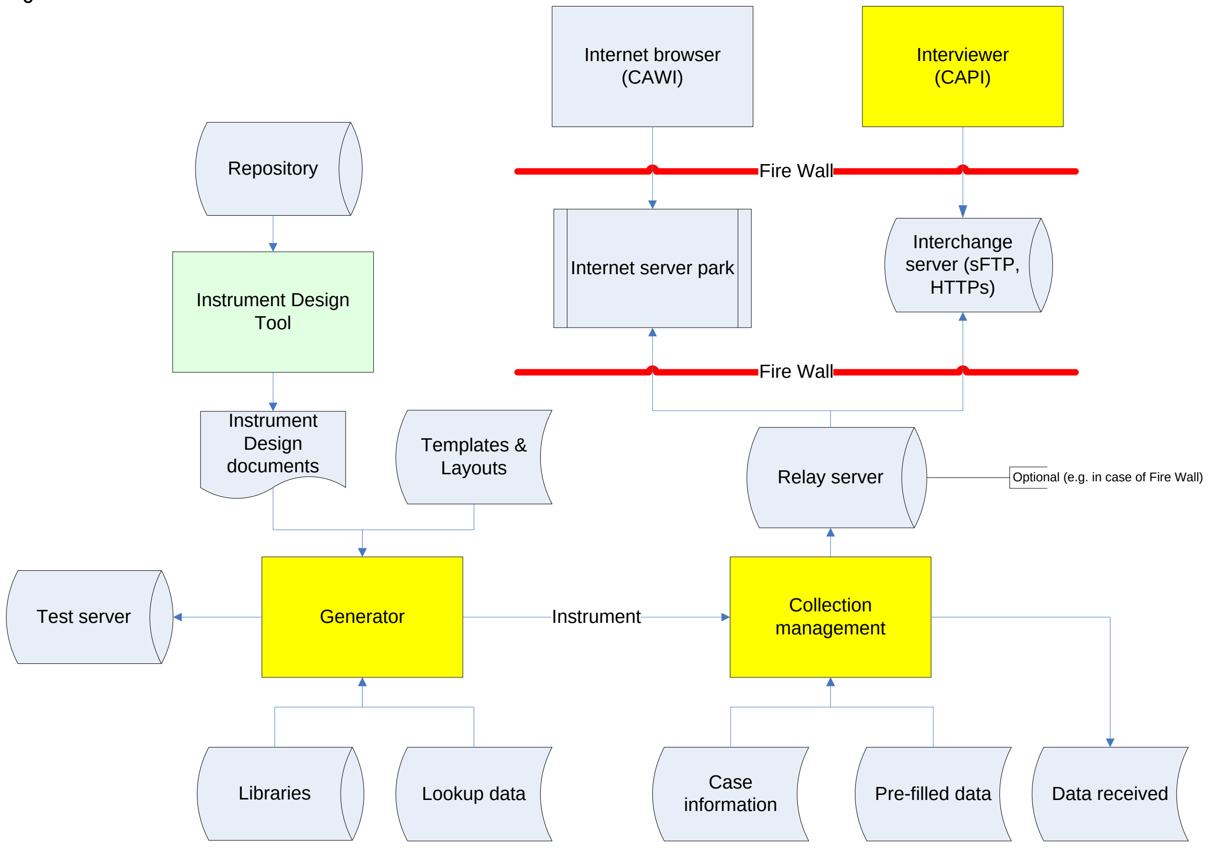
In the Blaise community the involved organisations are already for some time looking for a standard to describe a data collection instrument in a more functional language to communicate between different disciplines in the field of data collection. Although the majority probably use their own standard for documentation a few of them are focussing on DDI. DDI has the advantage to be an open standard expressed in the global supported standard XML. This makes the standard also interesting for commercial software vendors (e.g. Colectica) to create solutions for DDI. For data collection institutions like NSI's this means that not all the software needed to manipulate DDI has to be created by themselves.

The University of Michigan used DDI for the output definition of their MQDS (Michigan Questionnaire Documentation System). ABS (Australia) is already for some time experimenting with DDI as the standard for their QDT (Questionnaire Design Tool). INSEE (France) is also looking for the use of DDI in relation to Blaise data collection instruments. Triggered by these initiatives I started end of last year a hobby project developing a POC for generating Blaise instruments from a DDI version 3.1 (the current version) instance. Therefore this paper focusses on DDI as a meta data source for data collection instruments.

4. The architecture

On first sight DDI looks like a clear path to standardisation of meta data definition of data collection instruments. However, there are different approaches possible. First of all (and some organisations are following this approach) you can try to express the Blaise source language directly in DDI. As Blaise is a meta language on its own the added value of this approach is questionable. The goal of the POC was to create a generic DDI instance for a data collection instrument and to apply a separate generator to create a Blaise instrument. As a BlaiseIS instrument generator was already available the POC was limited to create CAWI instruments. The idea is that in the future any kind of data collection instrument in any kind of programming language can be created from the same generic DDI definition. The future data collection architecture could look like shown in figure 1. Only CAWI and CAPI are presented but CATI and CADI should also be included. This architecture could fit in a “plug and play” initiative which is now trendy among a number of the statistical institutes.

Figure1. Data collection architecture.



In the POC the "Instrument Design Documents" were created as DDI 3.1 instances with a text editor (EditPad Pro 6). The existing generator (BLS-Wizard) was extended to be able to uses these instances for input. The generated CAWI instruments were deployed on the test server.

5. DDI, a very open standard

To learn about the possibilities and restrictions of DDI 3.1 a number of examples were examined at the start of the POC. Most of them were downloaded from the Internet but also a result from MQDS and an example from ABS were taken into account. Looking at these examples it became soon clear that the "standard" DDI was very open. In other words: there were a lot of ways a data collection instrument could be defined in the XML-structure and still verify against the XML-schemas. After several attempts the basic structure as was found in the examples downloaded from DDI Alliance related web sites was taken as they seemed the best fit.

The BLS-Wizard generator supported already an instrument definition in an ASCII-file. Elaborating on that definition and the chosen basic structure a DDI instance was created as well as a Maniplus script to convert the content of the XML-file into the ASCII-file. A Maniplus script to read any kind of XML and convert it into flat ASCII was already available too. Including these Maniplus scripts in the workflow of the generator a working CAWI instrument was generated and installed on the test server. This instrument contained a simple IF-structure as well as some enumerated questions. After that success more (even multilingual) instrument specifications were defined in the DDI structure including download and upload portals. The majority of the specifications fitted into the DDI 3.1 definition without a problem. However, in some cases some definitions were forced beyond their boundaries to be able to store the necessary information. Some generic definitions had to be used too. Were possible an attribute was included which generator (BLS-Wizard) was the target. This attribute is only used for documentation. Finally all the meta specifications could be placed in the DDI XML-tags.

The current meta specifications are partly based on the specifications as used by the BLS-Wizard generator. In their turn these specifications are based partly on the Blaise language, e.g. in the case of formulas in calculations and checks. But formulas can be defined in another open standard called EBNF (Extended Backus–Naur Form ISO/IEC 14977). This will make the DDI instance more generic and less programming language dependent. The generator should then convert the EBNF formulas into the grammar of the used programing language.

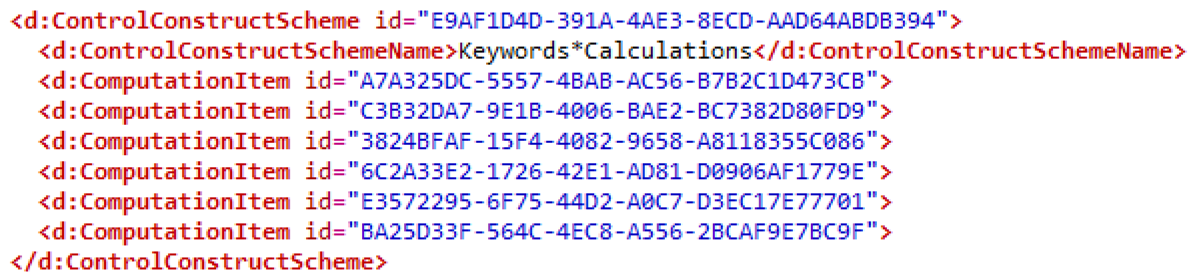
We should take into account that the target language can have influence on the definition of the instrument. As we know the Blaise language is very rich. This can result in functions that are not supported in other programming languages for electronic forms such as XForms or even HTML/JavaScript solutions. These conflicts are to be solved by the generator even as this results in lesser functionality in the resulting instrument. Perhaps adding a target language attribute at more places in the DDI definitions can be helpful.

6. DDI, a technical or a functional standard?

One thing is very clear: an DDI instance is almost unreadable for human beings! As XML-files are normally already hard to read these XML-definitions are moreover infested with internal references (see figure 2). An enumerated question which takes a few lines in the Blaise language consist of many XML-tags referring to other tags in the DDI instance. And these referring tags are not even grouped together but spread throughout the XML-file. Probably this has been done to support the reuse of enumeration specifications within the DDI instance. A small instrument soon occupies hundreds of lines in a DDI instance.

DDI is a meta data standard and therefore functional. However, as it is expressed in XML the implementation is technical. Thanks to this technical implementation it can be used to generate instruments from it using software. In the POC a text editor (EditPad Pro 6) was used to create the DDI instances but for really extended and robust use a more sophisticated instrument design tool is a must. This instrument design tool could be developed by the a commercial software vendor or the Blaise community (like the Questionnaire Design Tool from ABS) or in some kind of consortium involving both parties. Such a tool should be using a repository and (as much as possible) a WYSIWYG user interface. The output of that tool should be the complete DDI instance of a data collection instrument. The Blaise community can then be leading in creating the generator for the Blaise language (versions 4 and 5), the software vendor for other languages. This could be profitable for everyone involved.

Figure 2. A lot of references.



```
<d:ControlConstructScheme id="E9AF1D4D-391A-4AE3-8ECD-AAD64ABDB394">
  <d:ControlConstructSchemeName>Keywords*Calculations</d:ControlConstructSchemeName>
  <d:ComputationItem id="A7A325DC-5557-4BAB-AC56-B7B2C1D473CB">
  <d:ComputationItem id="C3B32DA7-9E1B-4006-BAE2-BC7382D80FD9">
  <d:ComputationItem id="3824BFAF-15F4-4082-9658-A8118355C086">
  <d:ComputationItem id="6C2A33E2-1726-42E1-AD81-D0906AF1779E">
  <d:ComputationItem id="E3572295-6F75-44D2-A0C7-D3EC17E77701">
  <d:ComputationItem id="BA25D33F-564C-4EC8-A556-2BCAF9E7BC9F">
</d:ControlConstructScheme>
```

7. DDI alone is not enough

Not all of the features for a CAWI instrument can or should be defined in the DDI instance. As is shown in figure 1 several data sinks (Templates, Layouts, Libraries and Lookup data) are linked directly to the generator. The BLS-Wizard generator uses the predefined templates for the different type of instruments like questionnaire, election form, download/upload portal etc. Colour schemes and logos are also included in the templates as well as the division of the window in separate panels. The libraries contain predefined procedures e.g. to access the Google Maps API and web services but also to check an e-mail address or telephone number. These all belong to the environment of the generator as they are programming language specific.

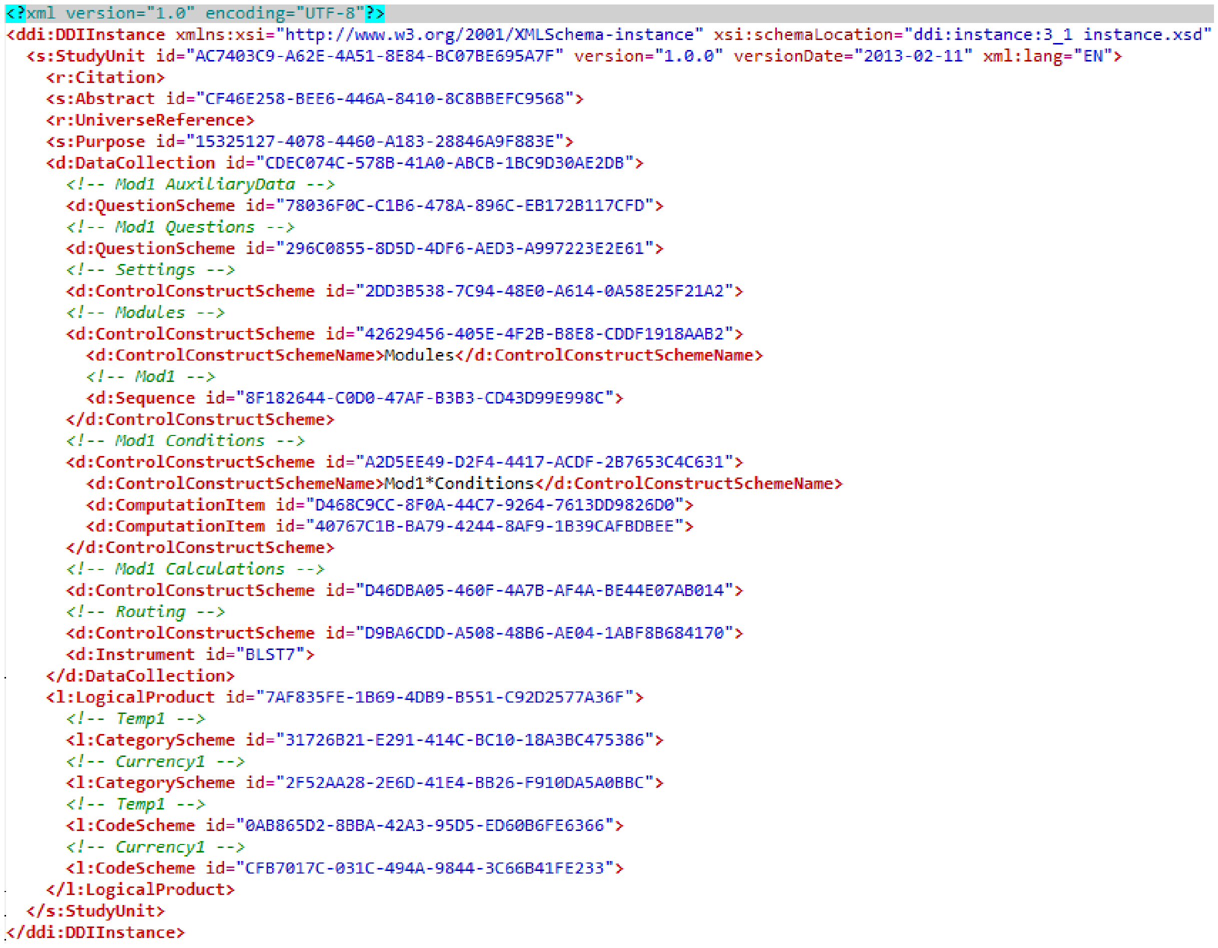
Another major issue is the layout of the questions. In Blaise 4 the layout definitions are stored in the Mode Library, in Blaise 5 in a resource data base. Other programming languages will store the layout definitions in a complete other form like CSS (Cascading Style Sheets). However, in the instrument design definition (the DDI instance) the instrument designer should be able to define which layout must be applied as this can have methodological consequences. In the POC the name of the field pane in the Mode Library was included in the DDI instance. Of course this is programming language dependent and therefore not the wright solution. For the moment it could be improved by defining a generic list of layout definitions with standard names that the generator links to the technical definition in the layout repository (like the field pane in the Mode Library). A final open standard solution must yet be found.

In the POC the lookup data files were already present in Blaise format as the generator was expecting this. In chapter 9 is described that these lookup files can also be defined in DDI and created in Blaise format using a Maniplus script as was developed during the POC.

8. Some technical details

The basic container used in the POC for a DDI instance for data collection is a <StudyUnit>. Besides some minor tags it contains the main tags <DataCollection> and <LogicalProduct>. The <LogicalProduct> tag contains the values and texts for the enumerated and set questions. In DDI 3.1 the values (codes) and texts are stored in separated containers and connected by references. Within the main tag <DataCollection> the questions and the rules are stored. The questions in the container <QuestionScheme> and the rules in a set of linked <ControlConstructScheme> containers. A separate <ControlConstructScheme> container is used for storing the so-called (BLS-Wizard language) settings of the CAWI instrument. These settings cover amongst others languages and libraries used, navigation buttons and external file references.

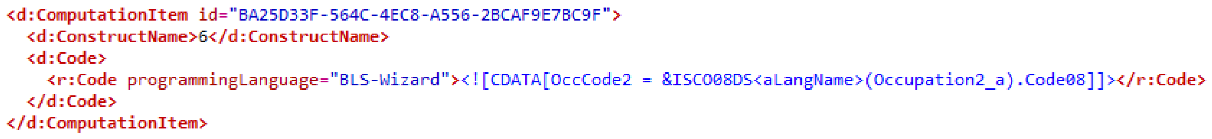
Figure 3. The basic structure of a DDI instance for a data collection instrument.



```
<?xml version="1.0" encoding="UTF-8"?>
<ddi:DDIInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ddi:instance:3_1 instance.xsd"
  <s:StudyUnit id="AC7403C9-A62E-4A51-8E84-BC07BE695A7F" version="1.0.0" versionDate="2013-02-11" xml:lang="EN">
    <r:Citation>
    <s:Abstract id="CF46E258-BEE6-446A-8410-8C8BBEFC9568">
    <r:UniverseReference>
    <s:Purpose id="15325127-4078-4460-A183-28846A9F883E">
    <d:DataCollection id="CDEC074C-578B-41A0-ABCB-1BC9D30AE2DB">
      <!-- Mod1 AuxiliaryData -->
      <d:QuestionScheme id="78036F0C-C1B6-478A-896C-EB172B117CFD">
      <!-- Mod1 Questions -->
      <d:QuestionScheme id="296C0855-8D5D-4DF6-AED3-A997223E2E61">
      <!-- Settings -->
      <d:ControlConstructScheme id="2DD3B538-7C94-48E0-A614-0A58E25F21A2">
      <!-- Modules -->
      <d:ControlConstructScheme id="42629456-405E-4F2B-B8E8-CDDF1918AAB2">
        <d:ControlConstructSchemeName>Modules</d:ControlConstructSchemeName>
        <!-- Mod1 -->
        <d:Sequence id="8F182644-C0D0-47AF-B3B3-CD43D99E998C">
      </d:ControlConstructScheme>
      <!-- Mod1 Conditions -->
      <d:ControlConstructScheme id="A2D5EE49-D2F4-4417-ACDF-2B7653C4C631">
        <d:ControlConstructSchemeName>Mod1*Conditions</d:ControlConstructSchemeName>
        <d:ComputationItem id="D468C9CC-8F0A-44C7-9264-7613DD9826D0">
        <d:ComputationItem id="40767C1B-BA79-4244-8AF9-1B39CAFBDBEE">
      </d:ControlConstructScheme>
      <!-- Mod1 Calculations -->
      <d:ControlConstructScheme id="D46DBA05-460F-4A7B-AF4A-BE44E07AB014">
      <!-- Routing -->
      <d:ControlConstructScheme id="D9BA6CDD-A508-48B6-AE04-1ABF8B684170">
      <d:Instrument id="BLST7">
    </d:DataCollection>
    <l:LogicalProduct id="7AF835FE-1B69-4DB9-B551-C92D2577A36F">
      <!-- Temp1 -->
      <l:CategoryScheme id="31726B21-E291-414C-BC10-18A3BC475386">
      <!-- Currency1 -->
      <l:CategoryScheme id="2F52AA28-2E6D-41E4-BB26-F910DA5A0BBC">
      <!-- Temp1 -->
      <l:CodeScheme id="0AB865D2-8BBA-42A3-95D5-ED60B6FE6366">
      <!-- Currency1 -->
      <l:CodeScheme id="CFB7017C-031C-494A-9844-3C66B41FE233">
    </l:LogicalProduct>
  </s:StudyUnit>
</ddi:DDIInstance>
```

The common CDATA-container was applied in most of the string type tags to avoid conversion of XML unfriendly signs like “<” and ”>”. Texts were defined in UTF-8.

Figure 4. The use of a CDATA container.



```
<d:ComputationItem id="BA25D33F-564C-4EC8-A556-2BCAF9E7BC9F">
  <d:ConstructName>6</d:ConstructName>
  <d:Code>
    <r:Code programmingLanguage="BLS-Wizard"><![CDATA[OccCode2 = &ISCO08DS<aLangName>(Occupation2_a).Code08]]></r:Code>
  </d:Code>
</d:ComputationItem>
```

9. Defining Externals in DDI

In questionnaires external lookup files are often used. In a Blaise instrument these externals are Blaise files based upon Blaise data models. Although a Blaise instrument is also based on a data model there are differences. As a data collection instrument is mainly meant for reporting data the purpose of an external is to retrieve data to be used in a data collection instrument or script. Question text is therefore normally not present as it is not used but a filled data file is a requirement. Even though in the Blaise language the same type of meta data is used (a data model) in the POC a different DDI definition was applied.
The main container of the definition of the DDI instance is in this case a <ResourcePackage>. Within this main container a number of sub containers can occur. These containers hold the definition of BLOCKS and the data. One container is holding the definition of the data model on the highest level.

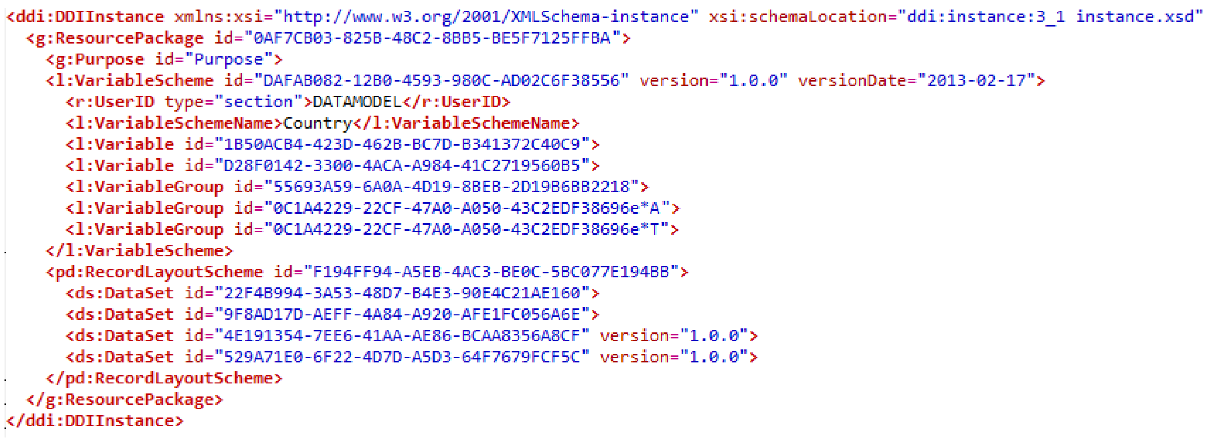
Within a BLOCK another BLOCK can be included by using the tag <VariableSchemeReference>. Inside the BLOCK a concatenated reference is needed to put in on the route with a local name. A tag <VariableReference> is referring to a tag <VariableGroup> which contains the reference to the included BLOCK and its local name. A FIELD is defined by the tag <Variable>.

The data records are stored in the sub container <RecordLayoutScheme>. Each record is stored within this container with the tag <DataSet>.

It is obvious that defining even simple externals in DDI is complex. It becomes even more complex as enumeration and set fields are involved. In that case one or more sub containers <CategoryScheme> and <CodeScheme> must be included too.

Also in this case an user friendly tool is an absolute prerequisite to be able to define an external with or without data. In the POC a separate Maniplus setup was created to translate these DDI definitions to a Blaise data model and related data file.

Figure 5. The basic structure of a DDI instance for an external with data.



```
<ddi:DDIInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ddi:instance:3_1 instance.xsd"
  <g:ResourcePackage id="0AF7CB03-825B-48C2-8BB5-BE5F7125FFBA">
    <g:Purpose id="Purpose">
    <l:VariableScheme id="DAFAB082-12B0-4593-980C-AD02C6F38556" version="1.0.0" versionDate="2013-02-17">
      <r:UserID type="section">DATAMODEL</r:UserID>
      <l:VariableSchemeName>Country</l:VariableSchemeName>
      <l:Variable id="1B50ACB4-423D-462B-BC7D-B341372C40C9">
      <l:Variable id="D28F0142-3300-4ACA-A984-41C2719560B5">
      <l:VariableGroup id="55693A59-6A0A-4D19-8BEB-2D19B6BB2218">
      <l:VariableGroup id="0C1A4229-22CF-47A0-A050-43C2EDF38696e*A">
      <l:VariableGroup id="0C1A4229-22CF-47A0-A050-43C2EDF38696e*T">
    </l:VariableScheme>
    <pd:RecordLayoutScheme id="F194FF94-A5EB-4AC3-BE0C-5BC077E194BB">
      <ds:DataSet id="22F4B994-3A53-48D7-B4E3-90E4C21AE160">
      <ds:DataSet id="9F8AD17D-AEFF-4A84-A920-AFE1FC056A6E">
      <ds:DataSet id="4E191354-7EE6-41AA-AE86-BCAA8356A8CF" version="1.0.0">
      <ds:DataSet id="529A71E0-6F22-4D7D-A5D3-64F7679FCF5C" version="1.0.0">
    </pd:RecordLayoutScheme>
  </g:ResourcePackage>
</ddi:DDIInstance>
```

10. Compatibility with newer versions: a must

In the past the Dutch government initiated several projects to create a standard for the data collection at enterprises by the different governmental organisations including Statistics Netherlands. The main issue was to involve the vendors of administration software and accountant organisations. One of the first attempts appeared to be successful. Software vendors included at their own costs the standard in their administrative software. Then, before it was even taken into production, a next project changed the definitions making the already developed software useless. From that moment on for a lot of years software vendors were not so willing anymore to implement any governmental standard in their software unless someone else was paying for it. A basic demand for the survival of a standard is compatibility between versions.

At this moment DDI version 3.1 is still the current version. A DDI 3.2 version is still under development but the provisional XML-schemas are already available. When a simple DDI 3.1 instance was taken from the POC and verified against the XML-schemas of version 3.2 not less than 365 errors occurred! Such an incompatibility between 2 MINOR versions of a standard is an attempt to commit suicide.

11. Conclusion

The lesson that was learned from this exercise is that, if we (the Blaise community) really want a standard instrument definition in DDI some kind of manual should be produced describing how to apply DDI for this purpose. The DDI instances created in this POC can be used as a start. We should also feed the DDI Alliance with proposals for improvement of existing definitions and besides that filling in some gaps. As some members of the Blaise community already are involved in the DDI Alliance this should not be a problem.

The use of DDI for instrument definitions will not be successful without the necessary user friendly tools. A text editor (as EditPad Pro 6) is totally insufficient for the job.

Subsequent DDI versions of DDI should be compatible as much as possible. Only then organisations are willing to invest in their systems to adopt this powerful standard. The DDI Alliance should be very aware of that.

12. References

1. The home page of the DDI Alliance:
   http://www.ddialliance.org

2. The Wikipedia link:
   http://en.wikipedia.org/wiki/Data_Documentation_Initiative

3. Sample of DDI instances created in the POC:
   https://www.dropbox.com/s/3c6hx2oq1ax4uyu/POC%20samples_DDI.zip