WHITE PAPER

SCHEMAS FOR A XML STANDARD

A CASE STUDY

ERIK SIEGEL
V1.0 - 15-DEC-2003

0. TABLE OF CONTENTS

0.1. USAGE AND COPYRIGHTS

0.2. ACCOMPANYING SOFTWARE

On my site (`www.siegel-ict.nl`) you can find the Designer output to Excel software described in paragraph 3.2.1 on page 7. It is provided in two formats: As an installation executable and as a zip file containing all the sources. An example Designer output file is included so you can try it.
This software is provided “as is”: I will not provide any support for it. On the other side, you can try it, study it, modify it and do with it whatever you like.
I would appreciate it if you told me about using the software or the source code. You can do this by simply sending me an e-mail about it (`erik@siegel-ict.nl`). Also, if you have any comments (good or bad), other ideas or complementary information, please send me an e-mail. I am looking forward to your reactions.

1. INTRODUCTION

When companies need to communicate a lot to get work done, they will sooner or later start thinking about automating this. In most cases this is a complicated and time-consuming process. You need to agree about the functionality, format and content of the messages, the technical details for actually sending and receiving them, the legal side of it, and much, much more. Setting up a business-to-business (B2B) communication standard isn't easy.

However, the reward can be enormous: Increased volumes of business transactions, decreased number of errors, etc. Sometimes because of the automation whole new markets open up.

Nor so long ago, this kind of B2B communication was done using the EDI standards. For several reasons, with a few notable exceptions, EDI never really made it as an important standard. Nowadays, these kinds of processes are automated using XML technology.

However, whatever technology is used to implement things, the groundwork will stay the same: Somebody must analyze the communication processes, create a common data model, design messages and message flows, etc. After this is done and agreed upon, the technical people come in and design the actual messages, the infrastructure and all the other necessary technical bits and pieces to make it work.

This is a case study about a small but significant step in a XML standard creation process: The analysts were ready, we knew what we wanted to do and there was a huge formal data and communication model. Now, from this, we needed to design the actual messages and create XML schemas. Maintenance was an important issue, because this was only version one and things would evolve.

Since the standard was quite large, hand crafting all message schemas was definitely not a good idea: It would have been next to impossible to get all the details, like field types and sizes, completely right. But inconsistencies between standard and actual messages were not acceptable…

This whitepaper explains how we automated the conversion from data/communication model into the actual XML message schemas. It involves an interesting mix of technologies, resulting in not only the schemas themselves but also a full set of documentation, all generated and therefore easily maintained.

The reader is expected to have a basic understanding of XML and what an XML schema is. Detailed understanding of the XML schema or other standards is not necessary, except for some minor details in paragraph 3.3.

2. THE SETTING

This chapter explains a little bit about the setting the standardization effort took place in.

If somebody wants to place an advertisement in a newspaper or magazine, lots of information needs to be transferred. Some examples:

- What newspapers/magazines should it be in and in what section?
- Is this only once or a recurring placement? If recurring, what frequency/which dates?
- What kind of advertisement is this?
- What is the format of the advertisement: Color (if so, what color scheme), black and white, what size, etc.
- Is this advertisement part of an overall contract so you get a discount?
- What is the status for this advertisement: Option only, reservation, actual placement, etc.

In most cases all this information is transferred using phone, fax or e-mail. As you can imagine, the result will not always be correct and in most cases time and labor consuming and therefore expensive.

A number of big players on this market in the Netherlands decided to do something about it. In 2002 they started a standardization project called Ad\Venture. Most of the work was done in 2002/2003.

The main idea was to standardize all the administrative communication around advertising. This does not only include the communication described above, but also invoicing, complaint handling and contract handling.

Ad\Venture was a huge and complex undertaking that was complicated even more because little standardization existed so far. Most companies involved did things their own way, resulting in incompatible business processes and back-end systems. They all looked alike but were very different in the details that counted.

The way Ad\Venture solved this incompatibility problem was to create an overall virtual data and communication model. It was the common denominator of the models of the parties involved. You must view it as the data/communication model for an overall media company that does not (yet) exist.

Of course, this wasn't as simple as it sounds. Companies had made conflicting decisions on how to handle things. Simple field formats, like for a name or an address, did not match. Every company had their own algorithm for calculating the price for an advertisement, etc. etc.

Luckily, all parties were very committed in making this project a success. So after long debates about lots of details a virtual model was decided upon. It consisted of:

- A large functional data model (also called Entity-Relationship Diagram, ERD) designed with and stored in Oracle Designer 2000.
- State diagrams for all communication processes
- Detailed documentation

With this as input, the technical people (like me) came in. My job was to design XML schemas for all the messages. Other people worked on the actual communication, implementing a message hub and communication protocols.

The rest of this white paper explains how the schema creation was designed and structured. Because the details of the standard are still under non-disclosure, I cannot convey too much about it. However, I think the way the schemas were created is interesting enough on its own and has a much wider applicability than this project alone.

3. CREATING THE XML SCHEMAS

As explained in the previous chapter, the starting point for the XML schema creation consisted of:

- A functional data model (ERD) stored in Oracle Designer 2000.
- An informal model of the communication, consisting of some state diagrams and written documentation. A lot of information about message content was still inside people's heads.

This chapter explains how, with this as input, the actual XML schemas were created.

3.1. CONSTRAINTS AND DESIGN DECISIONS

In creating the XML schemas, I had some important constraints:

- **Repeatability**: Since the standard was very likely to evolve, the whole process must be repeatable.
- **Traceability and understandability**: XML Schemas are not easy to read and understand. So the whole process, from inputs to schemas, must be traceable: A non-XML expert should, with some effort, be able to understand what was happening. In this light, the end results must be understandable: Why are certain constructions there, why is this field here, where is the information, etc.
- **Documented**: Linked to the constraint above was that the resulting schemas should have human readable documentation attached. Luckily, most of the information for the documentation was already in the data model.
- **Cooperation with non-XML experts**: To create the schemas some extra input was needed. This input had to be created by an analyst that had some knowledge of XML but preferred working in a more user friendly environment than a text editor typing XML tags.
- **Not real-time**: Also important to keep in kind is that the whole schema creation process was not real-time and not, within reason, time bound. So performance of the software was not an issue.
- **Time and budget constraints**: Of course (I haven't been in a project that didn't), the project had tight time and budget constraints. So an all-out software development effort was out of the question. To save time and effort, I had to use tools that were known and proven to me.

With all this in mind, I made the following design decisions:

- **Excel as intermediate format**: XML is ok for the technical users and of course, as a schema, it is the end product, but in this case it was not useable for extra input information or in-between results that needed to be manually checked.

  Of course, we could have used XML editors that made the input of data less cumbersome. Also, we could have created XSLT transformations that converted the XML files to more readable HTML pages for checking purposes. However, I decided not to do this because it would simply take me too much time.

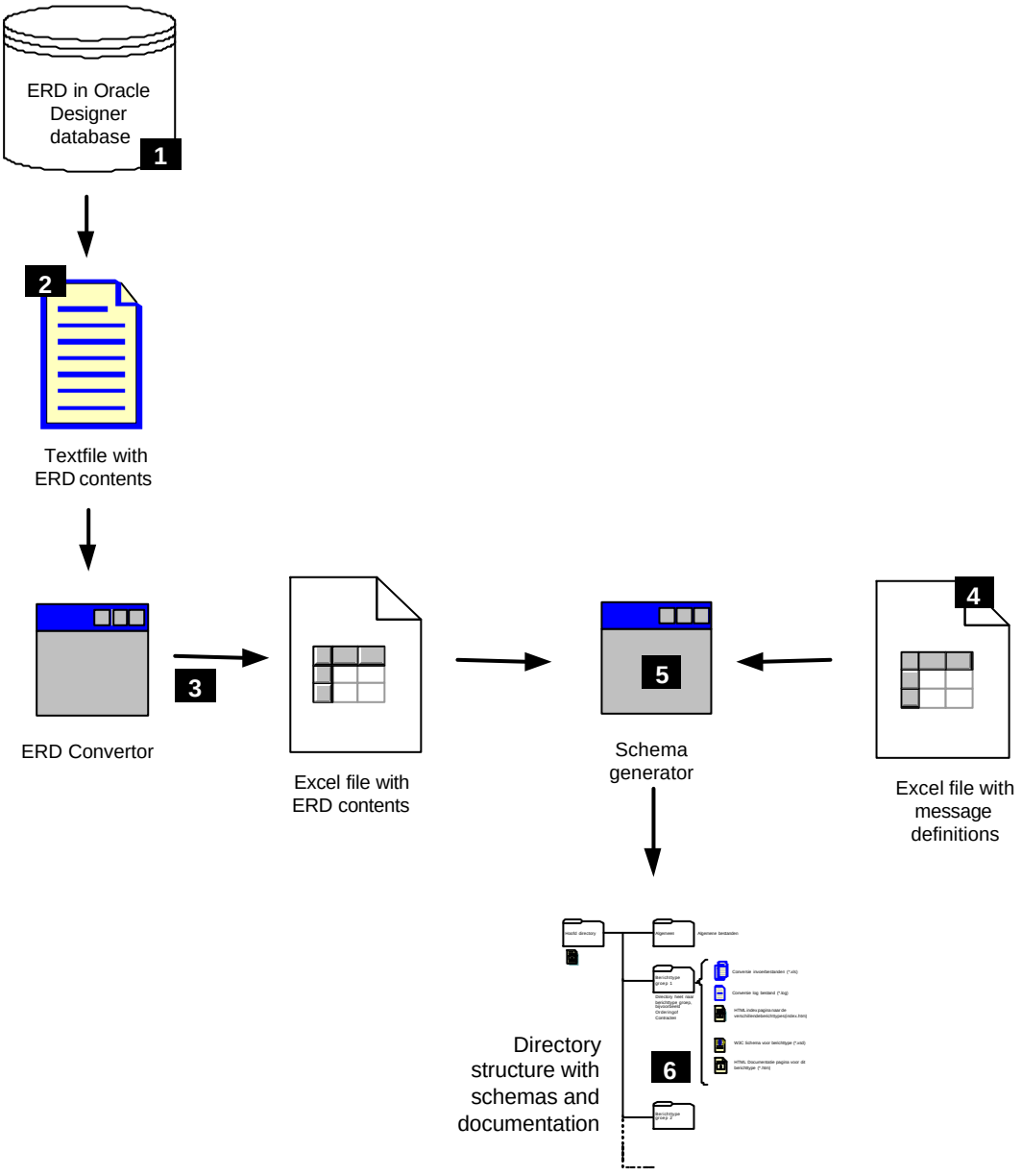
  After some initial tinkering I decided to use Excel as an intermediate format. It is very well structured, easy to generate and read (with COM enabled programming tools) and everybody knows how to handle it.
- **Use Visual Basic**: When you want to use Excel, you need a programming tool that can handle COM/ActiveX. From previous projects I have a lot of experience with Visual Basic (V6 EE), so I decided to use this, saving me the learning curve for a new tool.

- **Use XML Spy**: In working with XML, I like to work with a tool that helps me with this. I had some experience with XML Spy and decided to use it here also. Some tasks I used it for were:
  - Parts of the schemas were fixed and I needed a tool to design these. Also, I needed something to create example schemas before I could design the schema generator well enough. XML Spy has a graphical schema designer and generates high quality output.
  - I needed a way to easily validate the outcome of my schema generator: Was what I had produced a valid schema? XML Spy can do this for you.

3.2. THE GENERATION PROCESS

Given the inputs and the constraints/design decisions I had to create the XML schemas and the accompanying documentation. It did not take long for me to realize that creating them by hand was not the way to go: Too many schemas and almost impossible to get all the zillion of details right. So, creating a generator was the way to go.

In the diagram below is a simplified representation what the generator finally looked like:



1 An important starting point was the data model (ERD) stored in Oracle Designer 2000.

2 From this an export was made.

3 With a VB program, this ERD export was converted into an Excel file.

Steps 1 to 3 is explained in more detail in paragraph 3.2.1 on page 7.

4 The necessary information about message content was recorded in another Excel file.

Step 4 is explained ion more detail in paragraph 3.2.2 on page 10.

5 A generator program took both Excel files together and generated the end result

The generator program is described in paragraph 3.2.3 on page 12.

6 The end result is a directory structure, containing the schemas and HTML documentation.

Details about the end result can be found in paragraph 3.3 on page 13.

3.2.1. USING THE DATA MODEL

One of the inputs for the generation process was a huge functional data model (ERD) designed and stored in Oracle Designer 2000. This data model contained information about:

- The various entities (tables) that made up the data model.
- The attributes (fields) for these entities.
- The relations between the entities.

We tried several ways of exporting data from Designer, but none fitted our needs. Especially the version control of Designer got in the way. At last we decided not to use any formal export format, but an ASCII dump of a report that you would normally use for print.[1]

Here is an example of the definition of an entity in such a report:

```
12 June      2003                                                    Page  4  of 92
                                Entity Definition

     Container:  Adventure     Version:  1.12

     Entity
     ------
     Name           :  ADRES
     Short Name     :  ADR
     Plural Name    :  ADRESSEN
     Description
     Plaats waar iemand of iets gevestigd is of bereikbaar is.

     Attributes              * - Attributes in primary unique identifier
     ----------
          Name                  Domain               Opt Format     Length   Scale
          LAND                  ALFANUM050           N    VARCHAR2      50
          PLAATS                ALFANUM050           N    VARCHAR2      50
     *    POSTCODE              POSTCODE             N    CHAR           7
          STRAAT                ALFANUM050           N    VARCHAR2      50
     *    HUISNUMMER            ALFANUM010           Y    VARCHAR2      10
     *    TOEVOEGSEL            ALFANUM010           Y    VARCHAR2      10

     Relationships           * - Relationships in primary unique identifier
     --------------
     Each Occurrence Of This Entity :
        MAY BE      kent one or more                        PARTIJ ADRESSEN
```

[1] This report can be created with an Oracle Designer tool called “Repository reports”, generating an “Entity definition report”. Use the settings “file print”, “character mode” and “wide”.

```
Unique Identifiers
--------------------
Name                    Primary? Attrib.Name    Relationship       To Entity Name
----                    -------- ------------   ------------       --------------
PK_ADR                  Yes      HUISNUMMER
                                 POSTCODE
                                 TOEVOEGSEL
```

As you can see in the upper right-hand corner, there were 92 pages of this kind of detailed information. To be able to work with this, it needed to be converted it into something more suitable for direct processing.

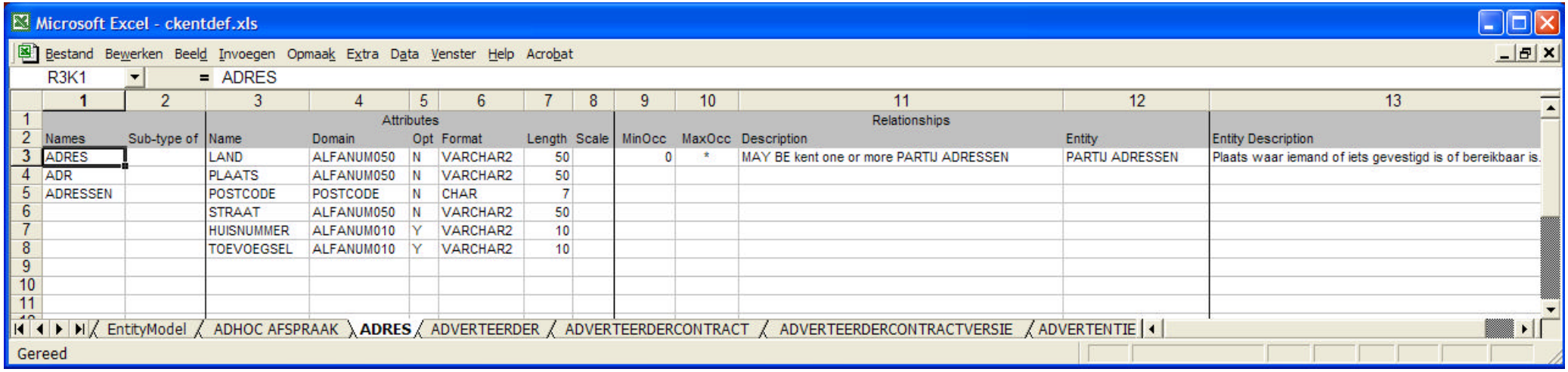
Automatic interpretation of this Oracle Designer output turned out not to be trivial. Some of the problems I encountered were:

- Large multi-line page header/footer sections that interspersed the text, sometimes in the middle of a data table. For some reason, not all header/footer sections had the same number of lines or were formatted the same.
- Text that printed well but because of creative use of CR/LF codes, split into lines in a very strange way. The convertor needed to read ahead or come back on decisions made earlier.
- Column settings in data tables (like for instance the attributes table in the example above) caused data to be wrapped to the next line. However, column data can also contain spaces. See the example below and guess the name of the attribute: `AFSPRAAKNUMMERME` or `AFSPRAAKNUMMER ME`? It took some experimenting with the conversion software to get it right most of the time but nothing is guaranteed.

```
Name                    Primary? Attrib.Name
----                    -------- ------------
PK_PRA                  Yes      AFSPRAAKNUMMER
                                 ME
```

Options for the conversion output format that came to mind first were an Access database or, of course, an XML file. However, since the conversion process was not guaranteed to be 100% right, it became more and more important to be able to view the conversion results directly and also to alter or amend them easily. So I decided to put the results in an Excel file.

For every entity definition a new sheet is created that hold the most important data. On the next page is an example of such a sheet. It is the conversion result of the `ADRES` entity in the example above.



	1	2	3	4	5	6	7	8	9	10	11	12	13
1			Attributes						Relationships				
2	Names	Sub-type of	Name	Domain	Opt	Format	Length	Scale	MinOcc	MaxOcc	Description	Entity	Entity Description
3	ADRES		LAND	ALFANUM050	N	VARCHAR2	50		0	*	MAY BE kent one or more PARTIJ ADRESSEN	PARTIJ ADRESSEN	Plaats waar iemand of iets gevestigd is of bereikbaar is.
4	ADR		PLAATS	ALFANUM050	N	VARCHAR2	50						
5	ADRESSEN		POSTCODE	POSTCODE	N	CHAR	7						
6			STRAAT	ALFANUM050	N	VARCHAR2	50						
7			HUISNUMMER	ALFANUM010	Y	VARCHAR2	10						
8			TOEVOEGSEL	ALFANUM010	Y	VARCHAR2	10						
9													
10													
11													

Example of a generated Excel worksheet holding the information for an entity from the original ERD.

3.2.2. DEFINING THE MESSAGE CONTENT

Besides the data model, we needed a way to define the actual message content. We choose to use Excel for this also.

Using Excel enabled us to create the definitions fast without the need for custom data input software. After we decided on the format for the Excel sheets, the analyst was able to define the messages quickly. They were send by e-mail to me and I used them to generate the actual schemas and documentation.

For every group of messages (all messages in a group have the same schema) we created an Excel sheet. There is an example of such a sheet on the next page. The messages this sheet defines are about complaint handling.

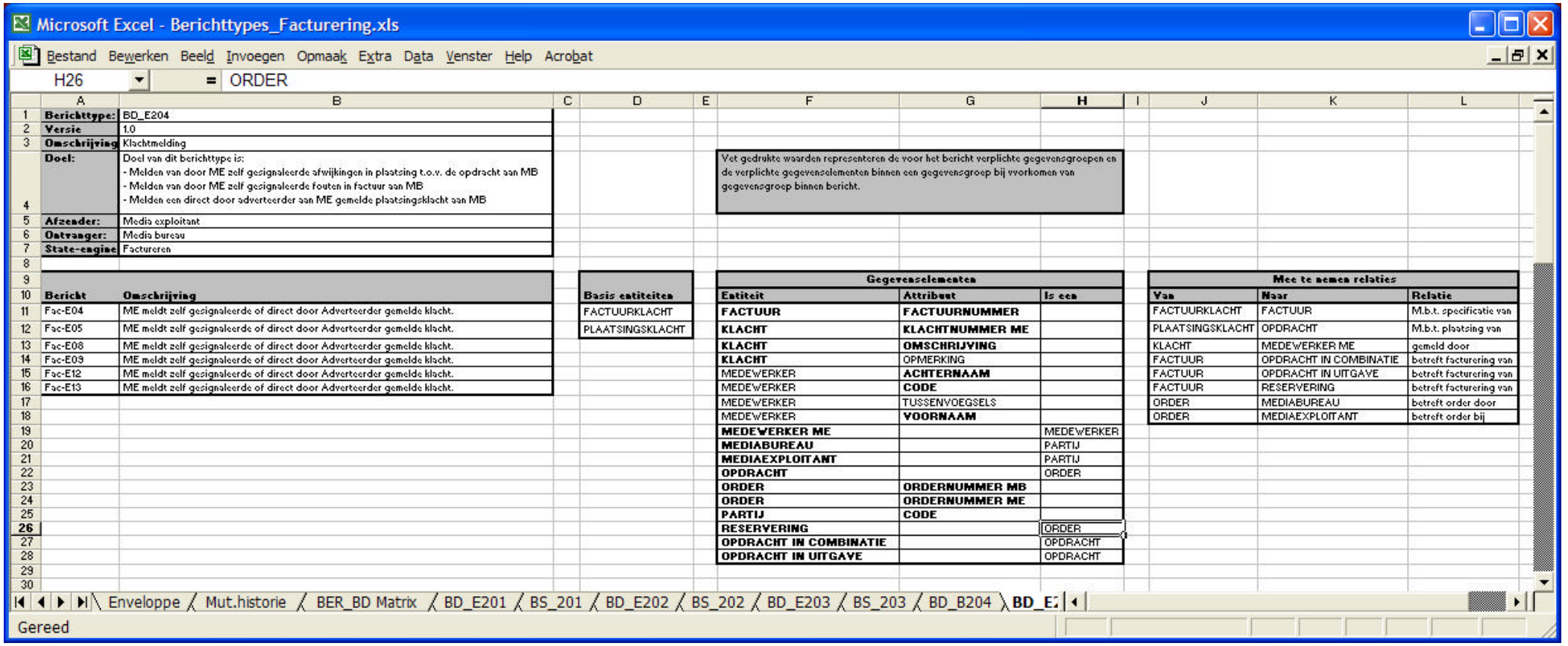
If you look at the example, you see several blocks of data:

- The two blocks on the left are for documentation purposes only. They contain important information about what the messages are for and their code numbers. This information returns in the documentation that is generated with the schemas.
- The block labeled "Basis entiteiten" ("Base entities") contains the starting entities for the schema generation. The resulting schema in the example will start with a choice element between a `FACTUURKLACHT` and a `PLAATSINGSKLACHT`. Both these names exist as an entity in the data model.
- The block to the right labeled "Mee te nemen relaties" ("Relations to use") defines what relations should be used in creating the schema.

  Take for instance the first line of this block: A `FACTUURKLACHT` (invoice complaint) is always about a `FACTUUR` (invoice). The relation between `FACTUURKLACHT` and `FACTUUR` is defined in the data model:

```
Relationships           * - Relationships in primary unique identifier
--------------
Each Occurrence Of This Entity :
    MUST BE     betreft one and only one                    FACTUUR
```

  Since we are defining a message about an invoice complaint, this message must identify the invoice the complaint is about. So this relation is important and should be part of the resulting schema. The result is that some attributes of `FACTUUR` will be part of the schema for these messages.
- The block labeled "Gegevenselementen" ("Data elements" or "Attributes") contains the name of the attributes/fields that must be in the schema. No further definition of attributes is necessary here (type, size, etc.), because the convertor can look up the details in the data model.



Microsoft Excel - Berichttypes_Facturering.xls

Bestand Bewerken Beeld Invoegen Opmaak Extra Data Venster Help Acrobat

H26 = ORDER

	A	B
1	**Berichttype:**	BD_E204
2	**Versie**	1.0
3	**Omschrijving**	Klachtmelding
4	**Doel:**	Doel van dit berichttype is: - Melden van door ME zelf gesignaleerde afwijkingen in plaatsing t.o.v. de opdracht aan MB - Melden van door ME zelf gesignaleerde fouten in factuur aan MB - Melden een direct door adverteerder aan ME gemelde plaatsingsklacht aan MB
5	**Afzender:**	Media exploitant
6	**Ontvanger:**	Media bureau
7	**State-engine**	Factureren

Vet gedrukte waarden representeren de voor het bericht verplichte gegevensgroepen en de verplichte gegevenselementen binnen een gegevensgroep bij voorkomen van gegevensgroep binnen bericht.

Bericht	Omschrijving
Fac-E04	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.
Fac-E05	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.
Fac-E08	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.
Fac-E09	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.
Fac-E12	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.
Fac-E13	ME meldt zelf gesignaleerde of direct door Adverteerder gemelde klacht.

Basis entiteiten
FACTUURKLACHT
PLAATSINGSKLACHT

Gegevenselementen		
Entiteit	**Attribuut**	**Is een**
FACTUUR	**FACTUURNUMMER**	
KLACHT	**KLACHTNUMMER ME**	
KLACHT	**OMSCHRIJVING**	
KLACHT	OPMERKING	
MEDEWERKER	**ACHTERNAAM**	
MEDEWERKER	**CODE**	
MEDEWERKER	TUSSENVOEGSELS	
MEDEWERKER	**VOORNAAM**	
MEDEWERKER ME		MEDEWERKER
MEDIABUREAU		PARTIJ
MEDIAEXPLOITANT		PARTIJ
OPDRACHT		ORDER
ORDER	**ORDERNUMMER MB**	
ORDER	**ORDERNUMMER ME**	
PARTIJ	**CODE**	
RESERVERING		ORDER
OPDRACHT IN COMBINATIE		OPDRACHT
OPDRACHT IN UITGAVE		OPDRACHT

Mee te nemen relaties		
Van	**Naar**	**Relatie**
FACTUURKLACHT	FACTUUR	M.b.t. specificatie van
PLAATSINGSKLACHT	OPDRACHT	M.b.t. plaatsing van
KLACHT	MEDEWERKER ME	gemeld door
FACTUUR	OPDRACHT IN COMBINATIE	betreft facturering van
FACTUUR	OPDRACHT IN UITGAVE	betreft facturering van
FACTUUR	RESERVERING	betreft facturering van
ORDER	MEDIABUREAU	betreft order door
ORDER	MEDIAEXPLOITANT	betreft order bij

Enveloppe / Mut.historie / BER_BD Matrix / BD_E201 / BS_201 / BD_E202 / BS_202 / BD_E203 / BS_203 / BD_B204 / BD_E

Gereed

Example of an Excel sheet holding a message definition.

3.2.3. PUTTING IT ALL TOGETHER

Another Visual Basic program was written to put it all together and to create the requested schemas and documentation. It performed the following actions:

- It opened both the Excel files and checked if they had the expected format.
- It cleared the output directory.
- It copied some fixed information to the output:
  - A HTML page containing general documentation
  - Schema files that were common to all the generated schemas. The generated schemas included these files.
  - Some other documentation files.
- It took a worksheet in the message content definition file and generated the schema and documentation for this. Every time this definition referenced something from the data model (relations between entities, attributes, etc.), it looked in the data model Excel file for the details.
- After the actual generation it checked whether it had used all the information on the message content sheet. If not, it generated an error because in most cases this meant some mistake was made.
- It took the generated schema and validated it using an XML Spy library function. This was necessary to detect possible compatibility problems with fixed schemas that were included.
- After all schemas were generated it created index HTML pages so you could navigate through the generated documentation more easily.

3.3. THE END RESULT

The end result was a directory tree containing the schemas and documentation.

To explain a little more about the structure of the generated schemas, let me first introduce an example message:

```
<?xml version="1.0" encoding="UTF-8"?>

<ADVENTUREBERICHT
  BerichtType="BD_B001"
  Versie="1.0"
  Bericht="Opt-B01">

  <!-- Start of envelope information -->
  <AFZENDER>
    <ADVENTURENR>12345678</ADVENTURENR>
    <NAAM>Jan Jansen</NAAM>
  </AFZENDER>
  <ONTVANGER>
    <ADVENTURENR>87654321</ADVENTURENR>
    <NAAM>Piet Pietersen</NAAM>
  </ONTVANGER>
  <VERZENDTIJDSTIP>2003-06-02T12:00:00</VERZENDTIJDSTIP>
  <ONDERWERP>Aanvragen optie voor ... </ONDERWERP>
  <OPMERKINGEN>Denk erom dat ... </OPMERKINGEN>
  <!-- End of envelope information -->

  <!-- Start of actual message -->
  <BERICHT>
    <OPTIE-IN-COMBINATIE>
      <OPTIE>
        <VERVALDATUM>2004-05-21</VERVALDATUM>
        <ORDER>

            ....

        </ORDER>
      </OPTIE>
    </OPTIE-IN-COMBINATIE>
  </BERICHT>
  <!-- End of actual message -->

</ADVENTUREBERICHT>
```

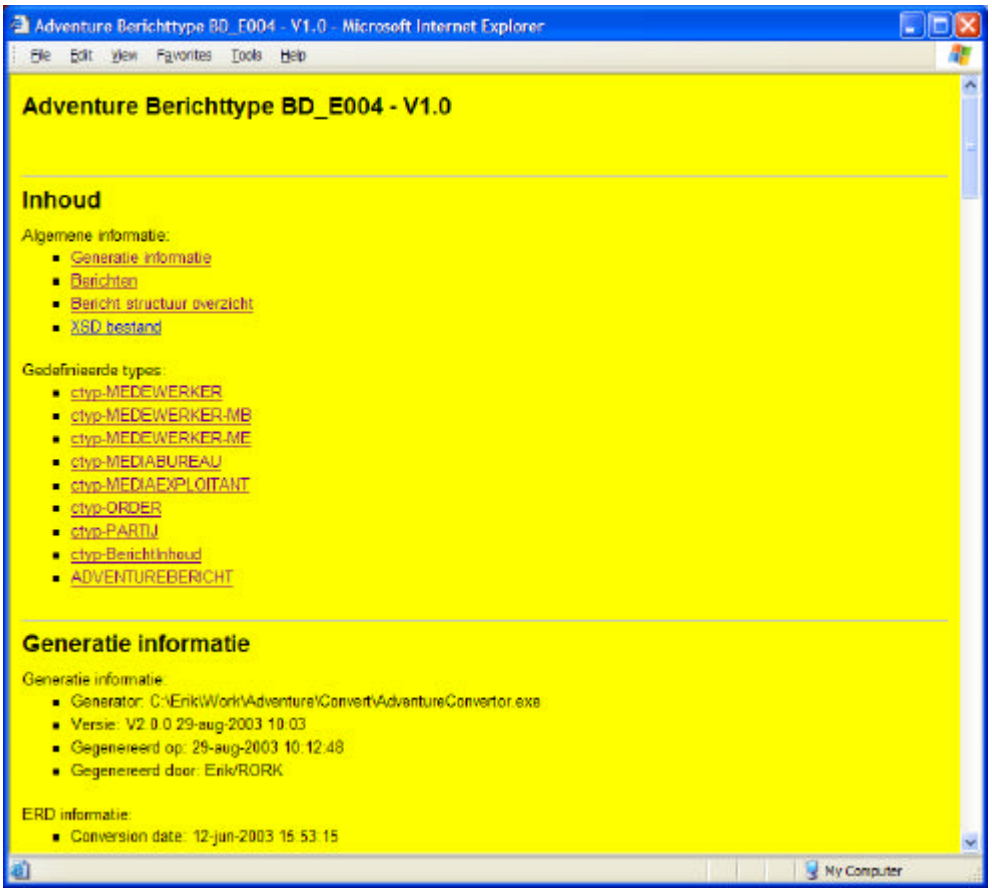
- All messages use `<ADVENTUREBERICHT>` as the root element. The attributes of the root element define what kind of message this is. With the in formation in these attributes you can find the schema for this message (in this example it would be in a file called `BD_B001-1.0.xsd`).
- All messages start with the same envelope information. This holds information about the sender and receiver, a timestamp, etc.
- The actual message is always in an element called `<BERICHT>` (Dutch for “message”). The content of this element varies with the message type.

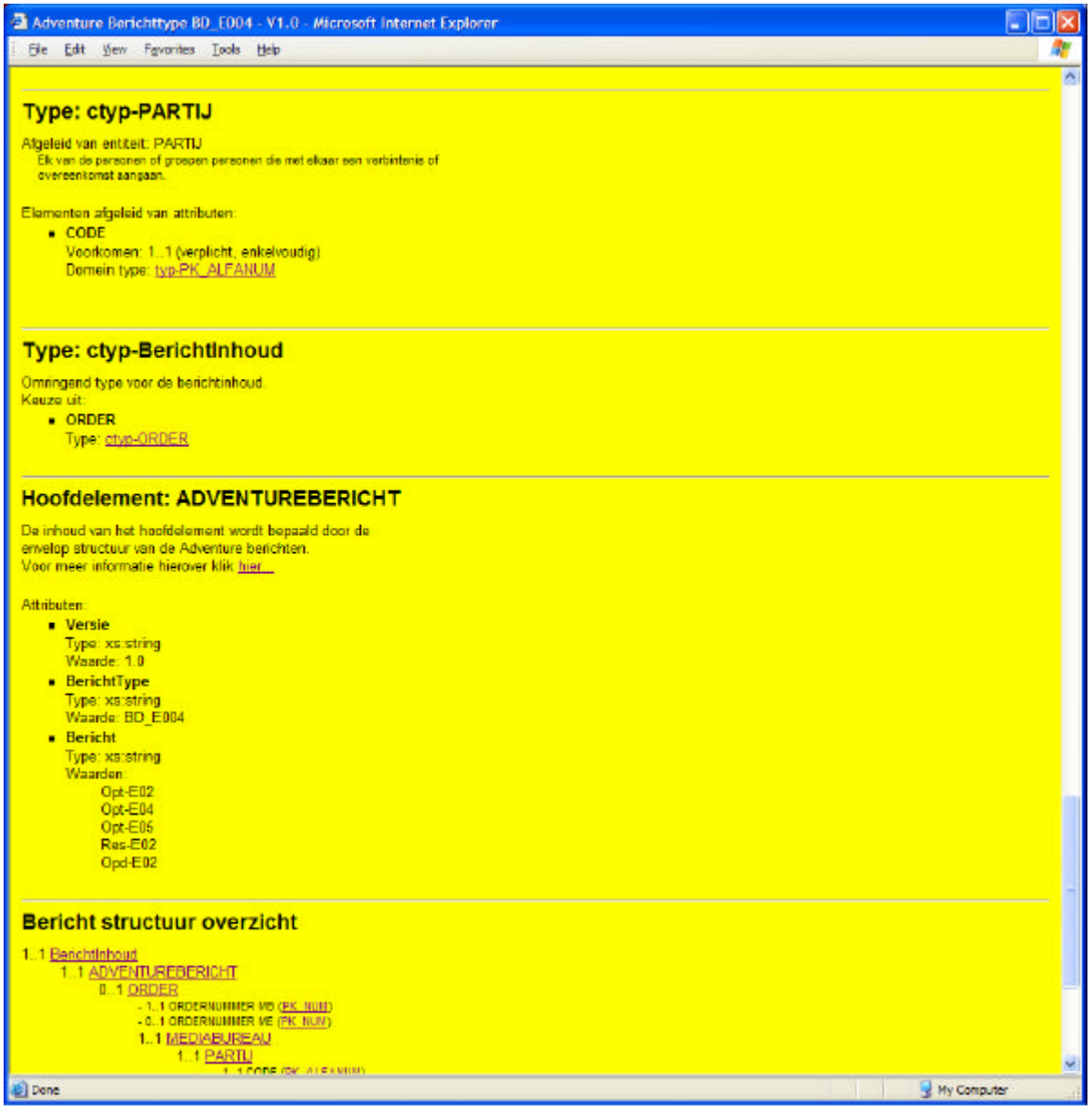
A schema that defines a message structure like the one above is build up as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- ============================================================= -->
  <!-- Generator: D:\Erik\Work\Adventure\Convert\AdventureConvertor.exe -->
  <!--    Generator version: V1.0.13 23-mei-2003 11:28 -->
             |
           Comments about how and when this schema was generated and
           what messages it is for.
             |
  <!-- Messages: -->
  <!--    Opt-E02: ME wijst optie af. -->
  <!--    Opt-E05: ME communiceerd het binnen 24 uur vervallen optie. -->
  <!-- ============================================================= -->

  <!-- Includes: -->

           Fixed include files with information about the message
           envelope and some general types.
             |
  <xs:include schemaLocation="../Algemeen/DomainTypes.xsd"/>
  <xs:include schemaLocation="../Algemeen/Envelop.xsd"/>

  <!-- ============================================================= -->

           For every entity used in this message a complex type is
           defined.
             |
  <!-- ===  Complex type definition for entity MEDEWERKER  === -->
  <xs:complexType name="ctyp-MEDEWERKER">
           ...
  </xs:complexType>

           etc.

  <!-- ============================================================= -->
  <!-- Main content type: -->

           The main content structure of a message is always defined
           in a complex type with the name ctyp-BerichtInhoud
           (Message Content).
             |
  <xs:complexType name="ctyp-BerichtInhoud">
    <xs:choice>
      <xs:element name="ORDER" type="ctyp-ORDER"/>
           etc.
    </xs:choice>
  </xs:complexType>
```

```
  <!-- ============================================================= -->
  <!-- Main element: -->

            Finally the root element is defined. This is always
            <ADVENTUREBERICHT>. It is an extension of a centrally defined
            complex type called ctyp-ADVENTURE-ENVELOP (defined in an
            include file).
            The extensions define the attribute values for the root element.
            The ctyp-ADVENTURE-ENVELOP type in turn defines an element
            called <BERICHT> with the type ctyp-BerichtInhoud (defined
            above) as its content.
              |
  <xs:element name="ADVENTUREBERICHT">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="ctyp-ADVENTURE-ENVELOP">
          <xs:attribute name="BerichtType" type="xs:string"
             use="required" fixed="BD_E004"/>
          <xs:attribute name="Versie" type="xs:string"
             use="required" fixed="1.0"/>
          <xs:attribute name="Bericht" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Opt-E02"/>
                <xs:enumeration value="Opd-E02"/>
                  etc.
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <!-- ============================================================= -->

</xs:schema>
```

Finally two screenshots of the generated documentation:





The documentation is fully bookmarked. Clicking on a link brings you immediately to the right page/location.

4. EVALUATION

Looking back on the project, I feel the following things are important lessons learned:

Things I think we did right:

- Although this had nothing to do with generating the schemas, creating a virtual data/communication model for a standardization effort like this is very efficient. It made all the choices and definitions very clear and unambiguous. And, of course, it was a very valuable input for the schema creation process.
- Do not hand craft schemas for a standard like this. It would have been next to impossible to get every detail right. A new version of the standard would have been a maintenance nightmare. Generating them is the way to go.
- The same applies to the accompanying documentation. If you take care that your data model and other inputs are documented well, you can generate high quality documentation automatically. If you use HTML, you cross-reference the documentation (with hyperlinks) easily.
- Excel files are a very well suited as an intermediate data format in these kinds of processes. It allows you to view and amend the result without the need for extra data input/reporting software. Any COM/ActiveX enabled programming environment can read/write an Excel file without problems.

Things to do better next time:

- The format for the message content file (as described in paragraph 3.2.2 on page 10) was a little awkward and hard to do right and understand. It took several tries and versions to get it completely right. I would advise experimenting with some other formats and put a little more effort in this than we did.
- I defined most of the actual message structure as the XML specialist on the project. The results were messages that were technically correct and easy to generate, but a little hard to understand. I think it would have resulted in better messages as (one of) the analyst(s) would have done this. If you do this, it is not necessary to design the XML directly: An overall structure idea would have been sufficient.