An XML-grammar-based Data Conversion Utility

Simon Kranzer[1], Robert Merz[1], Andreas Unterweger, Bernadette Himmelbauer[1], Peter Ott[1], Gerhard Jöchtl[1]
[1]Fachhochschule Salzburg GmbH

In industry, many devices, systems or other data sources produce huge amounts of information in a large variety of different and proprietary formats. In order to connect the data flow between individual systems to form effective and integrated tool chains, to store or analyze data, or to retrieve information for model creation and optimization, these different data formats constantly need to be converted in an automated fashion. Software for translation is required, which needs to be tailored to each individual case. If data formats change even in a diminutive way, those programs have to be modified and retested. To overcome the major drawbacks of data translation, we developed a graphical data conversion utility to simplify and facilitate this process. In our approach, the transformation is steered by rules implemented as an XML grammar. The user is able to define and reproduce a format using graphical blocks which represent nodes in an XML document. A parser and output generation tool are then controlled by this XML description. As this procedure is the same for both, the input and the output direction, all defined data formats can be converted into each other, using just a single description of each data format. This paper presents our approach, a data model, the XML grammar and storage in a relational database. Furthermore, it describes the components of a Rich Internet Application consisting of tools for grammar definition, validation, data conversion and generation.

1. Introduction

In the last decades, the amount of stored data has increased dramatically. In industry, thousands of thousands of process variables are gathered every day. Today, nearly every device collects data. For example, in an assembly line, data is gathered by individual machines, processor boards, sensors, cameras, counters, interfaces, and, last but not least, the workers themselves.
To transform that data and extract useful information from it a common data format is essential. However, most of the data is generated from different data sources and is therefore only available in non-standardized different proprietary representations. Appraisal, manipulation, permanent storage and exposition are often a vain endeavor, because information semantics of the used heterogeneous formats do not fit larger scale data containers used for evaluation and analysis. Therefore, though big data has become the fourth factor of production (see e.g. Capgemini, 2012), a lot of information is lost because of the amount and complexity of data which has to be transformed into different exploitable formats.
In order to overcome the above issues, data transformation software is needed.

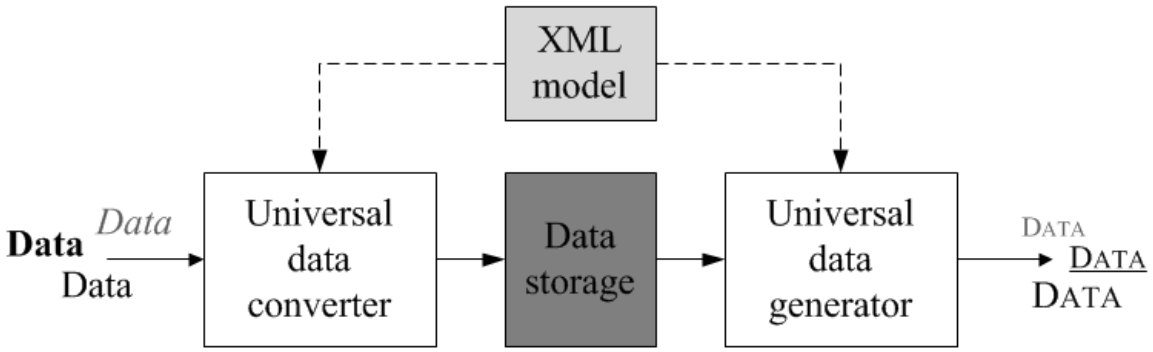

Figure 1: XML-grammar-based Data Conversion

One approach for data conversion is to implement a special parser and generator every time a new format has to be translated. Another method is to use adaptable data transformation tools and/or languages.
In this paper we present a universal data converter which enables the transformation of the acquired data sets into a general model with tree based structure, using a parser, whose rules are specified graphically or by programming . For a more detailed view on that work, please also refer to Unterweger, Himmelbauer, Kranzer, et al. (2012) and Unterweger, Himmelbauer, Kranzer, et al. (2011). Furthermore, to store that information, we propose a generic data model which allows storing arbitrarily complex data types in binary form, including metadata which simplifies the conversion from and to the format specified by the model. Data stored using the proposed data model can be arranged and rearranged, augmented and modified intuitively. This is enabled by a graphical component for data manipulation which offers statistical functionality to evaluate the existing data, and, if desired, generates new data sets, e.g., to supply test data. Erroneous or deficient data can be amended or deleted. At the end of the developed tool chain (see Figure 1), our data generator provides graphical blocks and accordingly classes for arbitrary transformation of the stored and generated data. Hence, our proposed system provides the data-driven transformation from and to arbitrary formats using a generic data model. Thus, we enable the use of distinct proprietary formats within one application as well as persistent and sustainable storage.

Similar to our model, a transformation language has been realized in XSLT by the W3C (W3C, 2007). While both approaches are Turing complete (see Brainerd & Landweber (1974) for Turing completeness) and support the conversion of arbitrarily complex formats, XSLT requires separate specifications for input and output. Furthermore, it is not specified how storage inside the database is handled. BiXid (Kawanaka & Hosoya, 2006), as well as XMLTrans (Walker, Petitpierre & Armstrong, 2000), are bidirectional transformation languages, but designed for the transformation of XML formats only and therefore lack usefulness, especially in industrial environments. DSLTrans (Barroca, Lucio, Amaral, Felix & Sousa, 2010) is another model, which, however, is not Turing complete. Parser generators like ANTLR (Parr, 2007) lack a generic data model, and in addition, cannot be

used for text generation. Standard text processing software like awk or commercial data converters like Altova MapForce (Altova, 2011) are limited to data formats and/or in the complexity of the conversion. All the above clearly address the same issue but are either not universal or less flexible than our approach.

The remainder of this paper is structured as follows. After the introduction and the presentation of other approaches as well as pointing out the difference to the proposed system in Section 1 above, Sections 2, 3 and 4 present the XML grammar, data generation and the generic data model. In Sections 4 and 5, the database for storage and the graphical user interface are discussed.

2. XML Grammar and XML Model

To enable the transformation of acquired data, we specified an XML model by using an XML schema (W3C, 2004). The resulting grammar is sufficient to model any computer program due to its Turing completeness (for a formal proof please refer to Unterweger, Himmelbauer, Kranzer, et al., 2012 or Unterweger, Himmelbauer, Kranzer, et al., 2011). To provide this, data is modeled using elements. Figure 2 shows the basic building entity of a data format structure, a *"block"*.

```
<xs:element name="block">
  <xs:annotation>
  <xs:complexType>
    <xs:all>
    <xs:attribute type="xs:string" name="name" use="optional" />
    <xs:attribute type="xs:string" name="type" use="required" />
    <xs:attribute type="xs:boolean" name="optional" use="optional" />
    <xs:attribute type="xs:boolean" name="lookAhead" use="optional" />
  </xs:complexType>
</xs:element>
```

Figure 2: XML-grammar-(block)

Four different types of blocks are used.*"static" and "variable* blocks allows representing static content, like fixed text literals, and variables to store dynamic values, respectively. The *"until"*-block enables the repeated processing of data of the same structure up to a specified token. The remaining type, the *"dummy"*-block, is used to provide a placeholder for later specification, when references to other blocks need to be used while parsing. Figure 3 illustrates the grammar for a *"for"*-loop which repeats the execution specified in *"iterate*" until the *"condition"* doesn't hold any longer.

```
<xs:element name="for">
  <xs:annotation>
  <xs:complexType>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element ref="condition"></xs:element>
      <xs:element name="iterate" type="allClasses"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3: XML-grammar-(for)

With variables, conditions, branches (see Figure 4) and loops the grammar is able to mimic any structure of input text formats. In addition, since the underlying XML model does not differ when either consuming or producing data, programs specified using our XML grammar are capable of both parsing, as well as data generation.

```
<xs:element name="if">
  <xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="condition" minOccurs="1" maxOccurs="1" />
      <xs:element name="true" type="allClasses" minOccurs="1" maxOccurs="1" />
      <xs:element name="false" type="allClasses" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 4: XML-grammar-(if)

3. Parsing and Data Generation

As already mentioned above, data generation and parsing share the same XML grammar, or XML model, respectively (see Figure 1). Once a data format is specified, either in textual form using the XML-grammar (see Section 2) or using the graphical interface (see Section 6), our conversion utility is able to read that format as well as to generate data, which is stored using our data model (see Section 4).
For the sake of simplicity, the validation and parsing steps shown below apply for data conversion. The same parser is also used for data generation, however, the order of the steps may differ. After the user has either graphically or textually specified the data format, the resulting XML model is validated against the XML grammar provided (see Section 3). If an input stream is defined, the parser starts to consume data according to the structure provided. In case of errors, several correction mechanisms are triggered. Anyway, all errors and warnings are reported to the user or the software/tool which is being used. Data, which has been parsed successfully, is either transformed into the proposed data model (see Section 4), or, in the case of online conversion without the need of storage, directly passed to data generation.

4. Data Model and Storage

As claimed above, our conversion tool and our XML model allow the representation of all existing and future data formats. Therefore, we propose a data model which is generic and allows storing complex data types in binary form, including metadata. This simplifies the conversion from and to the format specified by the model. If necessary, metadata is stored in addition to payload data. Our tool enables representing encrypted and compressed data, as well as binary values of any size.
Our model is able to represent lists, trees and structures which can be composed of the latter two. In addition, dependencies between single elements can be modeled. For example, Figure 5 illustrates a binary tree consisting of three stored temperature values. Aside from the nodes, which contain the payload, there are artificial nodes (shown in gray), which indicate the end of the tree or of one of its levels, respectively. For the sake of simplicity, artificial nodes store a NULL value.

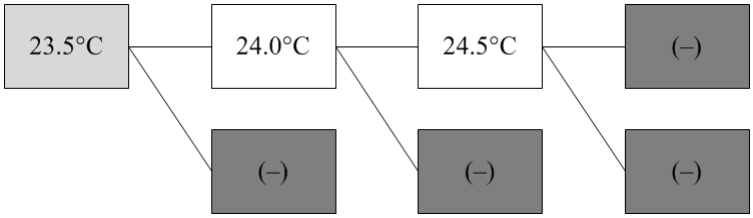

Figure 5: Binary tree representation

To physically store information, our data model has been realized as an ER model within a relational database. A more detailed view of the data model and its implementation can be found in Unterweger, Himmelbauer, Kranzer, et al. (2012).

5. Graphical Interface

As a proof of concept, all three main modules of the conversion utility depicted in Figure 1, i.e., the data converter, the data storage and the data generator, have been implemented prototypically as a Rich Internet Application.

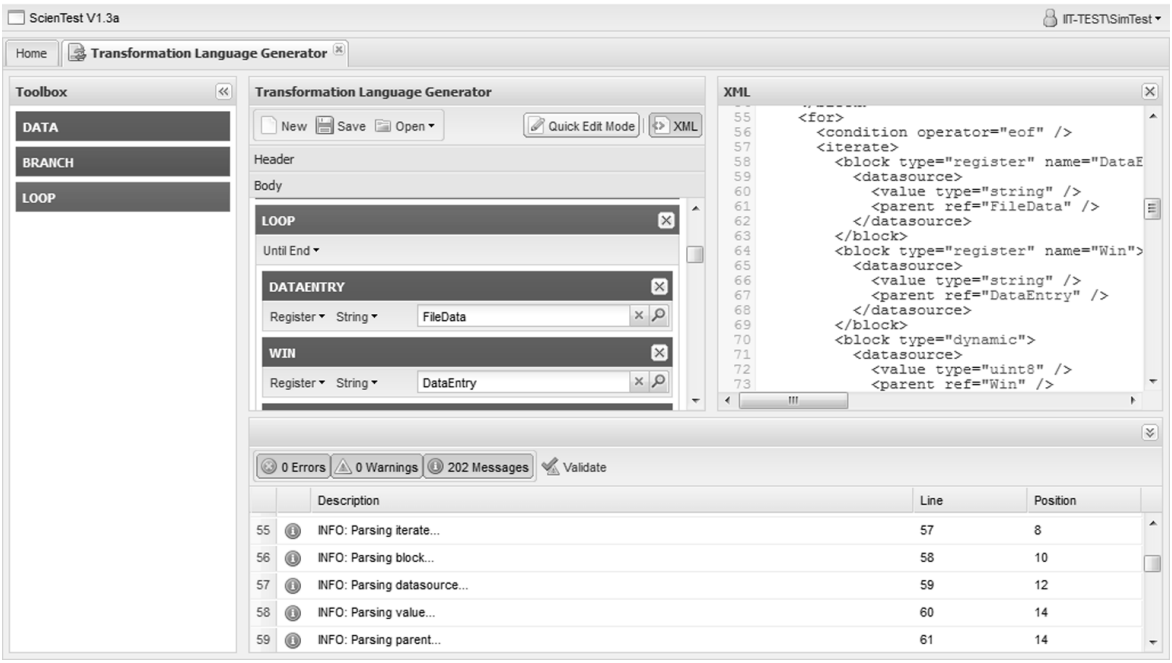

Figure 6: Transformation Model Generator

To specify a data format using the XML grammar (see Section 2), a Rich Internet Application for the GUI-based creation of transformation templates has been implemented (see Figure 6).
The GUI consists of four main areas: On the left, there is a list of available block templates, which can be added to the "Transformation Language Generator" in the middle. There, the list of blocks forms a graphical representation of the specified XML grammar. In addition, on the right, an XML editor is provided which reflects the blocks and their properties. If the user desires, it is possible to use the editor as a stand-alone tool, without making use of the graphical block representations. At the bottom of the GUI, the validation output window displays errors, warnings and informative messages as received by the parser in real time.
As soon as the user has finished designing the input format, test data in this format can be uploaded using a separate GUI. Subsequently, the file can be processed and its values are stored into the data base. During conversion, the overall progress, as well as detailed status messages are displayed. If errors occur, the user has the possibility to adapt the specification of the file format and retry.

6. Conclusion and Acknowledgements

In conclusion, we developed a method for universal data conversion for industrial purposes. Arbitrary input formats can be modeled using the proposed XML grammar. Once specified, data can be read from and generated into a data format using the same XML model for both directions. Between input and output, all data can be transformed into our generic

data model. Once data is in that format, it can be stored in a relational database for further processing and analysis. All the above steps can be triggered either by programming or using our prototypical Rich Internet Application.

Finally, we would like to thank our project partners for their generous support: B&R Industrial Automation Corporation, Bosch AG Hallein, COPA-DATA GmbH, Liebherr-Werk-Bischofshofen GmbH, PALFINGER AG and Wirtschaftskammer Salzburg.

7. Bibliography

Capgemini (2012). The Deciding Factor: Big Data & Decision Making, A global survey from the Economist Intelligence Unit commissioned by Capgemini, URL: http://www.capgemini.com/insights-and-resources/by-publication/the-deciding-factor-big-data-decision-making/ [30.01.2013]

Unterweger, A. Himmelbauer, B. Kranzer S. et al. (2012). A Generic Model for Universal Data Storage and Conversion and Its Web Based Prototypical Implementation. International Journal of Information Technology and Web Engineering (IJITWE), 7 (1), DOI: 10.4018/jitwe.2012010105

Unterweger, A. Himmelbauer, B. Kranzer S. et al. (2011). A generic model for universal data storage and conversion. Applied Electrical Engineering and Computing Technologies (AEECT), 2011 IEEE Jordan Conference on, pp.1-6, 6-8 Dec. 2011, doi: 10.1109/AEECT.2011.6132527

Brainerd, W. S., & Landweber, L. H. (1974). Theory of Computation. New York, USA: John. Wiley & Sons

W3C. (2007). XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20 [10.01.2013]

Kawanaka, S., & Hosoya, H. (2006). biXid: a bidirectional transformation language for XML. ICFP '06: Proceedings of the eleventh ACM SIGPLAN International Conference on Functional Programming (pp. 201-214). Portland: Association for Computing Machinery.

Walker, D., Petitpierre, D., & Armstrong, S. (2000). XMLTrans: a Java-based XML Transformation Language for Structured Data. Proceedings of the 18th International Conference on Computational Linguistics - Volume 2 (pp. 1136-1140). Saarbrücken: Association for Computational Linguistics.

Barroca, B., Lucio, L., Amaral, V., Felix, R., & Sousa, V. (2010). DSLTrans: A Turing Incomplete Transformation Language. In B. Malloy, S. Staab, & M. Van Den Brand (Ed.), Proceedings of the Third International Conference on Software Language Engineering (pp. 296-305). Eindhoven: Springer-Verlag.

Parr, T. (2007). The Definitive ANTLR Reference: Building Domain-Specific Languages. Raleigh, USA: Pragmatic Bookshelf.

Altova. (2011). Altova MapForce. http://www.altova.com/mapforce.html [05.02.2013]

W3C.(2004). XML Schema Part 0: Primer Second Edition. http://www.w3.org/TR/xmlschema-0/ [05.02.2013]