A Model-driven Process for Data Transformation of Heterogeneous Data

Haïfa Nakouri and Nadia Essoussi
LARODEC Laboratory, University of Tunis, Tunis, Tunisia

Keywords: Model Driven Engineering, Data Format Transformation, Heterogeneous Data.

Abstract: In this paper, a model-driven approach to transform data format is provided. Original data is extracted from heterogeneous sources and is initially presented in a generic format called the public view format. Our approach includes a complete process of data transformation starting with data in public view format and ending with generating different kinds of formats (Excel, XML, HTML, etc.). There are three main stages in the proposed model-driven process: the input file injection which inserts the public view's data in their corresponding model, the model-to-model transformation to bridge between the public view's model and the output file's model and finally the output file extraction to extract the output file from its corresponding model. Experimental results show that a model-driven approach for data transformation outperforms a code-centric approach in terms of execution time and automation rate.

1 INTRODUCTION

The growing diversity of the current systems and platforms makes it more and more challenging to have reliable and maintainable software systems. Moreover, third generation languages become not able enough to handle such a perpetual and fast evolution. This makes necessary to find a way to extract and present information in a meaningful way in order to make smart and fast decisions. Therefore, business intelligence (BI) tools are used to organize the data and then simplify the decision making process. The main issue with such decisional platforms is that they are used for reporting and analytical purposes regarding existing information systems which most of them are legacy and heterogeneous. A business intelligence system is mainly fed from several heterogeneous distributed data sources (MySQL, postgreSQL, SQLserver, flat files, etc.). Thus, the input data can be in different formats (Excel, XML, CSV, etc.). However, a good organization and structuring of data requires that all data have the same format even if it is extracted from different sources.

Model Driven Engineering (MDE) is an emerging software engineering methodology which focuses on creating models and transformations between them so that the whole development process becomes model driven. Using an MDE approach is convenient since it increases the value of models and allows handling specifications and requirements of systems from higher levels of abstraction. One of the most important objectives of software engineering is to address this platform complexity and the inability of third-generation languages to alleviate this complexity and express domain concepts effectively. MDE affords many techniques that help developers to build in the large more efficient, reliable and maintainable software systems.

This work was elaborated in the context of a customized business intelligence system that is used inter alia to evaluate the quality of a company projects'. This BI platform is used for reporting and analysis of existing data and support systems which most of them are legacy and heterogeneous. The main objective of this system is to have a single and unified architecture for reporting to develop more customized and relevant quality report. In this system, external data sources are not directly used for reporting due to security measures. Instead, a set of public views (PV) is used. We note that a public view is basically a SQL view. Each public view is the result of the mapping of different original data. Later, the generated public views are used for reports generation. Thus, the public view's data is exploited to generate reports of all kinds of formats (Excel, HTML, XML, etc.). A model-driven approach might be an interesting solution for the heterogeneity issue since we carry out data format transformation in a generic way, using models and model transformations at different levels of abstraction. Nevertheless, even with

Nakouri H. and Essoussi N..
A Model-driven Process for Data Transformation of Heterogeneous Data.
DOI: 10.5220/0004221901770181
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 177-181
ISBN: 978-989-8565-42-6

the success of many model-driven approaches within the researchers' community, there is still a somehow mystery regarding the efficiency of model-driven solutions by the industrials' community. Despite the availability of a plenty of MDE standard approaches and tools, model-driven applications are not as much developed as expected in software engineering industry.

The objective of the paper is twofold. First, we propose a basic version of a model-driven process for data format transformation to handle the systems heterogeneity issue. This process involves the different stages to transform data starting from a public view file format and resulting in other output file formats. In this work, we use the concept of public view model to denote a generic model of any type of data format (e.g. text, Excel, CSV, XML, etc.). The public view data corresponds to the result set of the SQL view query. This solution is about a complete automated process handling data format transformation. Then, we evaluate the proposed model-driven approach for data transformation to a code-centric one. Second, we show that it is possible to use model-driven engineering techniques for a real software engineering case such as data format transformation.

The rest of the paper is organized as follows. Section 2 details the three stages of the proposed approach and presents an example of a model-driven transformation from the public view format to the Excel format. Section 3 presents experimental results.

2 USING AN MDE APPROACH FOR DATA FORMAT TRANSFORMATION

In the context of our work, we precisely mean by data transformation, the data format transformation such as the Excel, CSV, HTML or XML formats. Then, we propose to apply a model-driven approach to the data transformation problem. Our approach was integrated in an existing reporting system where the generation of different types of reports (Excel, HTML, XML, etc.) is needed, starting from public views' data (in the CSV format). A public view consists of a stored query referencing real database tables and it is accessed as a virtual database table. The public view data corresponds to the result set of a SQL view query. In fact, there exist multiple heterogeneous data sources. These data sources can not be immediately accessed for security purposes. Besides, not all the available data is needed to generate reports. For this purpose, public views are used to store only relevant

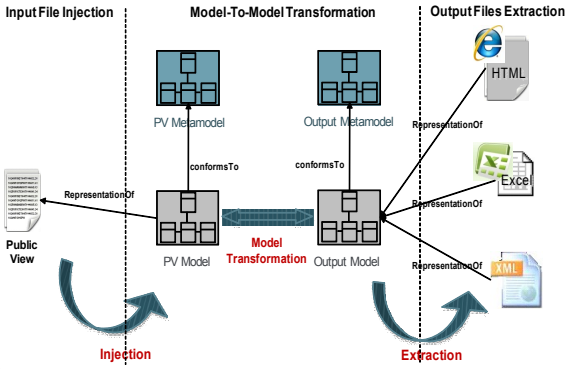

Figure 1: The data transformation process.

data for reports generation. At a latter stage, the public view data is used to generate different other formats such as Excel, HTML or XML which are ultimately used for reports generation, OLAP analysis, etc. In the context of our work, the point is to start from the public view's data and generate other formats (Excel, HTML, XML) containing exactly the same original data, using only models and model transformations. Thereby, we propose to model the public view layer in a model-centric way. Actually, we point to have a metamodel for both the public view format and the output files formats and then manage transformations between them in order to automate the process. Accordingly, we suggest to perform a complete data transformation process driven by models. During this process's development, four main features should be considered: the source of our process (the public view input file), the targets of our process (Excel, HTML, XML output files, etc.), the metamodels of both source and targets of the process and transformations between models. Figure 1 shows an overview of our model-driven process for the data transformation issue. It consists on three stages that are detailed below.

2.1 Input File Injection

The first stage of our model-driven process is the injection of the public view data in the associated public view model. First, we write the concrete syntax (grammar) of the public view data and then bridge it with the MDE technical space. A technical space (Bézivin, 2005) aims at representing different technologies at a higher level of abstraction which may allow capturing similarities and differences and eventually find a possibility of integration. To achieve the injection phase, we use the Xtext approach (Efftinge and Vlter, 2006) to allow the integration between the two environments. In Figure 2, we present a description of the injection steps:

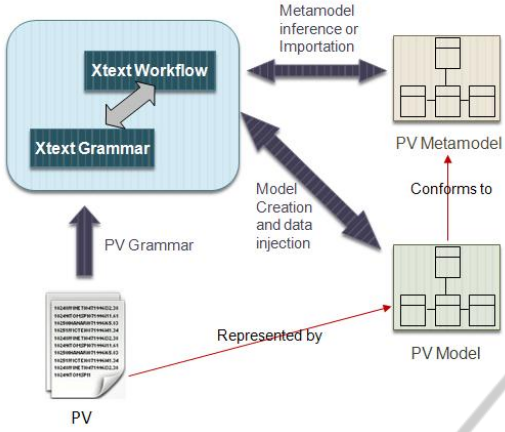

Figure 2: Data Injection with Xtext.

- Given a public view PV, we develop its corresponding Xtext grammar file. In the grammar file, we describe the exact structure of the data. In our case, the public view's grammar corresponds to the CSV's grammar and then we describe the CSV's format structure (i.e. a set of columns delimited by a separator).
- The public view metamodel in the ecore format is either imported if it already exists or inferred by the Xtext workflow. Ecore is the model and metamodel representation format used in EMF.
- Finally a public view model conforming to the public view metamodel and containing the public view data is generated.

2.2 Model-to-Model Transformation

At this stage the public view's data are injected in the public view's model. To obtain the wanted output file, we have to carry out a serial of model-to-model transformations first. The transformations are precisely between metamodels (the public view metamodel and the output formats metamodels). Thus, any public view's model (corresponding to the public view's metamodel) can be transformed to any output model (corresponding to an output metamodel such as those of Excel or XML). To define these model-to-model transformations, we use the Atlas Transformation Language (ATL) (Allilaire et al., 2006), a component of the AMMA platform (Bézivin et al., 2007).

2.3 Output Files Extraction

In the AMMA platform, XML is considered as a technical space with standard projectors: either injectors to the MDE technical space or extractors from the MDE technical space (Sun et al., 2008). In our case,

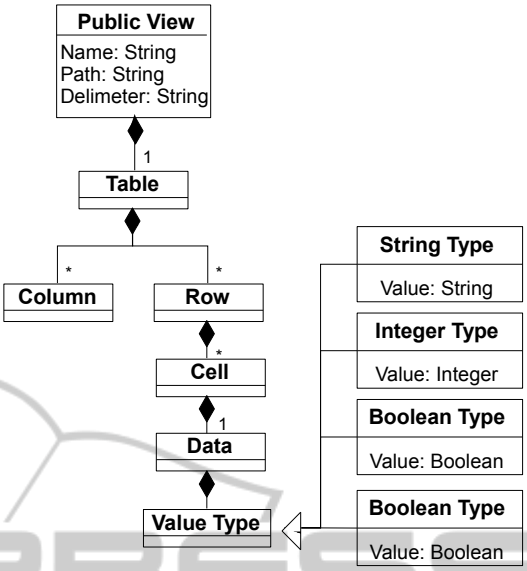

Figure 3: The public view metamodel.

we are only interested by the XML extractors to allow the extraction of the output files from their associated models. In what follows, we detail an example of a transformation from a public view format to Excel format. To carry out this transformation, we should first inject the public view file's content in the public view's model as shown in the details of the injection phase of the process (Figure 2). Then, we should define the different metamodels that will be used (the public view and the Excel metamodels in this example). Given that in our example the initial public view's data is in the CSV format, we define the public view's metamodel as a generic text file which data is organized in rows and columns and delimited by a given delimiter (see Figure 3). The Excel and XML metamodels have been already defined in previous works (Jouault et al., 2006) and we were pretty inspired by these works. Figure 4 shows an overview of the publicView-to-Excel model transformation. On the left side, we have the MDE technical space with the three levels of abstraction. The highest one (M3) corresponds to the ecore metametamodel, then in the second level (M2), we have our created metamodels (public view, Excel and XML) that conform to the ecore metametamodel and finally in the bottom-level (M1) we find the public view, Excel and XML models which are instantiations (examples) of their related metamodels. Accordingly all the models correspond to their corresponding metamodels. On the right side of Figure 4, we have the XML technical space and it is also composed of three levels of abstraction. The highest level (M3) corresponds to the XMLSchema metametamodel. In the second one (M2), we have the spreadSheetML metamodel. Finally, the lowest

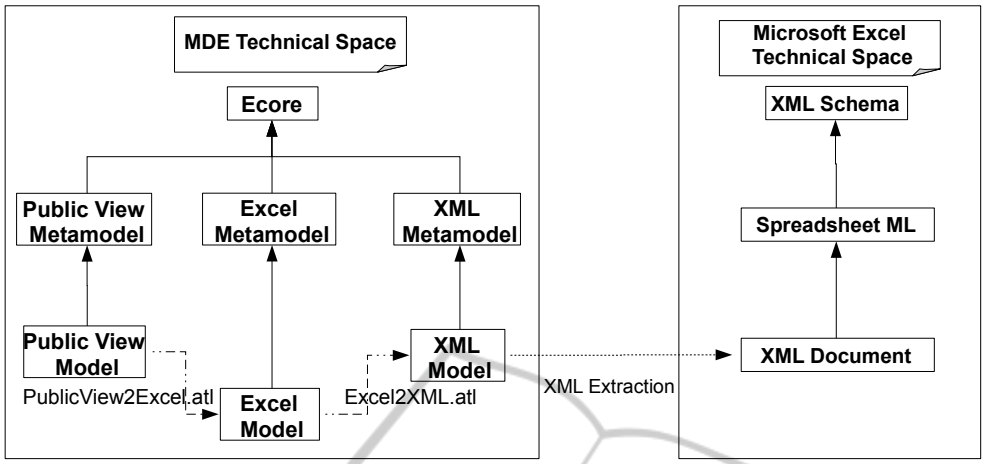

Figure 4: Public view to Excel Transformation.

level of abstraction corresponds to the XML document which is written in the spreadSheetML language and it can be viewed as an Excel document (Jouault et al., 2006). We integrated the XML metamodels in this transformation because it is not possible to extract an Excel file directly from an Excel model. Actually, all the possible extractors are of XML types and given that Excel is an XML-like language (with the Spreadsheet format) we can perform this extraction. Therefore, to generate the output Excel file, we should first perform a model transformation from public view to Excel (via the ATL transformation language, see Figure 5). This first necessary transformation is a model-to-model one and it consists in transforming the data contained in the public model into the Excel model. Thus, the result of this transformation is an Excel model containing all the public view's data. As a second step, we need the usage of a projector and precisely an extractor in order to extract the Excel file. Here, the projector consists in a transformation from Excel to XML (via the ATL transformation language and the predefined XML extractor). Therefore, we need a transformation from the Excel model to an XML model. The result of this transformation is an XML model containing all the Excel model data. Then, we perform an Excel file Extraction and it aims at extracting each of the elements composing the input XML model into an output XML file. No model-to-model Transformation is required for this. We just need an XML extractor. At the end of the extraction step, all the necessary information stored in the public view file should be correctly formatted into the Excel format.

Other types of data format transformation can also be completed such as transforming the view's data format to the XML and HTML format in a model-centric way. Given that HTML is also an XML-like language, we carry out the transformation exactly like the PublicView-to-Excel one but we should define HTML metamodel and model instead of the Excel ones. As for the PublicView-to-XML transformation, it is much simpler because we just need the XML metamodel and model used for both model-to-model transformation and XML file extraction.

```
module Text2SpreadSheetML; -- Module Template
create PublicView_Excel_SpreadSheetML :
    Excel_SpreadSheetML from DDTSTextFileModel : publicViewMetamodel;
--###--Rules--###-
--Rule 'Table2ExcelTable'
--This rule generates the initial Table
rule Table2ExcelTable {
    from
        t : publicViewMetamodel!Table

    using {
        tableRow : Sequence(TextFileMetamodel!Cell) = t.row->first().cell;
    }

    to
        wb : Excel_SpreadSheetML!Workbook (
            wb_worksheets <- Sequence{ws},
            name <- 'PublicView data'
        ),
        ws : Excel_SpreadSheetML!Worksheet (
            name <- 'Defect',
            ws_table <- et
        ),
        et : Excel_SpreadSheetML!Table (
            t_rows <-Sequence
              {t.row->collect(e | thisModule.resolveTemp(e, 'erow'))},
            t_cols <- Sequence
              {t.column->collect(e | thisModule.resolveTemp(e, 'ecol'))}
        )
}
```

Figure 5: A part of the ATL code from public view to Excel model transformation. The input of the model is the public view metamodel and the output is the Excel metamodel.

3 EXPERIMENTS

In the context of our work, we generated a complete working prototype for data format transformation using a model-driven approach. We validated the performance of our model-driven solution by carrying out a complete case study and we ob-

tained the expected results of the output formatted data files (Excel, HTML an XML formats). We also compared a model-centric approach to a classic one namely object-oriented approach (i.e. code-centric approach). We carried out this comparison based on average time execution and average automation rate. This experiment has been performed using the IBM's Eclipse Modeling Framework (EMF) (Budinsky et al., 2003) as modeling environment and the Atlas Transformation Language (ATL) (Allilaire et al., 2006) for model Transformations and tools interoperability performance. This latter is a sub-project of Atlas Model Management Architecture platform (AMMA) (Bézivin et al., 2007). We also used the Xtext framework (Efftinge and Vlter, 2006) to allow the integration between the text and model environments.

Evaluation is based on transformation from the public view format to Excel, XML and HTML data formats. Basically, the public view contains data from four projects upcoming from different external sources. Transformations from public view format to Excel, HTML and XML formats are performed using the proposed data transformation model and the existing object-oriented model. Table 1 shows the outperformance of the a model-driven approach in terms of execution time and automation rate. Execution time is computed as the average time of transformation between the public view format and the three other output formats. Average execution time of format transformation indicated in Table 1 shows that a the proposed model-driven approach is faster than an object-oriented one. Besides, data transformation using the proposed model is entirely automatic unlike the existing object-oriented model. The code-centric approach does not allow a complete automatic process and repeated revision of the code is required. With a the transformation model, data is accurately transformed from the public view format to Excel, XML or HTML format without confusion regarding the data type or semantic. Whereas, with the existing object-oriented system, data is not always as well formatted and by hand rectifications are necessary. The average automation rate is computed by dividing the number of data columns that were rectified by the total number or data columns of the source projects altogether.

Table 1: Accuracy rate and time execution.

	Model-oriented	Object-oriented
Execution time (s)	**10**	40
Automation Rate %	**100**	80

4 CONCLUSIONS

In this paper, we presented a model-driven approach to handle the problem of systems heterogeneity and proposed a model-driven process for data format transformation. Managing data semantics heterogeneity is not yet part of the proposed model-driven process. Handling different data semantic from external data sources can be considered in future works. We faced the challenge to prove that it is possible to apply current model-driven approaches to real industrial case studies as we showed in the data transformation example. Experiments show that a model-driven approach of data transformation reduces the domain-specific dependency. Nevertheless, the success of a model-driven approach is intimately related to the choice of the most relevant MDE approach and tools regarding the complexity of the application. MDE affords strong approaches namely Model Driven Architecture (MDA) and Software Factories (SF). Nonetheless, the main drawback of MDE is the lack of reliable and sufficient tool support. These tools are still in perpetual development and more stable releases are needed. The unsteady criteria of the current MDE tools can be a huge handicap for the validity and credibility of MDE applications especially for high-scaled systems.

REFERENCES

Allilaire, F., Bzivin, J., Jouault, F., and Kurtev, I. (2006). I.: Atl eclipse support for model transformation. In *In: Proc. of the Eclipse Technology eXchange Workshop (eTX) at ECOOP*.

Bézivin, J. (2005). Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64.

Bézivin, J., Jouault, F., and Touzet, D. (2007). An introduction to the atlas model management architecture., technical report, lina. Technical report.

Budinsky, F., Brodsky, S. A., and Merks, E. (2003). *Eclipse Modeling Framework*. Pearson Education.

Efftinge, S. and Vlter, M. (2006). oAW xText: A framework for textual DSLs. In *Workshop on Modeling Symposium at Eclipse Summit*.

Jouault, F., Bézivin, J., and Team, A. (2006). Km3: a dsl for metamodel specification. In *In proc. of 8th FMOODS, LNCS 4037*, pages 171–185. Springer.

Sun, Y., Demirezen, Z., Jouault, F., Tairas, R., and Gray, J. (2008). A model engineering approach to tool interoperability. In *SLE*, pages 178–187.