International Conference
FROM DELC TO VELSPACE
Plovdiv, 26–28 March 2014

DATAMORPHOSE: AN INTERACTIVE DATA MIGRATION FRAMEWORK

Vassil Vassilev, Petya Petrova, Martin Vassilev

Abstract. *The paper presents a framework, which defines and automatizes the necessary steps for data migration. We give an overview of a flexible, platform-independent plugin system. The plugins are categorized according to the enumerated data migration steps. The paper argues in favor of a visual programming language, which helps the non-experts to drive the data transformation process. We show an environment, which decouples the data model from the visualization and the interaction models. The work presents loose-coupled visualization techniques pairing visual representations with their non-visual counterparts.*

Keywords: Data migration, Visual Programming Languages, Interactivity, Multiple Models
Mathematics Subject Classification 2010: 68P15, 97R50

1. INTRODUCTION

Data migration is a process that occurs when a system needs to be upgraded or replaced by another system. The procedure is based on series of common steps in order that the legacy data can meet requirements of the target system.

Migrating data is a complex task, which requires methodical approach and specific knowledge in the field. Many details should be taken into account before a migration. The number of risks and mistakes could be minimized if the migration is automatized considering many factors, which could happen if done manually. An important aspect of a successful migration is the visualization of the data and of the tools that transform the data. A user must be able to import, export, analyze, change and validate the data with appropriate easy-to-use tools.

The main goal of our framework, called DataMorphose, is to provide a set of instruments, which simplify the process of data migration. There is a cross-platform, open-source prototype, demonstrating the framework. It defines a data-flow visual programming language (VPL). The VPL allows the non-expert users to perform data migrations.

The requirements for DataMorphose could be summarized as a tool, oriented to users having little or no knowledge in the field of data structure. Its environment ought to provide a convenient way of learning the building elements of data using a set of intuitive tools. The users can try as many transformations on the data as they want, having the data loaded directly into DataMorphose's live scratch area.

DataMorphose provides tools for preparing the data for the migration. That way the user learns about the risks and mistakes, which could happen in the process. Simultaneously, the system follows a set of steps, which gives to the users' valuable information over the required and iterative actions to achieve a successful migration.

Framework's goal is to provide reusable set of classes to automate the process, regardless the type of the data. The classes have to be generic enough so that various tools could be built on top of them. They have to be simple and intuitive to use by non-experts in the domain of data migration.

This paper is organized as follows: Section 2 describes related work; Section 3 gives an overview of the concepts on which is based the prototype; Section 4 highlights the important implementation details and supports the taken decisions with arguments; Section 5 discusses foreseen application scenarios; Section 6 concludes the presented work and gives future directions of the project.

2. RELATED WORK

Some modern systems have features and properties, which are essential for the data migration. We examine two types of applications – one concentrates on the mashup technologies [1], which is a great example of modern interactive interface with drag&drop components; the other is based on data transformations, from which we gain valuable ideas and information about the whole process of migration.

Presto Wires [2] is a user-friendly tool that allows users to create their own mashups by combining data from diverse web resources and extracting only the data of interest for the user. It draws our attention to the convenient way of using "filters" as the primary tools for applying complex live transformations on the data. The interface is very well structured, offering interactive graphical representation of the data and easy-to-use data manipulation components. However, it is not possible to perform a migration with this tool, because the purpose of the application is only to visualize extracted data from sources and changes on a dataset is not supported action.

DataSlave [3] is an ETL (Extract, Transform, Load) tool, which is designed to help users transform and migrate data. It supports numerous file formats and databases. Also, it provides a range of tools that help users go through the whole data migration process. The application gives the possibility to validate the dataset in various ways. Although DataSlave offers variety of functionalities, it has a few disadvantages – from user point of view working with this application is not a simple task even for people having some knowledge in the field. This is mainly due to the facts that some of the tools are not intuitive, also they are too many and some are very complex. Moreover, DataSlave works only on Windows OS and it is commercial application insufficient documentation. All these make the usage of DataSlave harder and not enough accessible for the public.

Targeting non-experts for performing such a complex process as data migration is not an easy task. However, implementing some features from DataSlave-alike systems and adopting the graphical presentations from in Presto Wires-alike systems may be beneficial. Combining the two items in one single application is a conceptual goal of the presented work.

3. CONCEPTS

Data migration is the process of transferring data between storage types, formats, or computer systems. Based on our practical experience in the field, the successful data migration follows the next few steps: *Data analysis* – the process of highlighting useful information about the data in the legacy system; *Data cleansing* – used mainly in databases, the term refers to eliminate incorrect, inaccurate, irrelevant and incomplete parts of the data, being found while analyzing; *Metadata restructuring* – metadata defines information about the content of the dataset. Usually, the metadata in relational databases is named schema; *Data transformation and mapping* – this is the step, which finds logically corresponding data between the legacy and the new system. The step links the data together, performing transformations if necessary; *Data migration* – executing the actual migration; *Data verification* – the final step required is verifying the data, which means checking the data accuracy.

All of the steps described above are iterative. The procedure continues until the desired results are achieved, i.e. whenever the user is satisfied with the new dataset. In DataMorphose, the term – ‘data migration’ is used to describe all transformations required for the data to fit for the needs of the new system. They are two main types – metadata transformations and data transformations. These two categories model the steps that should be executed until achieving high quality of the data.

In order to provide the tools required to perform, the highlighted migration steps, we developed a prototype application, organized as a set of modules and a core (see Figure 1) [7]. The core loads the modules from a configuration, while each module is designed to address a specific feature of the application. When necessary a module could be a combination of elements, delivering a specific set of features in broader sense. For instance, an import/export module provides IO support. It consists of two different elements – one for reading data from disk and one for writing data to disk but logically forming one module.

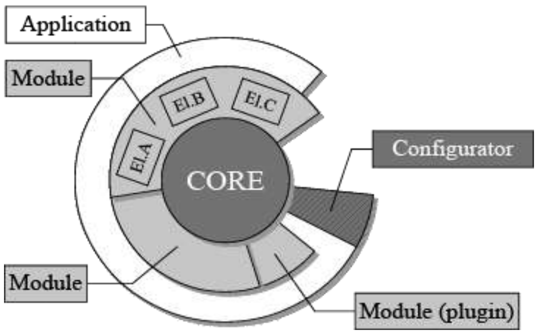


Figure 1. Abstract Architecture of the Prototype

The prototype uses the Model-View-Controller (MVC) [4] architectural pattern, where the model represents all elements of the data structure (row, column, etc.); the view contains the visual representation of the data; and the controller is responsible to be interface between the view and the model.

The prototype collects objects with the same properties into a single structure, which we refer to as data model. In order to be independent on data types and formats, the prototype abstracts out the data, turning it into a convenient model for applying diverse range of transformations. The visualization happens by relating a visualization model to the data model. Thus, the tool can provide more than one type of visualization of the same data entities. The different models can be organized in hierarchies. Analogically to the visualizers, we can have more than one controller. This forms a multichannel communication used to improve the interactivity of the application.

4. IMPLEMENTATION

An objective for DataMorphose is to cover as many use-cases on as many platforms as possible. DataMorphose is based on C# language and Gtk# [5] library. It works on Windows (using .NET), Unix and MacOS (using Mono). We present a prototype, which simplifies the analysis phase to the users by representing the data in the necessary details and allowing viewing different parts of it. The prototype supports making some basic modifications on the data and metadata. The migration phase is also possible by exporting the edited data in a new file or files containing the data sets.

4.1 MULTIMODELS

We chose to represent the data in the data model close to the relational database form. Many advantages could be taken from the relational database theory coming at the cost of a relatively low overhead. The data is organized in memory into a database object, which is a set of tables. A table is a set of columns, and the rows are generated on demand from the columns. The relational database organization allows attaching metadata to the data carriers. Metadata object can be attached to a column, a table and a database object. The prototype implements all the constraints this way. For example, if a column in a table must have unique values, this constraint is added to the metadata of the column. In general, providing extra information about the data set in the forms of relations and constraints can be helpful for validation the input upon insertion. In addition the provision of extra information could be useful for the transformation tools upon taking decision whether the transformation is feasible.

The combination between the data model (DataModel) and its corresponding visual model (ShapesModel) adds a layer of complexity when it comes to changing certain properties. Besides DataMorphose's DataModel and ShapesModel, there is a SelectionModel, which serves as a representation, helping the user-system interface by keeping track the selected items. In particular, the ShapesModel stores all graphical objects; the DataModel contains the imported data persistent storage. Multimodeling is a convenient way to avoid duplication of functionalities [6], [7]. For instance, filtering transformation tools require changes in their data model, i.e. specifying the filtering rule, whereas the border color of the table being visualized needs changes in the visualization model. This can become easily very misleading to the end-user when complex changes have to be made on both sides. The issue can be simplified by introducing an object inspector, providing 'runtime' information about the objects (it is on the left hand side of the application on Figure 2). Alongside with a live data grid, the object inspector introduces primitive 'visual debugging' of the transformation steps.

4.2 PLUGINS

Pluggable modules, called plugins allow further loose coupling, encapsulation and separation of concerns. That approach separates the transformation tools from each other and enforces communication only through limited, well-defined channels. The plugin system provides an open environment for many specialized plugins. Categorizing our plugins depending on the steps simplifies the plugin classification and helps the new developers orient themselves in the environment quickly:

1. Data analysis:
    - Plugins for data metrics – their objective is to provide statistics about the data any time. For example, plugins showing the size of the dataset;
    - Plugins for deductions – their objective is to find automatically the constraints between datasets and to suggest them to the user for acceptance. These plugins usually traverse the metadata trying to extract information about the payload. Deductions are applicable in many cases. The deduction system can rely on feedback from user in order to evaluate a deduction as true or false. For example, field names, which contain

   “id”, “pk” usually means restriction on the values in the corresponding set.

2. Data cleansing:
   - Plugins for data validation – they are responsible for the quality of the data. They make sure that the data or the data transformation match the defined set of constraints. Validation can be performed at any time of the migration process. Usually data validation is done as a first step to reduce the migrated set of data and improve its integrity. For instance, these plugins can be implemented as checkers, inspecting if a field has null values or if a set has only unique values;
   - Plugins for data filtering – they are responsible for slicing the data according to given criteria. Filter plugins could also be classified as helpers for any other of the enumerated steps, but here they usually play a central role. For instance, plugins can be used to filter the first 100 values from a given alphabetically ordered set.
3. Metadata restructuring – they are usually combined with the next step because any change in the metadata requires tight integration with the data.
4. Data transformation and mapping – they are accountable for all actions leading to changes in the data. The transformations can be done on the data or the metadata or both. A transformation can specify a relation between two (meta-) data sets. These kinds of transformation are usually named data mappings. Common data transformation is merging or splitting sets, trimming or appending data. Common metadata transformations are adding uniqueness constraint to a set.
5. Data migration – they are responsible for loading or storing the data in a specific format, i.e. responsible for importing and exporting data in different formats. Currently, we support import and export in comma separated values.
6. Data verification – this iterative step is the most tricky to automatize. One of the reasons is the difficulty to prove a migration was successful. Often, only the human migrator could certify a successful migration. The plugins helping this step are in the direction of assisting the user to find deficiencies in the procedures. These plugins provide tools for visual debugging of the data ready to be migrated.

Every change in the model is encapsulated into ‘reversible’ actions. Every transformation can be undone, which encourages the error and trial learning and development. The implementation relies on the Command design pattern [8], which requires a common interface between all transformations.

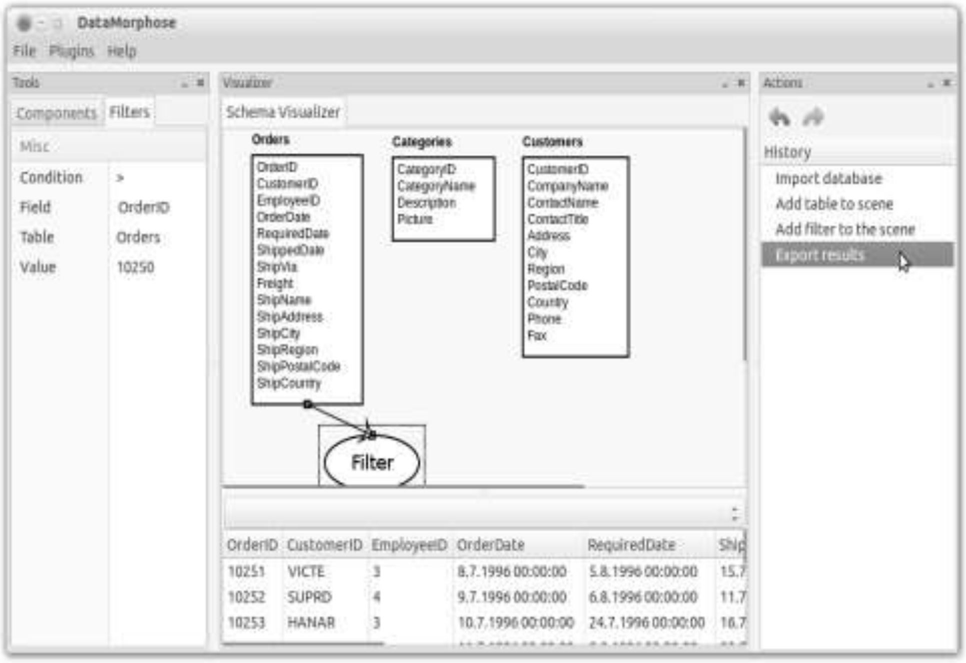
4.3 VISUALIZATION

Mapping a data model carrying the payload to a visualization model allows separating the visualization and interaction algorithms from the core business logic.

In addition, it enables creation of very rich, descriptive and interactive visualizations.

The powerful visualization and interaction support gives us the chance to implement handy data migration steering machinery. Since our target audience is mainly the non-experts and the education, we would like to avoid our users to write code. For this reason we designed a domain-specific visual programming language (VPL), based on data flow. It visualizes predefined transformations steps as components, which can be dragged&dropped on the scratch area and combined together. The user can build, visually, a set of steps on a specified data objects and connect them together into an executable migration program. On each change on the scratch area is all the steps are executed together on sample data, allowing the users to preview the results and tweak the migration program further (see Figure 2).

The specific to data migration VPL, combined with the flexible docking pads form a very user-friendly environment. The user can customize and personalize the layout of the environment such that it fits his/hers specific needs.



Condition	>
Field	OrderID
Table	Orders
Value	10250

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	Shi
10251	VICTE	3	8.7.1996 00:00:00	5.8.1996 00:00:00	15.7
10252	SUPRD	4	9.7.1996 00:00:00	6.8.1996 00:00:00	11.7
10253	HANAR	3	10.7.1996 00:00:00	24.7.1996 00:00:00	16.7

Figure 2. DataMorphose Interface

5. APPLICATION SCENARIOS

Users, without computer science background and data migration experience, are often intimidated by the syntax of programming languages and database queries. However, the situation changes when the objects, describing data (e.g. rows, columns, tables, databases), are represented by corresponding visual elements (e.g. pictures, icons, etc.)

Clients using the prototype, work with graphical objects by dragging and dropping components, linking and editing them in a user-friendly environment. Different structures that store data could be considered as graph nodes, while the links between them are the graph edges. Consequently, this leads us to the visual programming language (VPL) theory [9]. Defining a VPL allows these concepts to be generalized in a common methodology. In the case of DataMorphose, the textual form is usually the input and the output of the data migration process, whereas the graphical representation is an intermediate state that allows the user to manipulate the data regardless the input/output data format.

Another effective usage of the framework could be for educational purposes. Students can take advantage of the test data sets which could be reused as many times as needed. Thus, they learn about the building elements of data as well as the problems and challenges that occur while storing data. It is a good way to make students to understand how important is to plan carefully a data organization and architecture and to keep high quality of the data inside it. Students could be taught of the main principles of data migration methodology and what risks should be considered when executing a data migration.

6. CONCLUSION

We carried out an extensive research in the field of data migration. The outcome of which was a general migration methodology well applying in many cases. We investigated a few other tools, which use different technologies and methods for extracting and/or migrating data. Thus, we gained theoretical and practical knowledge in the domain and contributed for setting the requirements more convenient system.

This work presented a platform-independent framework simplifying a data migration. The combination between reusable tools and a VPL driver is very user-friendly even to non-experts. The chosen approach for the development of the prototype for data migration, results in a flexible and easy-extendable system. It acts as a basis of a powerful tool for transformations and manipulations of data.

Although the prototype is not large, we have a clear vision of the next set of enhancements. One of the important tasks is to improve the user-system interaction and components layout. Among the main goals for the future development of DataMorphose is visual data debugger, which could be an important feature for users operating with large amount of data or complex migration setups.

Another key objective is to make possible the import and export into/of DataMorphose environment in the most popular database formats. That will greatly extend the range and number of users.

REFERENCES

[1] Clarkin, L. and J. Holmes, Enterprise Mashups, *The Architecture Journal*, Vol. 13.

[2] JackBe Corporation, *Presto Wires*, http://www.jackbe.com/products/presto (visited March 2014).

[3] Baycastle Software Ltd, *Dataslave*, http://www.baycastle.co.uk/V2/DataSlave/DataSlave.htm (visited March 2014)

[4] Krasner, G. and S Pope, A cookbook for using the model-view controller user interface paradigm in smalltalk-80, *J. Object Oriented Program*, 1 (3), 1988, 26–49.

[5] Mono Project, *GTK#*, http://www.mono-project.com/GtkSharp

[6] Penev, A., Computer Graphics and Geometric Modelling – A Hybrid Approach, *International Journal of Pure and Applied Mathematics*, 85 (4), 2013, 781–811.

[7] Penev, A., D. Dimov and D. Kralchev, Open Hybrid System for Geometric Modeling, *17th International Conference on Systems for Automation of Engineering and Research*, 2003.

[8] Freeman, E., K Sierra and B Bates, *Head First Design Patterns*, O'Reilly, 2004.

[9] Rozenberg, G., *Handbook of graph grammars and computing by graph transformation: volume I. foundations*, World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

Faculty of Mathematics and Informatics,
University of Plovdiv „Paisii Hilendarski"
236 Bulgaria blvd, Plovdiv, Bulgaria
vvasilev@cern.ch, petya.petrova@hotmail.com, mrtn.vassilev@gmail.com

DATAMORPHOSE: ИНТЕРАКТИВЕН ИНСТРУМЕНТАРИУМ ЗА МИГРАЦИЯ НА ДАННИ

Васил Василев, Петя Петрова, Мартин Василев

Резюме. *Статията представя инструментариум, който определя и автоматизира необходимите стъпки за миграция на данни. Ние правим*

преглед на гъвкава и платформено-независима система, базирана на разширения. Разширенията са категоризирани според изброените им миграционни стъпки. Статията предлага визуален език за програмиране, който помага на неекспертни потребители да управляват процеса на миграция на данни. Ние демонстрираме среда, която разделя модела на данните от модела за визуализация и този за взаимодействие. Работата представя слабо свързани техники на визуализация, обвързващи визуалните представяния с техните съответни невизуални части.