ISSN 2658 – 7505
Выпуск №1, 2019
Электронный научный журнал «Вестник молодёжной науки России»

КОНВЕРТЕР ВХОДНЫХ ВЫХОДНЫХ ДАННЫХ ДЛЯ МАТЕМАТИЧЕСКОГО МОДУЛЯ ЭКСПЕРТНОЙ СИСТЕМЫ

Барковский С.А. *kharitonov.simon@yandex.ru*
Южный федеральный университет Институт компьютерных технологий и информационной безопасности (ИКТИБ), г. Таганрог.

Современные интеллектуальные системы, в том числе экспертные системы, часто имеют в своем составе разные программные модули для решения поставленных задач. Динамический характер и изменчивость линейки продуктов модулей порождает проблему адаптации новых структур входных/выходных данных после их подключения. Возникает потребность преобразования входных/выходных файлов в файлы с определенным видом и структурой, требуемые системой и/или другими подключенными модулями. Наше исследование сосредоточено на решении проблемы автоматизации подключения программных модулей и вычислительных алгоритмов. Мы предлагаем технологию управления входными/выходными данными в виде матриц разных программных модулей и вычислительных алгоритмов, объединенных в единый кластер, с инструментом извлечения, распознавания и преобразования этих структур данных. Предлагаемое решение процесса распознавания матричных данных осуществляется на уровне файловой системы и основано на выработке правил распознавания с использованием образцов структур данных.

INPUT OUTPUT DATA CONVERTER FOR THE MATH ENGINE IN AN EXPERT SYSTEM

Barkovskii. S.A.
Southern Federal University Institute of Computer Technologies and Information Security (ICTIS), Taganrog

Modern intelligent systems, including expert systems (ES), often contain various computational modules. The dynamic nature of applications and variability of the product line of modules generates the compatibility problem of formats, types, and structures of input / output data. In this case, there is a need to transform the input / output files into files with a certain type and structure required by the system and / or other plugged-in computational modules. Our research is focused on solving the problem of automating plugging-in computational modules. We propose a control tool of input / output data of various computational modules united in a single cluster, with an engine for semi-automated extraction, recognition and transformation of these data structures. We present the main phase of data post-processing – recognition, to be used in applications that require the input / output data in matrix form. The proposed solution of matrix data recognition is implemented at the level of the file system and is based on the development of generation rules using samples of data structures.

Introduction

We propose a simple and inexpensive mechanism for semi-automatic control of computational algorithms with heterogeneous input / output data structures. These data structures are matrices represented in different forms (semi-structured and unstructured data, arrays, unnamed lists, etc.). The proposed mechanism is geared towards data recognition and transformation methods for these data structures. Our methods are based on the generation of recognition and transformation rules from the data structure sample. Control is performed at the physical level of file system. The control process does not require special skills of the end-user. There is no need of end user's intervention into control process, except for the final approval of recognized data.

Related work

The need to develop an automatic engine to control external structured, semi-structured and unstructured data from various sources for business analytics is noted in [1]. Nath et al. offered the Semantic Extract-Transform-Load (SETL) engine for data semantic

ISSN 2658 – 7505
Выпуск №1, 2019
Электронный научный журнал «Вестник молодёжной науки России»

transformation based on the combination of Semantic Web and Data Warehouse technologies. Semantic Web provides data in Data Warehouse in XML, XML Schema, RDF, RDF Schema, OWL, CSV and several other formats. The extraction of the corresponding data from several heterogeneous data sources is performed through the extraction function using the R2RML mapping language. The definition of customized mappings from relational data into the Resource Description Framework is performed by the user. The authors in [2] proposed the visual dataflow programming language VisualTPL for generating report understandable to end-users. The basic idea is in the top-down decomposition; when large and complex structured content is divided into simpler layouts to arrange the content in a certain way. The authors used special extraction functions based on canonical tabular data. The engine they offer cannot be used for raw data recognition. VisualTPL includes programming techniques, therefore working with VisualTPL can be difficult to users not familiar with them.

The authors in [3] developed the TabbyXL console application to bring the tables to their canonical relational form. Their method is based on recognition of tagged documents in such formats as Excel, Word or HTML, functional and structural analysis of the logical structure of the table and its further transformation. Cells are defined as a set of specific attributes, such as Location, Style, Content, and Annotation. The method includes the modification of cells in the process of canonicalization. This type of modification can incorrectly change the contents of the cells in the tables, which may lead to incorrect interpretation of the data. For other data structures, data need to be reconfigured into a canonical table format.

Methods

In this section, we show the data post-processing to the document contained the matrix at the file system level. The program is written in C #, for the Windows; the software platform is .NET Framework. In this realization of the program, it is assumed that the matrices can be written in three different types: (i) as an $n \times n$ table, where n is the dimension of the matrix; (ii) as a single line of numbers that are elements of the matrix, including zero ones; (iii) as a table with three columns in which the row number, the column number and the numerical value of the nonzero element of the matrix are set. Our recognition process is brought into play by "input-recognizer.exe" manually or from the wrapper-program session. The diagram of the program components is shown in Figure 1.

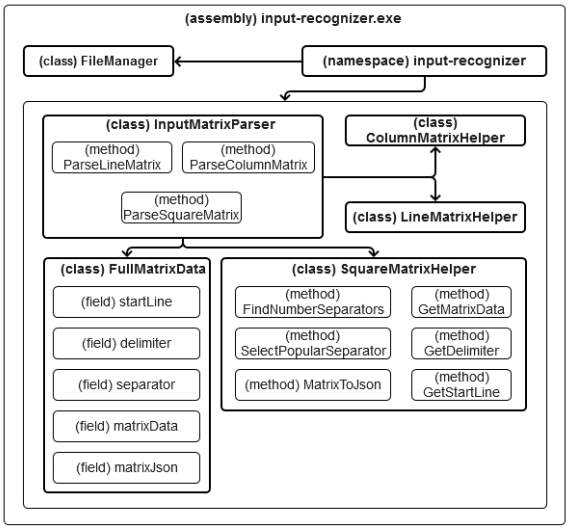

Fig. 1. Deployment diagram for the Input-Recognizer

The program takes the document with a sample of input data and notation (if any). The document and notation can be supplied to the program: (i) directly by the user through the command prompt of the program and (ii) automatically from the controlling wrapper-program in case of using the Input-Recognizer as a library.

Input-Recognizer contains six classes:

1) `FileManager` – downloads and writes the files.
2) `InputMatrixParser` – initializes matrix recognition.
3) `ColumnMatrixHelper` – recognizes the 3rd type matrix.
4) `LineMatrixHelper` – recognizes the 2nd type matrix.
5) `SquareMatrixHelper` – recognizes the 1st type matrix.
6) `FullMatrixData` – configures the output data.

The `InputMatrixParser` class includes three methods: `ParseSquareMatrix`, `ParseLineMatrix`, `ParseColumnMatrix`. These methods allow recognition for matrices written in three types. Method `ParseSquareMatrix(inputMatrix, fileName)` takes two arguments: (1) the text of the input data from the file and (2) the name of this file. Each tool of the `InputMatrixParser` class has six main functions:

1) `FindNumberSeparators(inputMatrix)` – identifies decimal separator types.
2) `SelectPopularSeparator(separators)` – searches and selects the predominant type of decimal separator (creation of the 1st generation rule "Separator").
3) `GetDelimiter(separator, inputMatrix)` – searches and selects the prevailing matrix delimiter (creation of the 2nd generation rule "Delimiter").
4) `GetMatrixData(inputMatrix, delimiter)` – creates the list of matrix cells.
5) `GetStartLine(matrixData, delimiter, inputMatrix)` – searches the initial line of the matrix entry in the document (creation of the 3rd generation rule "StartDrawMatrixAtLine").
6) `MatrixToJson (matrixData, fileName)` – rules-based transformation of the recognized matrix into json-format.

After the completion of all the functions above, we obtain a set of generation rules and the recognized matrix in json-format. We determined that each matrix has the following mandatory attributes, which must be identified in order to recognize the matrix image correctly: (1) Decimal separator – `Separator`; (2) Matrix cell delimiter – `Delimiter`. (3) The initial line of the matrix entry in the document – `StartDrawMatrixAtLine`. (4) Notation with the command description for getting the values of special cells (if any).

Consider the recognition process using the example of the `ParseSquareMatrix` tool to recognize the input data file from the `Fortran` program for computing the influence nodes. The input matrix `(inputMatrix)` of 1st type is shown in Figure 2.

During the recognition process, the following tasks must be solved:

1) Remove the "noise" and recognize matrix elements.
2) Recognize the elements of the matrix that have different decimal separators, and the elements that do not have decimal separators.
3) Create the generation rules regarding mandatory attributes `(Separator, Delimiter, StartDrawMatrixAtLine)`.

ISSN 2658 – 7505
Выпуск №1, 2019
Электронный научный журнал «Вестник молодёжной науки России»

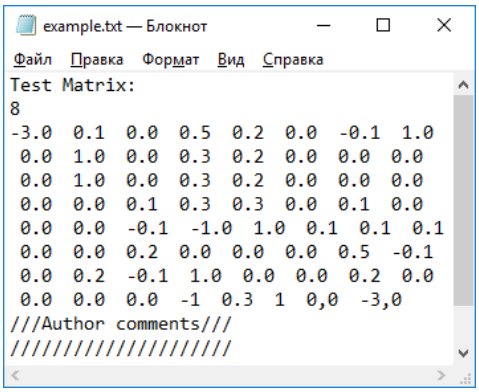

Fig. 2. Primary data of first type matrix

4) Recognize the author's notation (if any), and add to the existing rules the author's additional rules necessary to generate an input file to be transferred to another math algorithm.
5) Transform the recognized matrix into json-format for further work with the matrix data inside the program.
As a result of the work of the program, we will have:
1. Generation rules:
(a) The 1st rule: `Separator: "."` – decimal separator.
(b) The 2nd rule: `Delimiter: "  "` – the delimiter of the matrix cells, equal to two spaces in this case.
(c) The 3rd rule: `StartDrawMatrixLine: "2"` – the number of the line from which filling of the matrix starts (the count starts from zero).
(d) The 4th rule: `NotationRules: "E[0,0] = n"` – an additional rule to write a mandatory elements gotten from notation.
2. JSON-matrix having the following view:

```
[{
"name" : "example.txt",
"n" : "8",
"connections" : [{"from" : 0, "to" : 0, "value" : -3.0},
{…}, …]
}]
```

When a set of generation rules is created user can submit the raw data file for transformation to the required form for the math algorithm. The conversion of the user input data into the input data of the math module is shown in Figure 3.

Conclusion

We developed a prototype of the proposed solution for matrix recognition, a console application "input-recognizer", which can be used either separately by manual controls or as a module in other programs as class library. Our solution is as follows: (i) Recognition of the output data at the physical level; (ii) Generation of rules based on pattern recognition of data structures in the input data file; (iii) The recognition process does not require end-user intervention.

ISSN 2658 – 7505
Выпуск №1, 2019
Электронный научный журнал «Вестник молодёжной науки России»

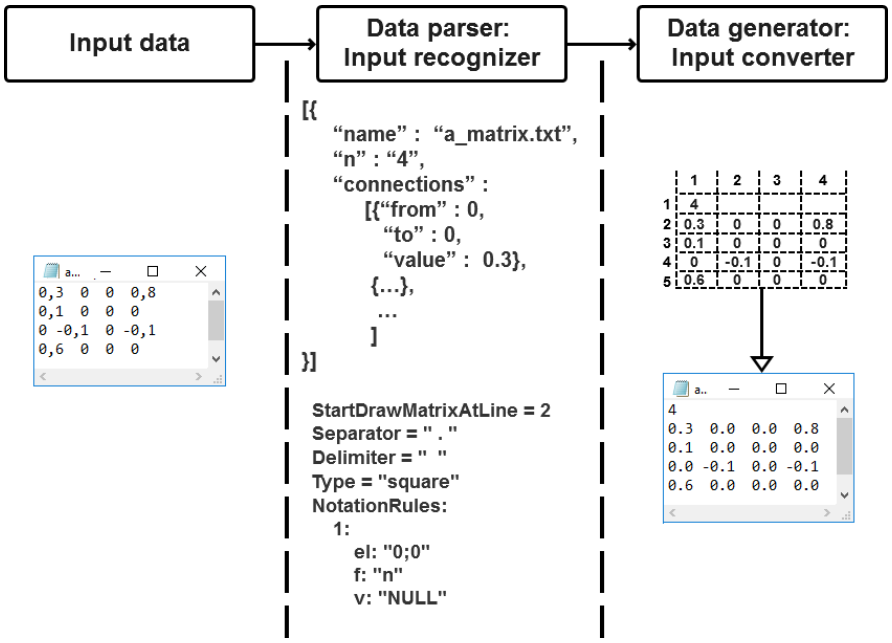

Fig. 3. Conversion of the user input data into the input data of the math module

References

1. Nath, R., Hose, K., Pedersen, T., & Romero, O.: SETL: A programmable semantic extract-transform-load framework for semantic data warehouses. Information Systems, 68, 17-43 (2017). doi: 10.1016/j.is.2017.01.005.
2. Chen, W.-K., & P.-Y., T.: VisualTPL: A visual dataflow language for report data transformation. Journal of Visual Languages and Computing, 25, 210–226 (2014). doi: 10.1016/j.jvlc.2013.11.003.
3. Shigarov, A., & Mikhailov, A.: Rule-based spreadsheet data transformation from arbitrary to relational tables. Information Systems, 71, 123–136 (2017). doi: 10.1016/j.is.2017.08.004.
4. Tselykh, A., Tselykh, L., Vasilev, V., & Barkovskii, S.: Expert system with extended knowledge acquisition module for decision making support. Advances in Intelligent Systems and Computing, 680(2), 21-31 (2017). doi: 10.1007/978-3-319-68324-9.