TabbyXL: Software Platform for Rule-Based Spreadsheet Data Extraction and Transformation

A. Shigarov, V. Khristyuk, A. Mikhailov

Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of the Russian Academy of Sciences, 134 Lermontov St., Irkutsk, Russia, 664033

Abstract

Spreadsheets are widely used in science, engineering, business, and other activities. Overall, they conceal a large volume of data in a form intended to be interpreted by humans. We present a novel software platform facilitated for liberating such data. It provides rule-based spreadsheet data extraction and transformation to a structured form. Its core consists of a flexible table object model and a domain-specific rule language for table analysis. They serve to represent knowledge of table layout and content features, as well as their interpretation depending on transformation goals. This enables processing arbitrary tables originating from various domains. Our empirical results demonstrate that one ruleset can be applied to process arbitrary tables having the same features of layout, style, or content. The paper also describes two applications using the software platform to develop programs for rule-based converting data from arbitrary spreadsheet tables.

Keywords: table understanding, information extraction, unstructured data management, rule-based programming, spreadsheet data, software development

1. Introduction

Many spreadsheet tables are designed to be interpreted by humans. They lack metadata (explicit semantics) needed for computer programs to interpret them as intended by their author or as required by an application. Spreadsheet tools provide a variety of layout structures and formatting styles for presenting tables. Their human-centeredness leads to the fact that arbitrary tables often have an incorrect structure (e.g. one logical cell can be improperly split into several physical cells) and "messy" text data (e.g. typos, non-standardized values, extra spaces).

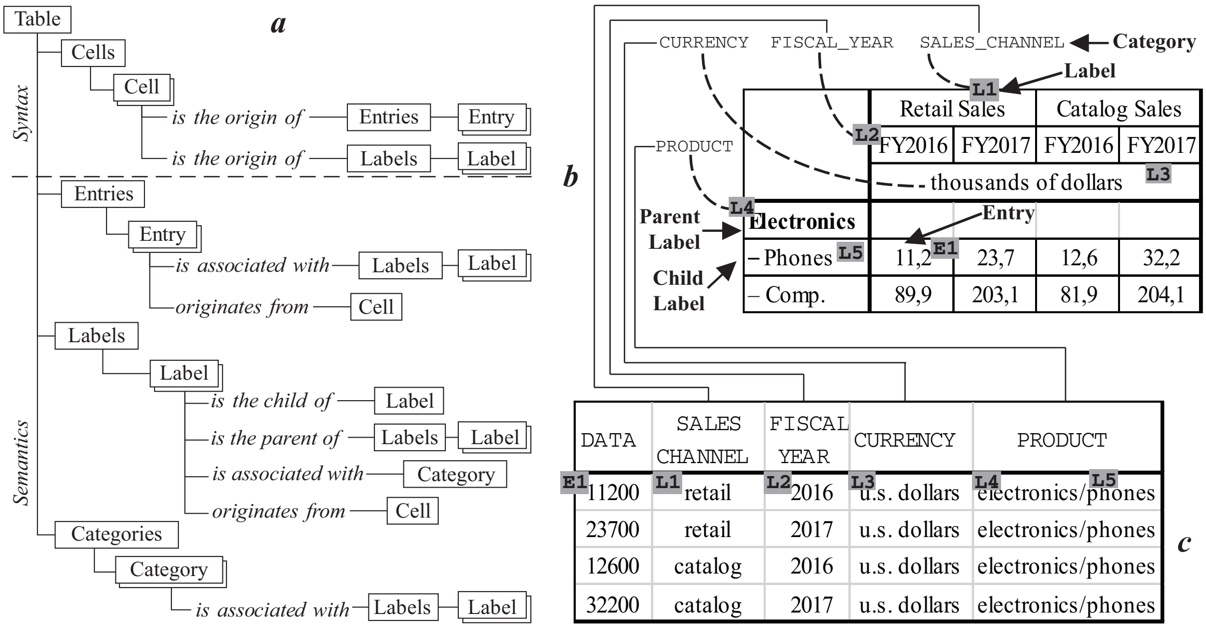

Figure 1: The concept of the table object model — *a*; an instance of an arbitrary table (source) — *b* and in a canonicalized version (target) — *c* (the tag `E1` indicates an entry, the tags `L1`,…,`L5` point out its related labels).

Spreadsheet data can be a valuable source in data science and business intelligence applications. However, difficulties that inevitably arise with extraction data from arbitrary tables often hinder the intensive use of them in these areas. Typically, experts faced with such tasks resort to general-purpose tools. In comparison with the latter, ad hoc software platforms or frameworks can allow shortening software development time, hiding inessential details and focusing on the issues under consideration.

`TabbyXL`[1], a novel software platform, aims at the development and execution of rule-based programs for spreadsheet data extraction and transformation from arbitrary (Fig. 1, *b*) to relational tables (Fig. 1, *c*). The platform implements our approach to rule-based table understanding (i.e. recovering the metadata of arbitrary tables) [1, 2]. It exploits a flexible table object model to represent knowledge of layout and content features, as well as user-defined rules to analyze and interpret tables, depending on transformation goals.

---

[1] `https://github.com/tabbydoc/tabbyxl/releases/tag/v1.0.4`

2. Background

The contemporary data integration solutions (e.g. `Talend`[2], `OpenRefine`[3]) provide some transforming operations (e.g. “unpivoting”, “deduplication”, or “reconciliation”) for flat file databases presented in spreadsheet-like formats. They expect only relational tables as input. Meanwhile we consider the transformation of arbitrary tables designed by humans for humans.

The spreadsheet data extraction and transformation consist in recovering missing metadata describing the structure and content of an arbitrary table. Typically, spreadsheets do not provide all metadata that are needed to interpret table structure and content. There are no functional roles, relationships, and external describing concepts of data items placed in an arbitrary table. Such metadata should be recovered to convert data from arbitrary to relational tables. We refer to the considered problem as *table analysis* (recovering the functional roles and internal relationships) and *interpretation* (recovering the external relationships).

The recent papers are dedicated to the related issues of table analysis and interpretation, e.g. layout features [3, 4, 5], code smells and formulas [6, 7, 8, 9], programming by examples [10, 11, 12], data models [13, 14, 15], linked open data [16, 17], domain-specific [18, 19, 20] and rule-based architectures [2, 21, 22]. There are works [23, 24] with goals similar to ours. They propose methods for transforming spreadsheet tables with predefined layout features to a canonical form based on heuristics [23] and machine-learning [24]. In contrast to them we propose the software platform supporting user-defined table layout features.

Only a few related projects exhibited software they developed. We mention here only those ones that published their software at least partially. The `SSaaPP`[4] project implemented two frameworks for mapping data from a spreadsheet to a relational database, `HaExcel` [25] and `MDSheet` [15], as extensions for `OpenOffice`[5]. The `DeExcelerator`[6] project aimed at development of a framework for information extraction from spreadsheet tables [26]. It is published only partially as `XCellAnnotator`[7], a desktop application for interactively annotating cell regions in Excel files. The `Senbazuru`[8] project on development of a spreadsheet database management system [27] pub-

[2] `https://sourceforge.net/projects/talend-studio`
[3] `http://openrefine.org`
[4] `http://ssaapp.di.uminho.pt`
[5] `https://www.openoffice.org`
[6] `https://wwwdb.inf.tu-dresden.de/misc/DeExcelarator`
[7] `https://github.com/elviskoci/XCellAnnotator`
[8] `http://dbgroup.eecs.umich.edu/project/sheets`

lished `Frame Finder`[9], a software for detecting functional cell regions in a spreadsheet table having a layout called "data frame" [28].

Typically, the related software solutions rely on a predefined table structure. They support only a few widespread layout types of tables with typical functional cell regions (e.g. "stub", "head", "body", "derived"). In contrast to them, we use a domain-independent table model that is not limited by predefined functional cell regions. It is designed for specifying layout features of arbitrary tables in user-defined rules.

Unlike others, our model associates functional roles with data items but not cells. This enables supporting a layout where one cell contains two or more data items (e.g. in some bilingual or statistical tables). Such data items originating from the same cell can be extracted separately and be linked with different external concepts.

Another distinctive feature of our approach consists in the use of user-defined rules for mapping a physical structure of cells (layout, formatting style, and text) to a logical structure (linked functional data items such as entries, labels, and categories). They can be executed by a rule engine or be translated to executable programs in a general-purpose language. Our software platform implements both cases.

3. Software Overview

`TabbyXL` is developed as Java application with the command-line user interface. As an input, it expects a spreadsheet file in Excel (*.xlsx) format containing one or more arbitrary tables and a rule-based program with some user-defined rules for cleansing, analyzing, and interpreting such tabular data. Our software platform uses the rules to transform data from arbitrary to relational tables. As an output, a spreadsheet file containing extracted data in the relational form is generated for each source table.

End-users can exploit the software platform to develop rule-based programs for goal-oriented extraction and transformation of data from arbitrary spreadsheet tables. `TabbyXL` provides two ways to implement and run the user-defined rules. One of them is to write the rules in a general-purpose rule-based language (e.g. Drools[10], Jess[11]) and execute them via an appropriate rule engine that is compatible with JSR-94 (Java Rule Engine API)[12]. The other is to express the rules in CRL [2], our domain-specific language,

[9] `https://github.com/chenzheruc/spreadsheet_framefinder`
[10] `https://www.drools.org`
[11] `https://www.jessrules.com`
[12] `https://www.jcp.org/ja/jsr`

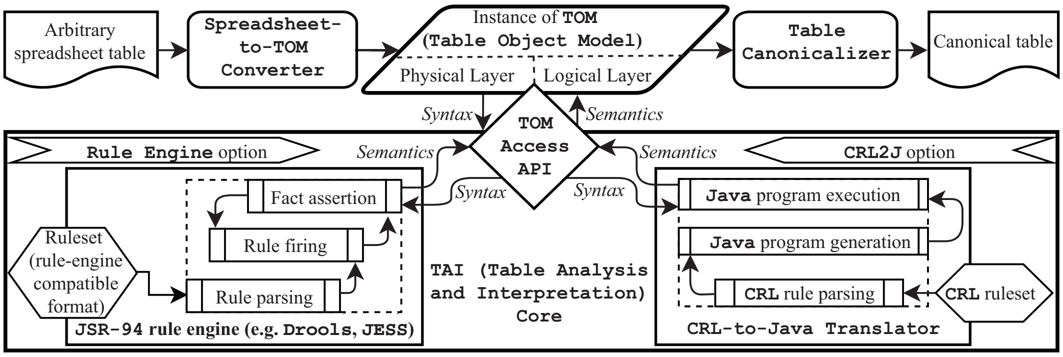

Figure 2: The architecture of the software platform.

and translate them to Java source code that can be compiled and executed as Java program. The detail user manual on developing the rules is available as a part of `TabbyXL` documentation[13].

3.1. Software Architecture

The architecture shown in Fig. 2 determines the interaction of the following main components.

`Table Object Model` (`TOM`) is designed for representing both physical *cells* and logical *data items* (Fig. 1). The model includes two interrelated layers: *physical* (*syntactic*) represented by a collection of cells and *logical* (*semantic*) that consists of three collections of *entries* (values), *labels* (keys), and *categories* (concepts). They are accessible through `TOM-Access API`, an application programming interface.

`Spreadsheet-to-TOM converter` puts cells of an arbitrary table into the physical layer of a `TOM`-instance.

`Table Analysis and Interpretation` (`TAI`) `Core` recovers the logical layer of the `TOM`-instance from its physical layer by performing one of two options. `Rule Engine` option executes a ruleset in an appropriate format with a JSR-94 compatible rule engine. The rule engine matches asserted facts (available data of the `TOM`-instance) against rules to create new facts (recovered data of the `TOM`-instance) and assert them into the working memory. `CRL2J` option provides CRL, our domain-specific language designed for expressing table analysis and interpretation rules. It uses `CRL-to-Java translator` to automatically generate Java source code from CRL rules and compile it to Java bytecode. The generated Java program recovers missing data of the

[13]`https://github.com/tabbydoc/tabbyxl/wiki/crl-rules`

`TOM`-instance.

`Table Canonicalizer` generates a canonicalized version of the processed table from the recovered data of `TOM` logical layer.

3.2. Software Functionality

The software platform enables developing programs for spreadsheet data extraction and transformation, supporting the following functions (actions) for cleansing, analysis, and interpretation of an arbitrary table represented as an instance of `TOM`.

Cell cleansing actions are intended to correct an inaccurate layout and content of a hand-coded table: `merge` combines two adjacent cells when they share one border; `split` divides a merged cell that spans n-tiles (row-column intersections) into n-cells; `set text` modifies a textual content of a cell; `set indent` modifies a text indentation of a cell. *Role analysis* actions aim at recovering entries and labels as functional data items presented in a table: `set mark` annotates a cell with a user-defined tag that can be used in subsequent table analysis; `new entry` (`new label`) creates an entry (label) from a cell content with the use of an optional string processing. *Structural analysis* actions enable recovering pairs of two kinds: entry-label and label-label: `add label` associates an entry with a label; `set parent` binds two labels as a parent and its child. *Interpretation* actions serve to recover label-category pairs: `set category` associates a label with a category; `group` places two labels to one group that can be considered as an undefined category.

The actions aim at mapping table data of syntactic layer to semantic one of `TOM`. They are driven by some rulesets or programs performed with one of the implemented options of `TAI-Core`. Our previous paper [2] explains the listed actions in detail.

4. Implementation

`TabbyXL` implements the presented architecture as follows.

`TOM`, the table object model, is a set of Java classes corresponding to naming conventions of JavaBeans specification. Their public interfaces define `TOM-Access API`. Data items of a `TOM`-instance (cells, entries, labels, and categories) are objects of these classes. This implementation enables us to assert these objects as facts into the working memory of any rule engine that is compatible with JSR-94 specification.

The `CRL2J` option is implemented as a `CRL-to-Java translator`, including the following components: (i) CRL-parser generated by ANTLR development tools, (ii) Java classes to represent `CRL Rule Object Model`, and (iii) utility programs to compile generated Java-programs for spreadsheet data extraction and transformation.

The `Rule Engine` option supports Drools as a rule engine by default. This expects rules expressed in DRL (the general-purpose rule language that is native for Drools) or in a dialect of CRL that is implemented as a domain-specific language (DSL) corresponding to Drools requirements. In the last case, CRL rules presented in DSLR format are automatically translated into DRL format through DSL-specification that defines CRL-to-DRL mappings. Unlike the pure CRL, this dialect supports DRL attributes in rule declarations. The option enables involving any JSR-94 rule engine specified in a configuration file. Our tests confirm that the Jess rule engine can also be used to execute rules represented in CLP format.

5. Empirical Results

This experiment exemplifies the use of our software platform for development and execution of a ruleset to extract data from arbitrary tables that are produced by different authors but pertain to the same document genre. The performance evaluation is based on `Troy200` [29] dataset. It contains 200 arbitrary tables as CSV files collected from 10 different sources of the same genre (government statistical websites). We added accompanying ground-truth data to automate the performance evaluation [30].

We designed a tested ruleset that transforms `Troy200` arbitrary tables into the relational form. It was implemented in three formats: CRL, DSLR (CRL-dialect as DSL for Drools), and CLP (Jess). The ruleset services to recover functional data items (entries and labels) and their relationships (entry-label and label-label pairs). All of its implementations were ran by `TabbyXL` to automatically transform the tested tables into the relational form. The recovered relational tables were the same for all three cases.

The performance evaluation was automatically carried out by comparing the ground-truth data with the results of the ruleset runs. The corrected functional data items and their relationships were compared with recovered ones. We adapted the standard metrics, *recall* and *precision*, as follows:

$$\text{recall} = {}^{|R \cap S|}/_{|S|} \quad \text{precision} = {}^{|R \cap S|}/_{|R|} \qquad (1)$$

Where R is a set of instances (entries, labels, entry-label pairs, or label-label pairs) in a target table and S is a set of instances in the corresponding source table. Table 1 presents the values of these metrics for each type of automatically recovered instances. Among 200 tested tables, only 25 are processed with errors (total 1256 false negatives in 25 tables, and 498 false

Table 1: The performance evaluation results of the tested ruleset for recovering data items and their relationships from tables of `Troy200` dataset.

metrics	entries	labels	entry-label pairs	label-label pairs
recall	0.9813 $\frac{16602}{16918}$	0.9965 $\frac{4842}{4859}$	0.9773 $\frac{34270}{35066}$	0.9389 $\frac{1951}{2078}$
precision	0.9996 $\frac{16602}{16609}$	0.9364 $\frac{4842}{5171}$	0.9965 $\frac{34270}{34389}$	0.9784 $\frac{1951}{1994}$

positives in 14 ones). Only one table was not processed. This resulted in about 72% of all errors.

All data and steps to reproduce this performance evaluation are available as the published dataset [30] and as a part of `TabbyXL` documentation[14]. We also prepared a Dockerfile[15] to build Docker image that contains the required software and data to reproduce the presented empirical results.

We compared our tested ruleset with `MIPS` [23], a state-of-the-art method for segmenting a table into typical functional cell regions. The *accuracy* of table segmentation we obtained was 0.9950 against 0.9899 `MIPS` that authors reported on the same dataset (`Troy200`). We also developed and tested an additional ruleset to compare it with the published results of `Senbazuru` [31] on extracting header hierarchies (label-label pairs) from spreadsheet tables. The testing was performed on a subset of 200 `SAUS` tables randomly selected. The F-score obtained by `TabbyXL` is 0.8657 against 0.8860 reported by `Senbazuru` [31] on `SAUS` tables. Our previous paper [2] presents this comparison in more detail.

Notice that the demonstrated results are close to these state-of-the-art solutions. However, both competitors use a table structure restricted by three predefined functional regions: single-column stub, "pyramid-like" head, and body. Unlike them, our platform allows processing tables with other user-defined layout features (e.g. cut-ins in body, footers, or an inverted "pyramid-like" head).

6. Illustrative Example

Fig. 3 illustrates a simple example of a transformational task that consists in converting tables similar to ones (a and c) to the relational form (b and d). These tables satisfy the following assumptions: $1, \ldots, n$ are entries; $a_1, \ldots, a_m$ are column labels of the category A; $b_1, \ldots, b_k$ are row labels of

[14]`https://github.com/tabbydoc/tabbyxl/wiki/performance-evaluation`
[15]`https://hub.docker.com/r/tabbydoc/tabbyxl`

a1		a2	
b1	1	b4	4
b2	2	b5	NA
b3		b6	6

a

DATA	A	B
1	a1	b1
2	a1	b2
4	a2	b4
6	a2	b6

b

a1		a2		a3	
b1		b3	3	b5	5
b2	2	b4	NA	b6	6

c

DATA	A	B
2	a1	b2
3	a2	b3
5	a3	b5
6	a3	b6

d

Figure 3: An illustrative example of the transformation of arbitrary tables with the same layout features (a and c) to their canonicalized versions (b and d).

a
```
when cell c: c.text.matches("NA")
then set text "" to c
```

b
```
when cell c: (cl % 2) == 0, !blank
then new entry c
```

c
```
when cell c: (cl % 2) == 1
then new label c
```

d
```
when
   entry e
   label l: cell.cr == e.cell.cr
then add label l to e
```

e
```
when
   entry e
   label l: cell.rt == e.cell.rt, cell.cl == e.cell.cl - 1
then add label l to e
```

f
```
when label l: cell.rt == 1
then set category "A" to l
```

g
```
when label l: cell.rt > 1
then set category "B" to l
```

Figure 4: A reference ruleset for transforming tables of the illustrative example shown in Fig. 3: data cleansing — (a), entry generation — (b), label generation — (c), associating entries with column labels — (d), associating entries with row labels — (e), categorizing column labels — (f), and categorizing row labels — (g).

the category B. A reference ruleset designed for this task is shown in Fig. 4. Note the demonstrated version of the ruleset is simplified by excluding `rule` and `end` keywords and some newline characters. Its complete version is available at GitHub[16]. All steps to reproduce this example are presented in the documentation[17]. A more complicated example is also available in the documentation[18].

7. Applications

We used `TabbyXL` in two real applications. The first of them aimed at developing a web-based statistical atlas of the Irkutsk region. Our platform enabled populating a database with tabular data extracted from statistical reports distributed by the Irkutsk Regional Committee of the Russian Federal State Statistics Service. The original tables from the statistical reports were

[16] `https://github.com/tabbydoc/examples/blob/master/tabbyxl/example1/results/crl2j/rules.crl`

[17] `https://github.com/tabbydoc/tabbyxl/wiki/example-1`

[18] `https://github.com/tabbydoc/tabbyxl/wiki/example-2`

presented as Word objects. They were converted to Excel spreadsheet format. Then a ruleset we developed in CRL format was executed by the `CRL2J` option. It recovered functional roles and relationships inside each table. As a result, the original tables were transformed into the canonical form by running the ruleset. Finally, they were converted to CSV format, aggregated, and loaded into a database of the statistical atlas.

The second application was dedicated software development from generating ontologies from data of arbitrary tables used in Industrial Safety Inspection (ISI) services. The platform provided extracting data from arbitrary tables presented in ISI reports. Such reports were originally presented as PDF documents. They described technical diagnostics, analysis (including interpretation) of diagnostics results, calculation of durability and residual resource, etc. Their content and layout were specified by some corporative standards. In the considered case, all tables could be separated into two types by their layout forms. We used two rulesets in CRL format for transforming tables of both arbitrary forms to the canonical form. The extracted data (relational tables) enabled us to generate fragments of a conceptual model as class diagrams in UML notation. Finally, such fragments were aggregated into one conceptual model to construct a knowledge base on inspected objects.

The detail description of both applications (including the workflows, rulesets, samples of real data, and steps to reproduce) is presented as a part of the platform documentation[19].

8. Impact

Data analysis needs structured data. However, in practice, data often are available only in a weakly structured form, such as arbitrary spreadsheet tables. For example, there is a large volume of tabular data presented in statistical reports, financial statements, safety data sheets, or business credit assessments. Many applications of data science and business intelligence potentially can use such data.

The data extraction from semi-structured tabular documents such as spreadsheet workbooks can be a time-consuming process. When there is a necessity to process a large volume of arbitrary tables with various layouts then manual processing should be reduced as much as possible. In recent years, this challenge attracted the attention of the scientific community in the area of document analysis and data management. However, a few efforts

[19]https://github.com/tabbydoc/tabbyxl/wiki/statistical-atlas
https://github.com/tabbydoc/tabbyxl/wiki/industrial-safety-inspection

of the community were devoted to develop and publish software for tasks of the table understanding.

`TabbyXL` shows new possibilities of software development for spreadsheet data extraction and transformation. The existing software packages support only a couple of predefined types of table layouts. They usually embed some rules in their internal algorithms. In contrast to them, our platform implements a general table model and allows extending its functionality with user-defined rules.

Such software can significantly facilitate the data extraction from semi-structured tabular documents. Particularly, the preliminary implementation of our platform was used for filling up a data warehouse with socio-economic data on Mongolian provinces. The current version of the software platform was used in two real ETL (Extract, Transform, Load) workflows for extracting data from arbitrary spreadsheet tables (Section 7). In one case, it enabled populating the database of the web-based statistical atlas from tabular data of government statistical reports. In another case, it facilitated generating ontologies from data of arbitrary tables used in industrial safety inspection.

We believe that the design principles of our platform can be used as a basis for the development of software for the conversion of tabular data from weakly or semi-structured sources to databases.

9. Conclusions

The contribution of the presented software consists of the following results. The implemented platform provides developing and executing rule-based spreadsheet data extraction and transformation programs. In contrast to the existing ETL tools, our platform supports arbitrary tables where an implicit semantics is concealed by a complex of layout, style, and content features.

The novelty of the software platform architecture consists in providing two rule-based ways to implement goal-oriented workflows. In the first case, a ruleset for table analysis and interpretation is expressed in a general-purpose rule language and executed by a JSR-94-compatible rule engine (e.g. Drools or Jess). In the second case, a ruleset expressed in CRL is translated to an executable Java program.

As an important part of the architecture, our two-layered table object model proposes a novel approach to associate functional roles with data items but not cells. Unlike other models, we assume that functional data items can be placed anywhere in a table. Thus, this provides processing such specific table layouts.

CRL, our domain-specific rule-based language, determines queries (conditions) and operations (actions) that are necessary to develop programs for spreadsheet data transformation from an arbitrary to relational form. CRL rules map a physical structure of cells (layout, style and text features) to a logical structure (linked functional data items such as entries, labels, and categories). In comparison with general-purpose rule languages (e.g. DRL or Jess), CRL enables expressing rulesets without any instructions for management of the working memory such as updates of modified facts or blocks on the rule re-activation. This allows end-users to focus more on the logic of table analysis and interpretation than on the logic of the rule management and execution.

While the competitive solutions are limited by a few predefined table layout types, our software platform supports the table analysis and interpretation for various layouts. The empirical results showed that one ruleset (program) developed and performed with the software platform can process arbitrary tables originating from different sources of the same genre.

The limitation of this work is that we do not involve the use of spreadsheet formulas. In practice, many arbitrary tables contain formulas. In particular, they can be used to detect and validate derived data in applications of the table transformation. It would be interesting for further work to incorporate formulas in user-defined rules to recover derived data items and their internal relationships.

The current version of the software platform implements the table interpretation limited by grouping and associating labels with user-defined categories. Further work can overcome this limitation by adding a new functionality based on the named entity recognition and linking. We believe that the linking of extracted tabular data with the global structure of linked open data (LOD cloud) would enable them to be interpreted in terms of external ontologies in third-party software applications.

Acknowledgements

This work was supported by the Russian Science Foundation [grant number 18-71-10001].

Conflict of Interest

The authors declare that there is no conflict of interest.

References

[1] A. Shigarov, Table understanding using a rule engine, Expert Systems with Applications 42 (2) (2015) 929–937. doi:10.1016/j.eswa.2014.08.045.

[2] A. O. Shigarov, A. A. Mikhailov, Rule-based spreadsheet data transformation from arbitrary to relational tables, Information Systems 71 (2017) 123–136. doi:10.1016/j.is.2017.08.004.

[3] E. Koci, M. Thiele, O. Romero, W. Lehner, Table identification and reconstruction in spreadsheets, in: Proc. 29th Int. Conf. Advanced Information Systems Engineering, 2017, pp. 527–541. doi:10.1007/978-3-319-59536-8_33.

[4] Z. Chen, S. Dadiomov, R. Wesley, G. Xiao, D. Cory, M. Cafarella, J. Mackinlay, Spreadsheet property detection with rule-assisted active learning, in: Proc. ACM on Conf. on Information and Knowledge Management, 2017, pp. 999–1008. doi:10.1145/3132847.3132882.

[5] W. Dou, S. Han, L. Xu, D. Zhang, J. Wei, Expandable group identification in spreadsheets, in: Proc. 33rd ACM/IEEE Int. Conf. on Automated Software Engineering, 2018, pp. 498–508. doi:10.1145/3238147.3238222.

[6] F. Hermans, M. Pinzger, A. Deursen, Detecting and refactoring code smells in spreadsheet formulas, Empirical Softw. Engg. 20 (2) (2015) 549–575. doi:10.1007/s10664-013-9296-2.

[7] W. Dou, C. Xu, S. C. Cheung, J. Wei, CACheck: Detecting and repairing cell arrays in spreadsheets, IEEE Transactions on Software Engineering 43 (3) (2017) 226–251. doi:10.1109/TSE.2016.2584059.

[8] D. W. Barowy, E. D. Berger, B. Zorn, ExceLint: Automatically finding spreadsheet formula errors, Proc. ACM Program. Lang. 2 (OOPSLA) (2018) 148:1–148:26. doi:10.1145/3276518.

[9] P. Koch, B. Hofer, F. Wotawa, On the refinement of spreadsheet smells by means of structure information, Journal of Systems and Software 147 (2019) 64–85. doi:10.1016/j.jss.2018.09.092.

[10] D. W. Barowy, S. Gulwani, T. Hart, B. Zorn, FlashRelate: Extracting relational data from semi-structured spreadsheets using examples, SIGPLAN Not. 50 (6) (2015) 218–228. doi:10.1145/2813885.2737952.

[11] R. Singh, S. Gulwani, Transforming spreadsheet data types using examples, SIGPLAN Not. 51 (1) (2016) 343–356. doi:10.1145/2914770.2837668.

[12] Z. Jin, M. R. Anderson, M. Cafarella, H. V. Jagadish, Foofah: Transforming data by example, in: Proc. ACM Int. Conf. Management of Data, 2017, pp. 683–698. doi:10.1145/3035918.3064034.

[13] D. Amalfitano, A. R. Fasolino, P. Tramontana, V. De Simone, G. Di Mare, S. Scala, A reverse engineering process for inferring data models from spreadsheet-based information systems: An automotive industrial experience, in: Data Management Technologies and Applications, 2015, pp. 136–153. doi:10.1007/978-3-319-25936-9_9.

[14] J. Cunha, J. P. Fernandes, J. Mendes, J. Saraiva, Embedding, evolution, and validation of model-driven spreadsheets, IEEE Transactions on Software Engineering 41 (3) (2015) 241–263. doi:10.1109/TSE.2014.2361141.

[15] J. Cunha, M. Erwig, J. Mendes, J. Saraiva, Model inference for spreadsheets, Autom Softw Eng 23 (3) (2016) 361–392. doi:10.1007/s10515-014-0167-x.

[16] D. Ritze, C. Bizer, Matching web tables to dbpedia - A feature utility study, in: Proc. 20th Int. Conf. on Extending Database Technology, 2017, pp. 210–221. doi:10.5441/002/edbt.2017.20.

[17] Z. Zhang, Effective and efficient semantic table interpretation using TableMiner^{+}, Semantic Web 8 (6) (2017) 921–957. doi:10.3233/SW-160242.

[18] M. de Vos, J. Wielemaker, H. Rijgersberg, G. Schreiber, B. Wielinga, J. Top, Combining information on structure and content to automatically annotate natural science spreadsheets, Int. J. Human-Computer Studies 103 (2017) 63–76. doi:10.1016/j.ijhcs.2017.02.006.

[19] T. D. Cao, I. Manolescu, X. Tannier, Extracting linked data from statistic spreadsheets, in: Proc. Int. Workshop Semantic Big Data, 2017, pp. 5:1–5:5. doi:10.1145/3066911.3066914.

[20] A. Swidan, F. Hermans, Semi-automatic extraction of cross-table data from a set of spreadsheets, in: S. Barbosa, P. Markopoulos, F. Paternò, S. Stumpf, S. Valtolina (Eds.), End-User Development, 2017, pp. 84–99. doi:10.1007/978-3-319-58735-6_6.

[21] S. Yang, J. Guo, R. Wei, Semantic interoperability with heterogeneous information systems on the internet through automatic tabular document exchange, Information Systems 69 (2017) 195–217. doi:10.1016/j.is.2016.10.010.

[22] S. Yang, R. Wei, A. Shigarov, Semantic interoperability for electronic business through a novel cross-context semantic document exchange approach, in: Proc. ACM Symposium on Doc. Eng., 2018, pp. 28:1–28:10. doi:10.1145/3209280.3209523.

[23] D. W. Embley, M. S. Krishnamoorthy, G. Nagy, S. Seth, Converting heterogeneous statistical tables on the web to searchable databases, Int. J. Document Analysis and Recognition 19 (2) (2016) 119–138. doi:10.1007/s10032-016-0259-1.

[24] Z. Chen, Information extraction on para-relational data, Ph.D. thesis, University of Michigan, US (2016).

[25] J. Cunha, J. a. Saraiva, J. Visser, From spreadsheets to relational databases and back, in: Proc. ACM SIGPLAN Workshop Partial Evaluation and Program Manipulation, 2009, pp. 179–188. doi:10.1145/1480945.1480972.

[26] J. Eberius, C. Werner, M. Thiele, K. Braunschweig, L. Dannecker, W. Lehner, DeExcelerator: a framework for extracting relational data from partially structured documents, in: Proc. 22nd ACM Int. Conf. on Information & Knowledge Management, 2013, pp. 2477–2480. doi:10.1145/2505515.2508210.

[27] Z. Chen, M. Cafarella, J. Chen, D. Prevo, J. Zhuang, Senbazuru: A prototype spreadsheet database management system, Proc. VLDB Endow. 6 (12) (2013) 1202–1205. doi:10.14778/2536274.2536276.

[28] Z. Chen, M. Cafarella, Automatic web spreadsheet data extraction, in: Proc. 3rd Int. Workshop Semantic Search Over the Web, 2013, pp. 1:1–1:8. doi:10.1145/2509908.2509909.

[29] G. Nagy, TANGO-DocLab web tables from international statistical sites (Troy_200), 1, ID: Troy_200_1 (2016).
URL `http://tc11.cvc.uab.es/datasets/Troy_200_1`

[30] A. Shigarov, V. Khristyuk, TabbyXL: Dataset for the performance evaluation of a software platform for rule-based spreadsheet data extraction

and transformation, Mendeley Data, v5 (2018).
URL http://dx.doi.org/10.17632/ydcr7mcrtp.5

[31] Z. Chen, M. Cafarella, Integrating spreadsheet data via accurate and low-effort extraction, in: Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2014, pp. 1126–1135. doi:10.1145/2623330.2623617.

Required Metadata

Current code version

Nr.	Code metadata description	Description
C1	Current code version	1.0.4
C2	Permanent link to code/repository used of this code version	*https : //github.com/tabbydoc/tabbyxl/releases/tag/v*1.0.4
C3	Legal Code License	Apache License 2.0
C4	Code versioning system used	Git
C5	Software code languages, tools, and services used	Java
C6	Compilation requirements, operating environments & dependencies	Java Development Kit 8 or more, Apache Maven
C7	If available Link to developer documentation/manual	*https : //github.com/tabbydoc/tabbyxl/wiki*
C8	Support email for questions	shigarov@icc.ru

Table 2: Code metadata (mandatory)

Current executable software version

Nr.	(executable) Software metadata description	Description
S1	Current software version	1.0.4
S2	Permanent link to executables of this version	*https : //github.com/tabbydoc/tabbyxl/releases/tag/v*1.0.4
S3	Legal Software License	Apache License 2.0
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	Java SE Runtime Environment 8 or more
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	*https : //github.com/tabbydoc/tabbyxl/wiki*
S7	Support email for questions	shigarov@icc.ru

Table 3: Software metadata (optional)