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

Alexey Shigarov    Vasiliy Khristyuk
Andrey Mikhailov   Viacheslav Paramonov

Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of the Russian Academy of Sciences

shigarov@icc.ru

October 2019

* This work was supported by the Russian Science Foundation, grant number 18-71-10001
Table of Contents

1 Introduction
   - Motivation
   - Background
   - Contribution

2 Approach

3 Software Platform

4 Empirical Results

5 Application Experience

6 Conclusions & Further Work
Motivation

About arbitrary spreadsheet tables
- A large volume of valuable data for science and business applications
- A big variety of layout, style, and content features
- Human-centeredness (incorrect structure and messy content)
- No explicit semantics for interpretation by computers

Challenges
- How to extract tables from worksheets
- How to recognize and correct cell structure anomalies
- How to recover semantics needed for the automatic interpretation
- How to conceptualize extracted data by using external vocabularies
Background

*Table understanding* [Hurst, 2001] includes the following tasks

1. **Extraction** — detecting a table and recognizing the physical structure of its cells
2. **Role analysis** — extracting functional data items from cell content
3. **Structural analysis** — recovering internal relationships between extracted functional data items
4. **Interpretation** — linking extracted functional data items with external vocabularies (general-purpose or domain-specific ontologies)
The related issues of the *table analysis and interpretation*

- **Layout properties** [Koci et al., 2017, Chen et al., 2017, Dou et al., 2018]
- **Code smells and formulas**
  [Hermans et al., 2015, Dou et al., 2017, Barowy et al., 2018, Koch et al., 2019]
- **Programming by examples**
  [Barowy et al., 2015, Singh and Gulwani, 2016, Jin et al., 2017]
- **Data model inference**
  [Amalfitano et al., 2015, Cunha et al., 2015, Cunha et al., 2016]
- **Linked Open Data** [Ritze and Bizer, 2017, Zhang, 2017]
- **Domain-specific models**
  [de Vos et al., 2017, Cao et al., 2017, Swidan and Hermans, 2017]
- **Rule-based programming**
  [Yang et al., 2017, Shigarov and Mikhailov, 2017, Yang et al., 2018]
The projects with goals similar to ours

1. **TANGO**¹ (Data Extraction Group, Brigham Young University) 2005 – 2016
   - Heuristics-based role analysis (pre-defined functional cell regions) [Embley et al., 2016]

2. **Senbazuru**² (Database Research Group, University of Michigan) 2013 – 2017
   - ML-based role analysis (pre-defined functional cell regions) [Chen, 2016]
   - ML-based structural analysis (pre-defined layout properties of the header hierarchy) [Chen, 2016]

3. **DeExcelerator**³ (Dresden Database Systems Group, TU Dresden) 2013 – 2019
   - ML-based layout identification [Koci et al., 2016]
   - Heuristics-based role and structural analysis (pre-defined functional cell regions) [Koci et al., 2017, Koci et al., 2018]

---

¹[https://tango.byu.edu](https://tango.byu.edu)
²[http://dbgroup.eecs.umich.edu/project/sheets](http://dbgroup.eecs.umich.edu/project/sheets)
³[https://wwwdb.inf.tu-dresden.de/research-projects/deexcelerator](https://wwwdb.inf.tu-dresden.de/research-projects/deexcelerator)
**Contribution**

TabbyXL is a software platform aiming at the development and execution of rule-based programs for spreadsheet data extraction and transformation from arbitrary \((a)\) to relational tables \((b)\)

**Novelty**

- **Table object model** assigning roles to data items, not cell
- **CRL**, domain-specific language to express user-defined rules for table analysis and interpretation
- **CRL-to-Java translator** to synthesize executable programs for spreadsheet data transformation
Table of Contents

1 Introduction

2 Approach
   - User-Defined Rules
   - Table Object Model
   - Cell Cleansing
   - Role Analysis
   - Structural Analysis
   - Interpretation
   - Illustrative Example

3 Software Platform

4 Empirical Results

5 Application Experience

6 Conclusions & Further Work

7 References
The user-defined rules map the physical structure into the logical structure of a table

- **WHEN-part** queries facts about the structure by using constraints
- **THEN-part** modifies available facts and asserted new ones

The facts are represented by items of the *table object model*

The rules can be expressed in a rule-based language (e.g. Drools\(^4\), Jess\(^5\), or CRL\(^6\))

---

\(^4\)https://www.drools.org

\(^5\)https://jessrules.com

\(^6\)https://github.com/tabbydoc/tabbyxl/wiki/crl-language
## Table Object Model

### Physical Layer
Cells characterized by layout, style, and content features

### Logical Layer
Functional data items and their relationships:
- entries (values)
- labels (keys)
- categories (concepts)
- entry-label pairs
- label-label pairs
- label-category pairs
CRL Grammar

rule       = 'rule' <a Java integer literal> 'when' condition
            'then' action 'end' <EOL> {rule} <EOF>
condition  = query identifier [':' constraint {',' constraint}
               [';', assignment {',' assignment}] <EOL> {condition}
constraint = <a Java boolean expr>
assignment = identifier ':' <a valid Java expr>
query     = 'cell' | 'entry' | 'label' | 'category' | 'no cells' |
            'no entries' | 'no labels' | 'no categories'
action    = merge | split | set text | set indent | set mark |
            new entry | new label | add label | set parent |
            set category | group <EOL> {action}
merge     = 'merge' identifier 'with' identifier
split     = 'split' identifier
set text  = 'set text' <a Java string expr> 'to' identifier
set indent = 'set indent' <a Java integer expr> 'to' identifier
set mark   = 'set mark' <a Java string expr> 'to' identifier
new entry  = 'new entry' identifier ['as' <a Java string expr>]
new label  = 'new label' identifier ['as' <a Java string expr>]
add label  = 'add label' identifier | (<a Java string expr>
            'of' identifier | <a Java string expr>)
            'to' identifier
set parent = 'set parent' identifier 'to' identifier
set category = 'set category' identifier | <a Java string expr>
                'to' identifier
group     = 'group' identifier 'with' identifier
identifier = <a Java identifier>
Cell Cleansing

The actions 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

Example

```
when
  cell corner: cl == 1, rt == 1, blank
  cell c: cl > corner.cr, rt > corner.rb
then
  split c
```
The actions recover 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

Example

```plaintext
when
cell corner: cl == 1, rt == 1, blank
cell c: cl > corner.cr, rt > corner.rb
then
new entry c
```
The actions recover 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

Example

```plaintext
when
  cell c1: cl == 1
  cell c2: cl == 1, rt > c1.rt, indent == c1.indent + 2
no cells: cl == 1, rt > $c1.rt, rt < $c2.rt, indent == $c1.indent
then
  set parent c1.label to c2.label
```
Interpretation

The 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

Example

```
when
  label l1: cell.mark == "stub"
  label l2: cell.mark == "stub", cell.rt == l1.cell.rt
then
  group l1 with l2
```
Illustrative Example

The transformation of arbitrary tables with the same layout features (a and c) to their canonicalized versions (b and d)

The ruleset for the cell cleansing (a), role analysis (b, c), structural analysis (d, e), and interpretation (f, g)

\[
\begin{align*}
\text{a} & \quad \text{when cell c: c.text.matches("NA")} \quad \text{then set text "" to c} \\
\text{c} & \quad \text{when cell c: (cl \% 2) == 1} \quad \text{then new label c} \\
\text{e} & \quad \text{when entry e} \\
\text{e} & \quad \text{label l: cell.rt == e.cell.rt, cell.cl == e.cell.cl - 1} \quad \text{then add label l to e} \\
\text{f} & \quad \text{when label l: cell.rt == 1} \quad \text{then set category "A" to l} \\
\text{g} & \quad \text{when label l: cell.rt > 1} \quad \text{then set category "B" to l}
\end{align*}
\]

This example is reproducible at https://codeocean.com/capsule/5326436
# Table of Contents

1. Introduction
2. Approach
3. Software Platform
   - Architecture
   - CRL2J Translation
4. Empirical Results
5. Application Experience
6. Conclusions & Further Work
7. References
Two options are provided

**Rule Engine option**
Executing a ruleset in an appropriate format with a JSR-94 compatible rule engine (e.g. Drools, Jess)

**CRL2J option**
Translating a ruleset expressed in CRL to an executable Java program
CRL2J Translation

Workflow for generating a Maven-project of a spreadsheet data transformation program

Start

Generating an hierarchy of project directories

Generating POM file

Generating Java source code for spreadsheet data transformation program by CRL rules

Generating main-class of the project

End

Workflow for translating a CRL ruleset to Java source code

Start

Parsing CRL ruleset

Abstract syntax tree

Building rule object model of the ruleset

Instance of the rule object model

Synthesizing source code

Java source code for the spreadsheet data transformation

End
CRL2J Translation

In the Workflow

- CRL Rules
  - Lexer & Parser
    - Abstract Syntax Tree
      - Rule Model Builder
        - Rule Object Model
          - App Code Generator
            - App Source Code

Rule Object Model

- Rule
  - Conditions
    - Condition
      - Identifier
        - Data Type
        - Constraints
          - Constraint
            - Assignments
              - Expression
                - Identifier
                  - Expression

- Actions
  - Action
    - Code Generator
      - Action Type
        - is one of
          - merge
          - split
          - set text
          - set indent
          - set mark
          - new entry
          - new label
          - add label
          - set parent
          - set category
          - group
Example (Source Rule)

when
    cell corner: cl == 1, rt == 1, blank
    cell c: cl > corner.cr, rt > corner.rb, ! marked
then
    set mark "@entry" to c
    new entry c

Example (Fragment of the Generated Java Code)

... Iterator<CCell> iterator1 = getTable().getCells();
while (iterator1.hasNext()) {
    corner = iterator1.next();
    if ((corner.getCl() == 1) && (corner.getRt() == 1) && ... 
        Iterator<CCell> iterator2 = getTable().getCells();
    while (iterator2.hasNext()) {
        ...

# Table of Contents

1. Introduction
2. Approach
3. Software Platform
4. Empirical Results
   - Performance Evaluation
   - Comparison with Others
5. Application Experience
6. Conclusions & Further Work
7. References

Shigarov et al. (ISDCT SB RAS) | TabbyXL | October 2019
### Performance Evaluation

The results of the transformation of 200 tables of Troy200 dataset [Nagy, 2016]

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Role analysis</th>
<th>Structural analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type of instances</td>
<td></td>
</tr>
<tr>
<td></td>
<td>entries</td>
<td>labels</td>
</tr>
<tr>
<td>Recall</td>
<td>0.9813</td>
<td>0.9965</td>
</tr>
<tr>
<td>Precision</td>
<td>0.9996</td>
<td>0.9364</td>
</tr>
<tr>
<td>F-score</td>
<td>0.9904</td>
<td>0.9655</td>
</tr>
</tbody>
</table>

#### Metrics

\[
\text{recall} = \frac{|R \cap S|}{|S|} \quad \text{precision} = \frac{|R \cap S|}{|R|}
\]

*S* is a set of instances in a source table, *R* is a set of instances in its canonical form.

All data and steps to reproduce the results are available at [http://dx.doi.org/10.17632/ydcr7mcrtp.5](http://dx.doi.org/10.17632/ydcr7mcrtp.5)
Performance Evaluation

The comparison of the running time by using TabbyXL with three different options for transforming 200 tables of Troy200 dataset [Nagy, 2016]

<table>
<thead>
<tr>
<th>Running time of</th>
<th>CRL2J</th>
<th>Drools</th>
<th>Jess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruleset preparation ($t_1$)</td>
<td>2108* ms</td>
<td>1711† ms</td>
<td>432† ms</td>
</tr>
<tr>
<td>Ruleset execution ($t_2$)</td>
<td>367** ms</td>
<td>1974‡ ms</td>
<td>4149‡ ms</td>
</tr>
</tbody>
</table>

* $t_1$ — a time of parsing and compiling the original ruleset into a Java program
** $t_2$ — a time of executing the generated Java program
† $t_1$ — a time of parsing the original ruleset and adding the result into a rule engine session
‡ $t_2$ — a time of asserting facts into the working memory and matching rules against the facts

For testing, we used 3.2 GHz 4-core CPU
## Comparison with Others

### Role Analysis
- **Contest task**: The segmentation of a table into typical functional cell regions
- **Testing dataset**: Troy200 [Nagy, 2016]
- **Contestant**: MIPS (TANGO) [Embley et al., 2016]
- **Accuracy**: MIPS (TANGO) — 0.9899 vs. TabbyXL — 0.9950

### Structural Analysis
- **Contest task**: The extraction of header hierarchies from tables
- **Testing dataset**: A random subset of SAUS\(^a\)
- **Contestant**: Senbazuru [Chen and Cafarella, 2014]
- **F-score**: Senbazuru — 0.8860 vs. TabbyXL — 0.8657

\(^a\)http://dbgroup.eecs.umich.edu/project/sheets/datasets.html
Populating a web-based statistical atlas of the Irkutsk region — (b) via extracting data from government statistical reports — (a)

The more detail can be found at https://github.com/tabbydoc/tabbyxl/wiki/statistical-atlas
Generating conceptual models — (b) from arbitrary tables presented in industrial safety inspection reports — (a)

The more detail can be found at https://github.com/tabbydoc/tabbyxl/wiki/industrial-safety-inspection
1 Introduction
2 Approach
3 Software Platform
4 Empirical Results
5 Application Experience
6 Conclusions & Further Work
7 References
Conclusions & Further Work

- Impact on software development for spreadsheet data management
  - Table object model associating functional roles with data items
  - Table analysis and interpretation driven by user-defined rules
  - Formulated actions to recover missing semantics of arbitrary tables
  - Translation of rules to executable spreadsheet transformation programs

- Limitations
  - The inaccurate cell structure prevents the table analysis
  - The very limited interpretation (without external vocabularies)

- Further work
  - Rearrangement of cell structure by using visual (human-readable) cells
  - Detecting derived data by spreadsheet formulas
  - Enriching the table analysis by named entity recognition
  - Linking extracted data items with LOD cloud
# Table of Contents

1. Introduction
2. Approach
3. Software Platform
4. Empirical Results
5. Application Experience
6. Conclusions & Further Work
7. References
A reverse engineering process for inferring data models from spreadsheet-based information systems: An automotive
industrial experience.

ExceLint: Automatically finding spreadsheet formula errors.

FlashRelate: Extracting relational data from semi-structured spreadsheets using examples.

Extracting linked data from statistic spreadsheets.

*Information Extraction on Para-Relational Data.*
PhD thesis, University of Michigan, US.

Integrating spreadsheet data via accurate and low-effort extraction.
In *Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 1126–1135.

Spreadsheet property detection with rule-assisted active learning.
In *Proc. ACM on Conf. on Information and Knowledge Management*, pages 999–1008.


Thanks

Read more about the project at http://td.icc.ru

The project source code is available at https://github.com/tabbydoc/tabbyxl