Chisel: Sculpting Tabular and Non-Tabular Data on the Web

Johannes Doleschal
University of Bayreuth
johannes.doleschal@uni-bayreuth.de

Nico Höllerich
University of Bayreuth
nico.hoellerich@uni-bayreuth.de

Wim Martens
University of Bayreuth
wim.martens@uni-bayreuth.de

Frank Neven
Hasselt University and
transnational University of Limburg
frank.neven@uhasselt.be

ABSTRACT

Chisel is a tool for flexible manipulation of CSV-like data, motivated by the recent effort of the World Wide Web Consortium (W3C) towards a recommendation for tabular data and metadata on the Web. In brief, Chisel supports an expressive built-in schema language for CSV-like data, that can handle both tabular and non-tabular data. Furthermore, it supports a simple programming language for transforming tabular and non-tabular CSV-like data.

In the demo, we showcase the system for specifying and validating schemas, building transformations, and setting up a pipeline for automatic conversion of "wild" CSV-like data into structured tabular data. We present use cases for Chisel specifically targeted at exemplifying the ease of specifying, modifying, and understanding Sculpt schemas as well as extracting and transforming data.

CCS CONCEPTS

• **Information systems → Semi-structured data**; **Data extraction and integration**;

KEYWORDS

CSV, Schema languages, Semi-structured data

ACM Reference Format:
Johannes Doleschal, Nico Höllerich, Wim Martens, and Frank Neven. 2018. Chisel: Sculpting Tabular and Non-Tabular Data on the Web. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3184558.3186963

1 INTRODUCTION

> *Every block of data has structure inside and it is the task of the data wrangler to extract it.*
> *– Michelangelo (paraphrased)*

Despite the availability of numerous standardized formats for semi-structured and semantic web data such as XML, RDF, and JSON, a very large percentage of data and open data published on the Web remains in a CSV-like format. In fact, Jeni Tennison, co-chair of the W3C CSV on the Web working group, claims that much of this data is very cumbersome to work with, i.e., "2/3rds of CSV files on data.gov.uk aren't machine readable [with standard tools]" [9]. The current recommendations [6, 10] of the W3C for a model and metadata format for *tabular data* deal with a subset of CSV-like data but still cannot capture some data fragments in their use case scenarios [8] where rows can have different numbers of columns or where the number of columns in "well-formed" data is not bounded by a constant.[1] We refer to such data as *non-tabular*. Conversely, we use the term *tabular data* to refer to data that is more rigid and where every row has the same, fixed number of columns. We note that the W3C recommendation for meta-data for tabular data on the Web only focuses on *tabular data*.[2] We use the term *well-formed* to refer to data that satisfies the associated *schema*. Well-formed data can therefore be tabular or non-tabular, if the schema language can describe non-tabular data.

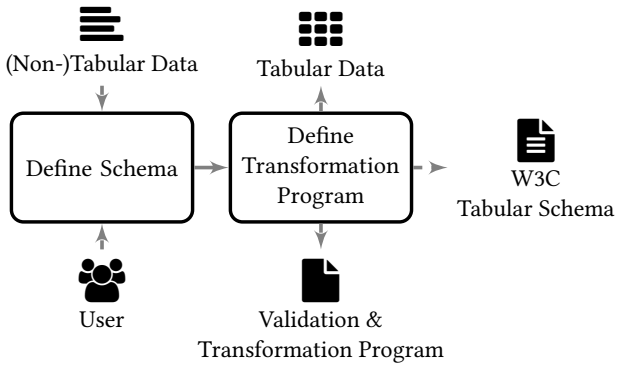

Figure 1: Chisel data wrangling process.

Our system Chisel supports a schema language that is powerful enough to describe all data fragments in the W3C use cases. It also supports a transformation language that can transform this data into a more structured (e.g., tabular) format that *can* be handled by other tools such as the W3C metadata format, SQL-on-Hadoop systems, etc. Since we work with both a schema and a transformation language, it is easy to set up a pipeline that automatically transforms new data that matches the schema. The source code and the tool can be obtained at https://github.com/PoDMR/Chisel.

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3186963

[1] The data can still be *well-formed* if there does not exist an a priori bound on the number of columns, e.g., as in "row 1 is a sequence of temperature measurements". (We even found data containing rows with over 90.000 columns.)
[2] Actually, the CSV on the Web working group defined non-tabular data as out of scope for the first recommendation [8, 3.3 Deferred requirements].

Table 1: Fragment of a CSV-like file, inspired by Use Case 13 in [8].

	1	2	3	4	5	6	7	8	9	10
1	subject	predicate	object	provenance						
2	:e4	type	PER							
3	:e4	mention	"Bart"	D00124	283–286					
4	:e4	mention	"JoJo"	D00124	145–149	0.9				
5	:e4	per:sibling	:e7	D00124	283–286	173–179	274–281			
6	:e4	per:age	"10"	D00124	180–181	173–179	182–191	0.9		
7	:e4	per:parent	:e9	D00124	180–181	381–380	399–406	D00101	220–225	230–233

```
TYPES
smallint:   base="integer" minimum="0" maximum="500"
rdfIRI:     base="IRI"
label:      base="string"
docID:      "D00" smallint
span:       smallint "-" smallint
certainty:  base="float" minimum="0" maximum="1.0"
REGIONS
provenanceRegion = col("provenance")/right*;
content = col("subject")/right*;
RULES
row(1) -> "subject" "predicate" "object" "provenance";
col("subject")   -> rdfIRI;
col("predicate") -> rdfIRI | label;
col("object")    -> rdfIRI | label;
provenanceRegion -> (docID span+ certainty?)*;
primary key rdfTriple: col("subject") {self, right, right/right};
```

Figure 2: Schema for files of the type in Table 1.

Roughly, the demo will showcase the data wrangling process in Figure 1. Chisel allows the user to define a schema for tabular or non-tabular CSV-like data using the schema language Sculpt [5]. Building further on the schema, the user can define programs to transform the data into more rigid tabular data. The validation and transformation program then can be used to set up a "parse and transform" pipeline. This pipeline enables automatic validation and transformation of new input data into tabular data that corresponds to a W3C Tabular Meta Data file (in JSON, which we can automatically generate), allowing further processing by any tools that adhere to the W3C Tabular Data on the Web initiative. Internally, Chisel uses an extension of the schema language Sculpt [5], which is heavily based on parsing cells by regexes.

2 DEFINING SCHEMAS WITH CHISEL

We illustrate Chisel and Sculpt by means of one example, which is inspired by Use Case 13 of the CSV on the Web Use Cases [8]. Table 1 shows an example CSV-file, to which we added row numbers on the left and column numbers at the top. The original file uses tab (`\t`) as a column delimiter and newline (`\n`) as row delimiter. The rows and columns divide the document into *cells*. In this example, rows can have different numbers of cells, e.g., row two has three cells, whereas row three has five. In Use Case 13, which describes a data extraction scenario, "a single row [...] comprises a triple (subject-predicate-object), one or more provenance references and an optional certainty measure" [8]. In Table 1, we see that the provenance information includes a document ID (e.g., the value D00124), pairs of string offsets within the document (e.g., 283–286), and an optional float representing a certainty measure (e.g., 0.9). This information can be repeated for several documents as it is the case in row seven. Since there may not be an a priori bound on the number of columns that are needed for representing the provenance information, the kind of data in Table 1 is *non-tabular*.

Sculpt [5] is designed for describing *tabular and non-tabular* CSV-like data in a flexible way. For the implementation we extended Sculpt by adding native data types (using the keyword `TYPES`), primary and foreign keys, and a data transformation language. Figure 2 contains an example schema for the data of Table 1. The schema is divided into three parts, defined by the keywords `TYPES`, `REGIONS`, and `RULES`.[3]

In `TYPES` the schema designer can define data types for the content in cells. The type definitions are based on the types used by the W3C in [10] (users can add custom base types if desired).

The second part, `REGIONS`, is optional and builds on top of type definitions to define named regions of cells in the data. It consists of rules of the form

```
<name> = <region selection expression>
```

A region selection expression selects cells by navigating through the table. For instance, our schema has the rule

```
provenanceRegion = col("provenance")/right*;
```

Here, the right hand side first selects the column of the cell matching `"provenance"` (by the subexpression `col("provenance")`) and then collects all cells zero or more steps to the right (using the `right*` operator). In the data, the selected region is the rectangular area with the cell in row 2, column 4 in its top left corner and the cell in row 7, column 10 in the bottom right corner. In the remainder of the schema, this area can be referred to by the name `provenanceRegion`. Formally, the schema language allows *propositional dynamic logic* [3], a very powerful XPath-like [7] navigational language, to define regions.

The last part, `RULES`, is the heart of the schema. It is a set of rules of the form

```
<region selection expression>
        -> <regular type expression>;
```

where `<region selection expression>` evaluates to a region R (i.e., a set of cells) in the data and `<regular type expression>` is a regular expression over constant values and types that describes the permitted structure of R. The first rule in our example only uses constant values and states that the first row of the data has four cells, matching the strings `"subject"`, `"predicate"`, `"object"` and `"provenance"` respectively. The second rule states that each cell below the `"subject"` cell must be of type `rdfIRI`. The most interesting rule is the fifth one, which says that every row[4] in the `provenanceRegion` region must match the expression `(docID span+ certainty?)*`.[5] In our sample data, this is indeed the case as every row in the region is an iteration of blocks consisting of `docID`, at least one `span` and an optional `certainty`. In general,

[3] There are further optional parts for defining delimiters, tokens, and transformation programs. We come back to these later.

[4] Using a different arrow `=>` instead of `->` it is also possible to describe the *entire region* by one expression, instead of describing each separate row (cfr. [5]).

[5] Our language has the power to add that the certainty value must be at least, say, 0.75.

Algorithm 1 An example transformation program for the data in Table 1. Here, content is the region, containing everything besides the first row of the input file.

```
1: TRANSFORMATION PROGRAM
2: output("subject", "predicate", "object", "docID", "from", "to", "certainty");
3: for each (s,p,o,x) in content.is(rdfIRI, label, label, SOMETHING*)
4:     if (x is empty) then output(s, p, o, "", "", "", "")
5:     for each (doc, spans, cert) in x.split(docID, span*, certainty?)
6:         for each (sp) in spans.split(span)
7:             output(s, p, o, doc, sp.smallint<0>, sp.smallint<1>, cert)
```

Table 2: Output of Algorithm 1 when applied to the data from Table 1.

	1	2	3	4	5	6	7
1	subject	predicate	object	docID	from	to	certainty
2	:e4	type	PER				
3	:e4	mention	"Bart"	D00124	283	286	
4	:e4	mention	"JoJo"	D00124	145	149	0.9
5	:e4	per:sibling	:e7	D00124	283	286	
6	:e4	per:sibling	:e7	D00124	173	179	
7	:e4	per:sibling	:e7	D00124	274	281	
8	:e4	per:age	"10"	D00124	180	181	0.9
9	:e4	per:age	"10"	D00124	173	179	0.9
10	:e4	per:age	"10"	D00124	182	191	0.9
11	:e4	per:parent	:e9	D00124	180	181	
12	:e4	per:parent	:e9	D00124	381	380	
13	:e4	per:parent	:e9	D00124	399	406	
14	:e4	per:parent	:e9	D00101	220	225	
15	:e4	per:parent	:e9	D00101	230	233	

region selection expressions are much more powerful than what we showed in the example schema (cf. Figure 2). Region selection expressions can select cells in tables in an XPath-like fashion, navigating in directions `right`, `left`, `up`, `down`, using transitive closures of these operators, and boolean combinations thereof. For the demo, we implemented a version of *propositional dynamic logic*, which is even more powerful than the navigational capabilities of XPath. We refer the interested reader to [2, 5] for more details.

3 TRANSFORMING DATA WITH CHISEL

In addition to a highly flexible schema language, Chisel supports a simple data transformation language, powerful enough to transform non-tabular into tabular data. This is interesting since some data is *non-tabular* in nature (Use Case 13, Table 1) whereas the W3C meta-data format only deals with *tabular* data. (Of course, Chisel can also do tabular to tabular transformations.) The basis of the data transformations is a Sculpt schema against which the data is validated first.

The control statements of the transformation language include loops, output statements, and conditional statements (see, e.g., Algorithm 1). In addition, if desired, Sculpt region selection expressions can be used for complex navigation. Finally, programs can exploit the `TYPES` of the schema to easily extract and manipulate substrings of information in single cells.

For example, the program in Algorithm 1 transforms the non-tabular data in Table 1 to the tabular data in Table 2. After writing header information in line 2 of the algorithm, it enters a for-loop that iterates over the region `content` defined in Figure 2. Using `content.is`, it parses each row in `content` using the predefined `TYPES` of the schema, where we allow the keyword `SOMETHING` as a wildcard that matches any cell content. In the example program, `SOMETHING*` matches an arbitrarily long list of cells with arbitrary content, which is stored in the variable x. Likewise, the first cell of each row, matching `rdfIRI` is stored in variable s, etc. The next interesting part is the for-loop on line 5, where the list of cells x is split into sublists, each consisting of docID, a list of span-cells, and an optional certainty. Again, this uses the `TYPES` from Figure 2. (We require the split to be unambiguous here.) In the innermost for-loop, we iterate through the list of span-cells using the variable sp and we can use sp.int<0> and sp.int<1> to address the two smallints that are used in the schema to define the type span. Notice that the transformation program outputs sp.int<0> and sp.int<1> in separate columns in the output file (cf. columns 5 and 6 in Table 2). Such parsing and manipulation of CSV-like input data into structured output is much more cumbersome with standard programming languages.

4 DEMO OVERVIEW

Chisel's implementation mainly consists of a *region selection engine*, supporting propositional dynamic logic and a powerful automata model, a *schema validator* for the extension of Sculpt, a *schema debugger* that allows users to retrieve defective cells in the data if it does not match the schema, and the *transformation engine*.

We give an overview of the actual demo where we showcase the different abilities and features of Chisel. We also highlight how attendees can interact with the system and perform all steps of the Chisel data wrangling process, shown in Figure 1. We focus on the following three scenarios:

(1) Chisel as an IDE for Sculpt: specification and validation of Sculpt-like schemas;
(2) Chisel as a schema cleaning tool: analyzing and debugging Sculpt-like schemas; and
(3) Chisel as a data transformation tool: transforming non-tabular into tabular CSV files.

Specification and Validation. The purpose of this scenario is to familiarize users with the Sculpt language and the basic Chisel features for defining and validating Sculpt schemas. Attendees can create schemas from scratch using real and artificial data. (E.g., any desired data set from the W3C Use Cases [8]). This scenario also addresses basic features of the GUI (loading a schema, loading a CSV file, associating them to one another), specification of types, tokens, regions, and validation. We demonstrate how Chisel can visualize how rules in the schema correspond to cells in the document. For instance, one can click a cell in the CSV data, upon which the types, tokens, and left-hand sides of rules that match the cell are highlighted (even if the data is invalid). Conversely, one can click a type, token, or a rule, and the cells that are matched by it in the CSV data are highlighted. This visualization helps users understand the schema and how it relates to the data, which is crucial in fast and accurate development of schemas. As another aid, we implemented highlighting features that allow users to see the union or intersection of sets of cells that are selected by rules.

Analyzing and Debugging. Here we want to showcase Chisel's functionality to improve, clean, and debug a schema. When a CSV file does not validate against a schema, Chisel reports on the specific places where the violations occur. Actually, when testing the

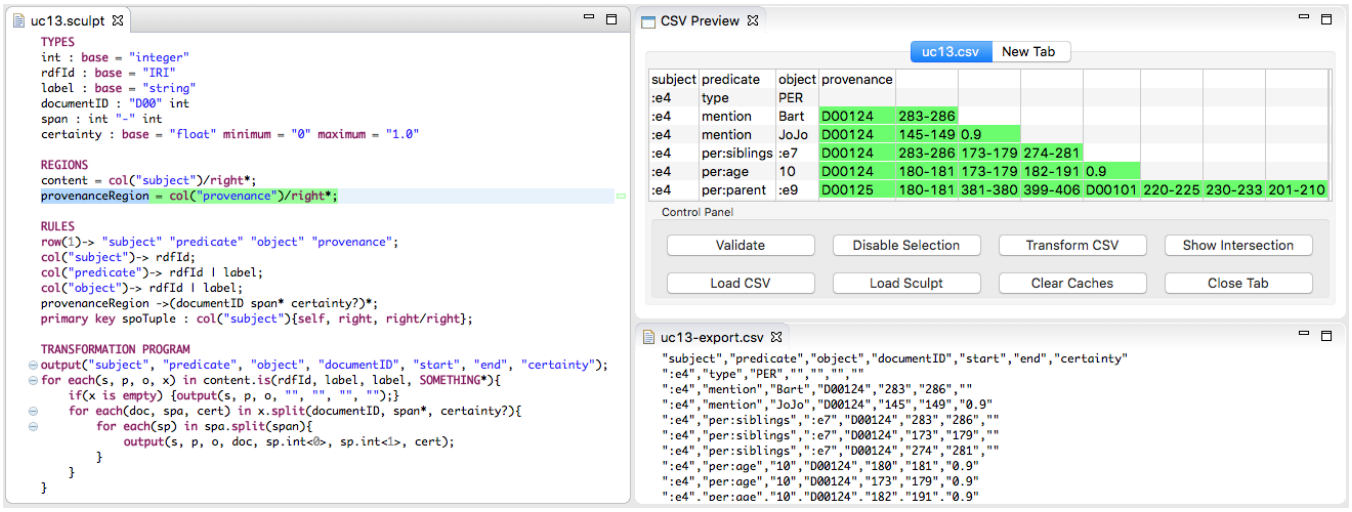

Figure 3: Screenshot of the system. The left tab has a Sculpt-like schema followed by a transformation program. The user selected the provenanceRegion on the left, upon which the corresponding region in the CSV file (top right) is automatically highlighted. The bottom right contains the output of the transformation program, as in Table 2.

system, in one case, such an error pointed us to a minor inconsistency with whitespaces in a data set in Use Case 24 of the W3C CSV Data on the Web Use Cases, as we can show in the demo.[6]

Chisel also allows to check if the data contains cells that are not matched by any rule or token. Such cells are effectively unconstrained by the schema and can therefore take on any form which might be an indication that the schema is still incomplete. Attendees will be able to interactively explore the analysis and debug features of Chisel.

Data transformation. Finally, we demonstrate Chisel's ability as a data transformation tool. This part of the demo shows how we can leverage the schema information to easily implement transformations (even beyond the scope of the W3C) on tabular and non-tabular CSV-like data. Since Chisel can also automatically generate a W3C Tabular Metadata file for the output (in JSON), it can easily be used as a preprocessing tool for whatever technology the W3C develops for transformations of *tabular* data.

Related Work. The philosophy of Sculpt is similar to that of BonXai [4] a flexible rule-based language that facilitates the development of XML Schema documents. Sculpt, as well as BonXai are designed to be easy-to-use yet expressive schema languages. Arenas et al. [1] define a language for validating and annotating CSV-like data which, like Sculpt, is based on regular expressions, but uses an orthogonal approach. It would be interesting to see if and how ideas from Arenas et al. and ours can be combined.

ACKNOWLEDGMENTS

We are very grateful to Thomas Buchmann for his help and expertise w.r.t. the XText framework, which we used for developing the Sculpt parser. This work is supported by the Deutsche Forschungsgemeinschaft under Grant No: MA 4938/2–1 and the Special Research Fund (BOF) of Hasselt University.

[6] In the "soc_structure_2010.csv" file, some lines (e.g. the line containing the group "13-1161") contain whitespace after the group number, while most of them do not.

AUTHORS


Johannes Doleschal is a PhD student at Bayreuth University. His research interests include grid- and graph structured data.


Nico Höllerich is a Master's student at Bayreuth University.


Wim Martens is professor at Bayreuth University and works on principles of data management.


Frank Neven is professor at Hasselt University and works on management of data.

REFERENCES

[1] Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. 2016. A framework for annotating CSV-like data. *PVLDB* 9, 11 (2016), 876–887.
[2] Johannes Doleschal, Wim Martens, Frank Neven, and Adam Witkowski. 2018. Satisfiability for SCULPT schemas for CSV-like data. In *ICDT*. To appear.
[3] Michael J. Fischer and Richard E. Ladner. 1979. Propositional Dynamic Logic of Regular Programs. *J. Comput. System Sci.* 18, 2 (1979), 194–211.
[4] Wim Martens, Frank Neven, Matthias Niewerth, and Thomas Schwentick. 2017. BonXai: Combining the Simplicity of DTD with the Expressiveness of XML Schema. *ACM Trans. Database Syst.* 42, 3, Article 15 (2017), 42 pages.
[5] Wim Martens, Frank Neven, and Stijn Vansummeren. 2015. SCULPT: A Schema Language for Tabular Data on the Web. In *WWW*. 702–720.
[6] Rufus Pollock, Jeni Tennison, Gregg Kellogg, and Ivan Herman. 2015. *Metadata Vocabulary for Tabular Data.* Technical Report. World Wide Web Consortium (W3C). https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/.
[7] Jonathan Robie, Michael Dyck, and Josh Spiegel. 2017. *XML Path Language (XPath) 3.1.* Technical Report. World Wide Web Consortium (W3C). https://www.w3.org/TR/2017/REC-xpath-31-20170321/.
[8] Jeremy Tandy, Davide. Ceolin, and Eric. Stephan. 2017. *CSV on the Web: Use Cases and Requirements.* Technical Report. World Wide Web Consortium (W3C). https://w3c.github.io/csvw/use-cases-and-requirements/.
[9] Jeni Tennison. 2014. 2014: The Year of CSV. https://www.theodi.org/blog/2014-the-year-of-csv. (2014). https://www.youtube.com/watch?v=a8piOmSsJ2I.
[10] Jeni Tennison and Gregg Kellogg. 2015. *Model for Tabular Data and Metadata on the Web.* Technical Report. World Wide Web Consortium (W3C). https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.