Detailed Provenance Metadata from Statistical Analysis Software

George Alter, *University of Michigan* Jack Gager, *Metadata Technology North America.*
Pascal Heus, *Metadata Technology North America*
Carson Hunter, *Metadata Technology North America.* Sanda Ionescu, *University of Michigan*
Jeremy Iverson, *Colectica* H V Jagadish, *University of Michigan*
Bertram Ludaescher, University of Illinois Urbana-Champaign Jared Lyle, *University of Michigan*
Timothy McPhillips, *University of Illinois Urbana-Champaign*
Alexander Mueller, *University of Michigan* Sigve Nordgaard, *Norwegian Centre for Research Data*
Ørnulf Risnes, *Norwegian Centre for Research Data* Dan Smith, *Colectica*
Jie Song, *University of Michigan* Thomas Thelen, *University of California Santa Barbara*

Abstract

We have created a set of tools for automating the extraction of fine-grained provenance from statistical analysis software used for data management. Our tools create metadata about steps within programs and variables (columns) within dataframes in a way consistent with the ProvONE extension of the PROV model. Scripts from the most widely used statistical analysis programs are translated into Structured Data Transformation Language (SDTL), an intermediate language for describing data transformation commands. SDTL can be queried to create histories of each variable in a dataset. For example, we can ask, "Which commands modified variable X?" or "Which variables were affected by variable Y?" SDTL was created to solve several problems. First, researchers are divided among a number of mutually unintelligible statistical languages. SDTL serves as a lingua franca providing a common language for downstream applications. Second, SDTL is a structured language that can be serialized in JSON, XML, RDF, and other formats. Applications can read SDTL without specialized parsing, and relationships among elements in SDTL are not defined by an external grammar. Third, SDTL provides general descriptions for operations that are handled differently in each language. For example, the SDTL MergeDatasets command describes both earlier languages (SPSS, SAS, Stata), in which merging is based on sequentially sorted files, and recent languages (R, Python) modelled on SQL. In addition, we have developed a flexible tool that translates SDTL into natural language. Our tools also embed variable histories including both SDTL and natural language translations into standard metadata files, such as Data Documentation Initiative (DDI) and Ecological Metadata Language (EML), which are used by data repositories to inform data catalogs, data discovery services, and codebooks. Thus, users can receive detailed information about the effects of data transformation programs without understanding the language in which they were written.

1. Introduction

There is often a need to add provenance to metadata files in standard formats (Data Documentation Initiative (DDI) [1], Ecological Metadata Language (EML) [2]) to record data modifications. This is a common issue in the social sciences and other domains. Data repositories rely on metadata which is often expressed in XML, for data catalogs, codebooks, and other tools. However, metadata files are difficult and time-consuming to update when the data are modified. Consequently, provenance information is lost or expressed in irregular and inconsistent ways.

To address this need, the "Continuous Capture of Metadata for Statistical Data" (C^2Metadata) Project (NSF ACI-1640575) has created a set of tools for automating the extraction of fine-grained provenance from statistical analysis software used for data management. Our tools create metadata about steps within programs and variables (columns) within datasets ("dataframes"), which allow the creation of histories for derived and transformed variables. Variable histories can be used by data creators to audit their work processes and by data repositories to populate data discovery services, codebooks, and other tools. This paper describes some of the lessons learned about creating granular provenance metadata and representing it in a standard format.

Our approach uses Structured Data Transformation Language (SDTL) as an intermediate language for describing data transformation commands [3]. Scripts from the most widely used statistical analysis programs (SPSS [4], SAS [5], Stata [6], R [7], and Python [8]) are translated into SDTL, and SDTL is added to XML files in supported metadata formats (DDI, EML). SDTL has been developed as a set of JSON schemas, which can be serialized into XML, RDF, or other formats. SDTL can be queried to create histories of each variable in a dataset. For example, we can ask, "Which commands modified variable X?" or "Which variables were affected by variable Y?" There is also a C2Metadata tool for translating SDTL into natural language. SDTL has been adopted by the DDI Alliance into its suite of international

standards, which means that it will be maintained and updated in the future [9].

In this paper, we first briefly describe the tools in C2Metadata that do all the work mentioned above. Then, we focus on SDTL, its relationship to ProvONE, and the lessons we learned in developing it as a lingua franca.

2. C²Metadata Workflow

The C2Metadata workflow has been implemented in a set of open-source tools (see Figure 1), which are also available by API.

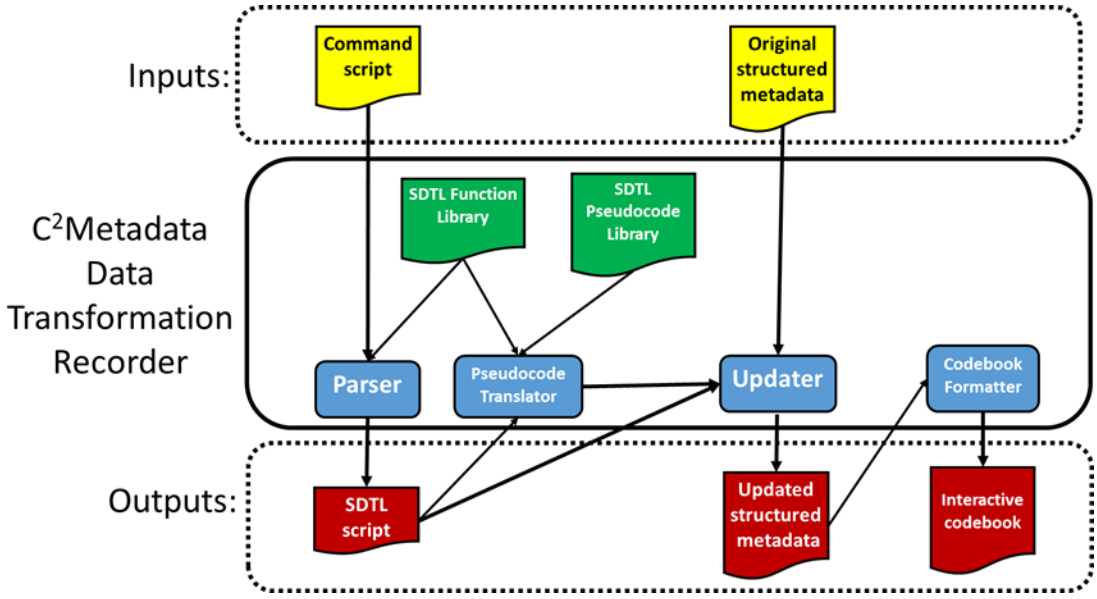

Figure 1. C2Metadata Workflow

2.1. Parsers

Parsers read a script in a statistical analysis language and translate it into SDTL JSON. We have developed Parsers for five widely used statistical analysis languages: SPSS, SAS, Stata, R, and Python, because each language requires a separate Parser adapted to its particular syntax. All of the Parsers use standard programming techniques that convert the source language into an abstract syntax tree and apply visitor methods for creating SDTL by referring to a language-dependent predefined translation mapping. Our focus has been on data transformation commands, which are a manageable subset of each language commonly used for data management tasks. For R and Python, which have many user-contributed libraries, we focus on the base languages and the most common data transformation libraries, tidyverse [10] in R and Pandas [11] in Python.

2.2. Updaters

Updaters modify a metadata file to reflect the changes to a data file described in an SDTL script. Like the Parsers, Updaters are specific to each metadata standard, and we have created Updaters for both DDI and EML, which are expressed in XML. The Updater reads an original XML metadata file for each data file, and converts it to a custom representation of the dataset. The Updater classifies commands in the SDTL script based on the actions that they perform: updates a dataset or variable, deletes a variable, selects variables to work on, saves a dataset, etc. Some SDTL commands are assigned to more than one of these categories. The Updater then processes the SDTL commands in order, performing the specified actions on the relevant metadata object. The Updater also traverses the SDTL to create a history of every prior command and variable that affected the state of each variable saved to an external file. SDTL commands are also translated into natural language by the Pseudocode Translator. Finally, an updated XML file is created, which includes both variable histories and natural language translations of SDTL commands.

2.3 Pseudocode Translator

The Pseudocode Translator makes natural language translations of SDTL commands. Translations are created by a fill-in-the-blank approach. A template for every SDTL command is available in the C2Metadata Pseudocode Library. Templates consist of text surrounding locations for inserting the values of properties of SDTL commands.

2.4. Codebook Formatter

To demonstrate the capabilities of SDTL-enhanced metadata files, we have created a tool that generates an interactive codebook from a DDI XML file. The codebook includes an entry for each variable in the data file with any available metadata (e.g. variable and value labels). If the variable was

created or modified by commands in the SDTL script, a variable history listing all relevant commands is included in the codebook entry. Variables in these commands are connected to the variable history by hyperlinks pointing to other places in the codebook. The codebook includes entries describing the data before transformations were performed and entries for temporary variables (or variables in intermediate states) that were not in the saved version of the data file.

2.5. Data Transformation Recorder

The Recorder orchestrates all of the other applications. The Recorder calls the appropriate Parser and Updater and passes intermediate files to APIs in the correct sequence. Recorders take two inputs: a metadata file and a data transformation script. The outputs produced are an SDTL version of the script, an updated metadata file, and an HTML codebook.

3. Structured Data Transformation Language (SDTL)

SDTL, which plays a central role in the C2Metadata workflow, was created to solve two problems. First, researchers are divided among a number of mutually unintelligible statistical languages. Scientists tend to be divided among the five main statistical languages by discipline. For example, 70 percent of scripts submitted in support of articles published by American Economic Association journals use Stata. SPSS is widely used in some disciplines, because it is easy to learn, and SAS has a strong following among producers of survey data and in some biomedical specialties. R and Python, which are both used in data science, have different strengths and weaknesses. SDTL provides a common language for downstream applications. In addition, we translate SDTL into natural language, so that users can receive detailed information about the effects of data transformation programs without understanding the language in which they were written.

Second, SDTL is a structured language that software applications can read without parsing. We currently provide SDTL in JSON, but it can be serialized into XML, RDF, and other formats. Parsing requires specialized programming techniques, and defining the grammar of a language is a difficult process. In contrast, SDTL uses tags and delimiters to minimize syntax rules. For example, statistical languages rely on rules to define the order of precedence of arithmetic operations in expressions like "y = a + b/c". Arithmetic operations are described by functions in SDTL, which are nested to make the order of operations unambiguous. In this example, SDTL shows the precedence of the division function by nesting it inside the addition function.

4. SDTL and PROV

With partners from the Whole Tale Project, we have been investigating how SDTL can be used in the PROV framework [12]. Thomas Thelen has written an application that converts SDTL JSON into SDTL RDF, which can be queried with SPARQL and other tools. Figure 2 shows a graph created from SDTL of a simple Python program that converts a temperature on the Fahrenheit scale into equivalents on the Celsius and Kelvin scales. SDTL RDF can be queried to answer questions like: "Which commands affected variable X?" and "Which derived variables were affected by variable Y?"

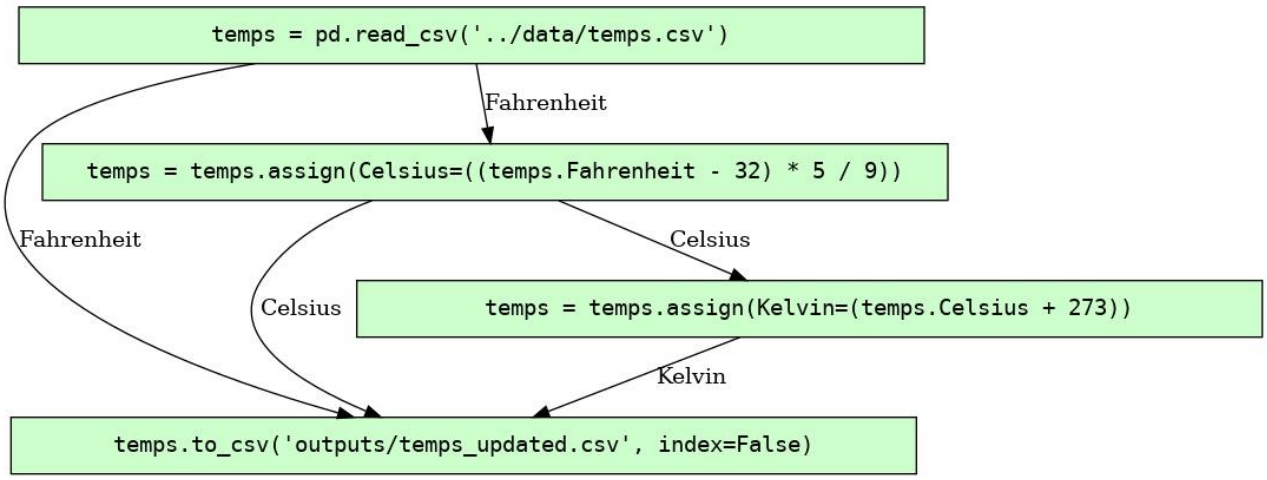

Figure 2. Graph of a Python Script Derived from SDTL RDF

The detailed nature of SDTL is an advantage for representing complex commands, but it creates challenges for querying SDTL RDF. SDTL expressions are often nested several layers deep. Even if we are only interested in finding variable names, we need to account for every level that may include a

variable name. Consequently, SPARQL queries on SDTL RDF can be very long and complicated.

Due to the specialization, simplicity and community usage of the ProvONE [13] data model, we are investigating mappings between ProvONE and SDTL. ProvONE's focus on describing provenance of data makes it a natural framework for representing the generation and usage of script artifacts. This allows for powerful queries about the lineage of data such as "Which commands modified a particular dataframe?" and "Which derived variables were affected by a particular original variable?". Queries of this kind focus on just a few classes and relations from the SDTL ontology, which have analogs in ProvONE. ProvONE supports both *retrospective* and *prospective* provenance. SDTL can be represented in both retrospective and prospective ProvONE, but the full detail of SDTL may not be needed for most Prov queries.

In the retrospective model, source code commands are represented by the *provone:Execution* class which are analogous to the *sdtl:Command* class. Representing nested commands is also possible by introducing hierarchy on the *prov:Entity* by using the *prov:hadMember* relation to declare that there is a child command. The *prov:Entity* class along with the *prov:used* and *prov:wasGeneratedBy* relations are enough to make assertions about the usage and generation of data.

Describing the SDTL as prospective provenance is also possible. The *sdtl:Command* class maps to the *provone:Program* class. The prospective analog to *prov:hadMember* is *provone:hasSubProgram* which gives rise to hierarchical structure support in a similar fashion. *provone:Port* classes represent data and map to *sdtl:Variable* classes. *provone:Channel* objects act as intermediate nodes between ports that are used by subsequent commands. The ports and channels are connected to *provone:Program* nodes via *provone:hasInPort* and *provone:hasOutPort*.

Our next step will be developing a path from SDTL RDF into ProvONE RDF to allow queries written for ProvONE, which are relatively simple, to be executed on SDTL RDF. We anticipate that reasoning implemented as SPARQL CONSTRUCT queries will be required to map between SDTL dataframes and variables on one hand and ProvONE channels and ports on the other hand.

5. Lessons

SDTL was created to represent common data transformation operations in five languages, and we learned a number of useful lessons about similarities and differences among these languages.

5.1. Function Library

The SDTL Function Library is possibly the most important way that we simplified the task of translating statistical languages into SDTL. All statistical languages use functions to perform common operations, like computing the logarithm of a number or the sine of an angle. Most of these operations are familiar, and we decided that recognizable explanations are more appropriate provenance than translating functions into a sequence of SDTL commands. We created a list of equivalent SDTL functions with natural language translations for each one. The Function Library is an externally stored crosswalk between functions in each of the source languages and the SDTL representations of the same functions. Thus, LN(x) in SPSS and log(x) in R both translate into natural logarithm(x) in SDTL. A script Parser can search the Function Library to find the SDTL version of a function, which also shows how parameters in the source language function are inserted into the SDTL function. Although there are exceptions in each source language, Parsers can use the same program code for many functions.

The boundary between functions and commands varies, and a function in one language may be a command in another. For example, in R the select() function is used to select a subset of columns, but this operation is a command in most other languages. In SDTL subsetting columns is performed by the KeepVariables and DropVariables commands.

5.2. Temporary variables

The dataset received by a data repository may not include variables that played important roles in creating or modifying other variables. For example, the items used to construct an index may be dropped after the index is created. Similarly, some variables remaining in the final dataset were changed several times before reaching their final values. For example, an income variable may be indexed for inflation and then recoded into categories. The C2Metadata workflow allows metadata to include information about variables not included in the final state of the dataset. Since we begin with metadata describing the data before transformations were performed, the updated metadata file can include descriptions of variables that were dropped from the final dataset.

We take advantage of a feature in the DDI metadata standard that allows a single XML file to describe multiple data files. Each file has an ID, and a variable description refers to the ID of the file in which it is found. Variables also have IDs, which are separate from the variable name. Using these features, both the pre- and post-transformation versions of the metadata can be included in the DDI XML file. The interactive Codebook uses these features to hyperlink derived variables to the variables used to construct them.

We also use variable and file IDs to describe multiple states of variables that are changed more than once. It is common for the values of a variable to be modified under the same variable name. By stepping through the SDTL script, it is possible to create a new ID and description for a variable every time that its contents change.

5.3. Iterating by rows

Most data transformation commands in statistical analysis software are designed to operate iteratively on every row in a dataframe. For example, a simple assignment statement, like "varX = varA + varB", implies that the value of varX on each row is derived from the values of varA and varB on the same row. However, commands that apply to an entire dataframe or file (e.g. append or merge commands) or to metadata (e.g. print format commands) are not iterative.

The difference between iterative and non-iterative commands is not always apparent. For example, these commands in the Stata language appear to be identical:

```
if varX>5 replace varY=3   /*** Version 1 *****/
replace varY=3 if varX>5   /*** Version 2 *****/
```

Both commands change the value of varY to 3 if the condition varX>5 is true, but the order of "if" and "replace" affects how Stata evaluates the condition and modifies the data. The Stata "if" command evaluates the condition only once, using the value of varX on the first row in the dataset, and then applies the command to all rows in the dataset. When "if" follows a Stata command, it is evaluated separately on each row, and the command is executed only on rows where it is true. These outcomes are illustrated in Figures 3 and 4.

In SDTL we created two kinds of "if…then" commands to distinguish between iterative and non-iterative operations. The "DoIf" command is evaluated once for the entire dataset, and the "IfRows" command is evaluated separately on each row.

varX	varY
9	11
4	11
1	11

→

varX	varY
9	3
4	3
1	3

Figure 3. Outcome of Stata "if varX>5 replace varY=3"

varX	varY
9	11
4	11
1	11

→

varX	varY
9	3
4	11
1	11

Figure 4. Outcome of Stata "replace varY=3 if varX>5"

5.4. Bridging languages by adding detail

Sometimes SDTL harmonizes commands from various languages by specifying properties in greater detail than in any of the source languages. The SDTL MergeDatasets command, which combines rows from multiple dataframes to make a single dataframe, illustrates this approach. The earlier statistical languages (SPSS, SAS, Stata) approached merging rows as a sequential process of matching rows from pre-sorted files. More recent languages (R, Python), which are modelled on SQL, use the language of "inner" and "outer" joins. The difference between these approaches is most apparent when keys are not used to match rows across dataframes. In the older languages, a merge that does not use keys will match rows sequentially from each dataframe, such that row i in dataframe A is matched to row i in dataframe B. If keys are omitted from R and Python, the result is a Cartesian join in which every row in dataframe A is matched to every row in dataframe B. Most results can be achieved by both approaches, but the language used to describe them is very different.

The SDTL MergeDatasets command provides a framework that covers all five languages by relying on specific properties rather than syntax and grammar. For example, "inner", "outer", "right", and "left" joins indicate when a new output row is created for unmatched input rows. In SDTL, each contributing dataframe has a NewRow property, which is True if an unmatched row from this input will result in a row in the output dataframe. Thus, an outer join is described by setting the NewRow property to True for all input dataframes, and an inner join sets all NewRow properties to False. Similarly, in SDTL, variables can be renamed as part of the MergeDatasets command. In R and Python, this happens automatically when a variable with the same name appears in more than one input dataframe, but SPSS, SAS, and Stata have rules about which dataframe takes precedence. We created an "SDTL Merge Gallery" which currently includes 38 examples to help application developers understand how to map various languages into SDTL.

6. Limitations

Software developed by the C2Metadata Project has several limitations. Our Parsers for the R and Python languages only include the base and tidyverse and Pandas libraries. The workflow described above operates only on metadata, which also limits functionality in several ways. However, these are limitations of software and not of the SDTL standard. The primary limitation of SDTL relates to data created by statistical analysis commands, like predicted values from regression models, which we hope to address in a future project.

7. Conclusions

While arbitrarily rich manipulations are possible, limited only by the expressive power of the transformation language

used, only a small subset of these are actually used in typical practice. We have designed SDTL to cover this typical set. Even so, we had several non-trivial translational challenges to address such as those described in this paper. However, having done that, we now have a flexible set of tools for deriving provenance from statistical transformations.

Acknowledgement

"Continuous Capture of Metadata for Statistical Data Project" is funded by National Science Foundation grant ACI-1640575. "Merging Science and Cyberinfrastructure Pathways: The Whole Tale" is funded by National Science Foundation grant OAC 1541450.

Access

Software described above is available in a Gitlab repository at https://gitlab.com/c2metadata. SDTL documentation and specifications are available at https://ddialliance.org/products/sdtl/1.0.

References

1. Vardigan, M., P. Heus, and W. Thomas, *Data documentation initiative: Toward a standard for the social sciences.* International Journal of Digital Curation, 2008. **3**(1).
2. E.H. Fegraus, S.A., M.B. Jones, M. Schildhauer, *Maximizing the value of ecological data with structured metadata: an introduction to ecological metadata language (EML) and principles for metadata creation.* Bulletin of the Ecological Society of America, 2005. **86**: p. 158–168.
3. Alter, G., et al., *Provenance metadata for statistical data: An introduction to Structured Data Transformation Language (SDTL).* IASSIST Quarterly, 2020. **44**(4).
4. IBM Corp., *IBM SPSS Statistics for windows, version 26.0*. 2019, IBM Corp.: Armonk, NY.
5. SAS Institute, *SAS®9.4 Product Documentation*. 2015, SAS Institute Inc.: Cary, NC.
6. StataCorp., *Stata Statistical Software: Release 16.1*. 2020, StataCorp LP: College Station, TX.
7. R Core Team, *R: A Language and Environment for Statistical Computing*. 2013, R Foundation for Statistical Computing: Vienna, Austria.
8. Python Software Foundation, *Python Language Reference, version 3.8*. 2019: Beaverton, OR.
9. DDI Alliance. *Structured Data Transformation Language*. 2020 December 1, 2020 January 21, 2021]; Available from: https://ddialliance.org/products/sdtl/1.0.
10. Wickham, H., et al., *Welcome to the Tidyverse.* Journal of Open Source Software, 2019. **4**(43): p. 1686.
11. The pandas development team, *pandas-dev/pandas: Pandas*. 2020, Zenodo.
12. Groth, P. and L. Moreau, *PROV-OVERVIEW: An Overview of the PROV Family of Documents*, in *W3C Note*. 2013.