AUTOMATING THE CAPTURE OF DATA TRANSFORMATION METADATA FROM STATISTICAL ANALYSIS SOFTWARE

George Alter[1], Darrell Donakowski[1], Jack Gager[2], Pascal Heus[2], Carson Hunter[2], Sanda Ionescu[1], Jeremy Iverson[3], H V Jagadish[1], Carl Lagoze[1], Jared Lyle[1], Alexander Mueller[1], Sigbjørn Revheim[4], Matthew A. Richardson[1], Ørnulf Risnes[4], Karunakara Seelam[1], Dan Smith[3], Tom Smith[5], Jie Song[1], Yashas Jaydeep Vaidya[1], Ole Voldsater[4]

Abstract

The C²Metadata ("Continuous Capture of Metadata for Statistical Data") Project automates one of the most burdensome aspects of documenting the provenance of research data: describing data transformations performed by statistical software. Researchers in many fields use statistical software (SPSS, Stata, SAS, R, Python) for data transformation and data management as well as analysis. The C²Metadata Project creates a metadata workflow paralleling the data management process by deriving provenance information from scripts used to manage and transform data. C²Metadata differs from most previous data provenance initiatives by documenting transformations at the variable level rather than describing a sequence of opaque programs. Scripts used with statistical software are translated into an independent Structured Data Transformation Language (SDTL), which serves as an intermediate language for describing data transformations. SDTL can be used to add variable-level provenance to data catalogs and codebooks and to create "variable lineages" for auditing software operations. Better data documentation makes research more transparent and expands the discovery and re-use of research data.

Acknowledgment

The Continuous Capture of Metadata for Statistical Data Project is funded by National Science Foundation grant ACI-1640575.

[1] University of Michigan
[2] Metadata Technologies North America
[3] Algenta Technologies
[4] Norwegian Centre for Research Data
[5] NORC

Table of Contents

Introduction

The C^{2}Metadata ("Continuous Capture of Metadata for Statistical Data") Project automates one of the most burdensome aspects of documenting research data: describing data transformations performed by statistical software. Researchers in many fields use statistical software (SPSS, Stata, SAS, R, Python) for data transformation and data management as well as analysis (IBM Corp., 2019; Python Software Foundation, 2019; R Core Team, 2013; SAS Institute, 2015; StataCorp., 2020). These software applications have limited metadata capabilities, and they do not support the detailed metadata standards used by data repositories, such as Data Documentation Initiative (DDI) and Ecological Markup Language (EML). Consequently, valuable information about the data is lost. The C^{2}Metadata Project translates scripts used by statistical software into an independent Structured Data Transformation Language (SDTL), which serves as an intermediate language for describing data transformations. SDTL can be used to:

- Update existing metadata files (e.g. DDI, EML, JSON-LD), so that both the original data description and changes to the data are preserved;
- Describe variable transformations in natural language for data users who are unfamiliar with the specific software used in variable transformations;
- Create "variable lineages" that describe the transformations performed on each variable for use in auditing scripts.

C^{2}Metadata differs from most previous approaches to data provenance by focusing on documenting transformations at the variable level. Most provenance initiatives record the name and identifier of the process (e.g. web service, program, or script) that changed the data, but each process is treated as a black box with little detail about what it does. C^{2}Metadata goes inside the black box and describes each step in the process for every variable that changes.

The C^{2}Metadata Project has developed an automated workflow that

1. Extracts data transformation information from scripts for the leading statistical software packages;
2. Expresses data transformations in a new Structured Data Transformation Language (SDTL) that is independent of the source languages; and
3. Incorporates SDTL and human-readable derivatives of SDTL into existing metadata standards (e.g. DDI, EML).

The Problem

All branches of science have been affected by growing demands for transparency, reproducibility, and data sharing (Freese & Peterson, 2017; King, 2011; Miguel et al., 2014; New York Times Editors, 2013; Powers & Hampton, 2019). These principles have been institutionalized by journals and funding agencies, which require authors to share both data and code used in their analyses (Adelson et al., 2019; American Economic Association, 2013; American Political Science Association Committee on Professional Ethics Rights and Freedoms, 2012; Coalition on Publishing Data in the Earth and Space Sciences (COPDESS), 2015; Holdren, 2013; Nosek et al., 2015; Political Science Journal Editors, 2015). As sharing research data has become an obligation, standards are emerging to make data FAIR: findable, accessible, interoperable, and reusable (Wilkinson et al., 2016).

Realizing the promise of research transparency and the FAIR principles requires provenance metadata i.e., documentation of the origins, contents, and meaning of data. For example, the ethical guidelines of the American Political Science Association obligate researchers to support "evidence-based knowledge claims through data access, production transparency, and analytic transparency so that their work can be tested or replicated" (American Political Science Association Committee on Professional Ethics Rights and Freedoms, 2012, p. 8). "Production transparency" means that researchers cannot simply share the data used in a publication without describing how it was created and managed. According to Lupia and Elman: "In order to achieve *production transparency*, researchers should provide comprehensive documentation and descriptive metadata detailing their project's empirical base, the context of data collection, and the procedures and protocols they used to access, select, collect generate, and capture data" (Lupia & Elman, 2014, p. 32). Viewed in terms of the FAIR model, data cannot be findable, interoperable, or re-usable unless they are well described in machine-actionable metadata.

In practice, most datasets are poorly documented, and the cost of creating detailed metadata is often prohibitive. Even though most data are "born digital," metadata are usually an afterthought. Data repositories generally receive data after a project has ended, and they often receive documentation in a PDF file that cannot be converted to a machine actionable format. Consequently, they can only re-create a small portion of the information that would be useful to researchers trying to re-use the data. In the social sciences, questionnaires frequently include thousands of questions, and recreating this information in a metadata file may involve weeks of work.

Data repositories have created detailed metadata standards that are capable of providing all the information that researchers need for findability and interoperability. In the social sciences, the Data Documentation Initiative (DDI) (Vardigan, Heus, & Thomas, 2008) offers a rich model for describing data down to the variable level. Social science data repositories, like the Inter-university Consortium for Political and Social Research (ICPSR) and the UK Data Service, provide variable-level data discovery services built on DDI metadata. Similarly, the DataONE catalog has advanced search capabilities built on Ecological Markup Language (EML) (E.H. Fegraus, 2005) and other metadata formats.

Data producers should be able to automate the creation of machine actionable metadata. Whether the data represent a questionnaire, a laboratory instrument, or an environmental sensor, a description of the data collection process could be transmitted with the data. For example, widely used social surveys, such as the American National Election Study and the General Social Survey, are collected by computer assisted interview (CAI) software. Even if a human interviewer asks the questions by telephone or in person, the survey is entirely digitized and never exists on paper. This means that all of the information in the CAI software can be transferred directly into a standard metadata format.

Why don't data producers capture metadata directly from the instruments that create the data? One of the main reasons is that they modify the data as soon as they receive it. Data elements that were included for administrative purposes are deleted. Some variables are combined into indexes. Variables like income and age are often converted into categories to make respondents less identifiable. These changes are usually performed with statistical software packages that have no way of recording changes performed on individual variables. Manually editing an existing metadata file to reflect changes performed by statistical software is much too time consuming. So, data producers are more likely to describe these changes in text documents or spreadsheets.

The C^{2}Metadata Project was designed to create an automated workflow for metadata that parallels the workflow that transforms the data. We extract provenance information from the script that transformed the data, and we use that information to update an existing metadata file. Since C^{2}Metadata software operates on exactly the same scripts that change the data, data producers do not need to change their workflows or practices.

The ALPHA Network (ALPHA Network, 2020) is one of the first organizations to recognize the potential of the C^{2}Metadata approach (Kanjala, 2019). A consortium of ten African research institutions, the ALPHA Network assembles and curates data from research sites in six countries for studies of HIV and AIDS. Since contributors collect data in different ways, the ALPHA Network has invested heavily in data harmonization procedures. SDTL will provide detailed documentation of each step in the data harmonization workflow, which may involve several data management tools and statistical packages. Variable lineages translated into natural language will increase the transparency of complex data preparation scripts.

C^{2}Metadata Workflow

An example of an automated metadata workflow based on C^{2}Metadata tools is illustrated in Figure 1. We assume that the user provides two files: a command script in a supported language (SPSS, Stata, SAS, R, Python) and a structured metadata file in a supported metadata standard or format (DDI, EML) describing the data taken as input to the script. The first step is performed by a Parser, which translates the command script into an SDTL script. The SDTL script is sent to an Updater, which also reads the user's metadata file. The Updater also communicates with the Pseudocode Translator, an application that creates a natural language version of the SDTL script. The output of the Updater is a revised metadata file that now includes the SDTL and natural language descriptions of all variables modified by the command script. The updated metadata file may be used in a number of different ways. When a data repository receives the updated data file, the updated metadata will be added to its online data catalog. The data repository may also use a Codebook Formatter to create a static (e.g. pdf) or interactive codebook (e.g. html) for users to download. Each variable in the catalog or codebook will include a derivation section that describes the origin of the variable and all of the transformations applied to it.

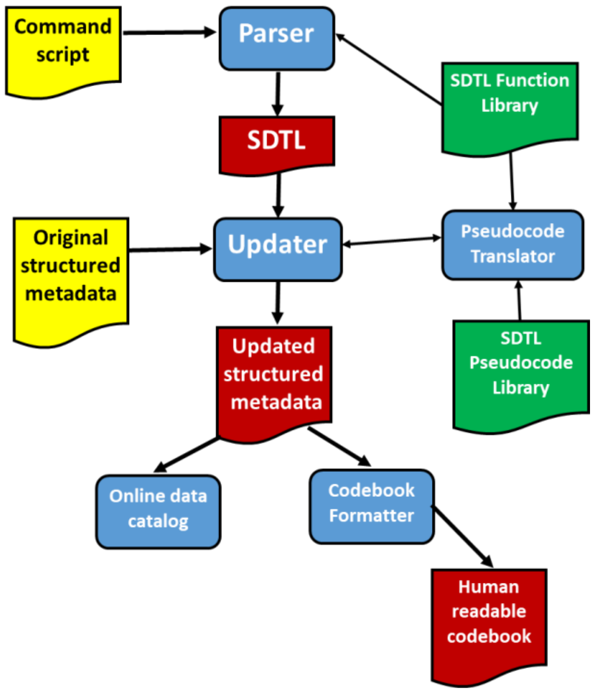


Figure 1 C²Metadata Workflow

Figure 1 includes one Parser and one Updater, but there are actually several versions of each. Each statistical package has its own language that must be parsed and translated into SDTL by a specially designed Parser. Similarly, every metadata standard requires a separate Updater. Since the Parsers and Updaters are separate modules, we can handle any combination of the supported statistical languages and metadata standards.

Figure 2 provides more detail on the operation of the Updater. The first stage in the Updater is an XML Reader that interprets and organizes the information provided in the original structured metadata file. The Dataset Updater creates a new version of the metadata that adds information from the SDTL file and the Pseudocode Translator. The XML updater creates the Updated structured metadata file by combining the original metadata with the updated metadata. This means that the final product includes descriptions of both pre- and post-transformation versions of the data file. We include the pre-transformation version so that users can hyperlink from the data derivation of a variable to examine pre-transformation descriptions of the variables that it came from.

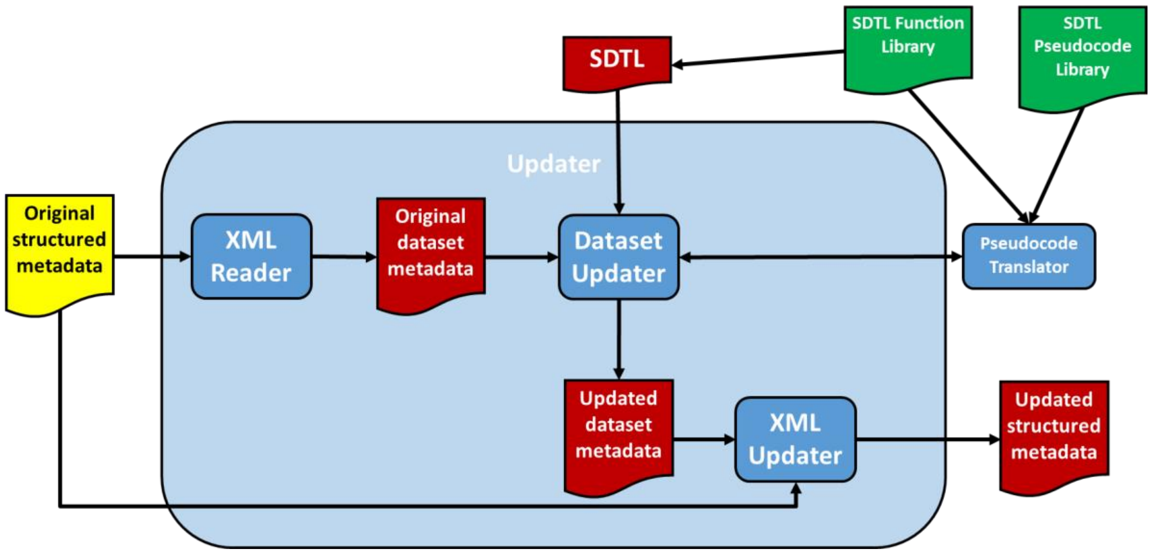


Figure 2 Workflow of the Updater

Figures 1 and 2 also include two files, the Function Library and the Pseudocode Library, which are parts of the SDTL standard. The Function Library is a crosswalk between the syntax for functions (e.g. sine, mean, maximum) in SDTL and in the statistical packages supported by C^{2}Metadata. Although there are thousands of functions, they can all be described by a common template, which simplifies the code in Parsers and Updaters. Similarly, the Pseudocode Library describes how to translate an SDTL command into natural language. The Pseudocode Library provides human-readable text to be inserted before and after variable names, numbers, and other expressions in SDTL commands. The result is a "pseudocode" version of the command that is comprehensible to a person unfamiliar with either the original statistical language or SDTL. Since both the Function Library and the Pseudocode Library are structured data files, they can be modified and expanded as SDTL is updated without changing any application code. The latest versions of the Function Library and Pseudocode Library are in JSON files accessed directly from Gitlab by C^{2}Metadata software modules.

C^{2}Metadata applications are available under an open source license from the project Gitlab repository C2Metadata Project (2020a).

Why a Structured Data Transformation Language?

SDTL was created to solve two problems. First, each of the five widely used statistical software packages has its own language, and our planned metadata workflow required a common intermediate language that would work for all of them. In preparation for our NSF proposal we examined download records at the Inter-university Consortium for Political and Social Research (ICPSR), the largest social science data repository in the U.S. ICPSR offers data for download in the formats of the four most common statistical packages, and researchers are divided among them. SPSS and Stata each account for about 25% of data downloads, and another quarter was divided between SAS and R. Researchers who did not select one of the leading four statistical packages downloaded data in ASCII files and sometimes in Excel. Thus, a solution that only worked for one of these four packages would reach at most a quarter of the research community. There was a clear need for a common language that could express the commands found in all of the statistical packages.

Second, this common language should be in a form that is easy for computers to process. Extracting meaning from a language is a complicated process, and a program customized to each language is required to process scripts into a form that a computer can use. We reduce the costs of sharing and re-using scripts in the common language by making SDTL computer-friendly. We developed SDTL in JSON (JavaScript Object Notation), but JSON can be easily translated into other formats used for transmitting complex information among software applications.

SDTL is "structured," because it follows a schema with defined tags and delimiters. For example, consider this SPSS command:

```
COMPUTE age_years=age_months/12.
```

This command will create a new variable named "age_years" by dividing the value of variable "age_months" by 12. The SDTL version of this commands is in Figure 3.

SDTL is obviously much more verbose than the SPSS language, but it is also more precise. How do we know that "age_years" and "age_months" refer to variables? Like a spoken language, the SPSS language has syntax rules that allow a person to assign meanings to text like "age_years" based on their order and position in a command. Computers can make these inferences too, but extracting meaning from text is a complicated problem. In SDTL the "$type" tells a computer program that "age_years" and "age_months" are variable names ("VariableSymbolExpression") that refer to columns in the dataset. SDTL relies much more on explicit tagging and less on syntax rules than the languages that it describes.

```
{    "$type": "Compute",
     "command": "compute",
     "sourceInformation": {  "originalSourceText": "compute  age_years=age_months/12" },
     "variable": {
                    "$type": "VariableSymbolExpression",
                    "variableName": "age_years"
                    },
     "expression": {
            "$type" : "FunctionCallExpression",
            "Function" : "division",
            "IsSdtlName" : true,
            "Arguments" : [ {
                        "$type" : "FunctionArgument",
                        "ArgumentName" : "EXP1",
                        "ArgumentValue" : {
                          "$type" : "VariableSymbolExpression",
                          "VariableName" : " age_months "
                        }
                      }, {
                        "$type" : "FunctionArgument",
                        "ArgumentName" : "EXP2",
                        "ArgumentValue" : {
                          "$type" : "NumericConstantExpression",
                          "Value" : 12
                        }
                        } ]     }
            }
```

Figure 3. Sample SDTL JSON

The SPSS COMPUTE command also uses a number of symbols that play a critical role in the meaning of the command: space, "=", "/", and ".", but these symbols have other meanings in different contexts. In the COMPUTE command "/" means division, but in the following SPSS RECODE command "/" is a separator between two variables that appear in one RECODE command.

```
RECODE age_years (0 THRU 14.999=1) (15 THRU 64.999=2) (65 THRU HI=3)

   / income (0 THRU 19999=1) (20000 THRU 99999=2) (100000 THRU HI=3)
```

The structured nature of SDTL removes ambiguities that would otherwise be resolved by a long list of syntax rules. For example, consider this SPSS command

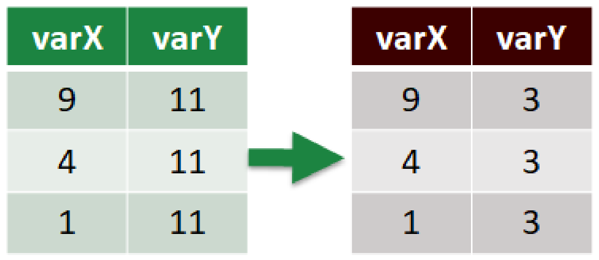
```
COMPUTE y = 1 + x/5
```

Which operation should be performed first, addition or division? Will the result be [(1 + x)/5] or [1 + (x/5)]? SPSS follows a common convention that division is performed before addition unless a different order of operations is specified by brackets in the formula. In SDTL the order of operations is never ambiguous. As the reader may have noticed in the previous example, arithmetic operations are implemented in SDTL as functions. The expression "x/5" is treated as "division(x, 5)" in SDTL. The basic arithmetic functions in SDTL have two parameters, but each parameter can be a function. This means that SDTL represents "1 + x/5" as "addition(1, division(x,5))". Since the division is nested within the addition, it must be performed first.

SDTL is also designed to handle several features of statistical software languages that are uncommon in most programming languages. First, some commands in these languages operate on the entire dataset but others operate sequentially on each row. For example, these commands in the Stata language appear to be identical:

```
if varX>5 replace varY=3   /*** Version1  ****/
replace varY=3 if varX>5   /*** Version 2 *****/
```

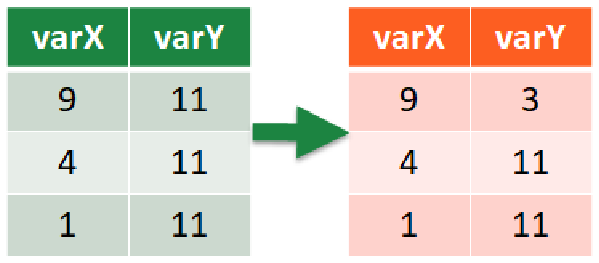
Both commands change the value of varY to 3 if the condition varX>5 is true, but the order of "if" and "replace" affects how Stata evaluates the condition and modifies the data. The Stata "if" command evaluates the condition only once, using the value of varX on the first row in the dataset, and then 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 4 and 5. In SDTL 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 4. 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 5. Outcome of Stata "replace varY=3 if varX>5"

Second, statistical software has special features for managing categorical variables. For example, the following SPSS RECODE command changes the values of variable *age_years* into three categories:

```
RECODE age_years (0 THRU 14.999=1) (15 THRU 64.999=2) (65 THRU HI=3) .
VALUE LABELS  age_years 1 "Child"  2 "Adult"  3 "Senior".
```

After recoding *age_years* remains a numeric variable, but it now has only values 1, 2, and 3. The VALUE LABELS command instructs SPSS to display labels "Child", "Adult", and "Senior" in place of the numeric values when *age_years* is used as the dimension in a cross-tabulation table.

For a more extended description of SDTL see Alter et al. (2020) and C2Metadata Project (2020b).

Translating SDTL into Natural Language

In addition to translating five statistical languages into SDTL, the C²Metadata Project has a simple way of translating SDTL into a more human-friendly form. We have created a set of templates for each SDTL command with text surrounding each of its properties. For example, the template for the SDTL Compute command is

Set {variable} to {expression}.

in which {variable} and {expression} are properties of the command. Each of these properties can be resolved into text, such as a variable name or a number. Using the example in Figure 3, {variable} resolves to "age_years" and {expression} resolves to "age_months/12". The result is

Set age_years to (age_months/12).

Note that {variable} resolves to "age_years" in one step, but the {expression} property is more complicated, as often happens in SDTL. In this case, the expression is a function with two parameters. The Function Library gives this template for division

(EXP1/EXP2),

and we find that EXP1 resolves to a variable named "age_months" and EXP2 resolves to the numeric constant "12". Since SDTL types are often nested several levels deep, resolving SDTL into natural language is a recursive process. The Pseudocode Translator application uses templates like these to convert SDTL into something approximating English.

Templates for SDTL commands are collected in a Pseudocode Library, which is a file in JSON format. (Pseudocode is a term used for the translation of a computer program into language that is easier for humans to decipher.) The Pseudocode Library can be revised and extended without changing any code in the Pseudocode Translator, and versions of the Pseudocode Library can be created for other languages or special purposes.

Embedding Variable Derivations in Metadata

The full benefits of variable-level provenance are realized when SDTL descriptions of data transformations are combined with other metadata. SDTL commands describe processes and their inputs. The C²Metadata model assumes that the data were described in a metadata file before the data transformation script was executed. C²Metadata Updater applications (see Figures 1 and 2) update the pre-transformation metadata file by embedding SDTL into the description of each transformed variable to make an updated metadata file. Since the updated metadata file includes the original metadata file in its entirety, data catalogs and codebooks can show the state of the dataset before transformations were applied. In addition, the Updater assigns each of the SDTL commands to each of the variables that it modifies. Since some SDTL commands use variable lists and variable ranges to modify many variables at the same time, identifying which variables are affected is not always obvious.

The C^2Metadata Project uses elements from DDI Codebook version 2.5 to attach data transformation descriptions to individual variables (DDI Alliance, 2014). A variable can be described by the "derivation" element in the DDI Codebook schema, which has two content elements "drvcmd" (derivation command) and "drvdesc" (derivation description). A derived variable can be described with multiple "drvcmd" elements, which allows us to include both the SDTL and the source language versions of every command that modified the variable. The "drvdesc" element is used for the natural language (pseudocode) version of each command. Commands that operate on the entire dataset, like AppendDatasets and MergeDatasets, are described with a "fileDerivation", which we have proposed for inclusion in the next version DDI Codebook.

DDI enhanced with SDTL provides a way to describe data transformations involving temporary variables and variables that change later in a command script. Every variable in a DDI metadata file receives an ID that is independent of the variable name. We can document the intermediate states of a variable by generating a new ID and variable description every time it changes. Derived variables can be linked to the relevant state of antecedent variables through their IDs even if the antecedent variable will be changed in later program steps. The DDI metadata file can also track temporary variables, i.e. variables that are created and deleted without ever being saved to a file. A DDI variable description is usually associated with a file, but a variable description without a file reference is either a temporary variable or a variable that will be transformed before being saved to a file.

Adding Variables to the PROV Model

The C^2Metadata approach to documenting histories of datasets and variables is compatible with the PROV model of provenance. The PROV model is a family of standards for describing data provenance recommended by the World Wide Web Consortium (Groth & Moreau, 2013). The original PROV model did not describe variables within datasets or commands within programs, but several extensions of PROV offer more granular approaches to data and data processing (Cuevas-Vicenttín et al., 2015; End to End Provenance Project, 2019; Garijo & Gil, 2013). The extended PROV model can be queried to find relationships between variables and data transformation commands. For example, we can ask which derived variables were affected by a specific antecedent variable or command in a script. Or, we can ask which antecedent variables affected a specific derived variable. Since SDTL is machine actionable, it facilitates reconstructing relationships between variables and program steps for representation in a PROV-compatible ontology.[6]

Documentation, Execution, Translation

The goal of the C^2Metadata Project has always been automated production of human readable documentation, but we are aware that more ambitious goals are possible. One of our early decisions was that SDTL would not be a machine executable language. Languages for machines must be precise and completely specified. Documentation can be valuable even when it is

[6] The C^2Metadata Project is grateful to the Whole Tale Project (https://wholetale.org/) for their work on integrating SDTL into the ProvONE model, especially Thomas Thelen, Matt Jones, Bertram Ludaescher, Tim McPhillips, and Craig Willis.

incomplete and sometimes imprecise. Humans are accustomed to making decisions under uncertainty and using other knowledge to resolve ambiguities. SDTL has been designed to describe data transformations, but it is not intended to guide the execution of data transformations.

During the course of the project we discovered that a third goal is possible – translation from one statistical language to another. Organizations often rely on large bodies of code in languages that have become difficult to maintain. Data management scripts in statistical analysis software may use features that are removed in later releases, and younger analysts may be unfamiliar with packages and languages that were prevalent a decade earlier. Under these circumstances, an application that can translate one statistical language into another could be very useful, even if the translation is less than complete. Although it is not part of the current project, we have come to believe that SDTL can be used as an intermediate step in translating between statistical languages.

Limitations

The scope of the C^{2}Metadata Project was limited in several ways to keep the project manageable with limited funding and time. We were aware from the start that we could not capture every data transformation feature available in large and complex languages like SAS and R. Our goal has always been to capture 80% to 90% of the commands that researchers use for data management. Moreover, the goal of the project has been automated production of documentation for humans not machines. We did not design SDTL as a language for directing computers to perform data transformations.

A basic limitation of tools developed on the C^{2}Metadata Project is that they operate only on metadata files and do not access any data directly. We rely on a description of the data prior to transformation in a metadata file with a standard format. This decision simplified the creation of Parsers, because they do not need to read and analyze data files, but it did prevent us from implementing some features.

- Metadata files often include descriptive statistics of variables, such as averages and frequency distributions, which are greatly appreciated by researchers. Since the current tools do not access the data, we cannot compute descriptive statistics for variables that have changed.

- Data transformation commands that depend upon the content of the data are not currently implemented. We have specified a ReshapeWide command in SDTL, but it cannot be supported in a metadata-only system. Reshaping data from a "long" to a "wide" format involves changing the unit of observation to a higher level, such as from individuals to households or counties to states. The new data has one row for every case at the group level (household, state) and separate columns for the attributes of every individual within a group. Suppose that data from a census are arranged with one row per person, and we want to reorganize the data to one row per household. The variables for each person in the household will become columns in the new data file, i.e. the age of the first person in the household will be in column Age1, the age of the second person in Age2, and so on. The number of columns for each variable (age, sex, occupation,…) depends upon the number of people in the largest household. Since we cannot know the size of the largest household without accessing the data, reshaping from long to wide is not possible using only metadata.

R and Python languages are much more open than earlier statistical packages, and the communities supporting each of these languages have contributed thousands of libraries that add new operations and analyses. To limit the scope of our project, we have focused on the "base" and most popular data transformation libraries in each language. The SDTL parser for R is implementing the tidyverse library (Wickham et al., 2019), and the Python parser works with the Pandas library (The pandas development team, 2020).

Since the C^{2}Metadata Project is designed to describe datasets, we do not describe the outcomes of analysis commands. The original text of an analysis command may be included in an SDTL script with the **Analysis** command, but we do not describe tables, graphs or other analytical results. However, analytical commands may also create new data. For example, regression models typically generate predicted values and residuals, which can be saved as new variables or separate datasets. We expect that support for data created by statistical procedures will be added to SDTL in the future.

Discussion

The C^{2}Metadata Project has demonstrated that it is possible to automate the capture of variable-level provenance metadata. Automation reduces the cost and increases the quality of documentation showing how users of statistical software transformed and manage their data. We have produced a set of applications that convert scripts from five statistical languages into a common intermediate language (SDTL), which is then embeded into two widely used metadata standards. By creating human-readable histories of variables, we provide metadata that is much more detailed and informative than a long list of commands in an unfamiliar language.

We believe that several innovations in our approach are worth noting. First among these is the creation of a Structured Data Transformation Language (SDTL) to serve as a standard way of representing data transformation commands. Since researchers are currently split among at least five statistical software packages, we created a new language that would work with all of them. SDTL is not intended to replace existing statistical languages, rather it is a *lingua franca* for applications like data catalogs, codebooks, and other data discovery and documentation tools. SDTL is expressed in a structured format (JSON) that is easily read by computer programs, and it is compatible with existing metadata standards.

Second, although SDTL has a small vocabulary, the SDTL Function Library makes it flexible and expandable. Functions are a familiar device in programming languages, and statistical packages rely heavily on functions for many operations, like generating random numbers and computing quantiles of probability distributions. SDTL extends this approach by using functions to describe arithmetic operations, logical conditions, and variables formed by aggregating over rows. The SDTL Function Library maps functions in other languages into their SDTL equivalents. Since all functions follow the same basic syntax, applications that parse other languages can translate functions into SDTL with a minimum of programming code. The Function Library can be expanded without any changes in applications that rely on it.

Third, we have also shown that translating SDTL into a human readable form is also a simple and extendable process. The Pseudocode Library is set of fill-in-the-blank templates for SDTL commands. Even complicated SDTL commands can be unfolded into properties that consist of pre-defined text, variable names, and numbers.

Finally, SDTL may serve as an intermediary in translations between statistical languages. Since the source languages have many idiosyncratic features, comprehensive translations are probably not attainable. However, translations covering 80 to 90 percent of a script will be extremely useful for many purposes. For example, many organizations have legacy scripts in statistical languages that their staff no longer understand. Incomplete translations accompanied by human-readable versions of the original scripts can be very helpful in redesigning out of date workflows.

References

Adelson, J. L., Barton, E., Bradshaw, C., Bryant, B., Bryant, D., Cook, B. G., . . . Dymond, S. (2019). A Roadmap for Transparent Research in Special Education and Related Disciplines.

ALPHA Network. (2020). ALPHA Network. Retrieved from http://alpha.lshtm.ac.uk/

Alter, G., Donakowski, D., Gager, J., Heus, P., Hunter, C., Ionescu, S., . . . Voldsater, O. (2020). *Provenance Metadata for Statistical Data: An Introduction to Structured Data Transformation Language (SDTL)*. ICPSR. University of Michigan. Ann Arbor MI. Retrieved from http://hdl.handle.net/2027.42/156015

American Economic Association. (2013). American Economic Review: Data Availability Policy. Retrieved from http://www.aeaweb.org/aer/data.php

American Political Science Association Committee on Professional Ethics Rights and Freedoms. (2012). *A Guide to Professional Ethics in Political Science*. In. Retrieved from http://www.apsanet.org/media/PDFs/ethicsguideweb.pdf

C2Metadata Project. (2020a). Gitlab Repository: c2metadata. Retrieved from https://gitlab.com/c2metadata

C2Metadata Project. (2020b). Structured Data Transformation Language. Retrieved from http://c2metadata.gitlab.io/sdtl-docs/

Coalition on Publishing Data in the Earth and Space Sciences (COPDESS). (2015). Statement of Commitment from Earth and Space Science Publishers and Data Facilities Retrieved from http://www.copdess.org/statement-of-commitment/

Cuevas-Vicenttín, V., Ludäscher, B., Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., & Leinfelder, B. (2015). Provone: A prov extension data model for scientific workflow provenance. In: DataOne Project, Tech. Rep., Mar. 2014.[Online]. Available: http ….

E.H. Fegraus, S. A., M.B. Jones, M. Schildhauer. (2005). 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, 86*, 158–168.

End to End Provenance Project. (2019). Extended Prov JSON. Retrieved from https://github.com/End-to-end-provenance/ExtendedProvJson

Freese, J., & Peterson, D. (2017). Replication in Social Science. In K. S. Cook & D. S. Massey (Eds.), *Annual Review of Sociology, Vol 43* (Vol. 43, pp. 147-165).

Garijo, D., & Gil, Y. (2013, 17 September 2013). The P-PLAN Ontology. Retrieved from https://www.opmw.org/model/p-plan/

Groth, P., & Moreau, L. (2013). *PROV-OVERVIEW: An Overview of the PROV Family of Documents*. Retrieved from http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

Holdren, J. P. (2013). *Increasing Access to the Results of Federally Funded Scientific Research. Memorandum from the Office of Science and Technology Policy. February 22, 2013*. Retrieved from http://www.whitehouse.gov/sites/default/files/microsites/ostp/ostp_public_access_memo_2013.pdf.

IBM Corp. (2019). IBM SPSS Statistics for windows, version 26.0. Armonk, NY: IBM Corp.

Kanjala, C. (2019). *Provenance of 'after the Fact' Harmonised Community-Based Demographic and HIV Surveillance Data from ALPHA Cohorts.* (PhD), University of London, London.

King, G. (2011). Ensuring the Data-Rich Future of the Social Sciences. *Science, 331*(6018), 719-721. doi:10.1126/science.1197872

Lupia, A., & Elman, C. (2014). Openness in Political Science: Data Access and Research Transparency. *PS: Political Science & Politics, 47*(01), 19-42. doi:doi:10.1017/S1049096513001716

Miguel, E., Camerer, C., Casey, K., Cohen, J., Esterling, K. M., Gerber, A., . . . Van der Laan, M. (2014). Promoting Transparency in Social Science Research. *Science, 343*(6166), 30-31. doi:10.1126/science.1245317

New York Times Editors. (2013, February 25, 2013). We Paid for the Research, So Let's See It. *New York Times,* p. A24. Retrieved from http://www.nytimes.com/2013/02/26/opinion/we-paid-for-the-scientific-research-so-lets-see-it.html?_r=0

Nosek, B. A., Alter, G., Banks, G. C., Borsboom, D., Bowman, S. D., Breckler, S. J., . . . Yarkoni, T. (2015). Promoting an open research culture: author guidelines for journals could help to promote transparency, openness, and reproducibility. *Science (New York, NY), 348*(6242), 1422.

Political Science Journal Editors. (2015). Data Access and Research Transparency (DA-RT): A Joint Statement by Political Science Journal Editors. Retrieved from http://www.dartstatement.org

Powers, S. M., & Hampton, S. E. (2019). Open science, reproducibility, and transparency in ecology. *Ecological Applications, 29*(1), e01822.

Python Software Foundation. (2019). Python Language Reference, version 3.8. Beaverton, OR. Retrieved from https://www.python.org/

R Core Team. (2013). R: A Language and Environment for Statistical

Computing. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org/

SAS Institute. (2015). SAS®9.4 Product Documentation. Cary, NC: SAS Institute Inc. Retrieved from http://support.sas.com/documentation/94/index.html

StataCorp. (2020). Stata Statistical Software: Release 16.1. College Station, TX: StataCorp LP.

The pandas development team. (2020). pandas-dev/pandas: Pandas: Zenodo. Retrieved from https://doi.org/10.5281/zenodo.3509134

Vardigan, M., Heus, P., & Thomas, W. (2008). Data documentation initiative: Toward a standard for the social sciences. *International Journal of Digital Curation, 3*(1).

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D. A., François, R., . . . Hester, J. (2019). Welcome to the Tidyverse. *Journal of Open Source Software, 4*(43), 1686.