IJDC | *Research Paper*

Bridging the Gap Between Process and Procedural Provenance for Statistical Data

George Alter
University of Michigan

Jack Gager
Metadata Technology North America

Jeremy Iverson
Colectica

Meng Li
University of Illinois at Urbana-Champaign

Bertram Ludäscher
University of Illinois at Urbana-Champaign

Timothy McPhillips
University of Illinois at Urbana-Champaign

Thomas Thelen
University of California at Santa Barbara

Dan Smith
Colectica

Abstract

We show how two models of provenance can work together to answer basic questions about data provenance, such as "What computed variables were affected by values of variable X?". Questions like this are central for understanding how data is managed and modified. W3C PROV is a widely used standard for describing the people, activities, and sources that create things like documents, a work of arts, and data sets. PROV associates processes with inputs and outputs, but it does not have a way to describe how data are changed within a process. PROV has no language for program components, like mathematical expressions or joining data tables. Structured Data Transformation Language (SDTL) was designed to provide machine-actionable representations of data transformation commands in statistical analysis software. SDTL describes the inner workings of programs that are black boxes in PROV. However, SDTL is detailed and verbose, and simple queries can be very complicated in SDTL. Structured Data Transformation History (SDTH) bridges the gap between PROV and SDTL. SDTH extends the PROV data model to answer questions about data preparation and management operations not available in PROV.

Submitted 9 February 2024 ~ *Accepted* 16 June 2025

Correspondence should be addressed to **George Alter**, Population Studies Center, University of Michigan, Institute for Social Research, 426 Thompson St, Ann Arbor, MI 48106-1248. Email: altergc@umich.edu

This paper was presented at the International Digital Curation Conference IDCC24, 19-21 February 2024

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Introduction

Structured Data Transformation History (SDTH) offers a standard way to describe and query programs that modify data used for statistical analysis. As we move toward increased reuse and interoperability of data from different sources, greater transparency about how data were processed becomes essential for FAIR data (Wilkinson et al., 2016). However, programs that manage and modify data can be long and complex, making it very difficult to see how variables were created and modified. SDTH contributes to making data FAIR by providing a simple, searchable representation of data transformation programs.

Our development of SDTH was motivated by the need to answer the following four guiding questions about dataframes (rectangular data structures) and variables (columns in dataframes)[1]:

1. What dataframes/variables affected the values of variable X or dataframe Y?
2. What dataframes/variables were affected by variable X or dataframe Y?
3. What commands affected the values of variable X or dataframe Y?
4. What commands were affected by variable X or dataframe Y?

These are fundamental questions for anyone trying to understand or audit a program or command script. For example, data repositories must understand data management scripts to evaluate requests for release of outputs derived from confidential data. Much of the data used in biomedical and social science research is only available in "data enclaves," such as the Federal Statistical Research Data Centers operated by the US Census Bureau. Researchers working in these facilities can analyse restricted-use data, but tables, graphs, and other files cannot be removed from the secure environment without approval. Since some variables are more likely to disclose sensitive information than others, a query like "Which variables were affected by the variable 'age'?" will be helpful in auditing code used to produce outputs requested for public release.

SDTH bridges the gap between PROV (Groth & Moreau, 2013), the widely used World Wide Web Consortium provenance model, and languages that are used to manage and modify data, such as R (R Core Team, 2013), Python (Python Software Foundation, 2019), SPSS (IBM Corp, 2019), SAS (SAS Institute, 2015), and Stata (StataCorp, 2020). We refer to these languages as 'procedural,' because commands are processed in a pre-defined order. In contrast, PROV is expressed in Resource Description Framework (RDF) schema, which describes relationships among things without a predefined order. PROV also lacks elements for describing complex objects, like data files with internal structures and computer programs with multiple steps. We use Structured Data Transformation Language (SDTL) (Alter et al., 2020; Alter et al., 2021; DDI Alliance, 2025e), a software-independent language for data transformations in statistical analysis software, as a stand-in for all procedural languages. SDTL is designed to describe data transformation operations in enough detail to be executable and predictable.

[1] The rows in a dataframe describe individuals, which may be persons, places, years, or other 'units of observation.' Columns in a dataframe are attributes of these individuals, like name, age, the response to a question, population, Gross Domestic Product, etc. We use 'file' to refer to data on persistent storage media, and 'data set' as a general term referring to any collection of data.

The difference between PROV and SDTL is not simply a question of detail. Each language uses fundamentally different concepts to describe data. Entities in PROV must have immutable attributes, while the dataframes and variables in procedural languages like SDTL are inherently changeable. An SDTL program can change the contents of a dataframe or variable but continue to use the same name to refer to the modified version.

We argue that the PROV model does not capture dependence at the fine grain needed. SDTL, on the other hand, remains a procedural language, in which variable names may be reused for completely unrelated concepts. Thus, SDTL is incompatible with the declarative approach to provenance exemplified by PROV. We propose SDTH as a full-fledged declarative provenance model that captures dependence at multiple granularities. SDTH uses the concept of a data "instance" to mediate between the immutable entities described by PROV and the mutable artifacts in SDTL.

PROV

PROV is a tool for describing the history of an object. It is designed to identify things, attribute them to persons or other entities, and represent the steps involved in creating and processing them (Groth & Moreau, 2013; Moreau et al., 2015). PROV enables interoperability across provenance gathered by different systems in diverse domains. PROV is expressed in RDF, which can be queried with the SPARQL query language and combined with RDF from other specifications.

PROV is based on three main concepts: Entity, Activity, and Agent. Entities are things, which includes physical, digital, and conceptual objects. Activities create entities and use entities to make new entities. An agent plays a role in an activity and has some responsibility for entities produced by the activity. PROV recognizes that a new entity is "generated" by an activity, and that other entities may be "used" in the creation of the new entity.

PROV defines an entity as "a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary." (Moreau & Missier, 2013, para. 5.1.1). PROV developed from experience with the Open Provenance Model (Moreau et al., 2008), in which artifacts were defined as immutable. This rule was relaxed in PROV, such that PROV entities may be mutable, as long as the attributes that matter are fixed (Moreau et al., 2015, pp. 243–244).

PROV is inadequate for answering the four questions posed above. First, PROV has no way of describing how a program works on a data set. PROV describes a workflow as a network of processes (activities) that input existing entities and output new entities. Each entity is immutable, and each process is a black box without any information about its internal processes. Second, PROV does not describe data sets. Even simple data formats, like comma-separated values (CSV), have meaningful internal structures, like records, variables, and variable names.

The ProvONE extension to PROV was intended to add more granularity for describing workflows (Cuevas-Vicenttín et al., 2016). ProvONE defines a Program as a subclass of the PROV Plan, which refers to a set of actions taken to achieve a goal. To make Programs more granular, ProvONE introduced the SubProgram, which is a component of a Program. ProvONE also models both *retrospective* and *prospective* provenance. Retrospective provenance is a history of a computation, including the tasks that were executed, objects used and created, and the environment in which the computation was executed. Prospective provenance is a recipe for performing a computation, including each of the steps to be performed. However, ProvONE inherits some of the limitations of PROV. For example, ProvONE does not have a way to describe structures within data sets like dataframes.

SDTL

SDTL was created by the "Continuous Capture of Metadata" (C2Metadata) Project (Alter et al., 2020; Alter et al., 2021) with the goal of automating the revision of metadata in standard formats, like Data Documentation Initiative (DDI Alliance, 2025) (Vardigan et al., 2008) and Ecological Metadata Language (EML) (Fegraus et al., 2005). Even when the original data was fully described in standard metadata, revising that metadata to reflect the results of command scripts is a time-consuming manual task. The C2Metadata workflow automates the re-creation of metadata files by using the command scripts that modified the data to update the metadata. SDTL plays an important role in the C2Metadata workflow by providing a common representation of data transformation commands in five widely used statistical analysis languages: SPSS, SAS, Stata, R (tidyverse), and Python (pandas). Unlike other languages, SDTL is structured in formats, such as JSON and XML, that can be interpreted by a computer program without parsing or syntax rules. SDTL is also potentially useful for translating between languages (Song et al., 2021).

Like the languages that it represents, SDTL is a procedural language. This means that SDTL commands are executed in a sequence of ordered steps, which we call a program or script. The outcome of an SDTL program depends on the order of the steps. The procedural approach is very different from PROV and ProvONE, which describe dataflows as networks.

SDTL uses three concepts to describe data used in statistical analysis: *dataframes*, *variables*, and *files*. Dataframes are rectangular data matrices in which rows are observations on individuals, which could be persons, countries, years, or any other entity. Columns are variables (attributes) describing those individuals. The intersection of a row and column in a Dataframe identifies a single data value, which may be a number or text. We use Dataframe to refer to data in computer memory during the execution of a Program. When a Dataframe has been stored on a persistent medium, we call it a File. Files must be loaded by a Program to create Dataframes, which are acted upon by the Program.

SDTL includes all the information needed to answer our four guiding questions. The C2Metadata Project demonstrated that SDTL can be used to derive variable-level provenance. As described in Alter et al. (2021), C2Metadata tools create an interactive codebook, in which every variable in a data set is hyperlinked to all commands and variables that affect its values.

Although SDTL includes all the information needed to answer our four guiding questions, querying SDTL is not easy. SDTL is verbose. SDTL elements may have multiple properties and may be nested several levels deep. Our guiding questions only ask about program steps, files, dataframes, and variables, and they do not involve the specifics of data transformation found in SDTL. In addition, the procedural structure of SDTL requires retaining the order in which commands are executed, which adds to the complexity of SDTL RDF. SDTH was created to provide a way to write simple RDF queries about key aspects of a Program without all the details included in SDTL.

Variables Versus Entities

The first step in transporting SDTL to a PROV-inspired schema is recognizing that Dataframes and Variables in procedural languages do not qualify as PROV Entities. As we noted above, PROV Entities are stable and immutable objects. In contrast, Dataframes and Variables change during the execution of a Program. We think of the cells in a Dataframe as containers. The contents may change, but the container is the same. A Program may use the same name to refer to a Dataframe or Variable, even though the values in that data structure have changed many times.

To describe Dataframes and Variables in a way consistent with PROV, we introduce the concept of the state or "instance" of a Dataframe or Variable. A DataframeInstance or VariableInstance is a specific set of values associated with the name of a Dataframe or Variable. Only one instance may be associated with this name at any given time, but any number of instances may share a name during the execution of a Program. Identifying instances of data structures also allows us to unambiguously link them to program steps and to each other via the prov:wasDerivedFrom relationship.

The definition of a VariableInstance in SDTH is equivalent to an "Instance Variable" defined in the Generic Statistical Information Model (GSIM) (United Nations Economic Commission for Europe (UNECE), 2024). It is also closely related to an "Instance Variable" in the DDI Cross-Domain Integration (DDI-CDI) standard recently released by the DDI Alliance (2025a).

SDTH assigns IDs to instances of Variables, Dataframes, and Files: sdth:VariableInstance, sdth:DataframeInstance, sdth:FileInstance. Each instance is a PROV entity, because any change in the values of its data results in a new instance.

SDTH Schema

The elements of SDTH are shown in Tables 1 and 2. SDTH has only five entities: Program, ProgramStep, FileInstance, DataframeInstance, and VariableInstance. These entities appear as subjects or objects of the SDTH predicates shown in Table 3. (See DI Alliance (2025b) for SDTH documentation.)

Table 1. SDTH Entities

sdth:Program	A program that modifies data, which consists of one or more ProgramSteps.
sdth:ProgramStep	A step in a Program. A ProgramStep may consist of several ProgramSteps.
sdth:FileInstance	A data file on a storage device. A FileInstance identifies a data file in a specific state.
sdth:DataframeInstance	A dataframe is a rectangular data array of rows (observations) and columns (variables). A DataframeInstance identifies a dataframe in a specific state.
sdth:VariableInstance	A variable is a vector of values in the column of a dataframe. A VariableInstance identifies a variable in a specific state.

Table 2. SDTH Predicates

Allowable Subjects	Predicate	Allowable Objects
sdth:Program sdth:ProgramStep	**sdth:hasProgramStep**	sdth:ProgramStep
sdth:ProgramStep	**sdth:hasSourceCode**	text
sdth:ProgramStep	**sdth:hasSDTL**	text

sdth:ProgramStep	**sdth:loadsFile**	sdth:FileInstance
sdth:ProgramStep	**sdth:savesFile**	sdth:FileInstance
sdth:ProgramStep	**sdth:consumesDataframe**	sdth:DataframeInstance
sdth:ProgramStep	**sdth:producesDataframe**	sdth:DataframeInstance
sdth:ProgramStep	**sdth:usesVariable**	sdth:VariableInstance
sdth:ProgramStep	**sdth:assignsVariable**	sdth:VariableInstance
sdth:FileInstance sdth:DataframeInstance	**sdth:hasVarInstance**	sdth:VariableInstance
sdth:FileInstance sdth:DataframeInstance sdth:VariableInstance	**sdth:wasDerivedFrom**	sdth:FileInstance sdth:DataframeInstance sdth:VariableInstance
sdth:FileInstance sdth:DataframeInstance sdth:VariableInstance	**sdth:elaborationOf**	sdth:FileInstance sdth:DataframeInstance sdth:VariableInstance
sdth:FileInstance sdth:DataframeInstance sdth:VariableInstance	**sdth:hasName**	text

ProgramSteps are PROV Activities that act upon FileInstances, DataframeInstances, and VariableInstances. Every ProgramStep is associated with both an input data entity (loadsFile, consumesDataframe, usesVariable) and an output data entity (savesFile, producesDataframe, assignsVariable) activity. ProgramSteps are linked to the original data transformation script through hasSourceCode and hasSDTL, which provide the original code and its SDTL equivalent respectively.

FileInstances and DataframeInstances are collections of VariableInstances, which are linked by hasVarInstance. A VariableInstance may exist in more than one FileInstance and DataframeInstance.

We use wasDerivedFrom and elaborationOf to describe two ways that a new data instance (i.e., a VariableInstance, a DataframeInstance, or aFileInstance) may be related to a previous data instance. When a ProgramStep changes the values in a data instance, we use wasDerivedFrom to link the new data instance to all previous data instances that affected it. Changing the order of rows in a Dataframe also results in a new DataframeInstance, because data transformation commands often use data from more than one row. For example, most statistical analysis software includes a "lag()" function that accesses the value of a Variable on the previous row.

The elaborationOf predicate is used when a new data instance has the same data values as its predecessor, but metadata have changed. Metadata includes labels, descriptions, and other explanatory information. For example, adding labels to the values of a categorical (factor) variable results in a new VariableInstance linked to the previous instance with elaborationOf.

The hasName predicate plays a simple but essential function by linking instances of Files, Dataframes, and Variables in SDTH to their mutable counterparts in SDTL and the source code.

An Example Program in SDTH

We discuss here an example program written in Python. Due to space limitations, the example program and related files are available at DDI Alliance (2025c). This program loads two CSV files derived from a study of political opinions into dataframes. One dataframe describes the subjects' personal characteristics, and the other holds their responses to questions. The program creates two new variables from a measure of household size: 'HHsize' and 'HHcateg'. 'HHcateg' is transformed from its original values into a set of six categories. The revised personal data set is merged with the political data, and then it is saved to a file.

The most complicated command in the program is:

PersonalData['HHcateg'] = pd.cut(PersonalData['HHsize'], [1, 2, 3, 5, 7, 10, 999], include_lowest=True, right=False, labels=['1', '2', '3-4', '5-6', '7-9', '10+'])

This command performs several actions, which are shown in Figure 1. First, the values of HHsize are assigned to a new categorical variable (HHcateg) using a set of cut points (#ProgStep0005a). Second, the data type of the new variable HHcateg is set to the Python data type "Factor" (#ProgStep0005b). Third, the command assigns labels to the values of HHcateg (#ProgStep0005c). Although this is one command in Python, we also show SDTL, which translates it into three simpler SDTL commands: Compute, SetDataType, and SetValueLabels. (SDTL commands are abridged to simplify the graph.) This decomposition helps SDTL to cover a variety of languages, each of which combines actions into commands in different ways.

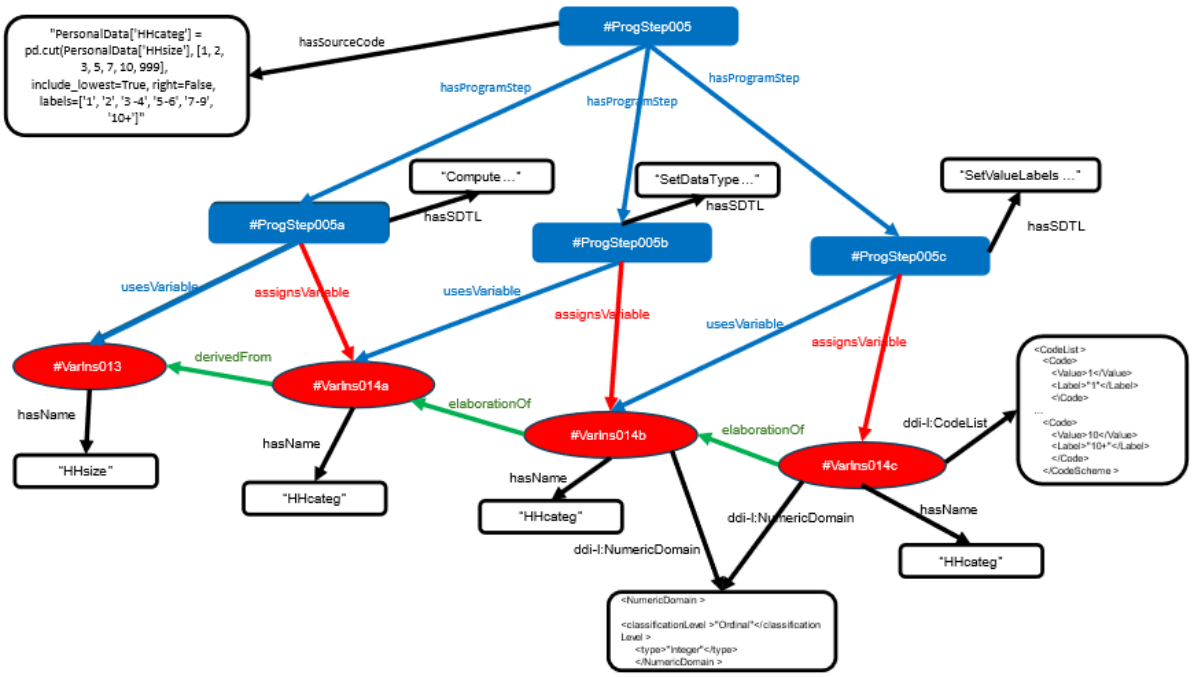

Figure 1. Graphical representation of line 5 in the Python script.

Figure 1 shows the SDTH for the command in line 5 as a graph. In SDTH, we can represent the relationship between the original Python command (#ProgramStep005) and the more granular SDTL equivalent (#ProgramStep005a, #ProgramStep005b, #ProgramStep005c), because an SDTH ProgramStep can be composed of other ProgramSteps. Each of these ProgramSteps creates a new VariableInstance (#VarIns014a, #VarIns014b, #VarIns014c). However, all three of these VariableInstances are associated with the same variable name in Python, "HHcateg".

Figure 1 also shows the difference between SDTH predicates “derivedFrom” and “elaborationOf”. The first command in this sequence (#ProgramStep005a: “Compute”) changes values in the data, and its output VariableInstance (#VarIns014a) is linked to its input VariableInstance (#VarIns013) by sdth:derivedFrom. In contrast, the next two ProgramSteps only affect metadata. The SDTL “SetDataType” command (#ProgramStep005b) changes the data type of variable “HHcateg” to a Python “Factor”, and the SDTL “SetValueLabels” command (#ProgramStep005c) adds value labels. We use sdth:elaborationOf to describe the relationships between the output and input VariableInstances of these ProgramSteps, because values in the variable do not change. We describe these metadata attributes with elements from the DDI-Lifecycle (DDI Alliance, 2025b) standard, ddi-l: NumericDomain and ddi-l: CodeList, to describe attributes of VariableInstances. The example in Figure 1 uses XML for these attributes, but the DDI-Lifecycle standard also allows artifacts like CodeLists to be identified by URIs.

Querying SDTH

We give an example here of a SPARQL query answering one of the questions posed above. (See DDI Alliance (2025c) for other SPARQL queries.) Specifically, we tailor the third question to the SDTH example program as: “What commands affected the values of variable HHcateg?” The SPARQL for this query traces the origins of HHcateg to earlier VariableInstances using the wasDerivedFrom predicate, and then it outputs the command that produced each VariableInstance by selecting program steps through the assignsVariable predicate:

Example SPARQL query

```
PREFIX sdth: <http://DDI/SDTH/>
PREFIX sdtest: <http://test/#>
SELECT ?sname ?oname  ?pscode
WHERE {
   ?s  sdth:wasDerivedFrom+ ?o .
   ?s  sdth:hasName ?sname .
   ?o  sdth:hasName ?oname .
   ?pstep sdth:assignsVariable ?o.
   ?pstep sdth:hasSourceCode ?pscode.
FILTER (?sname = "HHcateg")      }
```

Output from this query is shown in Table 4.

Table 4. SPARQL Output

Derived variable name (?sname)	Command (?pscode)	Source variable name (?oname)
HHcateg	PersonalData = pd.read_csv("SmallTestPersonal.csv")	PPHHSIZE
HHcateg	MergedData = PersonalData.merge(PoliticalData, on="ID", how="inner")	PPHHSIZE
HHcateg	PersonalData =	HHsize

	PersonalData.assign(HHsize=PersonalData['PPHHSIZE'])	
HHcateg	MergedData = PersonalData.merge(PoliticalData, on="ID", how="inner")	HHsize

Metadata

FileInstances and VariableInstances in SDTH provide a bridge between the PROV world and metadata standards, like DDI (Vardigan et al., 2008), EML (Jones et al., 2019), Dublin Core (DCMI Usage Board, 2010), Data Catalog Vocabulary (DCAT) (Albertoni et al., 2024), and schema.org (W3C Schema.org Community Group, 2024). A data set may be accompanied by many types of descriptive information in the form of metadata. Variables are characterized by labels, data types, display formats, value labels, and other attributes. Files may have authors, titles, version dates, etc. Rather than duplicating metadata properties that can be found in other standards, SDTH RDF can be used with types found in other standards. For example, an SDTH FileInstance corresponds to a dcat:Dataset, which can be described by properties dcat:creator, dcat:title, and dcat:description.

Related Work

Both noWorkflow (Murta et al., 2015; Pimentel et al., 2017) and the End-to-End Provenance Project (Lerner & Boose, 2015) provide query capabilities similar to those in SDTH. These tools collect fine-grained provenance during the execution of scripts in Python (noWorkflow) and R (End-to-End Provenance). To reduce the burden on researchers, they automate the collection of provenance by installing functions that operate in the background as a script is executed. After each program step, data objects (files, dataframes, variables, etc.) used, created or modified are recorded in a provenance information system and saved for examination. Since the provenance system registers every data transformation as a new object, they correspond to the data instances described in SDTH.

Although provenance maps in noWorkflow and the End-to-End suite identify the same nodes as SDTH, they differ in how nodes are linked by edges. The Data Derivation Graph (DDG) in the End-to-End suite creates links between program steps (procedure nodes) using the PROV "wasInformedBy" predicate. This creates an ordered sequence of program nodes to which data nodes are linked by "used" and "wasGeneratedBy" predicates. In contrast, SDTH creates ordered links between data nodes using the "wasDerivedFrom" predicate, and program steps are linked indirectly through their connections with data nodes. In other words, DDG focuses on program steps, and SDTH focuses on data. This is a minor difference, because the order of nodes can be inferred from the links between program steps and data nodes (SDTH:assignsVariable/DDG:wasGeneratedBy versus SDTH:usesVariable/DDG:used) without directly linking nodes of the same kind.

In general, the differences between the noWorkflow and the End-to-End Provenance projects and the C2Metadata Project, which developed SDTL and SDTH, are due to differences in the use cases that they were designed to serve. Both noWorkflow and the End-to-End suite were created to improve data analysis by making scripts more transparent and reproducible. Collecting provenance information during the execution of a program, allows them to collect both prospective and retrospective provenance and other information about the execution environment. In contrast, the C2Metadata Project (Alter et al., 2021) grew out of the need for variable-level provenance information

compatible with metadata standards like DDI and EML, which are widely used in the social and ecological sciences. Since SDTL was intended to describe data after transformations were performed, the tools developed by the C2Metadata Project create SDTL from scripts not during program execution. In most cases, SDTH can be derived from scripts rather than during program execution.

SDTH, Linked Data, and FAIR

SDTH addresses the R (Reusability) in the FAIR principles (Wilkinson et al., 2016). Data cannot be reused without an accurate description of its provenance, and SDTH shows how data was produced and transformed. SDTH can also be combined with other provenance information available in PROV and its extensions.

By applying persistent identifiers to describe data objects like files and variables, SDTH is compatible with FAIR and other applications of Linked Data. A typical data transformation program loads files from and saves files to storage devices. These data files can be described by persistent identifiers, such as the DOIs issued by DataCite used in many data repositories.

Data repositories could also provide persistent identifiers for variables, which would enable FAIR searching and data merging across repositories. Metadata formats, like the DDI standards, already accommodate URIs, and the infrastructure for variable-level persistent identifiers is being built. A growing number of data producers use the Colectica (2022) suite of tools, which are built around a DDI-compatible database of variables. The Statistical Data Exchange (SDMX) (2021) standard used by the official statistics community is developing standards for assigning URIs to variables, code schemas, and other data objects. Registries, like BioPortal, the National Library of Medicine Common Data Element Registry, and the European Language Social Science Thesaurus (ELSST) can use ontologies like OWL and SKOS to describe relationships between variables and concepts (National Center for Biomedical Ontology, 2005; National Institutes of Health, 2015; CESSDA, 2022; Horridge & Patel-Schneider, 2012; Miles & Pérez-Agüera, 2007).

Conclusion

Our development of SDTH began with four questions that are relevant to any program or command script modifying data for statistical analysis. The example presented above shows that all four questions can be answered by applying basic RDF tools, like SPARQL, to an SDTH graph, like Figure 1. All the queries in our example report the names of variables, dataframes, and files as they appear in the original source code. Program steps can be described by retrieving the text of the source code or its SDTL equivalent. Thus, these queries provide answers in terms of a procedural language, even though SDTH is a process language compatible with PROV.

The central insight in SDTH is that the data objects referenced in a procedural language are incompatible with the immutable entities described in PROV. In procedural languages, data objects (variables, dataframes, or files) change as the program is executed. We can only know the state of an object at a specific point in the program by working through all the steps that preceded it. Thus, the sequence of commands is essential to understanding the outcomes of the program in Python or SDTL. In SDTH, ProgramSteps are not explicitly sequenced, and data objects are immutable entities. Since an entity cannot change, a new entity is created every time any aspect of a variable, dataframe, or

file changes. We describe these states as unique instances of the data object, even though a program may use the same name to refer to multiple instances of a data object.

SDTH does not depend on SDTL, but there are advantages of creating SDTH from SDTL. Commands in SDTL tend to be simpler and more granular than in other languages, as we showed in the example above. Since code for prototype tools that convert other languages to SDTL already exists, a tool for converting SDTL into SDTH can serve multiple other languages. An SDTL-to-SDTH tool is under development at this time, which will be based on code used to create an interactive codebook from SDTL (Alter et al., 2021).

We are currently preparing documentation formally describing SDTH with the goal of proposing its addition to the suite of standards maintained by the DDI Alliance.

Acknowledgements

This paper is the product of a collaboration between two NSF-funded projects. “Continuous Capture of Metadata for Statistical Data Project” was funded by National Science Foundation grant ACI-1640575. "Merging Science and Cyberinfrastructure Pathways: The Whole Tale" was funded by National Science Foundation grant OAC 1541450. The authors are grateful for the helpful comments by anonymous reviewers.

References

Albertoni, Riccardo, Browning, David, Cox, Simon, Gonzalez Beltran, Alejandra, Perego, Andrea, & Winstanley, Peter. (2024). Data Catalog Vocabulary (DCAT)—Version 3. https://www.w3.org/TR/vocab-dcat-3/

Alter, G. C., Donakowski, D., Gager, J., Heus, P., Hunter, C., Ionescu, S., Iverson, J., Jagadish, H. V., Lagoze, C., Lyle, J., Mueller, A., Revheim, S., Richardson, M. A., Risnes, Ø., Seelam, K., Smith, D., Smith, T., Song, J., Vaidya, Y. J., & Voldsater, O. (2020). Provenance metadata for statistical data: An introduction to Structured Data Transformation Language (SDTL). *IASSIST Quarterly*, *44*(4). https://doi.org/10.29173/iq983

Alter, G. C., Gager, J., Heus, P., Hunter, C., Ionescu, S., Iverson, J., Jagadish, H. V., Ludaescher, B., Lyle, J., McPhillips, T., Mueller, A., Nordgaard, S., Risnes, Ø., Smith, D., Song, J., & Thelen, T. (2021, July 21). *Detailed Provenance Metadata from Statistical Analysis Software: TaPP Applications Track*. 13th International Workshop on Theory and Practice of Provenance (TaPP 2021).

Alter, G. C., Gager, J., Heus, P., Hunter, C., Ionescu, S., Iverson, J., Jagadish, H. V., Lyle, J., Mueller, A., Nordgaard, S., Risnes, O., Smith, D., & Song, J. (2021). Capturing Data Provenance from Statistical Software. *International Journal of Digital Curation*, *16*(1), Article 1. https://doi.org/10.2218/ijdc.v16i1.763

CESSDA. (2022). ELSST Thesaurus. https://www.cessda.eu/Tools/ELSST-Thesaurus

Colectica. (2022). Colectica. http://www.colectica.com/

Cuevas-Vicenttín, V., Ludäscher, B., Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., & Leinfelder, B. (2016). *ProvONE: A PROV Extension Data Model for Scientific Workflow Provenance*.

DDI Alliance. (2025a). DDI-CDI (DDI Cross-Domain Integration). https://ddialliance.org/ddi-cdi

DDI Alliance. (2025b). DDI-Lifecycle (DDI-L). https://ddialliance.org/ddi-lifecycle

DDI Alliance. (2025c). Example_A: SDTH with SPARQL Queries and Output. https://github.com/ddialliance/sdtl/tree/master/SDTH/Examples/Example_A

DDI Alliance. (2025d). Structured Data Transformation History (SDTH) Documentation. https://docs.ddialliance.org/SDTL/dev/SDTH/

DDI Alliance. (2025e). SDTL (Structured Data Transformation Language). https://ddialliance.org/sdtl

DCMI Usage Board. (2010). Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI. https://www.dublincore.org/specifications/dublin-core/dces/

Fegraus, E. H., Andelman, S., Jones, M. B., & Schildhauer, M. (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.

Groth, P., & Moreau, L. (2013). *PROV-OVERVIEW: An Overview of the PROV Family of Documents* (W3C Note). https://www.w3.org/TR/prov-overview/

Horridge, M., & Patel-Schneider, P. F. (2012). OWL 2 web ontology language manchester syntax (Second Edition). W3C Working Group Note.

IBM Corp. (2019). *IBM SPSS Statistics for windows, version 26.0*. IBM Corp.

Jones, M., O'Brien, M., Mecum, B., Boettiger, C., Schildhauer, M., Maier, M., Whiteaker, T., Earl, S., & Chong, S. (2019). Ecological Metadata Language version 2.2.0. https://doi.org/10.5063/f11834t2

Lerner, B. S., & Boose, E. R. (2015). RDataTracker and DDG Explorer: Capture, Visualization and Querying of Provenance from R Scripts. *Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9-13, 2014. Revised Selected Papers 5*, 288–290.

Miles, A., & Pérez-Agüera, J. R. (2007). SKOS: Simple Knowledge Organisation for the Web. *Cataloging & Classification Quarterly*, 43(3–4), 69–83. https://doi.org/10.1300/J104v43n03_04

Moreau, L., Freire, J., Futrelle, J., McGrath, R. E., Myers, J., & Paulson, P. (2008). The Open Provenance Model: An Overview. In J. Freire, D. Koop, & L. Moreau (Eds.), *Provenance and Annotation of Data and Processes* (Vol. 5272, pp. 323-+).

Moreau, L., Groth, P., Cheney, J., Lebo, T., & Miles, S. (2015). The rationale of PROV. *Journal of Web Semantics*, *35*, 235–257. https://doi.org/10.1016/j.websem.2015.04.001

Moreau, L., & Missier, P. (2013, April 30). *PROV-DM: The PROV Data Model*. https://www.w3.org/TR/prov-dm/#concept-software-agent

Murta, L., Braganholo, V., Chirigati, F., Koop, D., & Freire, J. (2015). noWorkflow: Capturing and Analyzing Provenance of Scripts. In B. Ludäscher & B. Plale (Eds.), *Provenance and Annotation of Data and Processes* (pp. 71–83). Springer International Publishing. https://doi.org/10.1007/978-3-319-16462-5_6

National Center for Biomedical Ontology. (2005). BioPortal. https://bioportal.bioontology.org/

National Institutes of Health. (2015). NIH Common Data Element Repository. https://cde.nlm.nih.gov/home

Pimentel, J. F., Murta, L., Braganholo, V., & Freire, J. (2017). noWorkflow: A tool for collecting, analyzing, and managing provenance from python scripts. *Proceedings of the VLDB Endowment*, *10*(12).

Python Software Foundation. (2019). *Python Language Reference, version 3.8*.

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

SAS Institute. (2015). *SAS®9.4 Product Documentation*. SAS Institute Inc.

Song, J., Jagadish, H. V., & Alter, G. C. (2021). *SDTA: An Algebra for Statistical Data Transformation*. 33rd International Conference on Scientific and Statistical Database Management, Tampa, FL, USA.

StataCorp. (2020). *Stata Statistical Software: Release 16.1*. StataCorp LP.

Statistical Data and Metadata Exchange (SDMX). (2021). SDMX. https://sdmx.org/

United Nations Economic Commission for Europe (UNECE). (2024). GSIM version 2.0. https://unece.github.io/GSIM-2.0/GSIMv2.html

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

W3C Schema.org Community Group. (2024). Schema.org. 6.0. https://schema.org/

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., & Bourne, P. E. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3. doi:10.1038/sdata.2016.18