PSFC/JA-17-45

Scientific data management with navigational metadata

J. Stillerman, M. Greenwald and J. Wright

January 2018

Plasma Science and Fusion Center
Massachusetts Institute of Technology
Cambridge MA 02139 USA

This work supported by NSF ACI-164029. Reproduction, translation, publication, use and disposal, in whole or in part, by or for the United States government is permitted.

Scientific data management with navigational metadata

J. Stillerman, M. Greenwald and J. Wright

Massachusetts Institute of Technology, Cambridge, MA, 02139 USA

Modern science generates large complicated heterogeneous collections of data. In order to effectively exploit these data, researchers must find relevant data, and enough of its associated metadata to understand it and put it in context. This problem exists across a wide range of research domains and is ripe for a general solution.

Existing ventures address these issues using ad-hoc purpose-built tools. These tools explicitly represent the data relationships by embedding them in their data storage mechanisms and in their applications. While producing useful tools, these approaches tend to be difficult to extend and data relationships are not necessarily traversable symmetrically.

We are building a general system for navigational metadata. The relationships between data and between annotations and data are stored as first-class objects in the system. They can be viewed as instances drawn from a small set of graph types. General-purpose programs can be written which allow users explore these graphs and gain insights into their data. This process of data navigation, successive inclusion and filtering of objects provides powerful paradigm for data exploration.

This paper lays out the underlying core ideas, progress to date, and plans for near term work.

Keywords: data science, metadata, annotation, navigation, graph data

1. Introduction

The size and complexity of the data collected during all kinds experimental science has been growing rapidly over time. This is due at least in part to improved measurement equipment, faster computers, larger storage, and better computer networks. In many spheres, experiments are also getting larger and more collaborative, and these collaborations are often spread out geographically. These factors combine to make descriptive metadata for these data sets imperative for their exploitation. Expressive metadata provides context documenting recorded measurements and computed results. If sufficient metadata is present, then the data can be understood and used by current and future collaborators, without relying on direct communication with the original experimenter, and relying on their memories. The smaller scale experiments of the past had a less urgent need to store and manage metadata, in that the data sets were smaller and more homogeneous, and the size of the scientific teams involved tended to be smaller. A modern experiment such as a tokamak or a particle accelerator may involve hundreds or even thousands of scientists. In order for them to work effectively they need to be able to access and understand the results recorded and stored by their colleagues. Data acquisition and data management tools have been evolving to meet these challenges.

This evolution has been not only in the capabilities and capacity of these tools but also in their level of abstraction. Hand recorded measurements were replaced by pen based chart recorders, oscilloscopes, and Polaroid scope cameras. These in turn led to purpose built hardware and software to record measurements with computers, general purpose hardware with purpose built programs, general data collection programs like LabView [1] and MDS [2], and finally general data collection and management systems such as MDSplus [3] and workflow engines like Kepler[4]. Over time these tools increased in their generality and allowed users to represent their data at higher levels of abstraction.

Most current experiments have ad hoc systems to store and navigate a subset of the relationships between their data. For example, they connect experimental proposals to experiment operations, comments on those operations, machine status, and data provenance. These systems have tended to encode data relationships explicitly, often in the user interface program code itself. Furthermore, the navigation is often asymmetric (e.g. a logbook entry refers to some particular stored data, but this fact is not easily discernable when the data is accessed). Many of the ideas for this project stem from previous work by the authors, Metadata Ontology Project[5], which represented data provenance information as graphs, and data object references as uniform resource identifiers, URIs[6]. It is a logical next step, generalizing the storage and retrieval of relationships between stored data.

2. Problem Statement

This project will provide a set of tools to define, store, and retrieve a general representation of the relationships between stored data. Distinguishing these from the underlying data stores allows users to create multiple organizations of the same underlying set of objects. These organizations can then be explored using a successive browse (navigate) and filter/search user interface. Users filter by specifying criteria to limit the result set. For example, find things that are ‘annotations’, ‘made by me’, ‘yesterday’. Once a reasonably small list of answers is returned, one can browse their summaries, reorder them and apply further filters or expansions to the list. After locating objects of interest, the user will be able to

author’s email: firstname.lastname@some.mail.server

discover what relationships the objects are members of, and the adjacent objects in those relationships. This adjacency is a key feature enabling data discovery.

A key element of the system is its ability to query the relationships of research objects, and then look at the adjacent objects with respect to those relationships. Figure 1 provides a cartoon of this multiple facet traversal.

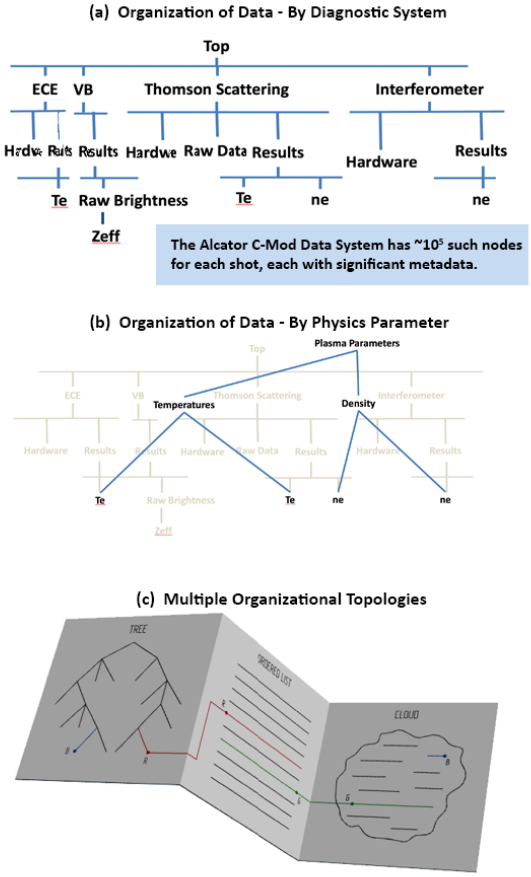

Figure 1 - Data relationships as graphs. (a) and (b) are alternate hierarchical representations of the same collection of nodes. (c) shows a node as part of a hierarchy, an ordered list, and an unordered set.

Because we will store relationships and adjacency internally, will be able to make them traversable symmetrically, addressing one of the limitations of previous ad hoc systems. Once target objects are located, the system will, where possible, invoke the native tools to examine them. At all stages, users will be able to add annotations which will become first class relatable elements in the system. Adding new relationships will be done by modifying the schema which is also stored as data in the system. Programs will automatically be aware of the new stored relationships, without requiring new coding in applications. A typical modern science experiment stores a large set of kinds of information, many of which must be integrated by data access tools, or by the user by hand, in order to understand the experimental results. For example, at a tokamak doing magnetic fusion research they collect and store:

- Hierarchical data stores with raw and processed data
- Relational databases with "high level" results
- Electronic logbooks & annotations
- Data provenance graphs
- Data catalogs
- Experiment planning documents
- Records about personnel
- Experimental proposals
- Simulation inputs & outputs
- Source code management systems
- Facility information, with details of
- experiment, measurement systems
- Document management systems
- Publications & presentations

as well as many other items. Some of these are navigable using purpose built applications, others are hermetic external systems. Each of these kinds of information, has a primary organizational paradigm, which orders them and facilitates access to their records. For example, the logbook might contain records organized around experimental run days, instances of the experiment and topic. The raw and processed data, stored in MDSplus for example, are arranged in a hierarchy defined by the site or by individual users. The new system will support multiple organizations of the same underlying data. A user could view the collection of stored records by diagnostic, by measurement type, or by the location / sightline of the diagnostic, see Figure 1. These multiple organizations will facilitate users finding data relevant to their research.

3. Solution

Our solution to these challenges is based on the idea that relationships can be thought of as graphs, that is collections of nodes and edges. Both of which are subclasses of a general object type which all of the user stored records in the system share. The 'object' meta type provides fixed properties that all instances are guaranteed to have: author, date-time-inserted, and type. Type is used both as a gross filter and to inform the user interface system how to display, modify and interact with the object. Each node type will have a list of required and optional properties. They can be either self-contained, with their 'data' stored as values of the properties, or refer to external data stores. In this latter case, one or more of their properties are URIs. Examples of these include: records in MDSplus trees, HDF5 files [7] or records in them, other files in a filesystem, documents or drawings in a document management system, etc… Similarly, the relationships, or edges in these graphs, will have types and properties. The required and optional lists of properties

for each type of object and relationship form the schema for the system.

A small set of graph topologies can accommodate most of the relationships we need to represent. Users of the system will define their schemas as instances of these metaschemas, and ontologies of the properties of their nodes and edges. The small number of graph types allows general applications to be produced that can be applied not only within local groups or domains, but between disparate communities of users. A hierarchical data explorer could be used on multiple data hierarchies at a site, and shared with users in other science domains.

We have initially identified five classes of graphs the system will need to support They are:

- Hierarchies
- Ordered lists
- Unordered lists
- Timelines
- Threaded annotations

While these are not formally topologically distinct, distinguishing them is useful both from both implementation and user understanding perspectives. . Figure 1 shows stored data as multiple hierarchies ((a), (b)) and multiple topologies (c). If we encounter data and relationships which we cannot fit into one of these, the list can be expanded. However, at this metaschema level, topology is dealt with in code as opposed to in data. This list will be quite static, and by design, short.

Application specific schema is stored as data by the system. Communities will curate collections of kinds of graphs which are useful to support their data discovery applications. Adding a new hierarchical organization of objects is a data addition as opposed to a code change. However, limiting the set of stored relationships is required in order for users to be able to exploit the system. If a group stores every possible relationship between all of their data, separating the useful and interesting ones from the rest would be impossible. Further, the relationship types, object types, and their respective required and optional property lists, must be labeled with a limited vocabulary for them to be useful. If some people objects have 'name' properties and others have a 'full name' properties, it would be difficult for users of the system to know that these refer to the same thing. Other examples of this are Plasma Current vs IP, H-mode, vs HMode plasmas, etc.. Accordingly, we will support the creation and curation of ontologies for object types, relationship types, their properties and values. Applications can take advantage of these ontologies to build user interfaces specifically tailored to their use. In addition the formalism of defining 'vocabulary' will help users of the systems think carefully about the organization, meaning and content of their metadata and data.

Given an object in the system, applications need to be able to query its type, the set of relationships it is a member of, and the adjacent nodes in any of those relationships. For example, an annotation object is of type annotation, it may be a member of a threaded conversation, it may refer to some stored data, it was made by a user and there are adjacent entries by time, or subject. This list, while not exhaustive, demonstrates the need to curate the list of data relationships that are stored. It would not be interesting to sort annotations alphabetically for example.

These basic operations will make it easy to refactor the common data organization tools in use for an experiment in a more general fashion. A scientific notebook application can be described as a selector for objects of type annotation which meet additional criteria, and then a rendering of those objects with a specified ordering and logbook like layout. These logbook entries can also be members of other graphs, and it will be easy for the application to allow users to explore the connected objects. Annotations will contain markup tags to embed references to other objects within their text. For example the text of an annotation might contain: '…density rise at <data-ref tree="cmod", shot="$shot", signal="\ne-bar", time = "3.5 Sec">. These 'tags' would also be stored and indexed separately so that applications wanting to do the reverse navigation could access them quickly. The details of the tag syntax has not yet been designed. However, there will be purpose built markup editors to help users to insert them. In addition, tags will be rendered by the GUIs in type specific ways. Images as thumbnails, people as navigable contact links, etc… Any tags the application does not have a specific renderer for, will be displayed as their source text. If an entry refers to data, the application can offer to display that data, or show its context in the data system.

Populating the system will require significant effort on the part of its users. This is exacerbated by several factors. Descriptive meta data most benefits its consumers, not its providers, demotivating users to provide them. Existing programs would need to be modified to provide (and consume) this new relationship metadata. Design decisions about what to store and how to represent it must be made in the absence of experience using the system. Our design attempts to take these into account and mitigate them where possible.

The system will store what it can automatically. When a user enters a record, the system knows who is entering it, when it is being entered, and what kind of record it is. Depending on the context the system may be able to populate more of the object's properties automatically. For example, if an object is replacing (overwriting) another object, the system can attach the new object to its original version.

Existing data stores can be mined to create object and relationship records that reflect their contents. This will be done as a combination of extraction/replication, and the construction of records which refer to the existing data in place.

Example schemas, both scientific domain specific (other tokamaks, other earth science data sets) and from other domains will inform users initial decisions about

what to store in the system. Pushing the specifics – the schema – to data, instead of having it encoded in programs, the cost of deciding to make changes to this, if the initial choices prove to be bad, is minimized.

4. Implementation Plan

We will begin with concrete applications of these ideas, and then proceed to generalize the system. By solving a real-world problem, with users that have real problems to solve, we can verify that the software we are developing meets their needs. At the same time we are collaborating with potential users from a variety science domains. These include: other plasma physics experiments, physics computer modelers, earth science researchers, and people working in digital humanities.

We will begin by focusing on two data sets and users at the MIT Plasma Science and Fusion Center. The first will be retrospective, applying the software to the well understood needs of an existing experiment. The Alcator C-Mod tokamak [8] has an ad hoc system for metadata associated with experimental operations[9]. Our first step is to refactor these into the new more general framework. This will provide immediate validation of the approach. The refactored software must at a minimum work as well as the purpose-built collection of programs it replaces. The current system uses relational database tables to store experimental proposals, experiment run plans, descriptions of run days, individual shots on those days and user comments about those shots. Graphical user interfaces are implemented in PHP as a web 2.0 application.

The second application will be a new research effort at the Plasma Science and Fusion Center concerning high temperature superconductor magnet development. This work has similar workflows and metadata needs to the Alcator C-Mod tokamak, though the primary indices of the records will be different. The tokamak's annotations are indexed by run, shot, topic, user, and time. These conductor and magnet tests will be organized by work breakdown structure (WBS), test samples, test runs, recorded trending data recorded test data (MDSplus), and time. The similarities and differences between these requirements, provides an excellent check on the generality of our solution. The active users will provide essential feedback on the system functionality and usability.

As we carry out this initial development, we will continually refer back to our overall design. Iterating the design and implementation so that in the next steps we have confidence that our metaschema matches our problem space. These initial applications will provide the design validation. Do they adhere to the design and are they extensible? If not, iterate.

The next step is to create records describing the schema, iterating the implementations as necessary. We can then write tools at the next level of abstraction, operating on the schema as data. Again, as required, we will iterate our designs and implementations.

The technology stack is still under discussion, though our overall design will allow us to change out the various pieces independently. There is only loose coupling between the backend, the API and the graphical user interface. The incremental approach to development reduces the costs of early changes to the toolsets. For the front-end user interface, we plan a web based single page application (SPA) built with a modern javascript framework. The current candidates are vue.js[10], AngularJS[11] and React[12]. For the API we are looking at either REST[13] or GraphQL[14] written in either javascript or python. For the backend we will choose an overall approach from graph databases - possibly neo4j[15] or orientdb[16], other nosql databases, and traditional sql databases.

5. Conclusion

Across all research domains the problem of locating and understanding recorded results exists. Our core idea is that the relationships between data can be represented as graphs – that is collections of nodes and edges. Nodes can have properties that are URIs for externally stored data. This allows the system to represent relationships between heterogeneous objects. The system will be based on data driven schema, providing flexibility and extensibility. This also allows it to apply to a wide range of research domains. We have designed a set of tools to address this and are starting on its implementation.

6. Acknowledgements

This work is funded by the National Science Foundation under DIBBS award no. 1640829.

[1] Jeffery Y. Beyon. 2000. *LabVIEW Programming, Data Acquisition and Analysis* (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

[2] Fredian, T.W., Stillerman, J.A.. "MDS/MIT high-speed data-acquisition and analysis software system." Review of Scientific Instruments 57.8 (1986): 1907-1909.

[3] Stillerman, J.A., T.W. Fredian, K. A. Klare. and G. Manduchi. "MDSplus data acquisition system." Review of Scientific Instruments 68, no. 1 (1997): 939-942

[4] Kepler https://kepler-project.org/, retrieved 6/8/2017.

[5] Greenwald, M., Fredian, T., Schissel, D., Stillerman, J., A metadata catalog for organization and systemization of fusion simulation data, Fusion Engineering and Design, Volume 87, Issue 12, 2012, Pages 2205-2208, ISSN 0920-3796, https://doi.org/10.1016/j.fusengdes.2012.02.128.

[6] URI https://www.w3.org/TR/uri-clarification/ retrieved 1/7/2018

[7] HDF5 https://support.hdfgroup.org/HDF5/ retrieved 6/8/2017.

[8] Greenwald, et.al., 20 years of research on the Alcator C-Mod tokamak, Physics of Plasmas 21, 110501 (2014).

[9] Fredian, T.W., Stillerman, J.A., Web based electronic logbook and experiment run database viewer for Alcator C-Mod, Fusion Engineering and Design, Volume 81, Issues 15–17, July 2006.

[10] vue.js https://vuejs.org/ retrieved 6/8/2017.

[11] AngularJS https://angularjs.org/ retrieved 6/8/2017.

[12] React https://facebook.github.io/react/ retrieved 6/8/2017.
[13] REST https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm retrieved 6/8/2017.
[14] GraphQL http://graphql.org/ retrieved 6/8/2017.
[15] neo4j https://neo4j.com/ retrieved 6/8/2017.
[16] OrientDB http://orientdb.com/ retrieved 6/8/2017.