Fusion Engineering and Design 112 (2016) 1007–1013

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

The MPO system for automatic workflow documentation

G. Abla[a], E.N. Coviello[a], S.M. Flanagan[a], M. Greenwald[b], X. Lee[a], A. Romosan[c], D.P. Schissel[a,*], A. Shoshani[c], J. Stillerman[b], J. Wright[b], K.J. Wu[c]

[a] General Atomics, P.O. Box 85608, San Diego, CA 92186-5608, USA
[b] Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[c] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

HIGHLIGHTS

- Data model, infrastructure, and tools for data tracking, cataloging, and integration.
- Automatically document workflow and data provenance in the widest sense.
- Fusion Science as test bed but the system's framework and data model is quite general.

ARTICLE INFO

Article history:
Received 11 July 2015
Received in revised form 17 March 2016
Accepted 13 April 2016
Available online 18 April 2016

Keywords:
Workflow
Provenance
Metadata
Ontology
DIII-D
EFIT

ABSTRACT

Data from large-scale experiments and extreme-scale computing is expensive to produce and may be used for critical applications. However, it is not the mere existence of data that is important, but our ability to make use of it. Experience has shown that when metadata is better organized and more complete, the underlying data becomes more useful. Traditionally, capturing the steps of scientific workflows and metadata was the role of the lab notebook, but the digital era has resulted instead in the fragmentation of data, processing, and annotation. This paper presents the Metadata, Provenance, and Ontology (MPO) System, the software that can automate the documentation of scientific workflows and associated information. Based on recorded metadata, it provides explicit information about the relationships among the elements of workflows in notebook form augmented with directed acyclic graphs. A set of web-based graphical navigation tools and Application Programming Interface (API) have been created for searching and browsing, as well as programmatically accessing the workflows and data. We describe the MPO concepts and its software architecture. We also report the current status of the software as well as the initial deployment experience.

1. Introduction

Both scientific experiments and computations produce data. This data is a critical asset for science activities and the base for reproducing scientific analysis results. Most times, data is more useable and meaningful with context and metadata. Therefore, it is important to keep track of metadata about the experimental and computational processes.

However, creating and managing metadata is not a simple task. Throughout history, scientists generated handwritten logbooks to keep track of data, hypotheses, experiments, setups and computational processes, as well as reflections about the process and data. In the last several decades, as personal computers and handheld electronic devices became generally available, the handwritten logbooks were replaced with electronic logbooks. The electronic logbooks brought many conveniences by supporting multi-media, hypertext, and fast searching capabilities. However, content creation and metadata capturing remained as a manual activity, same as with the classic logbooks. The increased complexity of scientific workflows led to the fragmentation of data, processing, and annotation.

As the pace of scientific research accelerated, keeping up with documenting the process and data became even more challenging and time consuming. The increased capability and precision of scientific instruments, the rise of exascale computing, and the arrival of the "big data" era further complicated the bookkeeping of research activities. Thus, capturing metadata, creating logs,

* Corresponding author.
E-mail address: schissel@fusion.gat.com (D.P. Schissel).

http://dx.doi.org/10.1016/j.fusengdes.2016.04.023
0920-3796/

and organizing content as well as associating the data with proper context increasingly became a daunting task for scientists.

The MPO project tries to ease the documentation process of scientific research activity. By modeling data tracking, cataloging, and integration across a broad range of scientific domains, it aims for developing tools to automate documentation of scientific workflows and associated information. The MPO software consists of a data model and API that can capture information from the creation and recording/importing of physical data, through various levels of analysis, data preparation, supercomputer code execution, storage, post processing, data exporting, and publication. Data and all processes that create or modify that data are represented mathematically as a directed acyclic graph, providing explicit information about the relationships between workflow elements. It provides tools for metadata capturing, provenance creation, and context information collection.

The motivation of the project and design principles of the prototype system have been described earlier [1–3]. This paper discusses the most recent results of the MPO project's research, including the current status of the MPO software system and deployment experiences.

2. The MPO concepts

The MPO system aims to document scientific research activities by tracking experimental and computational workflows. Its goal is to ease the creation of contextual metadata and provenance information, and help scientists quickly find answers for the questions "who, what, when, how, and why" for each data element related to their research.

In order to create a model for capturing and organizing metadata in the widest sense, and supporting systematic analysis, the MPO project defined several entity concepts. The following are the entities defined by MPO data model:

Data Object—A piece of information that is related to a scientific activity or research. It can be a large dataset, one single value, or a research paper. A Data Object can be either produced or consumed by a computer code. The Data Object can be stored as a file or put into a scientific data store (e.g., MDSplus) for later retrieval via a client-server API. MPO keeps pointers—in the form of a Uniform Resource Identifier (URI)—that uniquely identifies the data and its access method. It keeps track of Data Objects in two ways. First is the general information about the Data Objects themselves, such as URI, creation date, and username. Second is the specific use of the Data Object in a workflow. This is done to allow the system to keep track of Data Objects and their uses in a more organized manner.

Activity—Anything that creates, moves, or transmutes data from one form to another. An activity often both consumes and produces data. It can have multiple input Data Objects, and can produce Data Objects as output. As a special case, activities can connect to each other in order to permit data in memory to be passed from activity to activity. Examples of Activities include data importing, pre-processing, input preparation, executing codes, data storage, post-processing, and data exporting.

Connection—The causal link between multiple Activities and/or Data Objects.

Workflow—A series of connected Data Objects and Activities, which can be organized as a Directed Acyclic Graph (DAG) that consists of one-way flows with no loops allowed. A Workflow shows the individual steps in the processing chain and the parent-child relationship among its elements (Data Objects and Activities), which can be followed in either direction. Data Objects, Activities, and Connections are building blocks of a Workflow.

Collection—A group of related entities. It can contain any number of Data Objects and Workflows. A Collection also can include other Collections. Each element in a Collection can be shared among multiple Collections. Relevant use cases are: (1) A series of simulation runs in a parameter scan; (2) Multiple computational data analysis workflows and associated data used in a published paper.

Metadata—A text-based, arbitrary name-value pair that is associated with a Data Object, Activity, Workflow, or Collection. Using Metadata is a flexible way to keep track of values without having to update the MPO system to explicitly handle that type of information. Examples of Metadata include last-updated dates for Data Objects and informative notes for an algorithm in an Activity.

Comment—Text (including hypertext) information that is associated to a Data Object, Activity, Workflow, or Collection. A Comment can also be associated with another Comment, which means that Comments can be added recursively. Comments are unstructured and may come with a few fixed attributes, such as user ID and timestamp of creation. Comments are either automatically generated by programs or manually entered by scientists, and are essential in capturing the user generated metadata of all basic MPO entities.

Fig. 1 shows an example of a workflow for documenting the data preparation steps to execute the GYRO code, a large scale gyrokinetic or neoclassical simulation of plasmas [4].

In addition to the above basic entities, MPO also defines two other concepts needed for documenting workflows: Provenance and Ontology.

Provenance is the lineage of Data Objects. It traces the path of a Data Object, starting from creation, through every transformation until the end. Every time a workflow is executed, an instance of that execution is generated. That instance represents the provenance, which includes the Data Objects and parameters used as input for each step, the Data Objects and parameters generated as output, and the sequential relationships between the steps. Extensive information, such as where a piece of data came from, how it was created, why and by which Activity, are captured as the contents of Provenance.

The Ontology is a structure that captures the common terms used to describe object properties in a specific domain. This is also referred to as controlled-vocabulary or classification structures. It is common to represent such structures as tree structures, where the leaves of the trees are referred to as "narrower terms", and the higher level elements as "broader terms". "Broader terms" are used for general categories, and narrower terms used for the specific branches of a general category. Classification terms can be used for any purpose, depending on the domain, and need to be developed in cooperation with application scientists. In this project, Ontology is a description of main MPO entities (Data Object, Activity, Workflow, Comment, and Collection) via their hierarchical relationships. Each element of the Ontology category (also referred to as facets) can form a hierarchy of broader terms that spawn narrower terms. The main purpose for Ontologies is to allow for more accurate searching of the all the basic MPO entities and Provenance information.

The MPO entities described above are the basis for automatically generating visual representations and describe relations among multiple Data Objects, Activities, and Workflows. They are also essential in creating software framework for documenting Provenance and Ontology.

3. The MPO software system

3.1. The MPO architecture

The architecture of the MPO system has been driven by the need of supporting both experimental and computational research activities. Such research activities involve processing of raw data, with small or large computational codes often providing inputs to

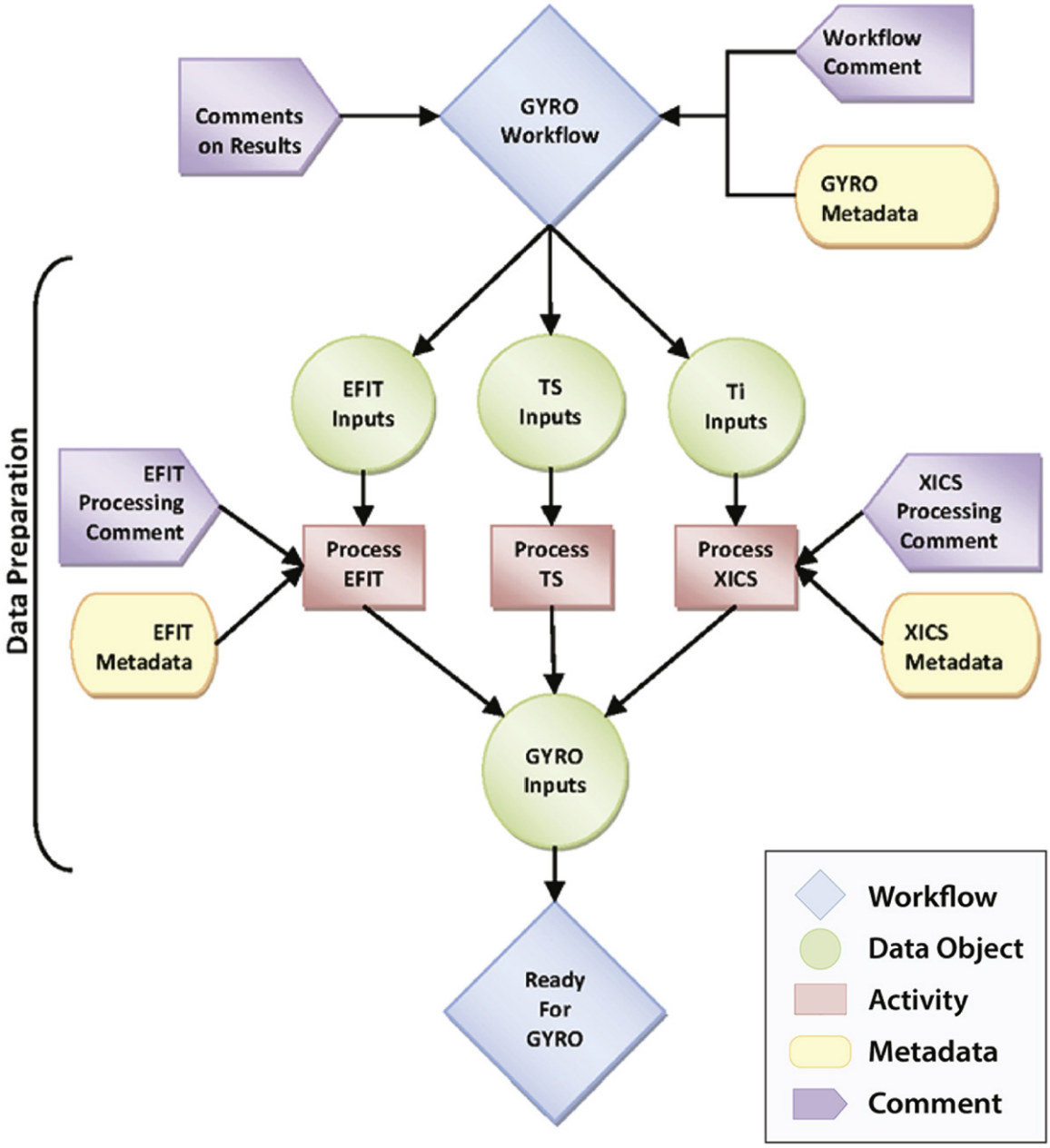

Fig. 1. Example data preparation workflow to execute the GYRO code.

larger simulations, whose output requires processing as well. The computational codes were often developed with a variety of languages (FORTRAN, IDL, C/C++, Python, shell scripts, Matlab), and they may require a variety of computational environments (e.g., operating systems, interconnects, software libraries). The computer hardware can be a laptop, desktop, computer cluster, and even supercomputers. The input and output data and storage formats can be different too, including, but not limited to, MDSplus, HDF5, NetCDF, JSON, and CSV. Some workflows may be repeated by multiple scientists, and the same Data Object may be used in multiple similar or different workflows. Some workflows represent single activities, and others can be a part of larger simulation efforts, such as parameter scanning carried out by many workflows.

To address the above reality, the MPO system was developed using multi-tier web services. It uses a RESTful API, which can be easily utilized in a variety of programming languages for instrumenting MPO client calls. The current architecture of the MPO system and its main components are shown in Fig. 2.

The building blocks of the MPO system are: (1) Database; (2) API Server and Event Server; (3) Interactive UI server; (4) Clients. The heart of the system consists of two main servers: the API Server and the interactive UI Server. Only the API server directly communicates with the Database. There are two types of clients: Native Application Client and Browser-based web client. While "Native Application" clients directly communicate with the API server, the web browser connects to the UI server.

The first production version of the MPO system architecture is slightly different than the prototype version reported earlier [2]. In the prototype version, both the API service and the interactive UI service were provided by a single web server and they used the same network port. The production version maintains those two services on separate web servers, and their responsibilities are clearly separated. The idea behind this separation is to keep the

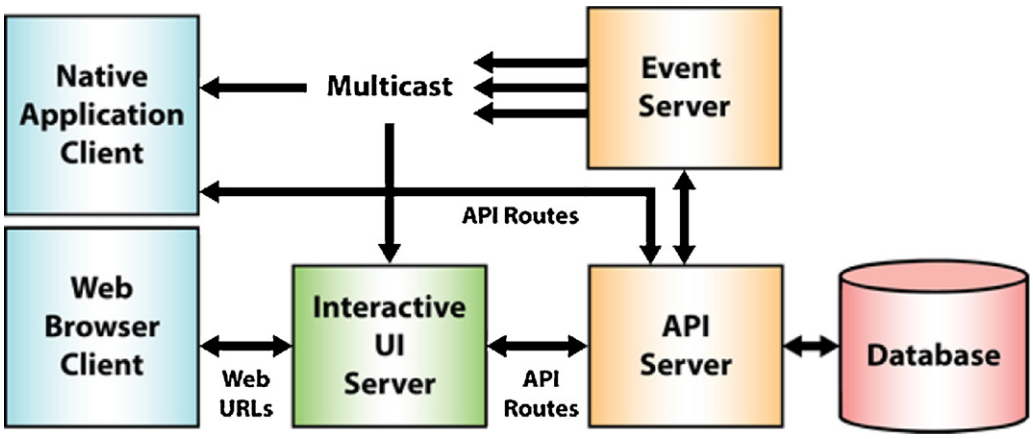

Fig. 2. The main components of MPO system.

API server independent so that other (desktop, mobile, web) client applications can be developed independently from the interactive UI server.

3.2. Database

The MPO database is responsible for storing all the MPO information. Its schema is based on these entities discussed in Section 2, and the MPO entities (Data Objects, Activities, Workflows, Collections, Metadata, Comments, and Ontology) are represented with database tables. There are also several additional tables such as the users table, which stores user information. Another additional Table is the connectivity Table for capturing relations between basic MPO entities in a DAG structure. The data stored in the connectivity Table is not intended for direct access; instead, it is used along with other entities, such as Workflows, Data Objects, and activities.

Currently, PostgreSQL, an open source relational database system, is chosen for the database server. However, the MPO database schema design is general enough that the database server can be easily replaced with any other relational database software. The generality of the schema also helps the separation of the database implementation from representing the concepts in the API as well as user interfaces. The team is working on furthering this separation by implementing Object Relational Mapping (ORM) with the SQLAlchemy toolkit [5].

3.3. The API server and event server

The MPO API server exposes its services via RESTful API and currently it only supports GET and POST methods. The basic entities in the MPO data model are represented with corresponding RESTful resources. All routes are accessed in the form of HTTP requests such as http://<server>/<type>/<version>/<route>/UID. <server> refers to the hostname and the port number of the API server. The API server uses <type> to refer the requests to the appropriate MPO environment such as production or testing. <version> refers to the version of the MPO API that is being used. This allows for multiple versions of the MPO API which is crucial due to the evolving nature of the software. UID is a universal unique identifier (UUID v4) created during a POST method and may be used with a GET method to refer to a specific entity. POST responses are in JSON format and includes specific information such as the UID of the new object, the URI of the object, the user who created the object and the time recorded when the entry for the object was made. Queries using GET may include key-value arguments to identify or filter metadata fields. <route> is the corresponding resource. All routes have a sub-route/<route>/<UID>, which provides information on a specific entity identified by the UID.

The following is the list of main routes along with the role of GET and POST methods./workflow—A POST method creates a new workflow and a GET method returns a list of available workflows./workflow route also has sub-routes for accessing a human readable alias for the workflow (instead of 32 hexadecimal digits in a UID) or a graph data structure representing the entire workflow. The GET method for/workflow/UID/dataobject route returns the dataobjects relevant to the workflow specified by the UID.

/dataobject—A POST method registers a new dataobject specified by a name, description and UID of the workflow in the message body. A GET method returns a list of available data objects. A GET method for/dataobject? instance returns all the instances of the specified dataobject. In this case, instance means a reference to a use of a dataobject in a workflow.

/activity—A POST method adds an activity to a workflow. Existing data objects may be specified as input or output parameters. GET method returns a list of activities.

/collection—A POST method creates a new collection. A GET method returns a list of collections.

/collection/<UID>/element—A POST method adds an element (entity) to the collection. A GET method returns a list of elements in the collection specified by the UID.

/comment—A POST method attaches a text comment to a workflow, dataobject, activity, collection, or another comment.

In addition to the core routes listed above, several supplementary routes are also provided. One such route is/workflow/UID/graph/. This route provides server side rendering of visualization for workflow graphs in SVG or PNG format. Other additional routes mainly provide convenience for reducing the number of server connections thus facilitating faster data retrieval.

The API server has been constructed using Flask, which is a lightweight micro web application framework written in Python. The simplicity and extendibility of this framework are the main reasons for choosing Flask for the MPO development.

The MPO event server is an additional service that runs side-by-side with the API server. It is implemented by utilizing the MDSplus system's event features and provides asynchronous events for real-time updates to clients. "Data updated" events are generated when new data is added to a certain workflow so that clients can asynchronously call the API server to retrieve the latest information. For example, web clients use this event server for triggering Asynchronous Javascript and XML (AJAX) calls and repainting the webpage to display the ongoing progression of the workflow in near-real-time.

3.4. The interactive UI server

The MPO interactive UI server provides visualization and interactive browsing of the MPO data via the web browser interface. There are four main sections available: Workflow, Data Object, Collection and Search.

In the Workflow page, each workflow is displayed with its Workflow ID, Description, and Creation Time. Additionally, corresponding quality ratings and number of comments for each workflow are also displayed. Users can choose to view or enter comments by clicking a comment link associated with a workflow. This page also provides an ontology-based filtering of the workflow list.

Clicking any of the workflows in the Workflow page opens the related Workflow details page. This page provides a rich interactive environment for viewing and exploring all the metadata about the specific workflow, thus creating a dynamic "notebook" interface. It utilizes an interactive and real-time graphical interface to present the data associated with a specific workflow. The interface is comprised of an interactive workflow diagram, an expandable list of nodes which include user comment log, list of metatdata and other linked workflows. The diagram offers pan and zoom capabilities, and also displays metadata associated with the node when selected. The workflow node listing provides users with a chronological list of workflow nodes. When a node is clicked, it expands and reveals all associated metadata. Users also have the ability to view and add node-specific comments on this workflow page. Using an event system, these respective workflow elements are updated in real-time as new data is added. For example, when a new workflow node is added, the node will appear in the workflow diagram and is also appended to the node list with its metadata. In the DAG visualization, if a workflow element is shared with other workflows, it is automatically detected and highlighted in blue. Fig. 3 shows a workflow list page partially overlapped with workflow details page of EFIT workflow.

The Data Object page and Collections page provide similar lists of corresponding entities. Clicking any item in the list opens the details page for the specific Data Object or Collection.

The Search page provides a single search box to search all recorded MPO entities by a keyword.

The MPO Interactive UI server utilizes the MVC design pattern. It does not maintain a local database; but its Model relies on the API server responses as its data source. Its View is the interactive web page; and its Controller is responsible for supporting user interactions.

Similar to the API server, the interactive UI Server development also relies on the Flask framework. Workflow visualization, and interactive browsing as well as searching capabilities were developed via open source web technologies, such as HTML, Graphviz, Ajax, JavaScript and Twitter Bootstrap.

3.5. MPO clients

The MPO API server provides necessary routes for implementing client commands and the RESTful API facilitates integrating them into a variety of applications. There are RESTful-supported tools and libraries in almost all the major programming languages. For example, the interactive UI server is actually a Python client of the API Server, and it is implemented using RESTful libraries in Python.

To further simplify the process and hide the unnecessary and repetitive details from the end-users, the MPO team developed mpo_arg.py, a command line interface package using the Python language. The package supports the following methods:

init/stop: begins/finishes a new workflow,

add: creates a new dataobject (optionally adds a reference),

step: adds an activity to a workflow,

comment: attaches a comment on an item in a workflow,

collection: creates a new collection or add an element to an existing collection,

meta: adds a key/value metadata to an item,

auth: authenticates using OpenSSL certificates,

archive: storing data in persistent store and/or creating dataobjects from persistent store,

These commands can be invoked in Python codes or directly from the command line. The mpo_arg.py package and its commands have been integrated with multiple computational workflows in instrumenting them as an MPO client. Those workflows have been developed with IDL, Python, Matlab or Linux shell scripts.

4. Deployment experiences

4.1. Workflow instrumentation

After testing the early prototype system with several workflows, many improvements have progressively been made to the MPO system and its first production version has been released.

The production version of MPO has been integrated with multiple new workflows.

One workflow is the automated calculation of the plasma shape (EFIT) during between-pulse computations of the DIII-D National Fusion Facility that was instrumented with the MPO API via the MDSplus data acquisition and data management system [6]. The MPO client commands have been adapted into workflow scripts and automatic metadata generation has been instrumented during the experiments. The generated metadata and DAG visualization along with user entered comments and metadata provide a rich document view. The connected workflow and shared data object representations provide improved context for relations among multiple experiments.

The other workflow is from the Simulation of RF Wave Interactions with Magnetohydrodynamics (SWIM) proto-FSP project that couples advanced fusion simulation codes into a computational framework that operates on computational clusters worldwide including supercomputers at ORNL and NERSC [7]. The SWIM system has been in use for a number of years to do production scientific data analysis and instrumentation via the MPO API was straightforward. After each simulation is completed, MPO clients methods are triggered automatically. The metadata information has been created by filtering status data stored in the SWIM portal service. Recently, SWIM researchers executed large simulations involving physics parameter sweeping. Each parameter sweep can include several hundred SWIM workflows, which is a good candidate for creating MPO collections. The automatic generation of SWIM collections is currently being tested.

MPO Instrumentation has begun with the Advanced Tokamak Modeling (AToM) project. The goal of the AToM project is to enhance and extend the predictive modeling capabilities that currently exist within the US magnetic fusion program [8]. OMFIT is the main workflow engine in the AToM project, and has significant worldwide user base in the tokamak modeling community. The MPO Python client package was used to instrument the workflow engine in OMFIT to create a hierarchical representation of scripts that has been executed. The kineticEFIT regression test cases executed through OMFIT collected MPO metadata along with graphical representation of the workflow. The initial results were well received by the OMFIT developers.

The success of adding these very different workflows validates MPO's approach of a simple API that can be used to instrument basically any existing workflow, a general data store and a Web-UI that can be used to display and navigate those workflows.

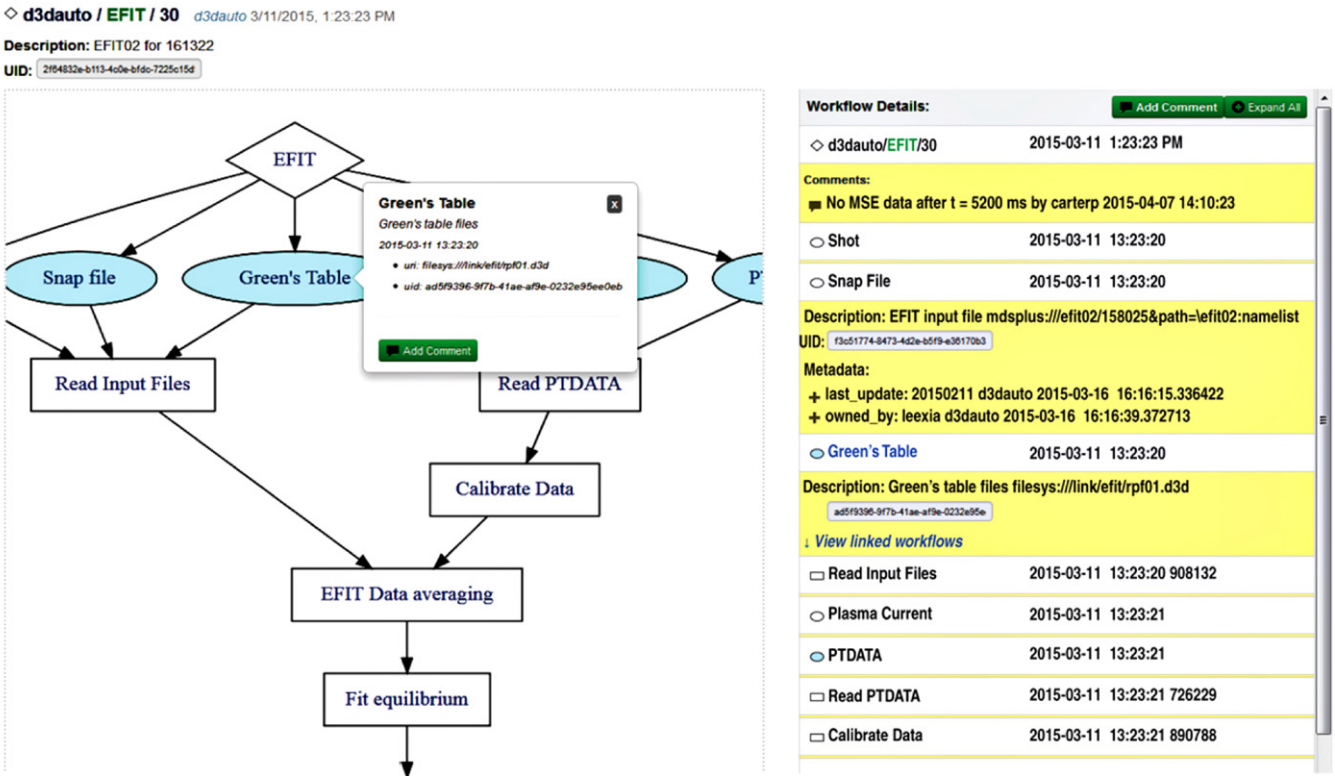

Fig. 3. EFIT workflow displayed on an MPO workflow details page with an interactive directed acyclic graph and listing of nodes.

4.2. Learned lessons

The above integration work also taught some lessons and brought some issues to the authors' attention. For example, when the MPO system collected over 50,000 data sets, it was realized that the response of the system slowed down. Therefore, additional efforts to improve the performance by implementing asynchronous client-server connections were made. As the result, the relations between the number of recorded workflows and performance has been minimized. During the instrumentation of SWIM, it was realized that SWIM monitoring component provides very detailed information and the full representation of every step with DAG representation can be overwhelming and unnecessary. Therefore, the instrumentation has been changed and only the "significant activities," as defined by the needs of the plasma science, have been documented. The "significant activities" are initially hand-picked, but an automatic detection of "significant activities" can be useful in the future. Another issue was realized with instrumentation of OMFIT workflows. Some OMFIT workflows can have many iterative loops and reruns of the same scripts, sometimes with parameter changes. The MPO and AToM teams working closely to represent this type of use cases.

5. Conclusions and future work

The MPO system automates documentation of scientific workflows and associated information and does so independently on the workflow orchestration. It also provides capabilities for organizing and analyzing documented metadata as well as presenting it with interactive visual representations.

In the near term, plans are to reach out to new users. The goal is to help them to instrument their workflows with MPO and see its value for enhancing their scientific work. Specifically, non-fusion scientists who have non-fusion workflows are being reached out to. One such project is the Calibrated and Systematic Characterization, Attribution and Detection of Extremes (CASCADE) project, which aims to advance scientific capabilities in studying climate extremes and in conducting extreme climate analysis [9]. Through collaboration, the CASCADE workflows will be instrumented with MPO. The experiences and feedback will help identify areas for improvements to the MPO system that serve the broader scientific community.

Early experiences demonstrate that the MPO is not only useful for individual researchers to document their own computations, but also for an environment for collaboration among researchers. Collaborating scientists can easily see what data and computational codes are being shared between them, or what kind of workflows others are running to solve a similar problem. Therefore, providing tools and features for promoting collaborations in the immediate future can be useful. Some examples are "subscribe" feature for getting notification on somebody else's workflow, and "like" feature for highlighting some high-value Data Object or Workflows.

Another potential area for enhancement is providing data exchange capabilities with relevant provenance tools and workflow engines. The World Wide Web Consortium (W3C) published a set of provenance standards and recommendations named PROV [10]. Multiple supporting applications exist and some of them are complimentary to MPO capabilities. Providing an export mechanism so that data collected by the MPO System can be used by PROV standard supported software is planned.

Acknowledgement

This material is based upon work supported by the US Department of Energy, Office of Advanced Scientific Computing Research and the Office of Fusion Energy Sciences under Award Numbers DE-SC0008697, DEAC02-05CH11231, and DE-SC0008736.

References

[1] M. Greenwald, et al., A metadata catalogue for organization and systemization of fusion simulation data, Fusion Eng. Des. 87 (2012) 2205.
[2] D.P. Schissel, et al., Automated metadata, provenance cataloging, navigable interfaces: ensuring the usefulness of extreme scale data, Fusion Eng. Des. 89 (2014) 745.
[3] J.C. Wright, et al., The MPO API: a tool for recording scientific workflows, Fusion Eng. Des. 89 (2014) 754.
[4] GYRO, https://fusion.gat.com/theory/Gyrooverview.
[5] SQLAlchemy, http://www.sqlalchemy.org/.
[6] D.P. Schissel, et al., Enhanced computational infrastructure for data analysis at the DIII–D national fusion facility", in: Twenty-Sixth European Physical Society Conference on Controlled Fusion and Plasma Physics, June 14–18 in Maastricht, The Netherlands, 1999.
[7] Don Batchelor, et al., Advances in simulation of wave interactions with extended MHD phenomena, J. Phys.: Conf. Ser. 180 (2009) 012054.
[8] O. Meneghini, et al., Integrated modeling of tokamak experiments with OMFIT, Plasma Fusion Res. 8 (2013) 2403009.
[9] The CASCADE Project, http://climatemodeling.science.energy.gov/sites/default/files/CASCADE_Fact_Sheet.pdf.
[10] W3C PROV-Overview, http://www.w3. org/TR/prov-overview/.