Dashboard V2 Specification For Comments

Ian Taylor
University of Notre Dame, Center for Research Computing & Cardiff University, UK

This report provides a short overview of the existing NDS dashboard and how we plan to evolve this design over the next several months. The current NDS dashboard interfaces with the Docker containers and data management systems that form the basis of the NDS Labs *Epiphyte* infrastructure. The resulting system has been shown at the NDS meetings in Washington DC and Austin, and was demonstrated at Supercomputing last year at the NCSA booth in two sessions. It is also checked into the NDS open source repository here:

https://bitbucket.org/nds-org/nds-dashboard

And a video demonstration outlining some of the features can be seen here:

http://ndspilot.com/nds/ndspilot1080p.mp4

With a live website being hosted here

http://ndspilot.com/ (user mturk, pass mturk for the demo account)

The existing design of the dashboard comprises three main themes:

- **Research Methods:** To provide an intuitive Web-based interface to expose fully *interactive research containers* to support the lifecycle of scholarly communication. Research containers enable executable and repeatable research by supporting methods, source code, and data within dynamically created Docker containers.
- **Research Data:** To provide a fully interactive means for a researcher to share their data via NDS Labs and to support the lifecycle of research progression by connecting the desktop to the NDS Labs data storage mechanisms.
- **Standardized Web APIs and Interfaces:** The Web dashboard backend has been modeled using the OAI-ORE Web aggregation standard to allow interoperability with other repositories and to provide a semantically annotated programmatical means of interacting with the NDS Labs content and services through a Web-based REST API.

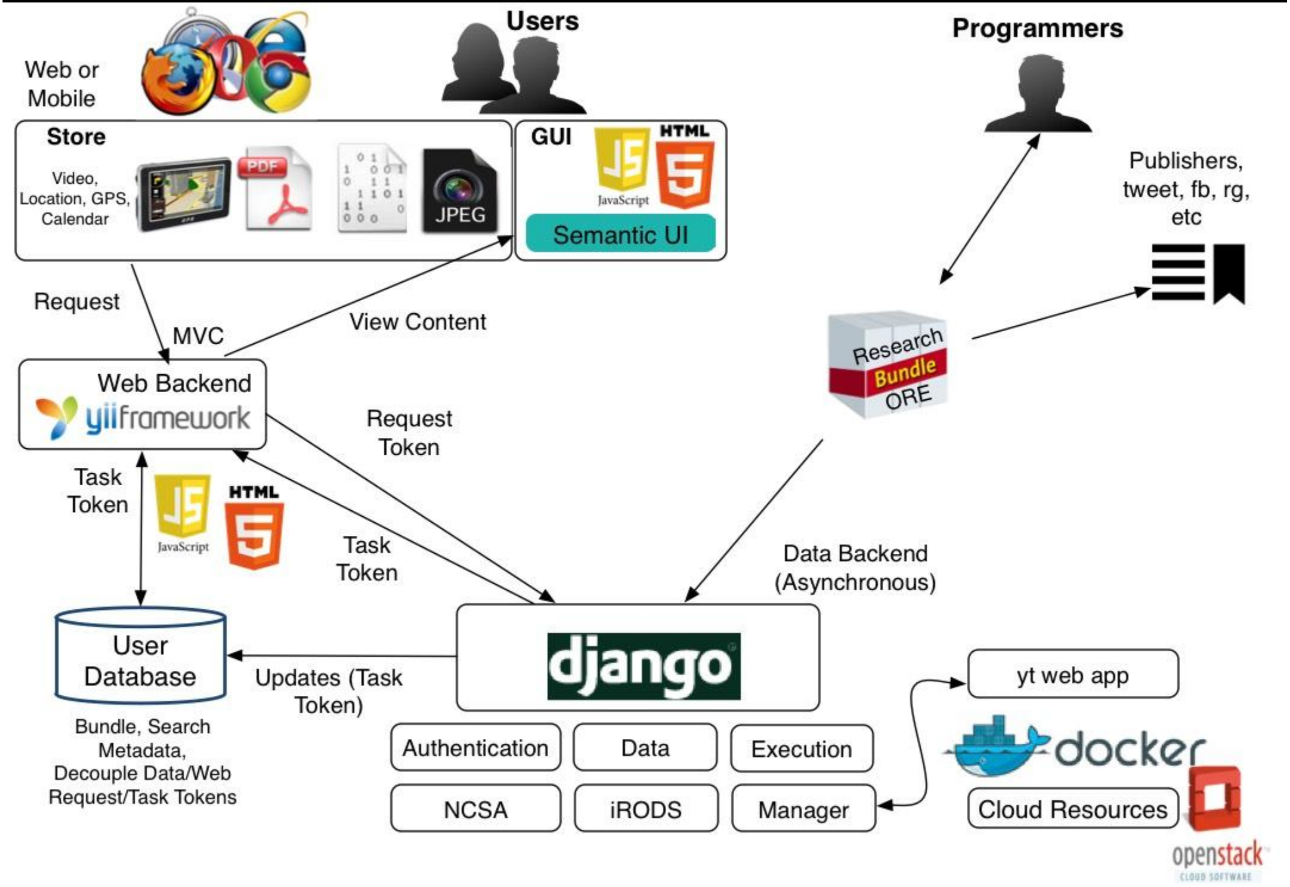


Figure 1: Current Dashboard Architecture and implementation

The existing implementation, shown in Figure 1, has several limitations:

1. We use the Yii and PHP templating model for generating the content for the views. The HTML for the views is injected with data at the backend and passed as generated HTML to the front end for displaying. This model has several disadvantages however, because HTML content is being mixed with the data and generated from the backend. This means that we have tie in to Yii for generating the views and it also means that there is PHP code that parses the models (database content) on the backend to generate HTML content. This is inefficient, it tightly couples the data and the view generation, it makes it difficult to create smaller-based services (i.e. microservices) and makes things hard to debug i.e. a Javascript files + a CSS file + HTML are being pulled into the view to generate the output. Also, since this approach tightly couples the front with backend code to access data, it makes it a lot harder to generate different versions of the GUI i.e. to create a customized mobile interface or different versions of the dashboard.
2. Although the code for the various supported services are created as different MVC pairs, the code is in one codebase and has various dependencies between one piece and the other e.g. the data toolkit has a dependency on the container execution, etc. A better model is to move away from this and try and building completely independent modules that can interact through defined REST interfaces. This makes the resulting code far more easy to use in different contexts.
3. The use of ORE is not recursive so currently an article is only allowed to contain one-level of data items. Data items cannot contain data items so versioning or provenance is difficult to integrate.

Future Direction

The new dashboard will be a complete rewrite and is outlined in Figure 2 below.

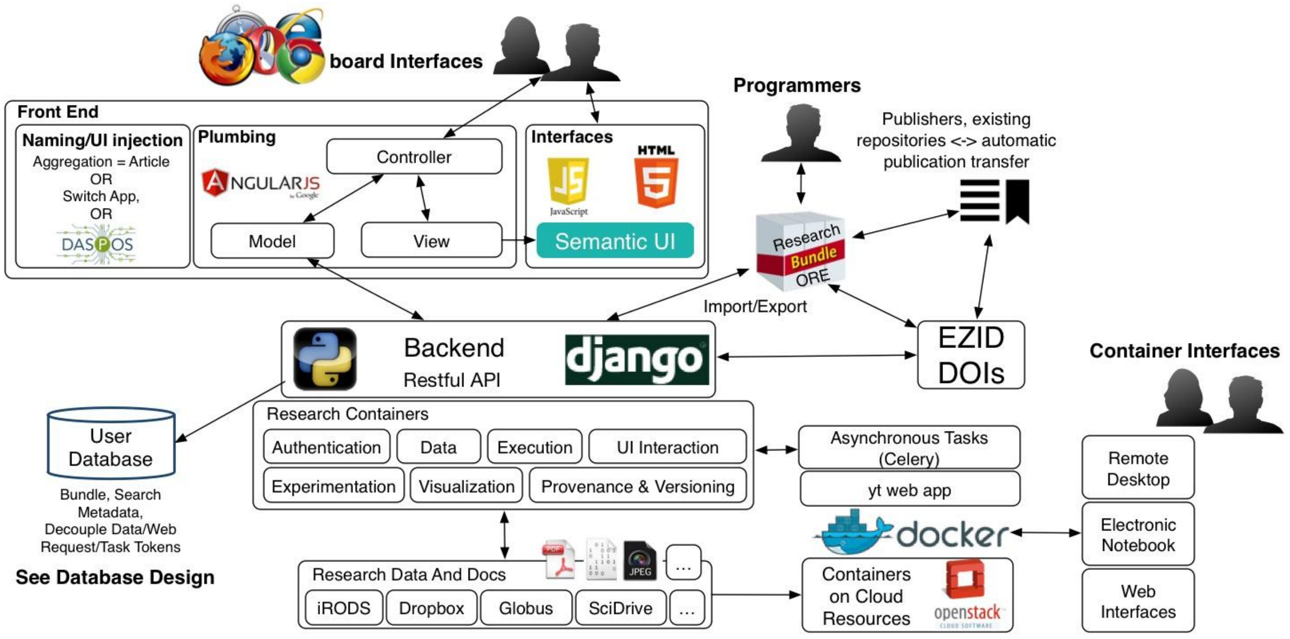


Figure 2: The New Dashboard Architecture

The new design specifies a generic set of components that can be used in multiple contexts. It will serve as a dashboard to the NDS but also a dashboard for other projects that require access to data and execution of code on containers. Therefore, it should be customizable and be capable of representing any experiment, along with the data, configuration information and an execution environment for running an experiment's code and data on a container. It has several extensions that collectively achieve this goal:

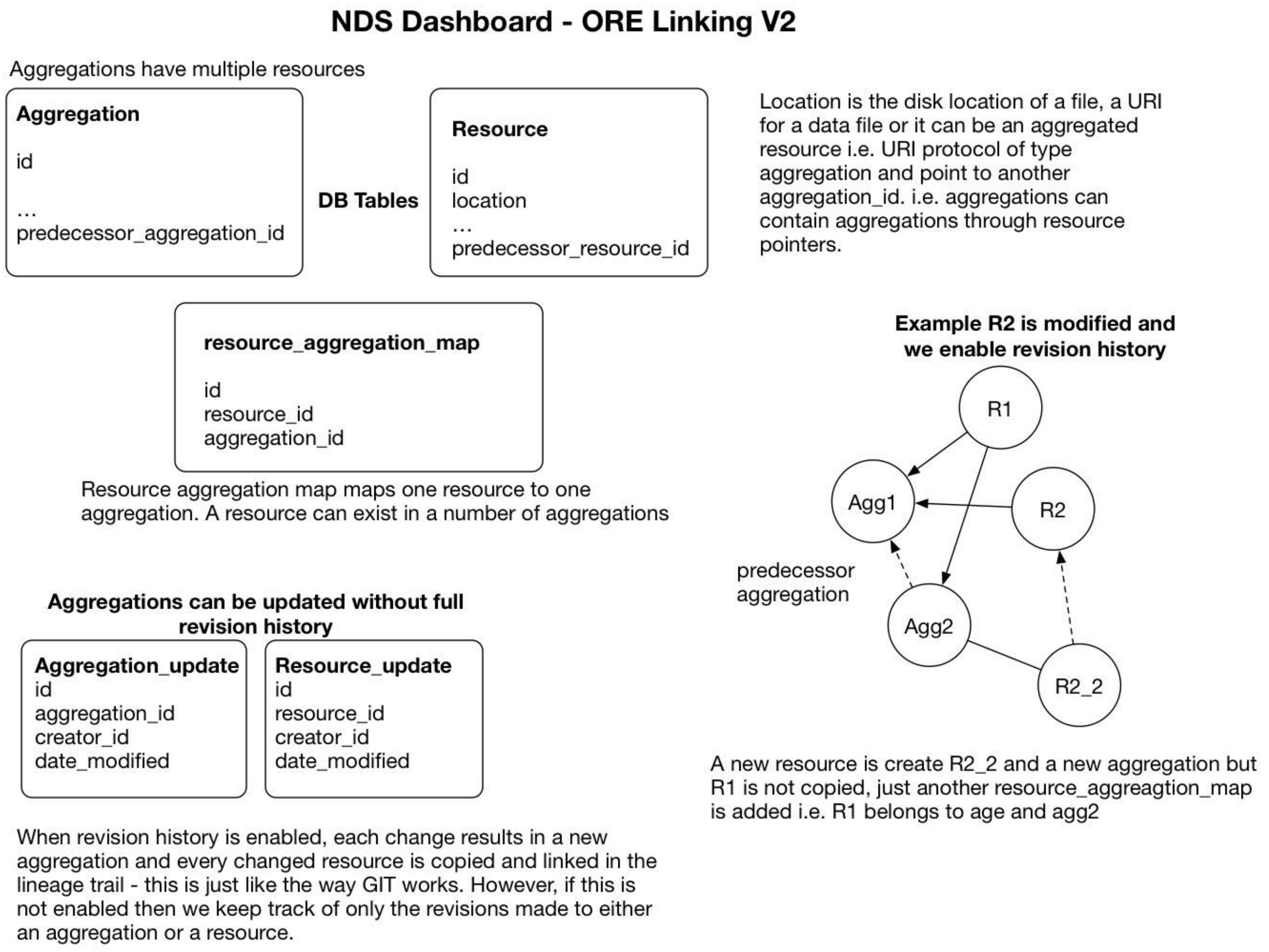
1. Standardized Web Aggregation Formats, Backend Representation and Web Interfaces

The dashboard's foundation is built on the Object Reuse and Exchange (ORE) standard, developed by the Open Archives Initiative (OAI). This allows a research study to be exposed using a single aggregated Web resource that can encapsulate all of the artifacts surrounding a scientific research piece. For example, it can be used to package the research paper, research techniques, tools, documentation and methods and data, using a standardized mechanism. This approach has three important features:

1. It provides a standardized means of being able to export and import the data to and from external repositories.
2. It provides a semantic resource map (RDF) which describes the relationships between the entities in the in the aggregation.
3. It facilitates the creation of Web APIs to interface with research publications, methods and data using a programmable interface i.e. it makes research publications machine processable.

Currently, we have implemented a Python-based REST API for extracting a research article (plus data and methods) as an ORE resource map. This has been integrated into the Dashboard so that researchers can export the ORE map as an RDF file.

The new design for the database is shown below in Figure 3. In the current version of the dashboard, an aggregation is called an article and an article can have multiple resources to represent the data and the scripts needed to repeat the research. In the new version these relationships will be more decoupled so that not only can an aggregation contain multiple resources, one resource can exist in multiple aggregations. We also extend the notion of a resource by using a allowing it to point to URI for other aggregations (i.e. the aggregated resource type), so that aggregations can contain other aggregations through resource data pointers.


NDS Dashboard - ORE Linking V2
Aggregations have multiple resources
Aggregation
id
…
predecessor_aggregation_id
DB Tables
Resource
id
location
…
predecessor_resource_id
Location is the disk location of a file, a URI for a data file or it can be an aggregated resource i.e. URI protocol of type aggregation and point to another aggregation_id. i.e. aggregations can contain aggregations through resource pointers.
resource_aggregation_map
id
resource_id
aggregation_id
Resource aggregation map maps one resource to one aggregation. A resource can exist in a number of aggregations
Example R2 is modified and we enable revision history
R1
Agg1
R2
predecessor aggregation
Agg2
R2_2
A new resource is create R2_2 and a new aggregation but R1 is not copied, just another resource_aggreagtion_map is added i.e. R1 belongs to age and agg2
Aggregations can be updated without full revision history
Aggregation_update
id
aggregation_id
creator_id
date_modified
Resource_update
id
resource_id
creator_id
date_modified
When revision history is enabled, each change results in a new aggregation and every changed resource is copied and linked in the lineage trail - this is just like the way GIT works. However, if this is not enabled then we keep track of only the revisions made to either an aggregation or a resource.
Figure 3: Generic ORE-based Database Design

The combination of these new features is quite powerful because it allows aggregations to contain aggregations and resources to be linked to multiple aggregations. This allows for features such as version control and branching through the use of a new aggregation types when a user makes a change. For example, consider the case where we make a deep copy (e.g. a revision history mode could be enabled) or a shallow copy. When revision history is enabled, each change could result in a new aggregation and every changed resource can be copied and linked in the lineage trail - this is similar to the way GIT works. If revision history is not enabled then we can still keep track of the revisions made to either an aggregation or a resource but not take new snaphots of the aggregation. When changes are made, the elements that have not been changed can point to the same resources across multiple revisions, which is a lot more efficient. This is illustrated in the bottom right part of Figure 3.

Through the ORE interface, we would like to enable a fully programmatical means of interfacing with the tools and technologies that are hosted within NDS. This could involve building on our existing work on the ORE API to provide an API to fully interact with the research aggregations hosted using NDS. Conceptually for example, using this approach, a third party application could query NDS to search for a particular research piece, programmatically discover the contents of that piece, create a remote container, execute the research methods and extract the results for local analysis. Such an approach would not only promote open research and methods but it would also allow research to be shared and used in external applications and Web interfaces in a completely interactive way.

2. Metadata Importing

The dashboard should be capable of automatically importing publications, data and methods, if available, from several popular external Websites, including Mendeley, ResearchGate, etc., or by using the DOI for the publication and accessing third party APIs to automatically pre-populate the data. There are a number of APIs and tools that can be integrated here to resolve DOIs and to import data from publications across various websites and APIs. For example:

- The http://dx.doi.org/ Website by appending the suffix and the prefix of the DOI reference will resolve to URL that is associated with that DOI name.
- It is also possible to get structured metadata about the article using content negotiation using a simple HTTP request. This includes all CrossRef and DataCite DOIs. For example, executing:

  ```
  curl -D - -L -H  "Accept: application/rdf+xml" http://dx.doi.org/10.1016/j.future.2008.06.012
  ```

- Will provide all authors, the title, journal etc, as RDF. This also supports Atom, Turtle and CrossRef's Unixref format.
- CrossRef also exposes a REST API for returning metadata about publications[1], which can be used to extract various fields about publications
- Chrome's DOI Resolver is a pluggin for chrome that can be used to resolve DOIs.
- Mendeley's importer utility provides a mechanism for importing of articles based on some basic information: an article URL or DOI, which works for almost 60 different sites. It seems it might be possible to interface with this utility to automate the importing of data into the dashboard[2].

As well as numerous APIs from publishers, for example:

- Springer, http://dev.springer.com/
- Nature Linked Data Platform, http://data.nature.com/
- PubMed API, http://www.ncbi.nlm.nih.gov/books/NBK25501/
- PLOS API, http://api.plos.org/
- ArXiv API, http://arxiv.org/help/api/index
- SAO/NASA Astrophysics Data System focuses on astronomy and related fields http://adswww.harvard.edu/
- And http://libguides.mit.edu/apis provides a list of further APIs for Scholarly Resources.

[1] https://github.com/CrossRef/rest-api-doc/blob/master/rest_api.md
[2] https://www.mendeley.com/import/information-for-publishers/

3. Naming/UI Injection

The underlying tools and technologies in the dashboard are applicable in a wide number of modern technological contexts. Although, these tools are to implement the NDS dashboard, they could be also used to provide a dashboard for other entities that would like to couple data, methods and documentation to provide repeatable experiments and configurable apps. We would therefore like to provide a naming interface that allows complete configurability of the naming of entities e.g. an aggregation and a resource, to match the particular application-hosting environment. Further, since every use will require different metadata that describes an aggregation, we need a further requirement to enable the integration of multiple different metadata tables and form data for applying customized metadata for a Web aggregation. For NDS, this includes things like the title of the publication, the authors, the journal (issue & colume), the DOI, the page numbers, and so on. For other deployments, this may require other information, such as the Name of the App, a short description, a creator, an aggregation type, a company/university/collaboration/project, a bitbucket repository, and so on.

We therefore plan to enable complete flexibility on naming and we will provide interfaces for overriding the default metadata provide for a form and the ability to specify backend CRUD operations for these for reading, writing, updating an view the new metadata models.

4. Interactive Research Containers

To support the execution of methods on the data for a research paper, NDS Labs provides *research containers*, which provide a fully dynamic means of creating a container-per experiment instance. Research containers encapsulate the methods (i.e. code) and the environment (i.e. libraries required to support the execution of the code) within a Docker container. Currently, Docker instances support a handful of environments and we have mainly been focusing on support for a Python-based interface to research applications. When a researcher wants to run the experiment, a Dockerfile configured with the necessary Python stack is used to instantiate a container, run the research experiment and output the results.

The existing dashboard is fully integrated with this process. It interfaces with a backend REST API (Django/Python) which instantiates a container and retrieves a UUID for tracking its progress. The "Run" dashboard asynchronously queries the API for updates and for the STD output from the running application, which results in the live streaming of the command line output from the container being piped into a command-style window within the browser. When the application has finished, the backend API exposes the results of the research experiment as a TAR file by taking a diff of the file system, before and after execution. Once this is ready, the "Run" dashboard exposes a Green "Download Results" button for the user to download and view the results.

There are several avenues for extension here. First, the dashboard (nor the backend Epiphyte infrastructure) has a common security model. We need to add the ability for users to upload certificates for authentication to the backend tools to make this process fully integration with personal accounts.

For the containers, we would like to make research containers fully interactive using off the shelf Web browsers and provide a user experience where the research container seems fully integrated with their desktop. This work would include researching and delivering mechanisms to support the complete research container lifecycle so that a researcher using any programming language (or tool) can export/import his/her work to/from a research container. This is wide reaching and would need to support multiple scenarios, including:

- For Python based applications, this would continue our existing work to include the hosting of personalised iPython Notebooks (Jupyter Notebooks) and tools that facilitate the sharing of those e.g. nbviewer.
- For most other programming languages, it would need to provide tooling that could be used to easily configure a container with the underlying toolkits that would support the running of their application – this includes not only the compiler tools but any third party libraries they need for execution. This could be achieved using Python or other scripts.
- For graphical tools, we would need to provide a lightweight X Windowing system and VNC server on the container so that NoVNC Javascript-based remote desktop sessions can be run within the dashboard. This would allow full GUI interaction with any remote application from within the Web browser. This could support multiple third party applications that scientists use on a regular basis to explore, analyze and visualize data e.g. workflows, Matlab, etc.
- For command line tools, we would need to support shell scripts and data processing pipelines.
- For custom applications we would need to support the ingestion of custom configured containers that users may develop with third party tools but would want to host in NDS Labs. To this end, we would need to provide an NDS Container Repository that could leverage existing tools, such as Docker Hub, to support the hosting and interaction with any container that a researcher may want to provide
- Mobile Phone Interaction. NDS should be accessible through the use of a smart phone, Customized mobile interface should be proved to allow advanced interaction from a mobile device.
- Support for multiple container deployments and cross-container communication may also be required to support more distributed science experiments.

5. Data Interaction

The dashboard provides an interface to the backend NDS Labs Epiphyte data management system (which is currently based on iRODS). It is also interfaced with the Dropbox picker API, the SciDrive picker API and also partially integrated with Google Drive API, to allow the researcher to import data from a desktop into the system. This works in the following way for Dropbox, although the others work in a very similar way. The user chooses the Dropbox picker option which results in a browser window being displayed, allowing them to select a file from their personal Dropbox folder that is connected to their account. Dropbox makes this file available using an openly referenceable HTTP URL and returns this to the dashboard. The dashboard then POSTs this information to the backend NDS Labs Data API to store this file into the system. This, in turn, results in the NDS data management system downloading the file using the HTTP endpoint, storing it into iRODS and exposing the file using an *Epiphyte* URL. Using the dashboard's file browser and discovery process, the user can then select this file in the "Run" screen to choose the dataset they want to run using their methods within the container. This process therefore allows the research to modify the datasets or upload new data to be processed by their research application.

We will rewrite the backend infrastructure using frontend tools where possible to connect directly from the backend to the data API. We will also continue to interface with other data systems to support the wider integration of data management systems into NDS to support open data. This would include, where possible to integrate Web based APIs, such as File pickers e.g. Globus is currently developing a Web based File Picker API to expose data through HTTP endpoints.

For the Epiphyte portion, we will focus on providing a sophisticated Javascript-based data browser/search facility to the backend NDS data store for easier navigation by researchers.

Versioning also plays an important part here, so integration with version control systems would be a key component. Further conventional forms of documentation, such as Web-based document editing and sharing (e.g. wikis, Google docs, digital notebooks, etc.) and collaborative tools, such as WebRTC for AV and multi-user chat (MUC) facilities between community members.

6. Pluggable Components

The current implementation of the dashboard uses Semantic UI and JQuery on the front end with PHP and Yii to implement the backend logic and to interface with the database. We have also implemented a python/DJANGO interface to the database and this has been used to extract an ORE resource map of the publication and its associated artifacts (see Figure 1).

Moving forward, we would promote a more distributed approach. Instead of building monolithic codebases we take an alternative approach, which leverages the Microservices philosophy. Using lightweight toolkits, such as Django/Flask, we will focus on implementing a loosely coupled collection of the services, which each participant group can define their own manage policies and access control. We further will decouple the front end Web interfaces from these backend toolkits to allow complete separation from the Web tools and the backend Web APIs that are queried to delver the content in a specific way. By decoupling the user view from the backend API this makes it far easier to implement UI changes and also to support multiple "views" of the data e.g. you can use the backend Web API through the console, sockets, the Web, or a tailored mobile phone interface. In other words, you drastically increase the code reuse.

The Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of services, and the services may be written in different programming languages and use different data storage technologies.

For the backend stacks, toolkits like Django (Python), Yii (PHP) and Spring (Java) typically implement the model view controller (MVC) approach where the: model manages the data, the view presents model data to the user; and the controller mediates between the model and view. For the NDS dashboard, we will decouple the view further by implementing an AngularJS front end for the Web interface development (or possibly AngularDART). By separating the user interface components and layout (the View that is typically implemented in backend MVC systems) from the logic that observes corresponding user gestures (the Controller) we can provide good decoupling to cope better with inevitable changes. This will allow us to completely separate the front and backend logic and provide a consistent development methodology across the dashboard components.

For the front end, AngularJS also follows the MVC pattern of software engineering but it implemented completely on the Web client in Javascript. It uses lightweight directives that bind to Javascript models, which work in tandem to provide mutual updates. The model can then bind (through Web APIs e.g. Django) to load the data dynamically from a database. This approach encourages loose coupling between presentation, data, and logic components on the Web client, and does not attempt to tightly integrate the client and server components. Using dependency injection, AngularJS also brings traditional server-side services, such as view-dependent controllers, to client-side web applications. Consequently, the burden on the backend is reduced, leading to much lighter web applications that focus the logic of the views at the front end. This approach fits very neatly with the Microservices architecture and a single Web page could be interacting through models that load data from multiple Microservice backend Web APIs.

AngularJS is built around the belief that declarative programming should be used for building user interfaces and wiring software components, while imperative programming is excellent for expressing business logic. It adapts and extends traditional HTML to serve dynamic content through two-way data binding that allows for the automatic synchronization of models and views. As a result, AngularJS de-emphasizes DOM manipulation with the goal of improving testability and performance.

Using such an approach, it will make it far simpler to componentize the Dashboard into a set of reusable components, for data, for execution, etc. This will provide the necessary flexibility to allow better integration, embedding, sharing and collaboration, into third party tools, such as existing Science Gateways, portals, and other frameworks, to support a wide range of users, from novice to advanced.

7. Curation environments for Scientific reproducibility

For the Computational Sciences, research artifacts include not just data, but the software that produced the data artifacts. Without access to software, it is difficult to form a proper scientific contextualization, and therefore judgment about the resultant data. One possible mechanism for preserving and sharing of data is to utilize the principles of Linked Open Data [LinkedData] for which Tim Berners-Lee has strongly advocated. These principles facilitate data to be discovered, shared, understood and reused for scientific research utilizing web standards for handling data and encourage publication of data under an open license. Additionally, it allows for human contextualization to be captured and shared in a "machine understandable" manner that facilitates machine agents to assist in discovery and reuse. However, it has been observed that linked data without proper context is just more data in a different schema [Prateek10]. This observation is particularly true in the sciences where requirements such as provenance, quality, credit, attribution and methods are critical to the scientific process. For the Computational Sciences, scientific papers need to connect to both data and software. The context and provenance of data artifacts require connection to the software that produced those artifacts as well as connection to conceptual and mathematical models that constitute algorithms that are instantiated in software. We propose a model of publishing science results with software that would facilitate connecting data artifacts to the software algorithms implementing scientific *models*, which produced those artifacts utilizing a Linked Open Data Model. This model extends work done by different scientific communities to share measurements in a standardized way that capturesboth provenance, methods, conditions, units, of the measurement process and extends conceptualization to include model and algorithm that constitute a "computational measurement" or "observation".

To this end, we plan on integrating the existing DASPOS container provenance infrastructure into the dashboard. This will allow us to track containers as they evolve through different usages and versions using PROV-based provenance trails. PROV [PROV] is an Ontology Pattern that is an official W3C recommendation for capturing the context of **Agents**, **Activities** that the agents initiated or participated in and the resultant **Entities** that are the digital artifacts of interest. There has been early work that shows PROV provides a flexible foundation for contextualizing digital artifacts [Xiaogang14] as "Things" that are extensible rather than "Strings" with limited non-human understanding [Compton14] and lower barriers for *interoperability* between diverse data sets. This provenance graph is serialized in the W3C JSON-LD [JSON-LD] representation of RDF data and *attached directly* to the docker image metadata as a docker label. Metadata can than be tightly coupled to the docker digital artifact *independent* of the NDS dashboard system. A python tool using the RDFLib module has been developed to automatically capture some of the information needed for the provenance graph by "wrapping" the standard docker command-line interface. JSON-LD has the advantage of being a standard JSON document, and consumed as such, while also providing the ability to view the document using the RDF data structure and queried as such. In addition, JSON-LD is a valid hypermedia profile facilitating the "follow your nose" principle between data link relations described in the document. These link relations include Ontology Design Patterns (ODP) or modular, formal vocabularies, that are aligned and "linked" using these principles to facilitate the scientific domain knowledge contained within the container to be captured. ODP vocabularies are linked open data in themselves and are already being developed (e.g. NIH PubChem, Protein Data Bank, OpenPhacts, Earthcube, OBO) and published as part of efforts organized by the scientific domains themselves. For example, the W3C Health Care and Life Sciences have recommendations [TR2015] for publication of dataset descriptions as linked open data and developed a recommended vocabulary set as well as a guideline for the proper use of the dataset description vocabulary. The guideline vocabularies contain elements of the previously mentioned ORE vocabulary and would allow extended annotation of archived docker relative to the Health Care and Life Sciences community. A set of guiding principles, "Five Stars of Linked Data Vocabulary Use" [FIVE-STARS] has been published as part of DASPOS activities and adopted as a recommendation by the Semantic Web Journal for publications that include linked data and vocabularies.

8. References

[LinkedData] http://www.w3.org/DesignIssues/LinkedData
[Prateek10] Jain, Prateek, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, and Amit P. Sheth. "Linked Data Is Merely More Data." In AAAI Spring Symposium: Linked Data Meets Artificial Intelligence, 2010. http://www.aaai.org/ocs/index.php/SSS/SSS10/paper/download/1130/1454.
[PROV] PROV w3c working group. http://www.w3.org/TR/prov-o/
[Xiaogang14] Ma, Xiaogang, Jin Guang Zheng, Justin C. Goldstein, Stephan Zednik, Linyun Fu, Brian Duggan, Steven M. Aulenbach, Patrick West, Curt Tilmes, and Peter Fox. "Ontology Engineering in Provenance Enablement for the National Climate Assessment." Environmental Modelling & Software 61 (November 2014): 191–205. doi:10.1016/j.envsoft.2014.08.002.
[Compton14] Compton, Michael, David Corsar, and Kerry Taylor. "Sensor Data Provenance: SSNO and PROV-O Together at Last." Accessed October 25, 2014. http://knoesis.org/ssn2014/paper_9.pdf.
[JSON-LD] http://www.w3.org/TR/json-ld/
[TR2015] http://www.w3.org/TR/2015/NOTE-hcls-dataset-20150514/
[FIVE-STARS] http://www.semantic-web-journal.net/content/five-stars-linked-data-vocabulary-use