The Data Structure Protocol for Image-Analysis Networking: Modules and Motivations

Recent years have seen an increasing emphasis, in the academic and scientific worlds, on *data publishing* — sharing research data and experimental results/protocols via web portals complementing those that host scientific papers. Published data sets now take a position alongside books and articles as primary publicly-accessible outputs of scientific projects. Coinciding with this increased volume of raw data, there has also emerged an ecosystem of tools allowing researchers to find, view, explore, and reuse data sets. These tools enhance the value of published data, because they decrease the amount of effort which scientists need to make use of data sets in productive ways.

Unfortunately, however, this ecosystem of tools does not include extensive work on software *applications* for accessing and using published data sets. Prominent publishes (Elsevier, Springer, Wiley, de Gruyter, etc.) have all developed suites of components for manipulating data sets and data/code repositories, including APIs, search portals, Semantic Web ontologies and other forms of Controlled Vocabularies, and cloud-based computing or visualization engines, founded on technologies such as Jupyter, Docker, and WebGL. However, none of these publishers actually provide *applications* for accessing data sets outside of the online resources where data sets are indexed. While these online portals can provide a basic overview of the data sets, publishers do not provide tools to help researchers rigorously use any data sets once they are downloaded. Moreover, the ecosystem for manipulating published research is largely disconnected from the software applications which scientists actually use to do research. The ability to work with data-publishing tools has not been implemented within most scientific-computing environments.

These lacunae may be explained in part by publishers' and scientists' hopes of creating cloud-hosted environments that can themselves serve as fully featured scientific-computing frameworks, with the ability to run code, evaluate queries, interactively display 2D and 3D graphics, and maintain user and session state so that researchers can suspend and resume their work at different times. In these cloud environments, users can run computations and generate complex graphics on remote processing units, with relatively little data or code-execution stored or performed on their own computers. Such employmenr of remote, virtual programming environments is sometimes necessary when interacting with extremely large data repositories; and can be a convenient way to explore data sets in general, especially if a user is unsure whether or not a given data set is in fact germane to their research. Investigating data via cloud services spares the researcher from having to download the data set directly (along with the additional software and requirements which are often needed to make downloaded data functionally accessable). However, cloud-based data access is limited in

important ways, which makes relying solely on cloud services to provide the filaments of a research-data ecosystm a very bad idea. The first problem is that cloud services are, despite their technical features, essentially just web applications under the hood; as such, they are susceptible to the same User Expericne degradation as any other web service — subpar performance due to network latency, poor connectivity, and the simple fact that web-based graphics can never be as responsive or as compelling as desktop software, which can interact directly with the local operating system and react instantaneously to user actions. The second, more serious problem is that could-computing environments are computationally and architecturally different than the native-application contexts where scientific software usually operates. Insofar as researchers develop new analytic techniques, implement new algorithms, or write custom code to process the data generated by a new experiment, these computational resources are usually formulated in a local-processing environment that cannot be translated, without extra effort, to the cloud.

To be sure, scientists can sometimes "package" their experimental and analytic methods into a coherent framework, such as a Jupyter notebook, which serves as both a demonstration and a precis of their research work. Indeed, tools such as Jupyter (which packages code, data, and graphics into a self-contained Python-based programming environment) are useful in part because the content shared via these systems (e.g. Jupyter "notebooks") needs to be deliberately curated; building a notebook is a kind of summarial follow-up to actual research work. The intellectual discipline involved in packaging up one's research via such tools may be a valuable stage in the scientific process, but even then the programming environment where research code and data is publicly shared is fundamentally different than the environment where the research is actually carried out. As a consequence, sharing research indirectly via cloud services and or "notebook"-oriented frameworks like Jupyter is not really conducive to either reuse or replication. To actually replicate a course of investigation, it is more thorough to employ the same (or at least functionally equivalent) software for data acquisition, analysis, and validation as the original software; and to incorporate published data in new projects, the data should be shared in such a way that the original research data, code, and protocols can be absorbed into a new research context, including the software used by the research team. Cloud-based services, which provide only an overview of research data, with limited analytic and imaging/visualization functionality compared to actual scientific software, do not substantially promote data replication and reuse insofar as these cloud services are functionally disconnected from scientific applications themselves.

This is the motivation behind x, a *native*, *desktop-style* application for accessing research data of different kinds — within the overall space of published data sets we can find specific variations, such as *data repositories* comprising multiple data sets; *image corpora* designed as test beds for Machine Vision and diagnostic-imaging methods; *simulations* which involve not only raw data but digital experiments that can be re-run as a way to access the data; and so forth. Each of these various kinds of data sets present different sorts of interactive specifications which must be implemented by the data-set explorer software. While executed as a native application — not a cloud service

— x nevertheless incoporates the important ideas from contemporary data publishing (including ideas originating in the cloud context): workflow models, notebooks, access to publishers' APIs, etc. Another feature of x is that it can be run as a standalone application *or* embedded in other applications — such as the software which researchers are already using. In short, x can be seen as akin to a cloud-based data-publishing platform where the "cloud" is replaced by a scientific-computing application. Instead of being hosted remotely ("on the cloud"), x is hosted within a local desktop application. This host application may be pre-existing program, or a custom host implemented to allow x data-sets to be explored in standalone fashion (with a default implementation that can, as desired, be modified for individual data sets/repositories).

The Four Layers of the D-SPIN Object Model