Eating Your Own Dog Food

Jackson, Nick John
University of Lincoln
nijackson@lincoln.ac.uk

Winn, Joss Luke
University of Lincoln
jwinn@lincoln.ac.uk

July 2012

Abstract

As part of its project to develop a new research data management system the University of Lincoln is embracing development practices built around APIs – interfaces to the underlying data and functions of the system which are explicitly designed to make life easy for developers by being machine readable and programmatically accessible.

1 The Goals of Orbital

The Orbital project at the University of Lincoln aims to build a new research data management platform, alongside creating new policies and guidance for the institutional management of data. This platform, also named Orbital, will include a variety of features which distinguish it from a traditional repository by encouraging programmatic use and re-use of research data not only after a project has ended, but throughout the entire life of the project. Once a project has ended, Orbital will encourage deposition of 'complete' research data in more formal repositories for long-term archiving by allowing the rapid condensation and collation of research data, automatically supplying metadata where possible, encouraging high quality metadata where it is manually supplied, and automating the process of depositing a record in a repository.

2 What is a Repository?

The majority of existing repository solutions are built on the premise that items deposited in the repository will be discrete digital objects[1] and that these objects will have associated metadata describing their contents. In some cases, a deposit will consist of multiple discrete digital objects grouped together with a common set of metadata. In the case of research data, repositories tend to adopt a similar approach — data exists in discrete objects (for example an Excel spreadsheet, a ZIP file of images, a binary blob) and has attached metadata which describes it. Occasionally these repositories will offer the ability to view the contents of certain types of file natively within the application, although this is dependent upon there being an available parser or interpreter for the data. While this is trivial to achieve for data stored in simple, open formats such as CSV, it becomes increasingly difficult as data is represented using complex structures in formats such as XML, and is nearly impossible in many proprietary binary blob formats.

Commonly, such repositories also offer a limited choice of ways to deposit and otherwise interact with content. In most cases they rely on web-based user interfaces which are accessed through a browser to manage the complete lifecycle of an item from uploading through curation, management and long-term archiving. The SWORD2 protocol, while offering a programmatic method of depositing and updating files, does not offer methods of interacting with the data itself, but rather its metadata and the container file of the data.

In designing Orbital, we are assuming that research data may be too complex or too large for storage and transmission in file containers such as CSV or XML. In surveying researchers at the University of Lincoln, the most pressing requirement was for managed storage of research data during the process of research itself[4]; among the research groups that Orbital works with in the School of Engineering, requirements also centre around the sharing and management of data during the research process. In our experience so far, working in the domain of RDM highlights that the approach taken by the repository

[1]Or, in some cases, digital pointers to physical objects.

community for the management of research papers is inadequate for the management of the research data itself and requires a reconceptualisation of what data actually is. A file-oriented approach is inappropriate for many use cases, where researchers are collecting and analysing relational data or real-time data from instrumentation, for example. By designing an RDM tool that respects the form of data (e.g. time-series) rather than imposing a format (e.g. CSV), an API-driven approach to depositing, querying, manipulating and archiving the data, provides interfaces that are native to the domain of research data, rather than research outputs.

3 Our Approach

3.1 Love Your Backend

The University of Lincoln is moving towards a software architecture in which APIs – interfaces which allow systems and pieces of software to communicate in an easily machine-readable manner – are defined and built before any interface intended for the end user. Orbital is built as an example of this thinking, that the 'backend' of any system should be given attention earlier than the user-facing aspects. This is not in conflict with a user-centric approach to application development, but rather acknowledges that user requirements should be deeply embedded in the design of the application and not simply manifest in a single user interface. In the case of applications which manage raw data, traditional user interface design may be entirely inappropriate and inadequate. Many researchers already have appropriate tools for working with data, which could not and need not be replicated by a research data management tool, such as Orbital. The value of RDM to such researchers is the ability to reliably store, share and selectively publish their data for scrutiny and citation.

The result of building a system in this manner is that it is inherently usable and extendable by any other system with a minimum of effort. In repository systems which lack an API the process of adding data to the repository may involve multiple steps of exporting data from a database, importing it to an application such as Excel, sorting the data, exporting a spreadsheet, creating a new repository item, uploading the exported data, adding metadata, saving and publishing. Where an API is available this process can be automated and, as part of Orbital's pilot scheme, users in the School of Engineering are directly loading sensor data from a variety of sources.

In addition to providing these interfaces to the data, Orbital is exposing the entire feature set of the repository over APIs. This is achieved by ensuring that the development team build APIs which can expose the required functions to an external interface application rather than including any end-user interface functionality in Orbital's 'core'.

Finally, since Orbital is adopting a loosely coupled RESTful[2] approach to APIs, no part of Orbital's core functionality has any particular notion of statefulness associated with it. What this means from a practical standpoint is that it is easy to scale the platform horizontally by adding additional servers or storage, since there is no requirement for any process to maintain communication with a particular instance of a component.

3.2 Eat Your Own Dog Food

The Orbital project consists of two distinct components: Orbital Core, which performs all repository, data management, security and processing functions; and Orbital Manager, which provides a web-accessible user interface. The Manager aspect of Orbital communicates with the Core exclusively over the same APIs which are publicly available and documented for general use, with no preferential treatment or additional permissions as shown below in Figure 3.

A large part of building APIs lies in the documentation of available functionality both of the underlying code and of external interfaces. As Orbital is developed using iterative, agile principles and 'Continuous Integration'[3], this documentation is automatically generated, and it is this documentation that is relied upon by the Orbital developers where they are unfamiliar with a part of the system. The use of a single canonical set of documentation for both internal and external development avoids the creation of undocumented APIs which can only be used with expert system knowledge.

[2]REpresentational State Transfer

[3]`https://en.wikipedia.org/wiki/Continuous_integration`

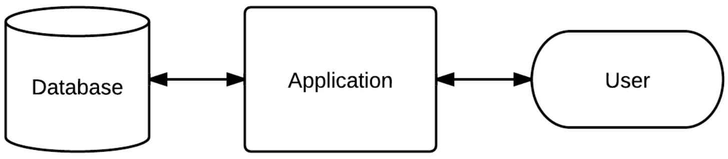

Figure 1: The only way to interact with this application is to either be a user, or pretend to be one (for example via screen-scraping).

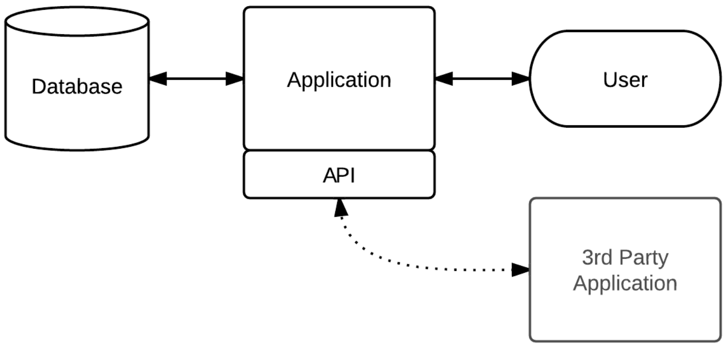

Figure 2: The most common form of API, consisting of a 'second view' on the data and functionality of an application. This style of API often exposes a limited subset of the application's functionality.

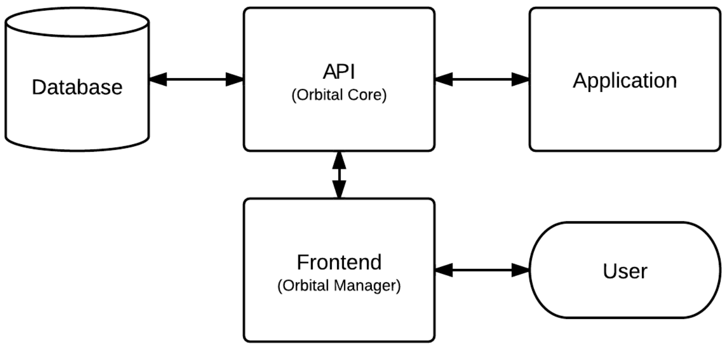

Figure 3: In an API-driven model the API is the only way to interface with the application.

3.3 Forget The Containers

There are already mature research output repository systems which store containers of data (i.e. ‘files’) for the long term, providing archival and discovery services. Orbital is not intended to be a replacement for such systems. However, there is a distinct lack of general purpose research data management systems which store and help manage the data throughout the actual research process. Although Orbital does allow researchers to upload and store discrete files[5], and we intend to provide a shared, synchronised filesystem for researchers[6], a defining feature of the approach of Orbital is that – as far as is possible – data can be removed from container files such as CSV files or Excel spreadsheets and stored in a database[2].

Access to and control of this data is provided by RESTful APIs, which align with Orbital’s overall design. These APIs can be used for the deposition of data either in large blocks or at a very granular level (roughly equivalent to a single row in a spreadsheet), the modification and retrieval of those individual data points, and querying of the data sets in real time using a variety of database operators, and exporting the results of those queries as a rich JSON document or a simplified CSV document which can readily be used for further analysis. APIs are also planned to allow for the description of data (including aspects such as the type of data) and the addition of metadata on a data set or data point level (for example allowing annotation of an anomalous result to be stored with the result itself).

Data itself is stored in the MongoDB No-SQL database, which provides a number of advantages over traditional SQL databases with regard to research data storage. MongoDB is a document storage database which holds representations of data objects as opposed to structured, tabular data. These objects may be heterogeneous with little or nothing in common, a feature which allows individual data points to contain differing quantities or properties depending on what is known about them. This means that a data point may initially be deposited with relatively little information and subsequently updated with additional facts, notes or analysis after the fact and with no impact on the underlying storage engine. As explored in Section 3.4, data may also be subject to automated processing which expands upon the original data.

By removing the data from container files and allowing direct deposition and access, Orbital exposes data for re-use the moment it has been stored. This dramatically reduces the amount of time spent waiting for data to be made available to researchers following its collection, which is of particular interest to the project’s pilot users in the Lincoln School of Engineering. Where previously data could take upwards of a week to be moved from its collection point through various storage systems and then manually exported in the appropriate format for further research, Orbital now produces daily datasets for the same work.

3.4 Make It Extensible

Due to the document storage model and API accessibility used within Orbital it is relatively simple for data objects to be updated with additional information by the means of single purpose, automated applications. Again in the case of the Lincoln School of Engineering, some data requires processing with a variety of signal filters in order to become usable for further statistical modelling. Previously this would have been a batch operation in which the researcher would take the original data, perform the processing, and store the output in a separate file. Using Orbital’s data storage engine the results of the processing are stored alongside the original data and are similarly queryable or usable in subsequent tasks.

By encouraging the breaking down of data into smaller chunks, Orbital also encourages the reduction of the processes involved in research into smaller, more manageable and above all re-usable aspects. It is envisioned that researchers will be able to build complex automated workflows, including import from external sources, processing of data using pre-built or custom modules, and subsequent export to files or specialist tools from directly within Orbital.

One thing that Orbital isn’t designed to do is replicate the functionality of more traditional and established repositories. Although Orbital is designed to be able to hold files in perpetuity following the end of a project it will also use protocols such as SWORD2 to allow rapid deposition of a record of research data – including all relevant metadata – with an existing institutional repository.

4 Why Data-Driven Development?

4.1 We Build Better Stuff

Adopting a data-centric, API driven approach to building Orbital has resulted in the development of APIs which are easily understood, easily implementable, behave in a reliable and predictable manner and which are robust enough to withstand heavy usage. APIs which lack these criteria are quickly identified and refactored, since they are being used by members of the Orbital team to build the Manager platform. Since APIs are the exclusive method of interfacing with the underlying data storage and the functions of Orbital the APIs must expose a complete set of functionality[3].

Reliability of these APIs is also critical to the usability of the entire data storage platform, which encourages better design of resiliency and error handling; and usability of the API is essential which encourages better documentation. By forcing the consideration of data types, methods and architecture early on it is easier to achieve a consistent design and behaviour across the entire Orbital platform, as well as an underlying structure which encourages the development of reusable, well-defined code.

Finally, by combining the aspects of reusable components and consistent APIs, Orbital is able to easily implement a robust universal security model using the OAuth 2 specification which encompasses all data access regardless of source. Since any frontend can only interface with the data via the controlled APIs, and since access to the data is governed on a per-user-per-application basis there is a reduced scope for accidental or intentional damage or loss of data, and a lower risk of the accidental disclosure of commercially sensitive data. OAuth 2 also enables the development of third-party tools which can interact with data in Orbital without exposing any user credentials[1], encouraging the building of tools which extend the ecosystem.

4.2 Improved Data Visibility from Day One

The DAF[4] survey conducted as part of Orbital's research shows that although most researchers know roughly how much data they have, that data is stored primarily on non-managed devices such as internal and external hard-disks, USB memory sticks, and third-party data storage systems[4]. During user meetings and observations of research by the Orbital team it was found that among these various locations there was often confusion over where a particular file was stored, which version any particular file may be, and if any given file actually contained the data in question.

By stripping away the container file and allowing searches to be run across the entirety of a data set – even up to the scale of millions of data points – the entire dataset becomes visible to the researcher at once for the purposes of finding what they're looking for.

Orbital also has the potential to provide improved information on the quantity of data being stored for the purposes of planning and reporting, giving academics a truer indication of the amount of resources being used and giving research administrators a clearer idea of the requirements for future research. In addition, since Orbital exposes its complete subset of functionality, it can quickly be integrated with other University systems such as staff directories or our Awards Management System to assemble disparate information regarding research data.

4.3 Scale To Infinity

A major benefit of using loosely coupled components which communicate using RESTful APIs, along with using an inherently scalable database solution such as MongoDB, is that the Orbital platform can scale horizontally to a very large system simply by adding new nodes. An example of an expanded, scaled deployment is given in Figure 4 on the next page.

Although in development Orbital is not using a deployment of this size, the application is designed to be inherently scalable as the number of users and size of datasets grows.

5 The Problems of This Approach

API-driven development, whilst having many benefits, does have downsides which have become apparent during the development of Orbital. Firstly, development must – in many instances – be 'doubled up' as developers create both the underlying functionality and API endpoints in Orbital Core, and then an

[4]Data Assets Framework

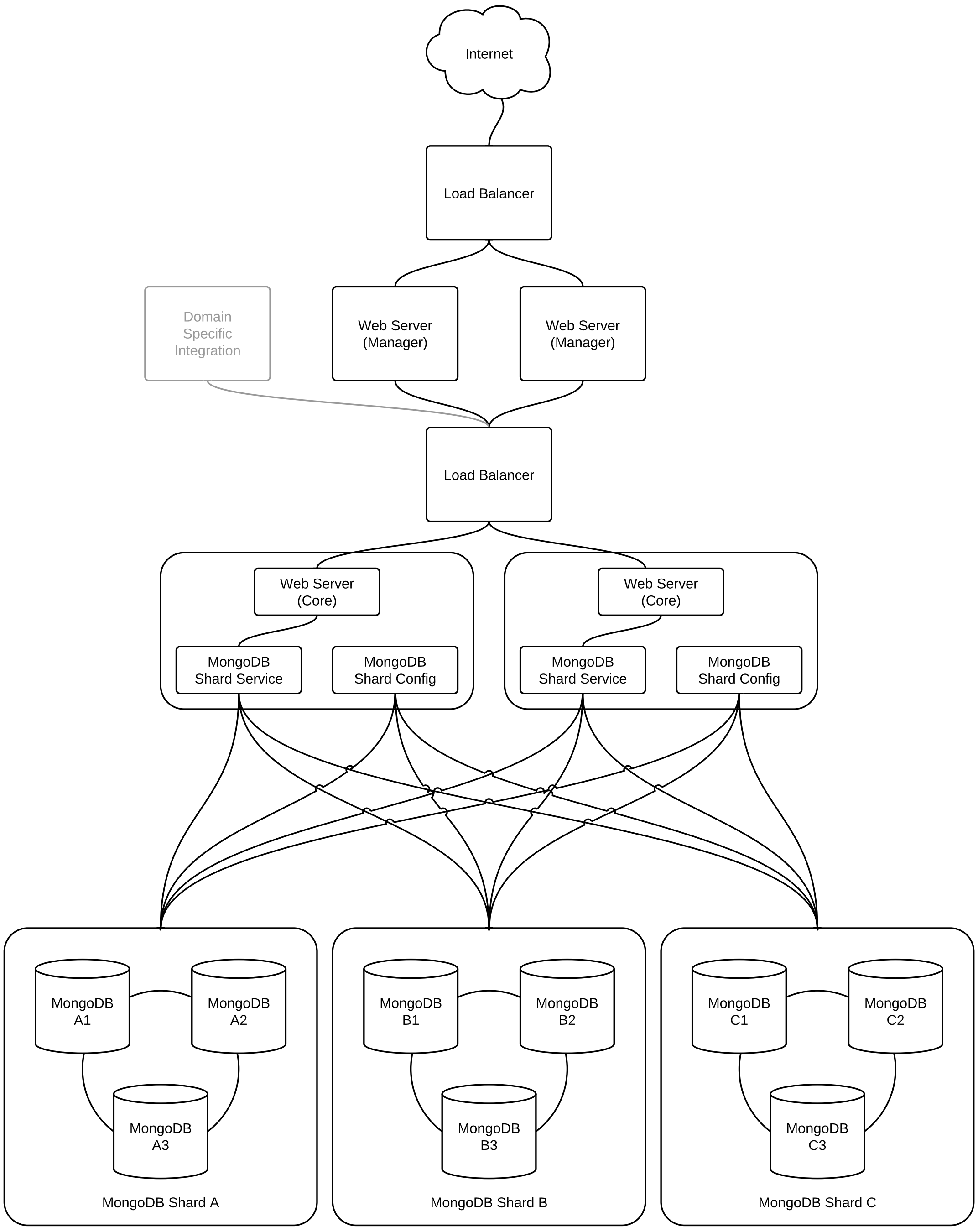

Figure 4: Example of a scaled Orbital environment, with sharded and replicated database, load balanced Core servers and load balanced Manager frontends.

API interface and user front-end in Orbital Manager. Although this does allow rapid development and development of better software at a later point initial work can appear slow and unnecessarily complex.

Secondly, there is also a significant challenge in ensuring that the transfer of data over APIs is fast and reliable enough for intended applications. The HTTP protocol can introduce an element of overhead which can rapidly accumulate in applications which are making a large number of API calls, and in some cases the Orbital project has modified APIs to be more lightweight or to allow for more intelligent grouping of requests. There has also been a significant amount of work and re-work around error handling — particularly with regard to correctly handling security-based errors, an area in which development is still ongoing.

References

[1] Alex Bilbie. What is OAuth? [online]. July 2012. Available from: `http://lncn.eu/dfps`.

[2] Nick Jackson. And now… dynamic data! [online]. June 2012. Available from: `http://lncn.eu/cjv2`.

[3] Nick Jackson. Why orbital is all about the api [online]. January 2012. Available from: `http://lncn.eu/euw5`.

[4] Joss Winn. Data assets framework survey summary [online]. April 2012. Available from: `http://lncn.eu/rs2`.

[5] Joss Winn. A minimum viable product: Orbital v0.1 [online]. May 2012. Available from: `http://lncn.eu/y26`.

[6] Joss Winn. Shared, versioned network drives [online]. May 2012. Available from: `http://lncn.eu/egj7`.