Towards Lensfield: data management, processing and semantic publication for vernacular e-science

Nick Day, Jim Downing, Lezan Hawizy, Nico Adams and Peter Murray-Rust
Unilever Centre for Molecular Science Informatics, University of Cambridge, CB2 1EW, UK.
ned24@cam.ac.uk

Abstract

Lensfield is a desktop and filesystem-based tool designed as a "personal data management assistant" for the scientist. It combines distributed version control (DVCS), software transaction memory (STM) and linked open data (LOD) publishing to create a novel data management, processing and publication tool. The application "just looks after" these technologies for the scientist, providing simple interfaces for typical uses. It is built with Clojure and includes macros which define steps in a common workflow. Functions and Java libraries provide facilities for automatic processing of data which is ultimately published as RDF in a web application. The progress of data processing is tracked by a fine-grained data structure that can be serialized to disk, with the potential to include manual steps and programmatic interrupts in largely automated processes through seamless resumption. Flexibility in operation and minimizing barriers to adoption are major design features.

1. Introduction

Reuse of scientific data is central to much of eScience. However, most of the data produced by individual researchers and groups is never made publicly available, and is eventually lost. Where data is made available, effective sharing is often prevented by lack of common resource discovery mechanisms, and by format interoperability issues. These problems are not unique to science, but also apply to, for example, census data, library catalogues, encyclopædias etc. "Linked Data" describes a set of semantic technologies that can be used to publish, discover and combine data on the world wide web in an interoperable way. When combined with open licenses / arrangements[1] as "Linked Open Data" (LOD), we believe it has great potential to deliver the same benefits to scientific data. However, a researcher must surmount considerable technological barriers to publish LOD. Our goal in this work was to develop a tool to make the preparation and publication of LOD painless and straightforward for researchers.

The first incarnation of our tool was a middleware that allowed researchers to manually or programmatically upload their data to a server for "semantification", the process of converting data into the triple-based RDF standard required for LOD. The tool therefore abstracted the semantic web technologies, but still required the researcher to write programs that dealt with URLs as references and interacted with persistent storage through a network API. This proved clumsy for researchers to deal with during iterative, day-to-day work.

Whilst working on this first incarnation, we observed the working habits of a range of chemoinformaticians, and it became clear that there was also an opportunity to assist in the preparation and handling of data. While several "big science" domains have built large automated data infrastructures, much, perhaps most, science is in the "long tail", in common with chemoinformatics. This is characterized by small groups with little formal computing and informatics expertise running on local machines.

In this paper, we present a set of pragmatic requirements derived from this early experience, explain how some modern technologies present novel opportunities in data management, and explain how we are combining these into Lensfield, the latest incarnation of our LOD tool.

Lensfield has four main areas of functionality: -

- Data management
- Data production description
- Process automation
- LOD publication

1.1 Computational chemistry as an archetype

We have piloted Lensfield in a number of chemistry-related domains, and some of these are described throughout this paper. One of our first pilot applications was in computational chemistry. CompChem can predict the properties and behavior of a very wide range of substances and processes and is frequently used in preference to experiment. Calculations can be precisely defined through parameterization and are reproducible across institutions and codes. It represents one of the largest users of computing in chemistry, yet most jobs are submitted manually and the results are also analysed manually. Billions of cpu hours are used each year to create results which are expensive and could be shared but almost none of this work is made available to the community for re-use; the same calculations are probably re-run many times. As members of the European COST D37 Semantics and Workflow Group we are developing standards and tools to automate workflow and are piloting Lensfield with several groups.

A typical CompChem project could involve many thousands of molecules (as *.cml files) run with a range of parameters ("parameter sweep"). Jobs are created as (say) *.gin files for all combinations and submitted asynchronously to Gaussian03 program on distributed computing resources (clusters, clouds, etc.). The jobs return after seconds, or hours or days (depending on size, rate of convergence, etc.) as *.g03 (output) files and as resources become freed more jobs are submitted. Not infrequently jobs crash and need re-running. The scientist needs someone to keep track of these jobs and "remember where she has got to". The *.g03 files are converted to *.cml (using a specific "JUMBO-converter"[2] – see section on JUMBO-Converters, below) which is further converted to RDF for LOD publishing and also to *.html and *.png for conventional web pages.

2. Design Requirements

In looking for design approaches we have been influenced in spirit by the "Principled Design" of Roy Fielding and Richard Taylor [3]. We take their seminal work on REST and abstract those fundamental ways of thinking that lead to successful modern shared systems.

In order to minimize the barriers to a researcher using a tool it must work in their own familiar environment. Taking this "vernacular" approach to design we arrived at the following functional and non-functional requirements for Lensfield:

Desktop based: must work on the researchers' workstation rather than through a client/server model.

Filesystem based: researchers most commonly use folder and files to manage their data.

Focus on description before automation: researchers use a wide range of tools and manual processes in data preparation; limiting the availability of these would hinder adoption of the tool. It is more important to know the provenance of data than to have the production of it fully automated.

Promote versioning best practice: in order to provide reliable rollback to reprocess data, it is necessary to hold versions of data and process (represented by configuration files, programs, scripts etc) in parallel.

Work "The Wiki Way": Users progress most quickly when simple things are simple, and complex things are possible[4]. Lensfield provides good out-of-the-box functionality, sensible defaults and automatic configuration for well-known program outputs.

3. Technology

In most modern software development, the strengths and weaknesses of an application are driven by the strengths and weaknesses of the underlying technologies. Lensfield's supporting technologies were carefully selected to make it as simple as possible to develop a system that meets the requirements described above.

3.1 Mercurial

In order to enable rollback to previous versions of processing code and data, some form of revision control is needed. Revision control is well understood, with a wide range of mature implementations; it would be senseless to reinvent this functionality without very good cause. Lensfield interacts with the Mercurial Source Control Management (SCM) system to provide simple, lightweight revision control. Mercurial works as a Distributed Version Control System (DVCS), an approach which has several useful benefits when applied to research data management in Lensfield. In a DVCS, there is no central, authoritative copy of the code (or "repository") that working copies must act as satellites to. DVCS systems enable sets of changes to be extracted from any repository and applied to another, and keep track of a full history of all the changes that have been applied. This brings a great deal of flexibility in how networks of repositories can be arranged, and is most effective when such an arrangement mirrors natural social patterns of data sharing. DVCS also allows a repository to participate in several networks. For example, a researcher might work on a Lensfield project on their workstation and send incremental changes to a server for backup. As the project progresses, they might collaborate with a colleague, exchanging changes as they improve the processing and data. Later still, they might wish to work with a wider community by creating a copy of their repository on a public server.

3.2 Clojure

A number of factors contributed to the selection of Clojure[5][6] as the programming language for Lensfield's implementation. The chemoinformatics community has developed a rich set of software libraries, primarily in the Java programming language (e.g. JUMBO-Converters, CDK[7]). To leverage these libraries most effectively we needed a language with strong Java interoperability for Lensfield. Clojure programs compile directly to Java bytecode and can call Java classes directly, without any intermediate interfaces or proxies.

The trend towards increasing CPU count and cores per CPU rather than increasing clock speeds means that applications of all types increasingly have to rely on concurrency for performance. Functional languages are perhaps the most promising alternative for writing concurrent applications as they avoid the difficulties inherent in managing concurrent access to mutable state. Clojure takes a pragmatic view, encouraging the use of pure functions whilst allowing controlled mutation of state through Software Transactional Memory[8]. Lensfield's build execution makes use of Clojure References[8]. These allow safe concurrent mutation of state using a similar paradigm to database transactions: changes to References are guaranteed atomic, consistent and isolated (although not durable, since the state is purely held in RAM, c.f. ACID[9]).

As Clojure is a dialect of LISP it provides a powerful macro system. Whilst the benefits of macros can be achieved in many programming languages, our experience in Lensfield is that they provide an extremely efficient way of abstracting implementation complexity from users.

3.3 Sesame

For LOD to be truly usable, one needs to be able to query and retrieve it. Whilst documents containing RDF data can be made available as simple file-like web resources, the most utility comes from exposing a "SPARQL Endpoint" for users to query. SPARQL is the standard query language of the semantic web, and comes with a standard protocol for making queries and returning query results[10]. Lensfield uses Sesame (Java Open Source RDF store) to provide RDF indexing and SPARQL language support.

3.4 JUMBO-Converters

Over this decade we have developed software thay converts legacy documents to CML, originally legacy2cml and now JUMBOConverters. These have been designed to be side-effect-free black boxes with normally one input and one output. These are strongly typed, and can be linked into chains with information being converted to and from CML. Besides being used for creating CML from legacy they are also capable of creating program input (since most computational programs have non-semantic input).

4. Architecture

Fig. 1 shows an overall view of Lensfield's architecture, showing the version control, build and publication elements. Rather than take a structure-based approach to explain architectural features in Lensfield, this section does so by describing an archetypal project that uses Lensfield; the processing and visualization of output files from the Gaussian03 program. The explanation assumes that Lensfield is already installed on the users system – Lensfield's build, distribution and installation are in development and will be the subject of a future publication.

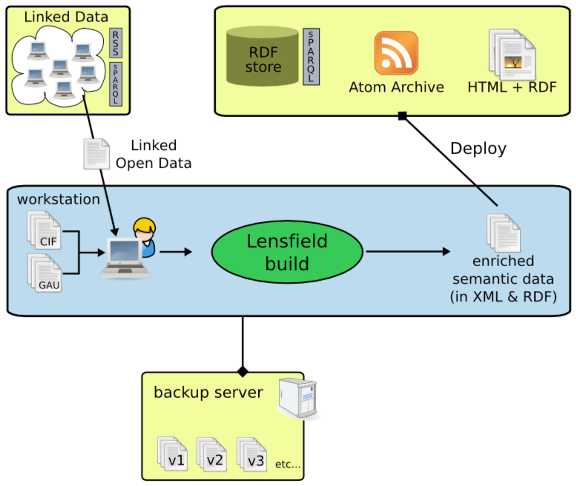

Figure 1 Architecture diagram for Lensfield.

The project is initialized by creating a folder on the file-system, as illustrated below:

```
> lensfield.sh init
Initialised!
```

This sets up an idiomatic file structure, and primes the mercurial version control by initializing the folder as a (mercurial) repository, adding the basic files to

this repository and instructing mercurial to ignore those files it is unnecessary to keep track of. It also generates a minimal `build.clj` (further described below).
The researcher would then begin the process of assembling their data files and describing their data processing in `build.clj`. From time to time the researcher can use `lf-snapshot` to create a rollback point in the project.

```
> lensfield.sh snapshot
Would you like to commit/ignore the
following files?
./data/gau/n0001.g03 (c/i)
```

The philosophy of Lensfield is to provide convenient shorthands for interactions with the enabling technologies, rather than limiting the user to a simple abstraction. In the case of `lf-snapshot`, Lensfield helps the user through the process of making sure all files are either added to version control or else ignored, and then commits these as a change set to the mercurial repository. If the researcher is familiar with mercurial, they are free to interact with the repository directly; this will not interfere with Lensfield's operation. Similarly, if the user has a backup server configured, they can run `lf-backup`, which simply pushes their changes to a remote server, creating the remote repository if necessary.

```
>> lensfield.sh backup
Project backup completed!
```

Despite this simplicity, these two functions provide a snapshot, backup and hence enable rollback and recovery of data processing projects.

4.1 Lensfield Components

Lensfield is designed to hide most of the complexity from the user. It contains and manages a mercurial repository and a sesame RDF engine. Lensfield (middle layer) provides generic macros (such as source and product) and functions such as template-match. The build.clj are specific to each project and may call additional custom programs (in Java or Python).

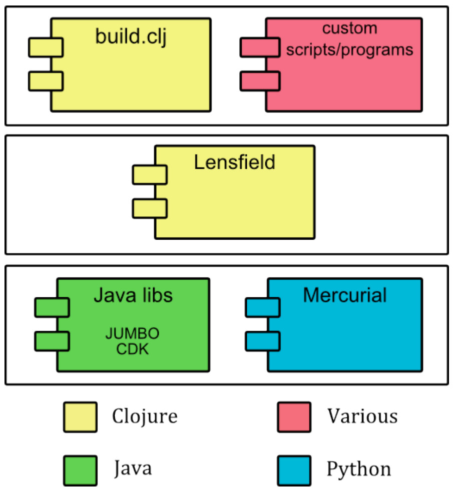

Figure 2 Layer cake of the technologies that comprise Lensfield.

4.2 Lensfield Build Description

Lensfield uses a software build paradigm for data processing; rather than describe the process imperatively with control flow (loops, conditionals etc.), the focus is on *describing* data artifacts, and the relationships between them (Fig. 3). [The examples and figures in this section describe a typical build for CompChem.]

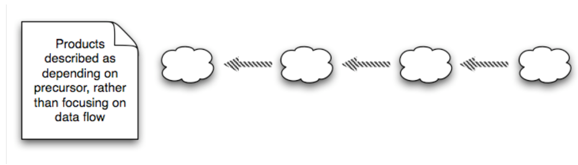
Products described as depending on precursor, rather than focusing on data flow

Figure 3 A build dependency chain.

In this way a Lensfield build is similar to those provided by common build tools such as `make` and `Ant`, which allow the specification of a target from a set of dependencies. These relationships may represent the artifacts being produced from a computational process, or by a manual process (e.g. manual adjustment of parameters / algorithm selection etc). This is an important principle as it values the recording of the provenance of the data produced over comprehensive automation. The `build.clj` file for the data build represented in Fig. 4 is shown in Fig. 5:

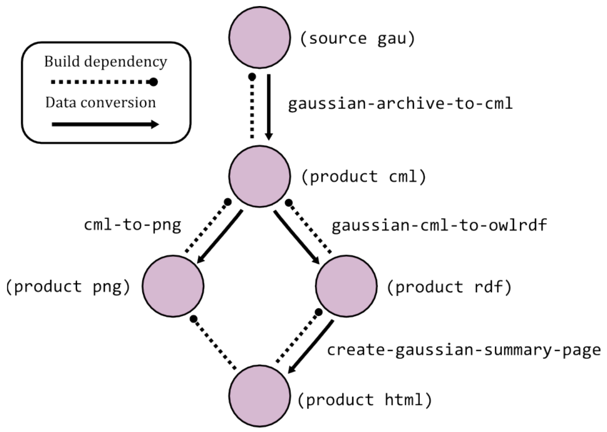

Figure 4 Directed acyclic graph showing data conversion flow and build dependencies in a Lensfield build.

`gaussian-archive-to-cml` is a function call to a g03toCml JUMBOConverter with a single input and output – many workflows will have a different converter for each step (see Fig. 5). Lensfield comes out-of-the box with many JC's, which are a mixture of generic transformation and (currently) chemical-format interconversions.

```
(source gau
  (template-match "./data" "{id}/{name}.g03"))

(product cml { :input gau
               :converter gaussian-archive-to-cml
               :output "{id}/{name}.cml"})

(product png { :input cml
               :converter cml-to-png
               :output "{id}/{name}.png"})

(product rdf { :input cml
               :converter gaussian-cml-to-owlrdf
               :output "{id}/{name}-gau-cml.rdf"})

(product html { :input [png rdf]
                :converter create-gaussian-summary-page
                :output "{id}/{name}.html"})
```

Figure 5 Example Lensfield `build.clj` file.

The build is represented as a directed acyclic graph of a `source` and a series of `products`. Through Clojure's macro feature Lensfield encapsulates the implementation efficiently – users solely focus on describing relationships and Lensfield expands these to form an appropriate execution strategy, without requiring the overhead of a Domain Specific Language or a configuration file and parser.

The build structure and principle of description still allow extensive data processing automation. Lensfield executes a build by expanding the `build.clj` file to a series of functions that must be run to complete the build. These are placed into a data structure that is updated as the build proceeds (Fig. 6), adding logging information and updating status. This build structure can be serialized to enable automatic checkpointing.

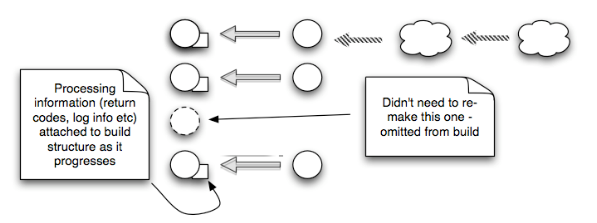
Processing information (return codes, log info etc) attached to build structure as it progresses
Didn't need to re-make this one - omitted from build

Figure 6 Description of how a Lensfield build structure is built up.

Lensfield stores the incremental build structure in a Clojure STM Reference[8], as explained in section 2.2, which allows a range of concurrent execution strategies safely and simply.

As the build structure may be serialized at any point during execution, it is possible for partial builds to be run. This is necessary if any manual steps have been defined in the `build.clj`. Upon resumption, Lensfield is able to read in the saved build structure and resume from the point that execution stopped.

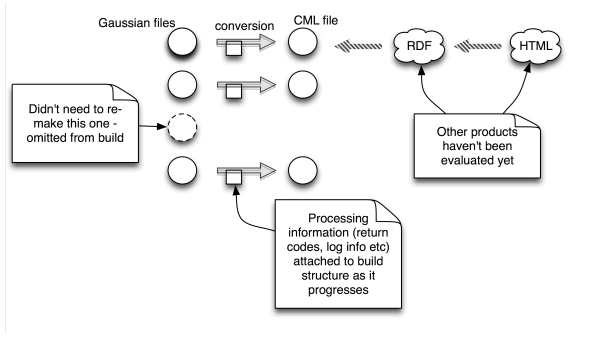
Gaussian files
conversion
CML file
RDF
HTML
Didn't need to re-make this one - omitted from build
Other products haven't been evaluated yet
Processing information (return codes, log info etc) attached to build structure as it progresses

Figure 7 Description of how a Lensfield build may be resumed after pausing.

After Lensfield has analysed all the dependencies and determined that no further work is to be done the RDF files are packaged to a standard Java web archive file (WAR - Java web archive files bundle the configuration, resources and programs needed for a full web application into a single, deployable file) file which may be independently deployed to a Java webserver.

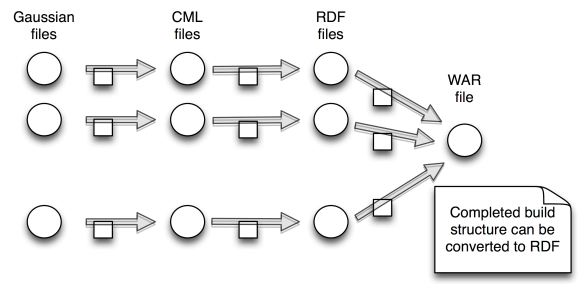

Figure 8 Description of how a Lensfield build may be resumed after pausing.

At any stage the build structure represents the state and history of the system; therefore the final build structure is the full data provenance for that run.

5. Further Lensfield Examples

We have created Lensfield builds to describe the processing in ongoing projects at the Unilever Centre. The `build.clj` file shown below describes the steps necessary to convert CIF files (standard crystallography interchange format) on a file-system into RDF, a process used in both CrystalEye[11] and C3DeR[12].

```
(source cif
  (template-match "./data" "{name}.cif"))

(product cifxml { :input cif,
   :converter cif-to-cifxml,
   :output "{name}-cif.xml"})

(product raw-cml { :input cifxml,
   :converter cifxml-to-raw-cml,
   :output "{name}-raw.cml"})

(product complete-cml { :input raw-cml,
   :converter raw-cml-to-complete-cml,
   :output "{name}-complete.cml"})

(product html { :input complete-cml,
  :converter create-cif-summary-page,
  :output "{name}.html"})

(product rdf { :input complete-cml,
  :converter complete-cml-to-rdf,
  :output "{name}-cif.rdf"})
```

Figure 9 An example Lensfield `build.clj` for converting a standard crystallographic format into RDF.

As described earlier, the source data for a Lensfield build need not come directly from a file-system, but may come from an external source. For instance, Lensfield contains `supplemental-file-crawler` which extracts supplementary data from published chemistry article from several major publishers. In the example below, the function will start a web-crawler that searches the latest issue of Nature Chemistry[13] and returns any CIF files that are found. If the source in Fig. 8 was replaced with this, then this would give a Lensfield build that created RDF for all crystallographic data found in the latest issue of Nature Chemistry.

```
(source nature-chemistry
  (supplementary-file-crawler :nature :chemistry
                              "./data"
                              "{article}.cif"))
```

Figure 10 A `source` step for scraping crystallographic data from Nature Chemistry.

6. Conclusion

Lensfield's attraction for the scientist is based on the following:
Principled design leading to low barriers to adoption. The success of REST in creating a community of practice has inspired us to design Lensfield for easy and rapid adoption before over other considerations. The users we aim at have a low tolerance of complexity and demands on maintenance, but appreciate systems which can be run and run repeatedly. Lensfield is designed as a scientist's amanuensis which understands a few simple commands and looks after the rest.
DVCS making good data practise easy. Data management is science is often a nightmare, which traditional backup systems do little to solve. Researchers frequently create different versions of work and "forget where they have put them". Lensfield can explore the archaeology of a project, for example in recreating the thought processes that went into tuning the parameters of a calculation
Recording build progress. Lensfield records the build at the granularity of each independent "arrow" step. This trivially allows resuming on the same machine, but also the project directory can be picked up and run in a different environment. An example is running a small number of test jobs before transferring to a more powerful system. Lensfield also supports the interjection of processes not under build.clj control such as the results of physical experimental experiments. If, for example, a researcher wishes to compare measurements with computation the lab data can be labeled as a dependency for a downstream process which will be able to be run when the data has been entered.
Easy publication of LOD from e-science. The packaging mechanism allows a complete project to be

tailored for publication and distributed to the appropriate server. It also can act as a deposition into the scholarly publishing process and ultimately trusted digital repositories.

7. References

[1] Guide to Open Data Licensing, The Open Knowledge Foundation Wiki, http://wiki.okfn.org/OpenDataLicensing, accessed on 2009-08-07.

[2] P. Murray-Rust, H. S. Rzepa and M. Wright, "Development of Chemical Markup Language (CML) as a system for handling complex chemical content", New. J. Chem., 2001, 25, pp. 618-634.

[3] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture", *ACM Transactions on Internet Technology*, 2 (2), 2002, pp. 115-150.

[4] B. Leuf and W. Cunningham, *The Wiki Way: Quick Collaboration on the Web*, Addison Wesley, 23 April 2001.

[5] R. Hickey, "The Clojure programming language", *Proceedings of the 2008 symposium on Dynamic language*, 2008.

[6] Clojure homepage, http://clojure.org/, accessed on 2009-08-07.

[7] C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttman and E. Willighagen, "The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics", *J. Chem. Inf. Comput. Sci.*, 2003, 43 (2), pp. 493-500.

[8] Clojure Refs and Transactions, http://clojure.org/refs, accessed on 2009-08-07.

[9] A. Reuter and T. Haerder, "Principles of Transaction-Oriented Database Recovery", *ACM Computing Surveys*, 15 (4), pp. 287-317.

[10] SPARQL Protocol for RDF, http://www.w3.org/TR/rdf-sparql-protocol/, accessed on 2009-08-07.

[11] CrystalEye, http://wwmm.ch.cam.ac.uk/crystaleye/, accessed on 2009-08-07

[12] Cambridge Chemistry Department Crystallographic Repository, http://wwmm.ch.cam.ac.uk/projects/C3DeR/index.html, accessed on 2009-08-07.

[13] Nature Chemistry homepage, http://www.nature.com/nchem/index.html, accessed on 2009-08-07.