A report to the Joint Information Systems Committee regarding

Nature Publishing Group's Project "ROSA".

by Ben Hammersley

An introduction for the shorter attention span.

Ok, so the top line: it's good. It's very good. It is, and this is not putting it lightly, quite possibly a milestone in its class. In funding this project, you not only got your money's worth, but also changed the world a little. Only a nudge here and a prod there, for sure, but a change nonetheless, and a change for good.

So, to summarise again: it's good. Nice one. In the rest of this report, we'll look into why it's good, and what you can do to make it even, as they say, shinier. But for now, relax, sip that coffee, slip off your shoes, and settle down into your chair. It is not every day you change the world, is it?

Quite.

Now, a word about this report. The first section will address what the Nature Publishing Group actually did: their achievements against their promises, and how they did what they did. That will necessitate some technical talk - for which I apologise - but this jargon can be safely ignored if it means nothing to you. Skip over it: feel free to bail out of paragraphs a few lines too early if the acronyms get out of hand. I know I do.

The second section addresses the future options for the project. This too might have jargon, but will explain such as it goes along. The more racy and daring of you might skip straight to this section, chequebook at the ready, and eager in your quest to make the world a better place. Either way, the conclusion of this report can be summed up thus:

Excellent so far, full of potential, give them more money.

So, firstly, some explanations…

Just who is this guy to tell us these things?

Firstly, I quite literally wrote the book on RSS: "Content Syndication with RSS", (O'Reilly and Associates, 2003), the key text in the field. I'm a member of the RSS 1.0 Working Group, the body entrusted with the upkeep of the standard, and have authored three major RSS module standards, including the Creative Commons module, and the Streaming Media vocabulary. I lectured at this year's Emerging Technologies conference in Santa Clara on the representation of message threading in RDF, and have contributed code to many of the major RSS packages. I am the master. Fear me.

Secondly, I'm a national newspaper technology journalist. I write for The Times, where I was internet correspondent during the dotcom boom, and for The Guardian. I specialise in the social effects of cutting edge technology. This is cutting edge social-software, so it's my speciality from an end-user point of view. I'm also available for parties.

and what are all these funny words?

A short glossary for the more socially adept in our midst.

RDF - Resouce Description Framework, a data format that provides a sematically complete way of expressing metadata.

FOAF - Friend of a Friend, an RDF vocabulary that allows you to describe people, and more specifically, their social relationships.

CPAN - the Comprehensive Perl Archive Network, a repository of Perl modules on the internet. Beyond useful.

Regexp - Regular Expressions, a way of expressing pattern matching criteria. Completely obscure to read without years of tortuous study, but a key tool in the Perl coder's bag.

Parse::RecDescent - the Perl module used within Urchin to define their search language. Fancy pants stuff, really.

Non-root users - users of a Unix system without sufficient privileges to control the entire machine. This is usually everyone but the system administrator (who has 'root' access).

input_type - a term internal to the codebase in question: what form the original data came in.

skipHours/skipDay/syndication - many RSS feeds contain these optional data elements that denote when, and how often, the feed should be queried. It's polite to follow the instructions, but few people do. The bounders.

rdfs:seeAlso - an optional element within RDF that basically means 'Look! More data here!' and provides a URL to some more RDF. It's especially common in FOAF files.

Project Overview, or did they do what they said they would?

The Project offered ten criteria against which it could be judged. Obviously, the temptation here is to choose criteria that are easy to reach, even if the project as a whole does not actually work. In this case, however, I think NPG did a good job in spacing out their milestones along the entire product development journey. They're just too damn honest, I guess.

1. *"A relational database schema for caching news feed information. This would include fields for holding not only straightforward RSS metadata (titles, links and descriptions) but also Dublin Core and Open Content Syndication metadata, thus allowing the feeds generated by ROSA to be extended beyond the limits of RSS."*

They say: delivered.

I say: agreed. The schema, reproduced within their report, is nicely done, at least to my eyes. Admittedly, I usually shun database schema like the dogs they are - finding them very very dull - but even to my jaded and slightly fearful brain, this one is good. It's certainly complete enough for the purpose.

2. *"General-purpose software modules written in Perl for retrieving data from the following types of sources and inserting them into the database described above: RSS feeds, Relational databases, XML, Arbitrary structured and semi-structured text files."*

They say: delivered.

I say: not quite, but well enough. The Urchin::Import modules are very well done, and will certainly have a life of their own outside of the complete aggregator project. I can certainly see myself using Urchin::Import::Scrape, for example, to provide a single feed from a web page that does not ordinarily provide one for my own personal use.

However, Urchin::Import::XML does not appear to be complete, or, in fact, in existence at all bar some place holders. The authors do not deny this: Ben Lund tells me that it is planned for completion soon. No matter, in my opinion: grabbing RSS-ish data from arbitrary XML sources is a nice feature to have, but the set-up time for Urchin::Import::XML for the end user could just as easily be spent writing some XSLT code, or a nice bit of XML::Simple code, or some Xpath stuff, or whatever to do the same job, and then use the existing modules. That this would have to be done for each different XML source anyway is rather to the point here. Indeed, apart from all the angle brackets, you could do just as well with the Urchin::Import::Scrape module. There is, as they say, more than one way.

3. *"A relational database schema for storing a list of information sources, whether local or Internet-based, and providing data in any of the formats listed"*

They say: delivered.

I say: agreed. And they're cheating on this one. Because the data describing the information sources cannot really be separated from the data that those sources provide, the success of point 1 pretty much presupposes success here, and vice versa.

This is, to me, a very good sign. It shows that the designers appreciate the fundamental nature of the data they are working with: that the details of the source of a piece of data is 'just another triple' - and that once the triples have been added into the database, there is no hierarchy to the information. In fact, holding source information in a separate database would have been completely detrimental to the extensibility of the project. We'll get on to that in my next section.

4. *"A master control program for reading the list of sources from the above database and polling them for new content using the relevant software modules already described.*

They say: delivered.

I say: delivered. I'm a bit grudging with this one. Sure, urchinadm does trigger the polling of the sources in the database, and does allow for new feeds to be added, but that's it. I'd like to see a lot more functionality here. I'll go into this in the next section.

5. *"A database schema for holding feed aggregation information ... and a software module for creating aggregate feeds based on this information."*

and

6. "A database schema for holding feed filtering ... Also, a software module for applying these filters to single or aggregate feeds, individually or in series. Note that for advanced users, this filtering will allow the use of regular expressions as well as simple text matches."

They say: delivered.

I say: agreed. The schema we could go on about all day, but we already know it's good. The Urchin::OutputFeed module is very nice. I'll talk later about the coding style, but this is very clear stuff.

7. *"A database schema for holding arbitrary combinations of aggregators and filters to create custom news feeds, whether defined by the administrator or by users. Also, a software module for taking aggregated, filtered output and delivering it in a variety of formats. These would include RSS 0.9, 0.91, 0.92 and 1.0; Meerkat XML and user-defined XML formats; JavaScript; and arbitrary HTML or other user-defined text formats."*

They say: delivered.

I say: agreed. Nice. I'm a fan of these two modules, especially Urchin::OutputFeed::XSLT, whose $serve_flag option is especially nifty. I agree with the author's judgement that some work could be done to provide example XSL files, and I would like some HTML::Template examples as well, but all in all, it's just great. There's a lot of potential here for exporting FOAF, for example.

8. *"Documentation detailing the installation, configuration, administration and use of ROSA."*

They say: delivered.

I say: no. Well, I'm being very harsh here, given that no version 0.81 project is likely to be completely documented, but having realized that I've agreed with everything the authors have said so far I'm feeling mildly embarrassed and need to show some venom. So: I'd have liked to see more here with regards to customization of the outputs. But other than that, the other documentation is good, especially the installation guide. In the next section of this report I talk about coding style and the internal documentation, but suffice to say, it's all pretty good. Boring already, right?

9. *"Installation packages for the software in appropriate formats, such as 'make' and 'rpm'."*

They say: partially delivered.

I say: give yourself a break, lads. The authors judge themselves to have only partially delivered on this point as they did not include an RPM package with the more usual 'make' build. I'd have never expected an RPM package for this sort of thing. An OS X installer, perhaps, but an RPM? No. Why? Partly because of the heavy configuring needed (which in the case of OS X would be taken care of with some form of beautiful UI), but mostly because I would say that this project is just as likely to be used for its parts as its sum. Tying everything up in an RPM would have been quite annoying. Having the main perl module specifically denote all the system's prerequisites would suffice, I think - leave it all to CPAN to worry about.

10. *"The source code of all of the above, including Perl scripts (using Perl version 5.6) and SQL database schemas (in at least MySQL and Oracle versions.)"*

They say: partially delivered.

I say: agreed, but no worries. Again, here's the thing with the lack of an Oracle database. I'm not too bothered about this: MySQL is perfectly good, and perfectly cheap enough (i.e. free in many cases) to be a viable option. Oracle is good, but is expensive: I'm not convinced that they lack of Oracle specific code is a show-stopper in any way.

A note on the architecture choices and coding style

One of the more difficult decisions to make regarding a project such as this is which platform to code upon. The choices are not simple: code a desktop system for Windows, and you will likely have more users, but will be hampered by massively greater costs and a difficulty in sharing your work. Code a web-based system, and you will need a server infrastructure. Code for Mac, and it will look beautiful, and will gain an almost religious following amidst its users - but they will number only a handful.

Language is also an issue, as is database compatibility and so on. NPG's choice of the LAMP platform - Linux, Apache, MySQL, Perl - is very much a fashionable one, but none the worse for that. All of these systems are available for zero cost for personal use - and only MySQL requires a license for commercial use. This brings the cost of both development and deployment down to easily manageable levels.

The use of Perl is also to be applauded and encouraged in other JISC funded projects of this sort. Perl (and the other 'scripting' languages, such as Python and PHP) is an 'interpreted' language, rather than a compiled one. This means that the raw source code is run through an 'interpreter' every time you use it, as opposed to running it once through a compiler and then shipping the compiled code. Shipping only source code allows others to work on and learn from the product, but also makes the development process faster.

This is all to the good.

Kudos is also due to Martin Flack, the coder from NeoReality responsible for many of the Perl modules. It is a common complaint against many Open Source projects that the code is both obscure and poorly documented. This is not the case with the Urchin:: modules. The code immediately strikes one as having been written by someone who knows what they're doing - it's beautifully laid out, shows good style, and is properly documented within each module. This is totally irrelevant to the end-user, of course, but makes the jobs of other developers much easier. The Urchin::Import:: modules, for example, could have many uses independent of the entire Urchin system - that they are documented so well makes it all the more likely that this further value will be unleashed. It was somewhat of a joy to see.

Comments on "Features that Urchin 0.8 should have, but currently doesn't"

1. **Full HTTP compliance - Use the If-Modified-Since -Respect 304 not modified responses -** Essential. Many of the more highly trafficked sites are being hammered by RSS applications. Slashdot.org, for example, is so affected that it regularly bans IP addresses for too frequently requesting feeds. Full HTTP 1.1 compliance ensures that the the feed is only requested when it has changed, making Urchin more efficient and far more friendly.
2. **Richer scraping capabilities** - Not so essential in my opinion. The scraping capabilities currently on offer cover at least 90% of all possible cases. For 0.9 and 1.0, it would be a better use of resources to work on feature request number 3, coming up:
3. **Full reconstruction of all RDF data for RSS 1.0 outputs** - Yes. Yes. Yes. Essential. This is tremendously important for the future of the use of metadata in the fields Urchin is aimed at. Ideally, I should be able to create my own RDF namespace, include it in data, and then be able to query for it on Urchin without having told the authors of its existence. Note that being able to query for it does not necessarily mean that it is included in the resultant output - that would be ideal, but much harder to implement given the difficulties in creating RSS 1.0.
4. **Richer Urchin metadata in the RSS 1.0 output** - That would be nice.
5. **Use urchin:aggregate to give the names of Urchin aggregates that an item's parent channel are be part of** - Interesting idea, but I'm worried that two different Urchin installations might clash. For example, I could know that NPG is running an Urchin installation and scraping my feed. I want my feed to appear in their category 'Florentine Information Scientists', so I add an urchin:aggregate element to my feed. Then along comes another Urchin installation, wherein I wish to be categorized as 'English Émigrés' and not under Florentine Information Scientists at all - how do I mark this up? Thought needed here, and perhaps some work with other aggregator authors.
6. **Routines to stop an Urchin installation importing data from itself** - Agreed.
7. **Deal with encoded HTML in titles and descriptions** - My own personal view is that encoded HTML in titles or descriptions (as apart from content:encoded elements) is that the authors of such feeds will be first up against the wall come the revolution. Still, a few lines of regexp would fix this.
8. **Ability to import own Parse::RecDescent vocabulary** - Agreed. Sharing new vocabulary items with other users would be great.
9. **Port to Oracle** - Not worth the time, in my opinion.
10. **Installation guidelines for non-root users** - Agreed
11. **SaveData.pm save input_type** - Agreed
12. **urchinadm to skip non-RSS channels** - I'd amend this to 'skip non-RSS and non-RDF channels' and include the ability to parse RDF triples in non-RSS form.
13. **Generate a query/slow log** - Agreed
14. **Insert skipHours/skipDay/syndication metadata into RSS output** - Agreed. Although no one honours any of these values as yet, but there is little harm in becoming the first (both to create-as-an-aggregator, and to honour, this information)

Improvements I'd like to see, and what they would do.

1. Urchinadm to have richer functionality. It is *essential* that the administrators can quickly and easily remove feeds from the both the database of content and the database of feeds to be polled. Within this project, the two databases are one, so this complicates things a little (in that you might want to retain old data, but gather no more), but that's workaroundable. Either way, there must be much finer control of feed polling.
2. Logging of feed reponses. As an administrator I need to know which feeds are slow to respond, which are down, which haven't changed in a long time, which are no longer valid data, which are HTTP 1.0 and not 1.1, and so on. All of these are signs that I need to manually look at the data I'm fetching. Urchin needs these logs quite urgently if it is to be used on a bigger scale.
3. Inclusion of other forms of RDF, especially FOAF. Urchin's beautiful RDF compliant searching would be greatly enhanced by the inclusion of other forms of RDF. For the scientific publishing world of NPG and JISC, the inclusion of personal information, such as the FOAF vocabulary, would enable a whole new form of searching. "Find me all the articles written by people who work with X", for example, or "Show me every geneticist who has co-authored a paper with someone who works in biological weaponry". This would be truly great, and quite easy to do with the existing code base. Talking this over with the developers (and other developers of other packages), I appreciate there is an issue with trust here. I'm working on a book about this sort of issue, actually, so start saving your pennies.) Onward…
4. The following and spidering of rdfs:seeAlso elements within RDF data. This would greatly aid the administrator, and also encourage publishers to provide references and semantic links.

Conclusion.

In all, I think Urchin/Rosa is a very good start. To my mind the next thing to do is include querying on all the RDF data, including namespaces unknown to the system's authors. If this can be done, and I see no reason why not (given the support of JISC, for example), then we would have a very interesting product indeed.

Why? Well, I'm undoubtedly preaching to the choir here, but the ability to create and query chains of metadata triples within published journals would give researchers and scientists a very powerful tool to keep track of their fields, and other related research.

The main advantage is the ability to chain together queries. For example, a simple text search system, looking at keywords, can do very little other than Boolean queries:

genetics AND carrots

but with an RDF based system, you can make much more interesting searches:

"genetics AND carrots, published before January 1999, with an author who works in the UK, and who speaks French."

That sort of search would be tremendously difficult without an Urchin type system. Such a system would also allow features like:

"Automatically alert me when someone in Belgium publishes a paper on Biological Weaponry, that cites any paper written by Professor Doom as a resource."

That would be pretty trivial to do, as would:

"Tell me all the people who have co-authored a paper with a Nigerian biologist in the last year. And then tell me all of their friends who work in Denmark."

Itself a massively complex query without the benefit of RDF triples, but relatively easy with.

There are problems, of course: the metadata would need to be of a high quality, which is quite a rare thing on the net. It is not a rare thing, however, in the scientific publishing field, provided that the publisher knows it is needed. Providing metadata at the point of authorship is by far the simplest way of doing things. Happily, an application such as Urchin provides the market impetus to the publishers to make their metadata available, where there frankly wasn't one before. That the Urchin::Import modules are so good is a great help here too, as the publishers needn't necessarily migrate to richer systems.
In all, it's a good system, and well worth JISC's support.

Ben Hammersley, Florence, Italy, 9 Oct 2003 - ben@benhammersley.com