Annalist
("keeper of records")

Graham Klyne
e-Research Centre, University of Oxford

http://annalist.net

(A copy of the workshop paper, and an older version of these slides, are linked from the page at annalist.net)

Acknowledgements

OeRC: FAST project (digital music)
- Kevin Page, David Weigl, David De Roure, Terhi Nurmikko-Fuller, Alan Chamberlain, Steve Benford, *et al*

JISC RDS: CREAM project (active metadata)
- Cerys Willoughby, Simon Coles, Colin Bird, Iris Garrelfs, Athanasios Velios, Mike Mineiter, *et al*

W4Ever (research objects, workflows)
- Jun Zhao, Kevin Page, David De Roure, Stian Soiland-Reyes, Sean Bechhofer, Khalid Belhajjame, Carole Goble, Daniel Garijo, Oscar Corcho, *et al*

OeRC: Claros project (classical art)
- Donna Kurtz, Robert Kummer, Reinhard Förtsch, *et al*

Oxford (Zoology): IBRG (image bioinformatics)
- David Shotton, Jun Zhao, Alistair Miles, Helen White-Cooper, *et al*

Annalist has been in gestation over many years, and the ideas behind it have been informed by many past and present projects, and people I've worked with, a few of whom are listed here. I apologize in advance for any I've omitted.

Origins

https://xkcd.com/974/

An early seed was planted over 15 years ago, at the conference dinner for the 1999 Semantic Web symposium in Stanford. I happened to be sitting at the same table as Prof Jim Hendler, who commented to the effect that the semantic web needed more RDF data on the web. At that time, DARPA funded projects were working to publish large scale datasets like the CIA World Factbook as RDF.

My focus has been on personal and small-scale data, and its attendant diversity. After a couple of earlier failed attempts, now with better tools, and more modest goals, I have developed Annalist as a working tool. I just hope that nobody has been waiting 15 years for me to pass the salt.

Goal

To make it quick and easy for individuals and small teams to create and share linked data on the web

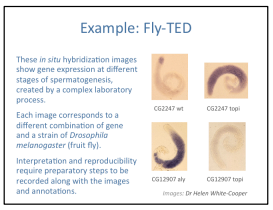
My overall goal in creating Annalist has been to make it quick and easy to create and share linked data on the web.

Working on a range of research projects over the past 10 years or so, from bioinformatics to classical arts, I've found the effort to create robust tools for linked data to be an impediment to deploying applications for research users. I feel this has put use of linked data beyond the reach of many small research groups who might benefit from facilitated exchange of data with a wider community of peers.

For myself, I've wanted to make it easy to prototype application ideas using RDF.

Motivation and Requirements
Annalist in use
Design and implementation
Status and future work
Discussion

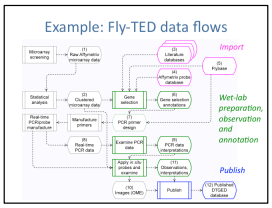
I'll start with a discussion of motivation and requirements, then continue with some examples of Annalist in use, technical design and project status.



One project that has motivated Annalist development is Fly-TED...

I was working with a zoology research team to create a database of gene expression images, showing activity of different genes at different development stages of fruit fly spermatazoa.

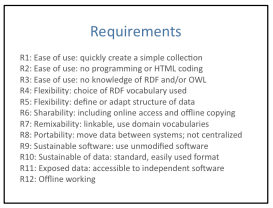
This was intended to become a community resource for researchers, augmenting existing resources on the web.



The preparation, capture and annotation of these images involved many kinds of related data, including published literature and data sources, statistical analysis, wet-lab work and microscopic observation.

A public database was duly created, and reported in the Nucleic Acids Research journal.

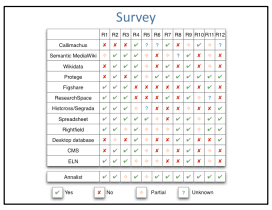
Unfortunately, this database was lost in an infrastructure failure. Although the original raw data still existed, lack of effort to rebuild the customized platform meant the public web presence was lost.



These Fly-TED sustainability travails have suggested several requirements that inform the design of Annalist as a tool to support sharing of research data.

I don't intend to discuss them all individually (they are covered in the paper), but the general thrust is towards ease-of-use, flexibility, and sustainability of data.

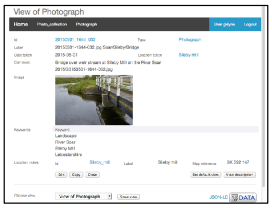
(pause)



A survey of existing tools for creating and sharing linked data indicates that all the identified requirements are addressed by some tool or other, but not all together in any single tool that we could identify.

Motivation and Requirements
Annalist in use
Design and implementation
Status and future work
Discussion

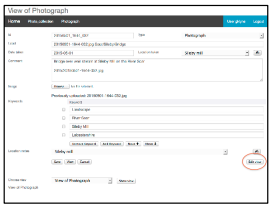
So what does Annalist look like in use?



The example screenshots I'll show are based on a tutorial that uses Annalist to create a catalogue of photographs and associated metadata.

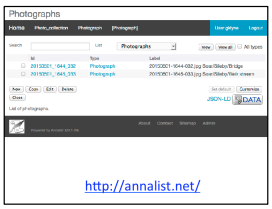
This slide shows a presentation of data describing a photograph.

Note the combination of free-form textual description, structured annotation, image media, and references to other entities.



This screenshot shows the same data, presented as a form used to create and edit the data.

The content of these displays is customizable within Annalist so that it can edit and present the data at hand (note the "Edit view" button).



http://annalist.net/

And this is a view that shows a list of photographs for which there is data in the containing collection.

These examples show the three main kinds of display provided by Annalist:
- a data record, for viewing entity data
- an editing form, used for creating and updating entity data
- an entity list, used for browsing and navigating the contents of a collection

These example data, and the Annalist tutorial they come from, are linked from the page at annalist.net.

I now turn to the technical design and implementation of Annalist

Design philosophy

- Data first, structure later
- Minimize impediments to data entry
- Annalist as a part in a wider linked data ecosystem
- JSON-LD as "view source" for linked data

The message that data can tell is not always clear at the outset, but may emerge though the process of observation and collection.

The philosophy adopted favours flexible data entry over conformance to formal schemas and ontologies.

But the design is intended to allow for formal structures to be crystallized (using new or existing ontologies) as data is accumulated.

Crucially, Annalist maintains data in a form accessible to other applications, rather than trying to be a one stop shop for every possible need.

Technical implementation

Web server application
Deployable in desktop, workgroup or cloud
- (Docker too)

Main logic is server-based
- Javascript for responsive interface

Data stored as JSON-LD files
- Can be published directly by HTTP server

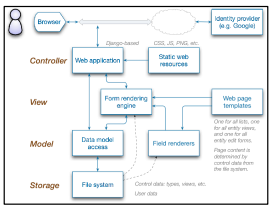
Customizable form generator
- Definition as JSON-LD, also managed by Annalist
- One definition used for view and edit forms

Annalist is implemented as a web server application. It can be deployed in different environments, from desktop to cloud.

There are also Docker containers available from DockerHub.

Application logic is implemented server-side, with some (optional) browser-based Javascript used for responsive user interface and enhanced interactions.

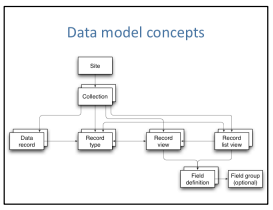
(move on...)



The Annalist server is implemented in Python using the Django web application framework.

It makes minimal use of Django's database facilities, but instead stores data as JSON-LD files – currently in the server file system, but with future plans for using external HTTP servers.

At the heart of the Annalist server is a customizable form generator, used to display, edit and navigate data, by combining editable view definitions with user data.

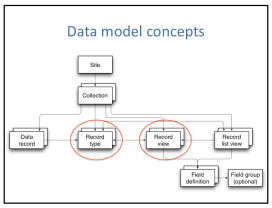


The data model is built around these conceptual elements:

A ***Site***, which generally corresponds to a single server.

A *site* contains data ***Collections***. Data and controlling metadata for a collection are stored in a single container directory, which may be copied or moved between systems.

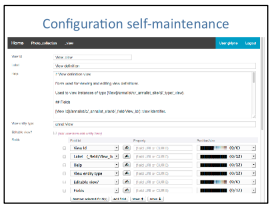
Within a collection, entities are described by ***Data records***, whose content and structure are controlled by ***Record types***, ***Record Views***, ***Record list views*** and ***Field definitions***, which are themselves data records.



Record types are used to organize entities, and do not of themselves define the content or structure of data records.

The *content* and structure is determined by the views that are used to create and edit the data records.

The association between types and views is quite loose:
- a single view may be used with several different types, or
- different views may be used with a single type.



The internal Annalist definitions for types, views, etc., are themselves all viewable (and editable) as Annalist data records.

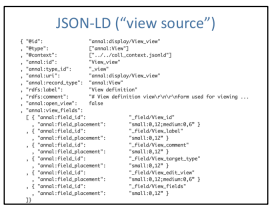
This is illustrated by this definition of the view that is used for displaying and editing view definitions.

So what you see here is defining its own presentation.

A close look reveals that the fields displayed in the view ("View Id", "Label", etc.) are reflected in the list of fields in the view definition.

And...

JSON-LD ("view source")



```
{ "@id":              "annal:display/View_view"
, "@type":            ["annal:View"]
, "@context":         ["../../coll_context.jsonld"]
, "annal:id":         "View_view"
, "annal:type_id":    "_view"
, "annal:url":        "annal:display/View_view"
, "annal:record_type": "annal:View"
, "rdfs:label":       "View definition"
, "rdfs:comment":     "A View definition view\r\n\r\nform used for viewing ...
, "annal:open_view":  false
, "annal:view_fields":
  [ { "annal:field_id":          "_field/View_id"
    , "annal:field_placement":   "small:0,12;medium:0,6" }
  , { "annal:field_id":          "_field/View_label"
    , "annal:field_placement":   "small:0,12" }
  , { "annal:field_id":          "_field/View_comment"
    , "annal:field_placement":   "small:0,12" }
  , { "annal:field_id":          "_field/View_target_type"
    , "annal:field_placement":   "small:0,12" }
  , { "annal:field_id":          "_field/View_edit_view"
    , "annal:field_placement":   "small:0,12;medium:0,6" }
  , { "annal:field_id":          "_field/View_fields"
    , "annal:field_placement":   "small:0,12" }
  ]
}
```

[And...]

... this is the JSON-LD that is stored for this view definition.

[skip...]

My experience as a developer has been that JSON-LD sits in a sweet spot between machine processable and human readable and editable data, which has been a key benefit for bootstrapping the Annalist system definitions.

It offers a good "impedance match" with programming language data types for manipulating data (arrays, dictionaries, etc.), and avoids some of uglier aspects of working directly with the RDF model (e.g. dealing with lists).

Motivation and Requirements
Annalist in use
Design and implementation
Status and future work
Discussion

The status of Annalist is that is a work in progress.

So what has been achieved to date?

Progress to date

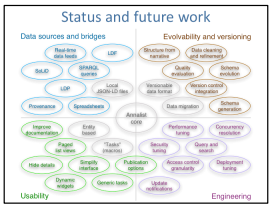
1. A viable tool to create and share linked data
2. Flexible to deal with diverse applications
3. Robust
   - even as a work-in-progress, I have never lost application data due to an Annalist software fault
4. At least approachable for users who are not familiar with RDF

I think I can claim that:
• Annalist is a viable tool for creating and sharing linked data,
• it is sufficiently flexible to be usable in diverse applications, and
• it is robust enough to be trusted with users' data.

Further, experience to date indicates that Annalist is at least approachable for users who are not steeped in RDF lore.

But...

Status and future work



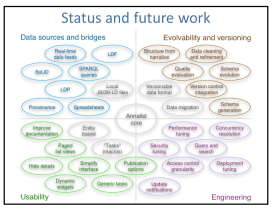
[But...]

... there remains plenty to do.

The gray elements indicate work that is completed or in hand.

The coloured areas suggest future work...

I don't propose to describe all of these individually, but they roughly divide into these areas...

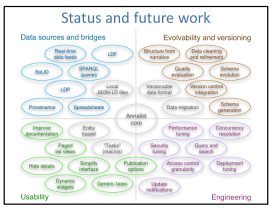


[Data sources and bridges]

In the blue corner, we have... ability to work with alternative storage and data from a variety of sources. (e.g. working with existing data provided as spreadsheets).

[Usability]

In the green corner... work to make Annalist easier to use, especially in the area of defining structure and presentation of data



[Evolvability and versioning]

In the brown corner: improving capabilities for dealing with evolving data and schemas

[Engineering]

And finally... engineering improvements to performance, security and other capabilities

Given that Annalist is intended to operate as a part of a suite of tools for handling linked data, not all of these functions should necessarily be part of Annalist itself.

For example, publication options and quality evaluation are features that I feel can be addressed by generic linked data tools used in conjunction with Annalist.

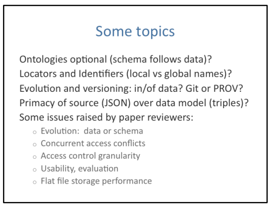
In addition to the various technical issues mentioned, there is the goal of transition to a community project.

Annalist has, from the outset, been an open development. All source code, documentation and design notes have been maintained in a public repository, but...

One of my hopes for the coming year is to start the transition from an "open source" project with a single developer to a wider community project.

This concludes what was intended to be the "linear" part of my presentation of Annalist.

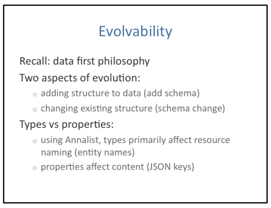
and I hope that there remains time for discussion of issues arising.



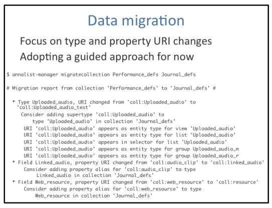
These are some topics raised in discussions with colleagues, and by the paper reviewers for this workshop.

I'd be happy to engage in discussion of these or any other questions.

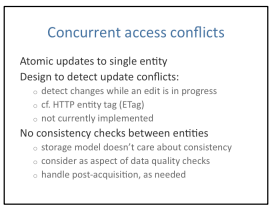
(and if no questions, I can plough on???)



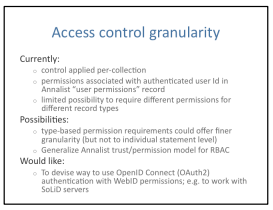
(Spare slide to support possible discussion)



(Spare slide to support possible discussion)



(Spare slide to support possible discussion)



(Spare slide to support possible discussion)

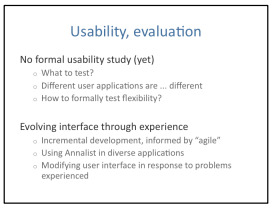
Access control notes

Currently permission records are stored as Annalist linked data records, so in principle could be referred to externally served resources.

I did look at UMA ~2 years ago, but at the time its requirements conflicted with Annalist goals.

Issues:

- granularity - currently per collection, with limited per-type (e.g. User Permissions access requires ADMIN)
- role-based - currently directly associated with authenticated user id, no provision for roles
- trust model - currently trust local user permission records
- SoLiD servers - how to combine OpenID connect with WebID?



(Spare slide to support possible discussion)

I'm not sure that it's a valid alternative to formal usability studies, but my approach has been to create minimum sufficient functionality for a task, and then to build on that incrementally. In some sense, usability evaluation is built in to the development process (but not formally recorded).

The "Chemistry personas" collection was developed in Annalist by a researcher not intimately familiar with RDF, but with detailed experience of experimental procedures in chemistry.
See: http://cream.annalist.net/annalist/c/Chemistry_Personas/d/type_Plan/plan_Synth_01/

I have noticed that the interface sometimes lends itself to appropriation – I've had required features on my TODO list, only to realize later on that I can implement them using existing capabilities (e.g. unified image link/import/upload fields.)

Discuss... ?