Annalist
("keeper of records")

Graham Klyne

e-Research Centre, University of Oxford

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 Mineter, *et al*

Wf4Ever (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 is a project that 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. There are surely many others whose thoughts have permeated the Annalist designs, for whose omission I must apologize in advance.
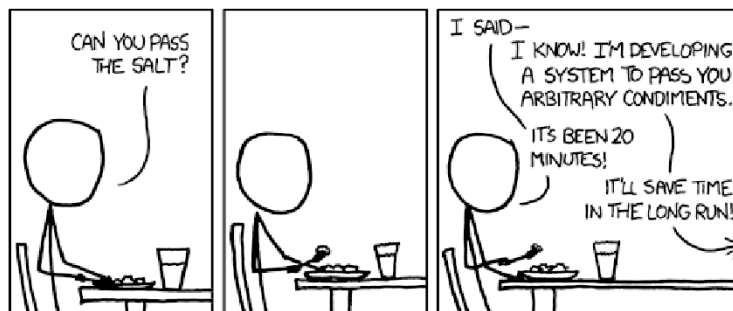
One of the early seeds for the project that became Annalist was planted over 15 years ago, at the conference dinner for the 1999 Semantic Web symposium in Stanford. The second RDF working group was active at about this time. I happened to be sitting at the same table as Prof Jim Hendler, who made a comment to the effect that what was really needed to help the semantic web gain traction was to have more RDF data on the web. At that time, DARPA funded projects were working to put large scale datasets like the CIA World Factbook on the web as RDF. In contrast with this, my focus has been on personal and small-scale data, and dealing with its attendant diversity.

Since then, I've made a couple of failed attempts to implement a generic web-based system for small-scale creation of linked data. But more recently, with the aid of better tools, and starting with more modest goals, I have managed to develop Annalist as a working tool.

I just hope that nobody's been waiting over 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.
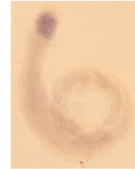
Working on a range of research projects over the past 10 years or so, covering bioinformatics to classical arts, I've found the level of effort needed to create robust tools for working with linked data to be an impediment to deploying applications for research users. I feel this lack of tools has put 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.

I've also wanted to create something that would it easy for me to prototype ideas for applications using RDF.
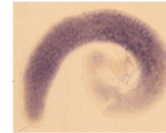
And I am interested in the idea that tools for manipulating RDF should themselves use RDF, in a kind of self-maintaining fashion.
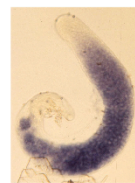
Here is an example of the kind of project that has informed requirements for Annalist.

I was working in Oxford's Zoology department with a genomics research team, who were creating a database of gene expression images that showed the activity of different genes at different stages of sperm development in fruit flies.

This database was intended to become a community resource for researchers, to sit alongside several other public resources on the web.

Example: Fly-TED data flows
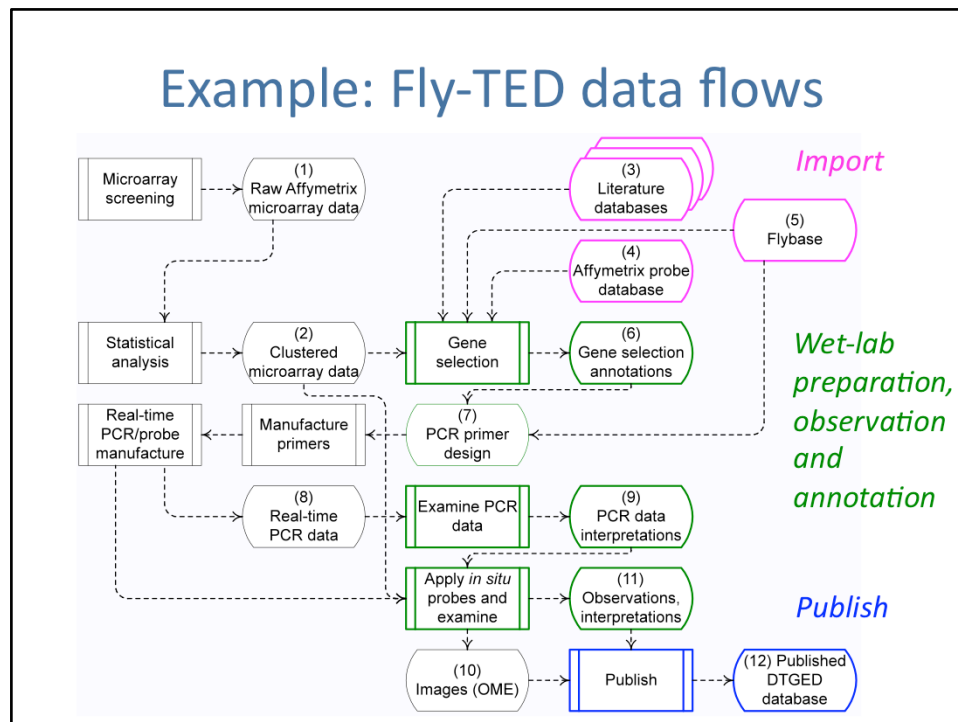
The diversity of data and activities that went into creating these gene expression images is illustrated here.

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

This work resulted in a published database, and was reported in the Nucleic Acids Research journal. Unfortunately, the database became inoperable due to a virtual machine infrastructure failure. Although the original raw data still existed, lack of available effort to rebuild the customized platform has meant the published data are no longer available on the web.

These sustainability travails with Fly-TED gave rise to several requirements that have informed the design of Annalist.

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

(pause)

Survey

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Callimachus	✘	✘	✘	✔	?	?	✔	✘	◈	✔	◈	?
Semantic MediaWiki	◈	✔	✔	✔	◈	✘	◈	?	✔	✘	◈	✘
Wikidata	✘	✔	✔	✔	◈	✘	✔	✘	✔	✘	◈	✘
Protege	✘	✔	✘	✔	◈	✔	✔	✔	✔	✔	✔	✔
Figshare	✔	✔	✔	✘	✘	✘	✘	✘	✔	✘	✔	✘
ResearchSpace	✔	✔	✔	✘	✘	✘	✔	✘	✔	◈	?	✘
Histcross/Segrada	✔	✔	✔	◈	?	✘	✘	✘	◈	✘	✘	✔
Spreadsheet	✔	✔	✔	✔	✔	◈	✘	✔	✔	◈	✔	✔
Rightfield	✔	✔	◈	✔	◈	◈	◈	✔	✔	◈	✔	✔
Desktop database	✘	◈	✘	✔	◈	✘	✘	✔	✔	✘	✘	✔
CMS	✘	✔	✔	✔	◈	✘	◈	✘	✔	✘	◈	✘
ELN	✔	✔	✔	◈	◈	◈	✘	✘	✔	✘	◈	✘
Annalist	✔	✔	◈	✔	◈	✔	✔	✔	✔	✔	✔	✔

✔ Yes	✘ No	◈ Partial	? Unknown

A survey of existing tools for creating and sharing linked data indicated that the identified requirements are all addressed by some tool or other, but not all together in any single tool that we could identify.
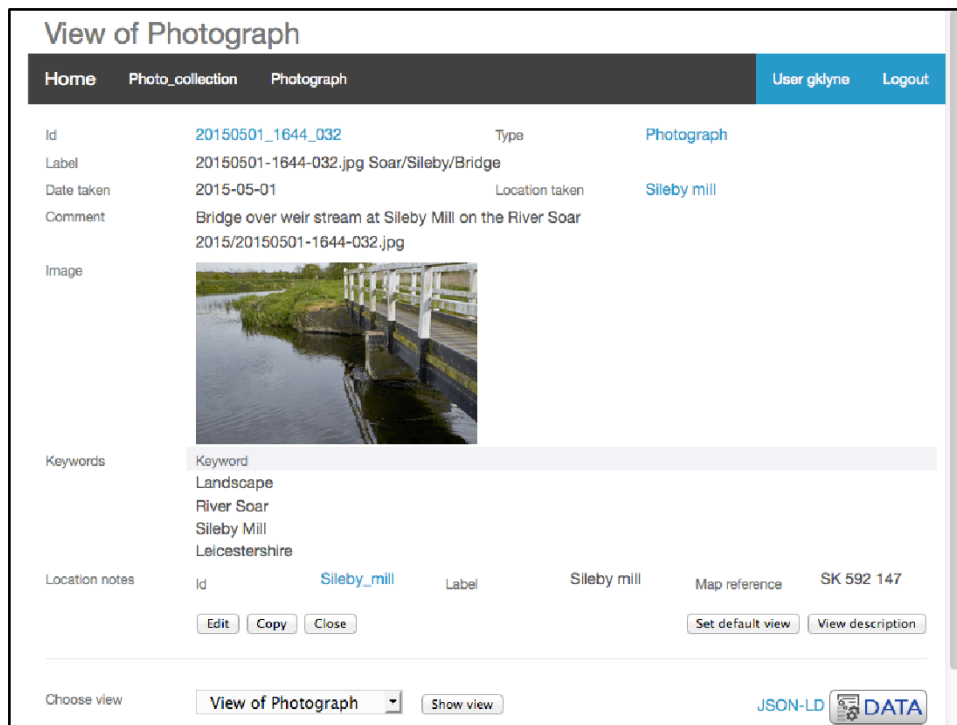
So what does Annalist look like in use?

This, and the following screenshots are taken from the Annalist tutorial, which uses Annalist to create a catalogue of photographs and associated metadata.

It shows a view of data describing a photograph, as presented by Annalist.

Note how the presentation combines free-form textual description, structured annotation, image media, and references to other entities.

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

Data:

http://demo.annalist.net/annalist/c/Photo_collection/

Tutorial:

http://annalist.net/documents/tutorial/annalist-tutorial.html

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

These three screenshots exemplify three main kinds of display provided by Annalist:
  1. data display, used for viewing entity data
  2. editing form, used for creating and updating entity data
  3. entity list, used for browsing and navigating the contents of a collection

The example data and the Annalist tutorial can be found at the links shown.

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
- Open source, open development

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

Turning now to the design of Annalist...

The philosophy adopted in the design of Annalist tends to favour flexible data entry over conformance to formal schema and ontologies in the acquisition of data.

But the design is intended to allow for formal structures to be crystallized (using new or existing ontologies)
as the underlying message that data conveys becomes more apparent through its acquisition.

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

Technical design

Web server application

Data stored as JSON-LD files

- Can be published by any HTTP server

Customizable form generator

- Definition as JSON-LD, also managed by Annalist
- One definition used for view and edit forms
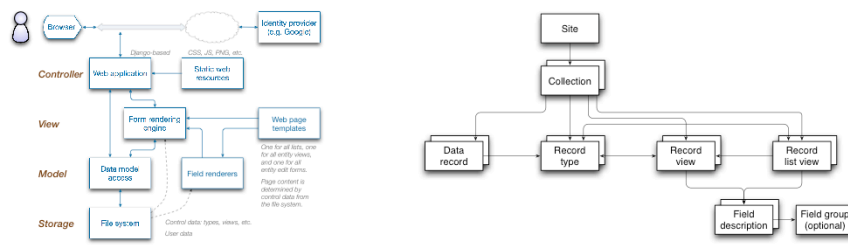
Annalist is implemented as a web server application. It can be deployed in different environments (desktop, local workgroup server or data centre), and has been used in each of these deployments.

There are also Docker containers available from DockerHub.

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

(move on...)
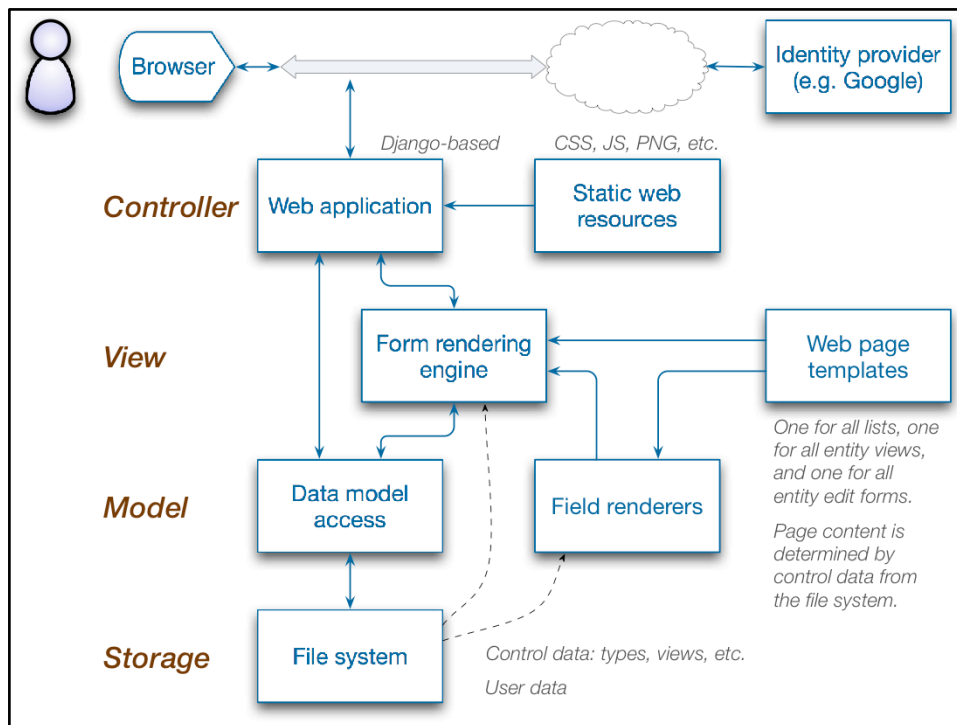
The Annalist server is implemented in Python using the Django web application framework.

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

At its heart is a customizable form generator, used to display data views, editing forms and lists of entities, combining editable definitions and user data records.

Data model concepts
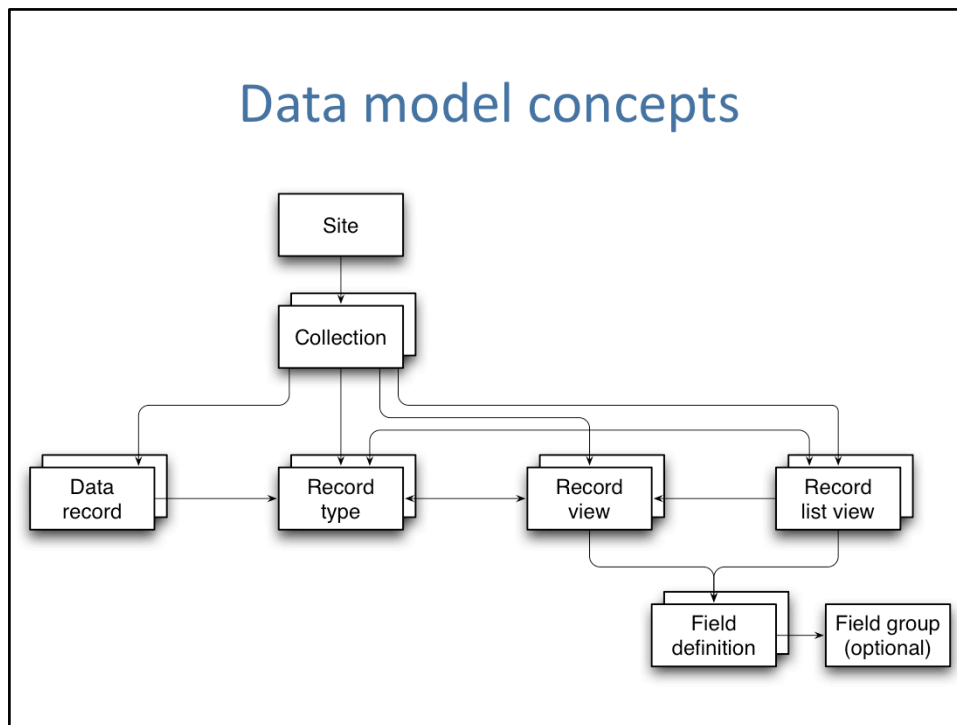
The data model is built around a number of conceptual elements:

Each Annalist deployment is associated with a **Site**, which generally corresponds to a single server machine (or container).

An Annalist site may contain any number of data **Collections**, which may be free-standing, or may use definitions from other collections in the same site. Definitions and data associated with a collection are stored in a single container directory, which may be copied or moved across 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.
(Some Field definitions refer also to a **Field group**.)

Record types are used mainly as a way of organizing entities into broad categories, and do not of themselves define the content the corresponding data records.

Rather, the *content* and internal structure of data records is defined by the view or views that are used to create and/or edit them.

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

Configuration self-maintenance

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

Just for fun, this screenshot shows the 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 that are referenced by the view definition.

And...

JSON-LD as "view source" for data

```
{ "@id":              "annal:display/View_view"
, "@type":            ["annal:View"]
, "@context":         ["../../coll_context.jsonld"]
, "annal:id":         "View_view"
, "annal:type_id":    "_view"
, "annal:uri":        "annal:display/View_view"
, "annal:record_type": "annal:View"
, "rdfs:label":       "View definition"
, "rdfs:comment":     "# 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 the view definition shown previously.

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 when bootstrapping the Annalist system definitions.

It also avoids some of uglier aspects of working directly with the RDF model (e.g. dealing with lists).

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

Annalist is a continuing project ... what has been achieved so far?

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

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

But...

Status and future work
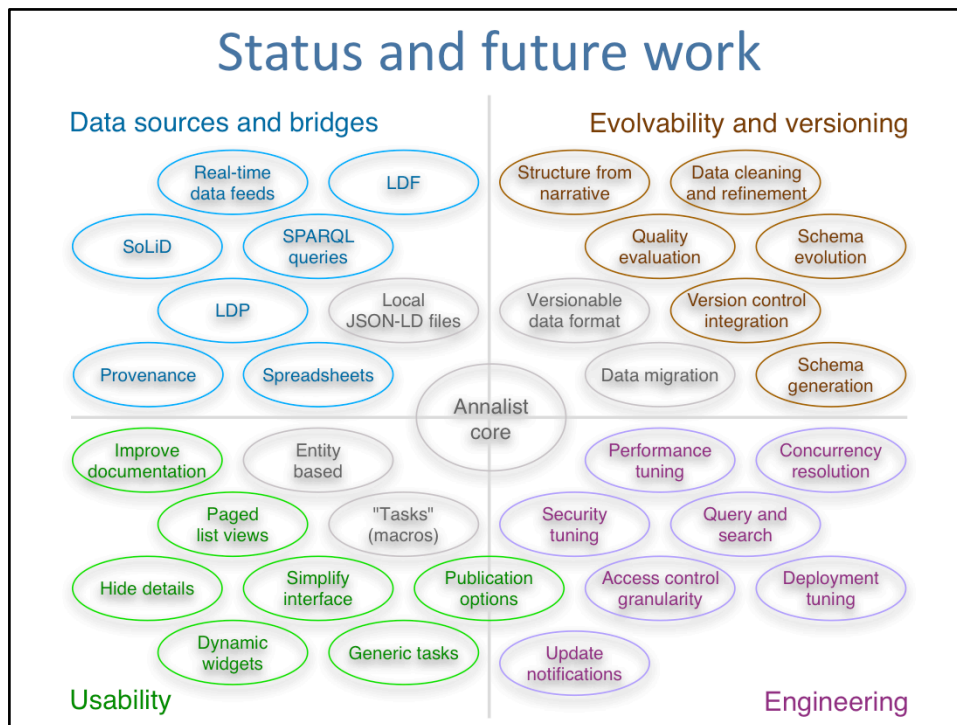
[But...]

... there remains plenty to do, which is the main message of this rather busy diagram.

I don't propose to describe all of these, but they roughly divide into the areas shown:

- [Data sources and bridges]
  In the blue corner, we have... ability to work with alternative storage and data from a variety of sources.
- [Usability]
  In the green corner... making Annalist easier to use, especially in the area of defining data structure and presentation
- [Evolvability and versioning]
  In the brown corner: improving capabilities for dealing with evolving data and schemas
- [Engineering]
  And finally... improvements to technical performance and capabilities

A few of these topics (the ones coloured grey) are already being addressed.

Also, I would point out: 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 best be addressed by generic linked data tools that can be used in conjunction with Annalist.

Transition to community project

A public repository is a start:
- https://github.com/gklyne/annalist
- MIT licence

But there remain many things to do...
- Governance
- Supporting documentation
- Engage other developers
- Integration with complementary systems
- Application data definition "libraries"

In addition to the various technical issues mentioned, there is transition to a community project that is yet to be achieved.

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

Assuming that there will be others who may find Annalist useful, one of my goals for the coming year is to start the transition from an "open source" project with a single developer to a community project.

Discussion

Flexible data vs ontology conformance?

Locators and Identifiers?

Some issues raised by reviewers:

- o Evolution of data and schema
- o Concurrent access conflicts
- o Access control granularity
- o Usability, evaluation
- o Flat file storage performance

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

I hope that there remains time for discussion of some issues arising from this work.

To prompt such discussion, this slide highlights some of the points raised by the workshop paper reviewers, and I'd be happy to enter into discussion of these or any other questions.

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

Evolvability

Recall: data first philosophy

Two aspects of evolution:

- adding structure to data (add schema)
- changing existing structure (schema change)

Types vs properties:

- using Annalist, types primarily affect resource naming (entity names)
- properties affect content (JSON keys)

Data migration

Focus on type and property URI changes
Adopting a guided approach for now

```
$ annalist-manager migratecollection Performance_defs Journal_defs

# Migration report from collection 'Performance_defs' to 'Journal_defs' #

 * Type Uploaded_audio, URI changed from 'coll:Uploaded_audio' to
   'coll:Uploaded_audio_test'
    Consider adding supertype 'coll:Uploaded_audio' to
        type 'Uploaded_audio' in collection 'Journal_defs'
     URI 'coll:Uploaded_audio' appears as entity type for view 'Uploaded_audio'
     URI 'coll:Uploaded_audio' appears as entity type for list 'Uploaded_audio'
     URI 'coll:Uploaded_audio' appears in selector for list 'Uploaded_audio'
     URI 'coll:Uploaded_audio' appears as entity type for group Uploaded_audio_m
     URI 'coll:Uploaded_audio' appears as entity type for group Uploaded_audio_r
 * Field Linked_audio, property URI changed from 'coll:audio_clip' to 'coll:linked_audio'
      Consider adding property alias for 'coll:audio_clip' to type
          Linked_audio in collection 'Journal_defs'
 * Field Web_resource, property URI changed from 'coll:web_resource' to 'coll:resource'
      Consider adding property alias for 'coll:web_resource' to type
          Web_resource in collection 'Journal_defs'
```

Concurrent access conflicts

Atomic updates to single entity

Design to detect update conflicts:
- detect changes while an edit is in progress
- cf. HTTP entity tag (ETag)
- not currently implemented

No consistency checks between entities
- storage model doesn't care about consistency
- consider as aspect of data quality checks
- handle post-acquisition, as needed

Access control granularity

Currently:
- control applied per-collection
- permissions associated with authenticated user Id in Annalist "user permissions" record
- limited possibility to require different permissions for different record types

Possibilities:
- type-based permission requirements could offer finer granularity (but not to individual statement level)
- Generalize Annalist trust/permission model for RBAC

Would like:
- To devise way to use OpenID Connect (OAuth2) authentication with WebID permissions; e.g. to work with SoLiD servers

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?

Usability, evaluation

No formal usability study (yet)
- What to test?
- Different user applications are ... different
- How to formally test flexibility?

Evolving interface through experience
- Incremental development, informed by "agile"
- Using Annalist in diverse applications
- Modifying user interface in response to problems experienced

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... ?

Links

Paper

https://github.com/gklyne/annalist/blob/develop/
documents/publications/LDOW2016-paper/
Annalist-paper-ACMSIG.pdf

Slides

https://github.com/gklyne/annalist/blob/develop/
documents/publications/LDOW2016-paper/
annalist-presentation-ldow2016.pptx, .pdf

Demo site

http://demo.annalist.net/

Tutorial

http://annalist.net/documents/tutorial/annalist-tutorial.html

Software

https://github.com/gklyne/annalist/