The OWL of Biomedical Investigations

Mélanie Courtot[1], William Bug[2], Frank Gibson[3], Allyson L. Lister[4], James Malone[5], Daniel Schober[5], Ryan R. Brinkman[1] and Alan Ruttenberg[6]

[1]Terry Fox Laboratory, British Columbia Cancer Research Center, Vancouver, BC, Canada
[2]National Center for Microscopy Imaging Research, UCSD, CA, USA
[3]School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
[4]CISBAN and School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
[5]The European Bioinformatics Institute, Cambridge, CB101SD, UK
[6]Science Commons, Cambridge, MA, USA
mcourtot@bccrc.ca, alanruttenberg@gmail.com

Abstract The Ontology for Biomedical Investigations (OBI), written in OWL DL, brings together a large consortium seeking to provide a cross-domain, shared framework for representing investigations in the biological and biomedical sciences. In this paper we report our experiences and describe our development process as it pertains to OWL, which includes a number of elements that might inform tool developers as well as suggest general development patterns. Finally, we review where improvements to OWL and OWL related tools might be beneficial.

1 Introduction

The Ontology for Biomedical Investigations (OBI) Consortium[1] is developing an ontology for the description of biological and clinical investigations, written in OWL DL. The OBI Consortium is a member of the OBO Foundry [1], a collaborative of developers of science-based ontologies who are establishing a set of principles for ontology development with the goal of creating a suite of interoperable reference ontologies in the biomedical domain.

OBI uses the Basic Formal Ontology[2] (BFO) as its upper-level ontology. Upper ontologies such as BFO aid interoperability by providing a higher-level framework that functions as a common structural and intellectual scaffold by way of which ontologies can share a common understanding of those aspects of the world that are independent of any particular application domain [2].

In order to enable development of OBI as a large collaborative project, a strategy was required that would allow concurrent editing, distributed development, version control, offline development, use of different tools and editors, and script-based augmentation of the ontology content. A review of the existing collaborative ontology development tools failed to identify a single application that met OBI's requirements. As a result we chose to rely on a small group of tools, augmented with a structured

[1] http://purl.obofoundry.org/obo/obi
[2] http://ifomis.org/bfo/

mechanism for development. For example, we chose Subversion[1] to address the need for version control, distributed and offline development, as well as logging history for change management.

To enable OBI development to proceed efficiently, the ontology structure was separated into 10 sections (biomaterial, data transformation, digital entity, function, instrument, plan, protocol application, qualities, role and relations) called branches, for concurrent development by different groups, with each group working more or less independently. Each branch is maintained in a separate OWL file, and contains closely related terms and definitions. For example, the *instrument branch* covers relevant kinds of instrumentation and parts of devices.

Although this concurrent branch development strategy proved effective, it also presented some challenges preparing OBI for distribution. Editing several OWL files concurrently and in a distributed manner can lead to non-unique class identifier assignment and conflicts within the ontology. Our set-up also required curators to be reasonably familiar with an ontology editor in order to be able to view the required multiple OWL files in harmony.

One of the fundamental principles of the OBO Foundry is to reuse, where sensible, existing ontology resources. While OWL provides a mechanism to import ontologies (*owl:imports*), this mechanism was not always suitable for OBI. Firstly, current editing tools are not effective for working with very large ontologies such as the NCBI Taxonomy [3] or the Foundational Model of Anatomy [4], making direct OWL imports of such ontologies, as a whole, impractical for day-to-day development. Secondly, other ontologies used by OBI are under active development and may not be aligned with OBI's design (e.g., not yet using BFO as an upper ontology, or not yet using OWL DL). Importing such ontologies as a whole could lead to inconsistencies or unintended inferences. An alternative to the OWL built-in import mechanism would be to copy only parts of the external ontology into *obi.owl*. Software that extracts a "module" [8] is in early development and we plan to try such strategies in the future.

In order to distribute up-to-date information, we need a mechanism that provides for automatic updating of the external information to be used regularly such as before OBI releases. We are aware of and accept that by copying only parts of an ontology there is the risk that inferences drawn may be incomplete or incorrect.

Our solution is a mechanism we call the Minimal Information to Represent an External Ontology Term (MIREOT). MIREOT provides guidelines on importing selected terms without the overhead of importing the complete ontology from which the terms derive.

2 OBI development practices

2.1 Minimal Information to Reference External Ontology Terms (MIREOT)

In deciding upon a minimum unit of import, our first step was to consider the practices of other ontologies. The practice of the Gene Ontology (GO) [5] is that the intended meanings of classes remain stable. Even when the ontology is repaired or

[1] http://subversion.tigris.org/

reorganized, the effects of such changes do not change the intended meaning of terms. Rather the changes are towards more carefully expressing the logical relations between them. If the meaning really changes, terms are deprecated [6]. Since a term is considered stable, in isolation from the rest of the ontology, we consider terms (i.e. classes) a basic unit of import.

The minimal amount of information needed to reference an external class is the ontology URI and the term's URI. Generally, these items remain stable and can be used to unambiguously reference the external class from within OBI. The minimal amount of information to integrate this class into OBI is the position in the OBI hierarchy, i.e., what OBI class the imported class is a subclass of. We store this minimal information set in a separate file called *external.owl*. We also want to provide extra information about our imported classes such as their label and definition. We map these to the corresponding OBI annotation properties. For example, in the current OWL rendering of OBO files, definitions are individuals and the *rdfs:label* of that individual records the text of the definition. That label becomes the value of OBI's *definition* property.

Such information is prone to change as the source ontologies evolve, and to allow for easy updates we store it in a separate file, called *externalDerived.owl*, created from *external.owl* and rebuilt via a script as needed.

When deciding to import an external term we review the textual definition and, if needed, talk with its editor. As we are importing from OBO Foundry ontologies we have a community process for monitoring change, a shared understanding of the basics of our domain, and the intention to eventually share the same upper-level ontology. Therefore, we expect that terms will be deprecated if there is a significant change in meaning, and expect to adjust our import of terms as the other ontologies start enhancing their logical definitions.

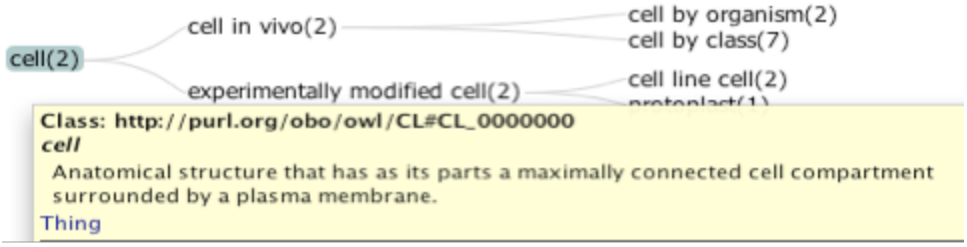

Figure 1 The to-be-imported *cell* term, as viewed in its original context in the Cell Type ontology class tree. The *cell by organism* and *cell by class* are examples of those we would prefer to not import into OBI.

As an example, we recently replaced the OBI class *cell* with that from the OBO Foundry Cell Type (CL) ontology [7] (Figure 1). Following the MIREOT guidelines, we identify the minimum information required in this case:

- the URI of the term *cell*: http://purl.org/obo/owl/CL#CL_0000000
- the ontology from which the term is imported: http://purl.org/obo/owl/CL
- the position of *cell* in the OBI hierarchy: as a subclass of *Anatomical entity*[1]

[1] This term will itself likely be replaced by the corresponding CARO term.

A set of templated SPARQL queries[1], taken together with *external.owl*, specifies which extra information about the class to gather, such as the definition and preferred label, and these are retrieved using queries against the Neurocommons[2] SPARQL[3] endpoint[4][5]. A script iterates through the minimal information stored in *external.owl*, substituting IDs into the appropriate SPARQL construct queries and gathers the combined results to create the supplementary information in *externalDerived.owl* file.

The second example presents a slightly more complicated challenge. OBI currently uses the NCBI taxonomy for its species terms. When importing those we decided that the information about the term itself was not sufficient on its own: for example if we want to import the term *Mus musculus*, we also want to import its rank information – genus, kingdom, phylum, etc. In this case the SPARQL query retrieves all direct superclasses up to one of a set of top-level classes in the taxonomy.

Correct inference using the external classes is only guaranteed if the full ontologies are imported. We expect to provide an option in the OBI distribution that replaces external.owl with a set of import statements generated by extracting the ontology URIs mentioned in external.owl. Other options are possible, for instance having the script that generates additional information or using software that extracts a module of the external ontology. However, in the case of the NCBI taxonomy we are guaranteed to obtain the desired module, as it currently uses only subclass relations.

A consideration using this approach is the status of OBI assertions made on external terms. In adding axioms such as the subclass axiom when placing the external term into OBI, the aim is to only assert true statements about the terms. We anticipate that some of these statements may migrate to the source ontologies at some point in the future, a fruit of the collaborative nature of OBO Foundry ontology development.

2.2 Releasing OBI

We required a mechanism that would allow the release of a public version of OBI[6] on a monthly basis. Such a process allows users to acquire a traceable version of the ontology that is essentially static and act as a stable reference point for usage, much like the process undertaken in the software industry.

Constructing a single OWL file that contained the entire ontology seemed an appropriate step towards this aim. Doing so removes some of the issues associated with importing multiple ontology files, and produces a view of OBI uncomplicated by choices made to accommodate development. The goal of producing a single file spurred development of our release and quality control process. Having a dedicated release process allows us to more carefully control and modify the ontology before making it available. Our release process involves a number of steps including checks for content quality (e.g., annotations compliant with our policy), syntax (e.g., OWL

[1] http://purl.obofoundry.org/obo/obi/repository/trunk/src/tools/build/external-templates.txt
[2] http://neurocommons.org/
[3] http://www.w3.org/TR/rdf-sparql-query/
[4] http://sparql.neurocommons.org/sparql
[5] http://www.w3.org/TR/rdf-sparql-protocol/
[6] The latest version of OBI is available at http://purl.obofoundry.org/obo/obi.owl

species validation), and reporting candidate release status to the ontology developers. To manage this, many of the tasks associated with release are automated.

2.3 Quality checks and reports

To ease curators' work whilst ensuring the quality of the ontology, we decided to provide reports to each branch that identified areas not compliant with our policies prior to each release. We use a Jena-based [9] script to read in our branch files and identify missing elements, duplicates, or misuse of any of our metadata properties. The reports are rated according what action needs to be taken: simple warnings for those errors that can be corrected automatically by script, or critical alerts for those issues requiring manual intervention from one of our curators. Reports are simple HTML pages displaying terms and associated issues. We explored different policies regarding what to do in case of significant errors (e.g. block release), but instead adopted a release early, release often approach in the hopes that this would encourage developers to correct mistakes in a timely fashion.

As an example of the sort of thing we need to correct, because of issues using the Protégé [10] editor, we would occasionally encounter a problem with one of our annotation properties being saved in the wrong branch file: for example, when adding a label to one of the instruments, this label could get serialized in the *Biomaterial.owl* file instead of the *InstrumentAndParts.owl* file.

In order to mitigate this, we are considering using an extra annotation property to indicate which branch classes belong in. By using this information we could automatically clean up and reorganize branch files.

Additional scripts perform other quality control checks, including listing terms missing a curation status instance, listing terms with extra curation instances (only one is allowed per term), listing terms missing a label, and listing classes that are asserted under a defined class.

2.4 Identifier maintenance policy

Having a stable and consistent ID policy is a fundamental OBO Foundry principle. In OBI, identifiers are prefixed with “OBI_” and followed by seven digits. Forcing developers to manage this was impractical, particularly given the distributed development process. Instead, we have curators ignore the identifier format while developing OBI. As an automated step prior to release we run scripts that find terms without standard IDs and rename them, as well as perform other checks such as whether all IDs present in the previous release are still present, since terms are not supposed to be deleted according to the GO policy OBI follows.

2.5 Managing disjoints

During the initial stages of our development process, we manually added disjoints to classes as we were building the ontology. However, we ran into consistency issues as we edited OBI, as a stated disjoint in one place of the OBI tree would not hold true when a term was moved to a different location in the class hierarchy. Instead, a script is used as part of our release process to automatically compute disjoint class statements, assuming that our asserted class hierarchy is not rearranged during reasoning. The sets of disjoints are computed traversing the asserted class tree,

ignoring placeholder classes and defined classes, making OBI classes at each level mutually disjoint, and OBI classes disjoint to non-OBI classes at the same level.

2.6 Distributing OBI with inferred superclasses

We are using defined classes, and want to provide an easy-to-use file that does not require the use of a reasoner on the end-user side. Therefore we add, via script, the inferred superclasses to our OWL file.

This allows end users to have the fully inferred hierarchy, while keeping the ontology "clean" according to Rector's [11] normalization recommendations by using defined classes and avoiding asserting multiple superclasses.

2.7 Assuming that all classes have instances

Consider the following example:

```
Namespace(e = <http://example.com/>)
Ontology(<http://example.com/>
 Class(e:manuf_role partial e:role)
 Class(e:role partial)
 Class(e:organization partial)
 Individual(e:Affymetrix type(e:organization))
 ObjectProperty(e:has_role )
 ObjectProperty(e:is_manufactured_by
     range(restriction (e:has_role someValuesFrom(e:manuf_role)))
  Class(e:hg133 partial e:microarray)
  Class(e:hg133 partial
     restriction(e:is_manufactured_by value(e:Affymetrix)))
  Class(e:manufacturer complete
     restriction(e:has_role someValuesFrom(e:manuf_role))))
```

Figure 2 Abstract syntax for an ontology for which the desired inference is not made.

In this example we define a *manufacturer* class, an object property *is manufactured by* with range *manufacturer role*, and add that a specific microarray type is manufactured by an organization *Affymetrix*. We were expecting the reasoner to classify *Affymetrix* as *manufacturer*. However this is not the case unless we explicitly add a *microarray* individual to the ontology.

This behavior arises because current OWL reasoners do not assume existence of instances when doing subsumption checks. Rather, satisfiability checks are done by asserting that at least one instance exists, serially, for each class. However, in the framework of BFO, all classes must have instances - classes that do not have instances do not exist. Ideally, we would indicate our assumption that all classes have at least one individual to a reasoner and have it compute subsumptions and other inferences on that basis. However the reasoners we use, Pellet [12] and Fact++ [13], do not offer this choice. Therefore we decided to script the addition of anonymous individuals of each type named in the ontology as part of our release process[1]. We do this for each leaf class, and before computing the inferred superclasses. Asserting a distinct anonymous individual as member of each leaf class means that the superclasses will also have one member and ensures that the type of entailment described above, that we depend on, will reliably be computed and that ontologies

[1] The latest version of OBI is available at http://purl.obofoundry.org/obo/obi.owl

that are not jointly satisfiable will be detected. We plan to suggest that a similar mechanism is adopted by the OWL versions of all OBO ontologies. We note that this choice is not without problems. OBI, augmented with these assumed individuals, becomes more difficult to reason with reliably - we have had problems with both Pellet and Fact++ and are at the moment communicating with the developers of those reasoners to determine the source of the problem. Therefore, we currently use the assumed individuals to compute the inferred class hierarchy, but do not include them in the released version of OBI.

2.8 Increasing the readability of the RDF/XML version of OBI

We use numerical identifiers for all our terms: classes, instances, but also for annotation, data and object properties. However, we sometimes need to edit the OWL RDF/XML directly, which is cumbersome because IDs are not easily remembered. To increase human readability we post-process the RDF/XML and generate XML comments for the released version of the file. We recommend that tool developers offer an option to use some annotation property as an XML comment when serializing OWL.

```
<owl:Class rdf:about="&obo;OBI_0000265"> <!-- report table -->
  <!-- definition editor -->
   <OBI_0000274 xml:lang="en">person:Allyson Lister</OBI_0000274>
  <rdfs:label xml:lang="en">report table</rdfs:label>
  <!-- definition -->
  <OBI_0000291 xml:lang="en">A report table is a report display
element consisting of a matrix of cells laid out in a grid, some set of which are
filled with some information content</OBI_0000291>
  <rdfs:subClassOf>
    <owl:Class rdf:about="&obo;OBI_0000001"/> <!-- report display element -->
  </rdfs:subClassOf>
</owl:Class>
```

Figure 3 Example of XML comments used to note what ids correspond to in RDF/XML serialization

2.9 OBI terms on the Web

In addition to supplying the OBI ontology as a single file, we are in the stage of prototyping responding with a bounded amount of useful information for each URI naming a term in OBI[1]. In doing so we follow httpRange-14[2] and use a HTTP response code of 303 with a redirect to RDF/XML describing the term. We do no content negotiation to emphasize that the URI names a single thing. In order to present readable information in web browsers, we use an XSL stylesheet, which is executed by the browser to generate HTML. Choices need to be made as to what the content of the RDF delivered at this URL should be. We chose to make each bundle of RDF delivered at this URL a valid OWL DL ontology by importing the full OBI ontology. A certain amount of relevant information is included for web clients that do not follow that import statement: for a class, the axioms defining it, inferred superclasses, properties that it is in the domain of or range of, and labels for any referenced terms are added. We also include project information using the DOAP

[1] For an example view http://purl.obofoundry.org/obo/OBI_0000225 in a browser
[2] http://www.w3.org/2001/tag/issues#httpRange-14

schema[1] including pointers to our repository, tracker, mailing list, and release information.

3 Discussion

Support for MIREOT and for reasoning services that assume the existence of an instance of each class, as discussed above, would be helpful. Below we discuss a number of additional features of OWL and associated tools that would ease development of OBI and other ontology projects.

3.1 Deprecation

As the obsolete terms are stored in a separate file (to make it easier to excise them from some versions of OBI), we constantly run into issues trying to move obsolete classes to the *ObsoleteClass* hierarchy. Protégé allows for editing of a single ontology file at a time (the *active ontology*), making this difficult and error prone.

In addition, our deprecation policy stipulates, among other things, that axioms involving deprecated terms should be removed. In order to support this practice we wish there were either better tool support for it or OWL language support that would cause axioms involving deprecated/obsoleted terms to be considered annotations.

3.2 Annotations on annotations

OBI is used in a variety of fields and we need to address the fact that one term can mean different things in different communities.

For example, the term *probe* is a synonym for the term *reporter* in a community, whereas it is a synonym for the term *detector* in another. Annotations on annotations are an appropriate representation of these community-specific labels, and would allow us to "tag" any of our synonyms with extra information noting pertinence to a specific community.

3.3 Versioning

OBI's policy is to release frequent updates and to maintain access to all versions. We create dated versions of each release, currently using the Persistent Uniform Resource Locator (PURL) [14] system to provide access to successive revisions. This leaves to the end-user control over the choice between prefering stability and being up to date with the latest developments. OWL 2's versionURIs[2] will make it possible for users to easily choose which version of the ontology to use.

We would appreciate if this practice became the generally accepted way of managing versioning of ontologies. While developing OBI we prefer stability (i.e., not being surprised by unplanned-for changes), and we work around the lack of published ontology versions by relying on local copies of imported ontologies.

---

[1] http://trac.usefulinc.com/doap

[2] http://www.w3.org/2007/OWL/wiki/Syntax#Ontology_URI_and_Version_URI

3.4 Support for Rector-normalization style editing

The dominant paradigm for editing ontologies is that of a single rooted hierarchy. However the style proposed by Rector and others is to develop a series of single inheritance ontologies and a separate set of classes defined in terms of elements of the single inheritance trees. An ontology interface that supported fluidly moving between the component trees, the defined classes, and the inferred composite view, as well as providing easy access to common patterns for the composite definitions would significantly benefit ours and other's efforts.

3.5 Disjoints

Our solution for disjoints is not entirely satisfactory. Declaring disjoint policy for whole trees where the siblings are all mutually disjoint is appealing, but there are exceptions., Consider the classes *kit* and *instrument* which are subclasses of *device*. *Device*'s subclasses remain disjoint if we decide to modify the hierarchy by moving *kit* to be subclass of *instrument*. However, if *kit* and *instrument* were each declared disjoint with each other we would arrive at an inconsistency. Upon closer examination we found potential exceptions - cases where, for the most part the siblings were disjoints, but not always. One example is the *Role* hierarchy, and within that *biological specimen role* and *assay input role*. We are currently debating whether these two roles overlap with each other - certainly the processes in which they are realized do. We might wish to note this pair as an exception - that they are not disjoint.

There are additional complications involving the choice of whether disjoints should be added relative to the asserted or inferred class hierarchy. If the former and the author misses an inference that results in a rearrangement of the class hierarchy, we might get an inconsistency. If disjoints are added after reasoning then we need to not add disjoints for completely defined classes.

4 Acknowledgements

In memory of our friend and colleague William Bug, Ontological Engineer.

The OBI consortium is **(***in alphabetical order***):** Ryan Brinkman, Bill Bug, Helen Causton, Kevin Clancy, Christian Cocos, Mélanie Courtot, Eric Deutsch, Liju Fan, Dawn Field, Jennifer Fostel, Gilberto Fragoso, Frank Gibson, Tanya Gray, Jason Greenbaum, Pierre Grenon, Jeff Grethe, Mervi Heiskanen, Tina Hernandez-Boussard, Allyson Lister, James Malone, Elisabetta Manduchi, Luisa Montecchi, Norman Morrison, Chris Mungall, Helen Parkinson, Bjoern Peters, Matthew Pocock, Philippe Rocca-Serra, Daniel Rubin, Alan Ruttenberg, Susanna-Assunta Sansone, Richard Scheuermann, Daniel Schober, Barry Smith, Holger Stenzhorn, Chris Stoeckert, Chris Taylor, John Westbrook, Joe White, Trish Whetzel, Stefan Wiemann. The author's work is partially supported by funding from the NIH(R01EB005034), the EC EMERALD project(LSHG-CT-2006-037686), the BBSRC(BB/C008200/1), the EU

NoE NuGO(NoE 503630), the CARMEN project EPSRC(EP/E002331/1), and the Michael Smith Foundation for Health Research.

5 References

1. B. Smith and M. Ashburner and C. Rosse and J. Bard and W. Bug and W. Ceusters and L. J. Goldberg and K. Eilbeck and A. Ireland and C. J. Mungall and N. Leontis and P. Rocca-Serra and A. Ruttenberg and S. Sansone and R. H. Scheuermann and N. Shah and P. L. Whetzel and S. Lewis (2007) The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration . Nat Biotech, 1251-1255.
2. Pierre Grenon, Barry Smith and Louis Goldberg: "Biodynamic Ontology: Applying BFO in the Biomedical Domain". From D. M. Pisanelli (ed.), *Ontologies in Medicine*, Amsterdam: IOS Press, 2004, 20–38
3. Wheeler DL, Chappey C, Lash AE, Leipe DD, Madden TL, Schuler GD, Tatusova TA, Rapp BA (2000). Database resources of the National Center for Biotechnology Information. Nucleic Acids Res 2000 Jan 1;28(1):10-
4. Golbreich C, Zhang S and Bodenreider O. (2006) The foundational model of anatomy in OWL: Experience and perspectives. Web Semantics: Science, Services and Agents on the World Wide Web, 4 (3). 181-1
5. The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. Nature Genet. (2000) 25: 25-29
6. The Gene Ontology Consortium. The Gene Ontology editorial guide. http://www.geneontology.org/GO.usage.shtml
7. Jonathan Bard, Seung Y Rhee, and Michael Ashburner An ontology for cell types. Genome Biol. 2005; 6(2): R21.
8. Grau BC, Horrocks I, Kazakov Y, and Sattler U. (2007) Extracting Modules from Ontologies: A Logic-based Approach. Proc. of the Third OWL Experiences and Directions Workshop, number 258 in CEUR
9. Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. 2004. Jena: implementing the semantic web recommendations. In Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters (New York, NY, USA, May 19 - 21, 2004). WWW Alt. '04. ACM, New York, NY, 74-83.
10. Protégé. http://protégé.stanford.edu
11. Alan L. Rector (2003). Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. Proc K-CAP: 2003 (ed J Genari)
12. Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. Web Semant. 5, 2 (Jun. 2007), 51-53.
13. Dmitry Tsarkov, Ian Horrocks (2006) FaCT++ description logic reasoner: System description. In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)
14. KE Shafer, SL Weibel, E Jul (2001) The PURL Project. Journal of Library Administration, 2001