Automated Heuristic RDF Repair

Peter Coetzee
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2006-126
October 10, 2006*

semantic web,
RDF, eyeball, jena,
heuristic, repair,
web service

This report covers the abstract concepts behind, as well as an implementation of, an automated heuristic RDF repair system for the Semantic Web. This student project formed a part of the Eyeball software originally written by Chris Dollin. This document also details the new availability of Eyeball as a web service, including its gathering of statistics. The web service was made available to the Jena community on 6th September 2006 on an HP external server at http: //jena.hpl.hp.com:5500, and analysis of some of the statistics gleaned from its usage since that time are included.

Automated Heuristic RDF Repair

Peter Coetzee < peter.coetzee@hp.com >
Semantic and Adaptive Systems Department
Digital Media Systems Laboratory
HP Laboratories Bristol
September 2006

This report covers the abstract concepts behind, as well as an implementation of, an automated heuristic RDF repair system for the Semantic Web. This student project formed a part of the Eyeball software originally written by Chris Dollin. This document also details the new availability of Eyeball as a web service, including its gathering of statistics. The web service was made available to the Jena community on 6th September 2006 on an HP external server at http://jena.hpl.hp.com:5500, and analysis of some of the statistics gleaned from its usage since that time are included.

Table of Contents

Preface

This document describes the work performed by Peter Coetzee under the guidance of Chris Dollin as a software intern working for Hewlett Packard Labs Bristol, during the months of June – September 2006. This work centred on extending the functionality of Eyeball[1] (a part of the Jena[2] RDF/OWL toolkit) to include heuristic analysis and repair. Work was also carried out for the creation of an Eyeball web service; this is documented in the latter stages of this text.

Introduction

Ontologies, schemas, and class definitions are often evolved over time, in order to support the creation of instance data. Keeping track of all of these definitions is a non-trivial activity for an RDF[3] author (whether a developer or otherwise), and attempting to do so can hinder creative flow. As a result of these conditions, and of the open world assumptions made by the Semantic Web[4], it is not unusual for mistakes to arise. Eyeball originally grew with the goal of heuristic RDF inspection and accuracy checking in order to allow an author to detect any errors they may have made post-hoc. The results of this checking is delivered as an RDF model, which can be reported either as an RDF XML[5] / N3[6] output or rendered as simple text (made more readable using the `rdfs:label` statements in the Eyeball schema). It is important to note here that the issues which Eyeball seeks are not necessarily *illegal* usages of RDF. One may, for example, perfectly legally use properties in an RDF document without first declaring them. However, it may be that an author wishes to ensure that they have declared those which they may use; an undeclared property might indicate a typographical error, or an omission in their ontology.

However, inspection is only half way to the solution. It is important that an RDF author is capable of not just understanding and locating a mistake, but is also able to fix it. While one may assume that most authors have little to problem with doing this manually, it will not always be the case that they the time or indeed the inclination to do so; a particular problem if their application works with the data as it is. Remedying this situation is the goal of the activity around Eyeball Repair; a *very* heuristic set of attempts at automatic non-interactive analysis and repair of an RDF model based on the reports issued by Eyeball.

Where Eyeball Stands

Eyeball is built upon the Jena Semantic Web Framework, a project from the Semantic and Adaptive Systems Group of Hewlett Packard Laboratories, Bristol. It reads in an RDF model in any format Jena is capable of understanding, applies an inference engine over the top, and inspects the model for errors. The process of inspection is carried out by a series of pluggable Inspectors, each of which may issue zero or more report items. Each of these items goes into a report model as a blank node of `rdf:type eye:Item`, and describes something which the `Inspector` deemed to be a fault, as well as enough information to inform the reader where in the model this fault was located (e.g. on this URI, this `rdf:Statement`, this literal etc.)

Shipped Inspectors

The following Inspectors ship with the standard Eyeball 2.0 distribution:

- AllTyped – Ensures that all URI and bNode resources (and optionally literals) have an `rdf:type` defined. If one isn't present, one should be defined.
- Cardinality – Checks whether all usages of a class' property conform to its cardinality restrictions. If not, it may be necessary to add or remove usages of a property, or possibly alter the class definition.
- Class – Every Class that is used should be defined; if one is used that is not defined, it may be useful to define it
- ConsistentType – Checks to make sure that any subject in the model is of consistent `rdf:type`, taking into account inferred types. If not, all usages of that subject should be set to the same `rdf:type`.
- Literal – Ensures that datatypes and language codes are accurately specified. Repairing this may require detecting a suitable datatype or setting a default type / language code.
- Prefix – Inspects prefix namespaces, reporting well-known prefixes that aren't used with their well-known URIs (or vice-versa). Repair should attempt to resolve any mismatched prefixes and update the model accordingly.
- Property – Ensures all used properties are declared in a provided schema; if not, it is feasible to assume that a spelling mistake has occurred, and attempt to locate the desired property.
- URI – Checks for syntactically correct URIs using Jena's IRI code. The various errors this creates should be dealt with according to the IRI error details.
- Vocabulary – Every used URI must be declared in its namespace's schema (if available). Once again, if not then it is a reasonable assumption that a typographical error occurred, and a suitable URI should be substituted accordingly.

Design Goals

Eyeball was written to be extensible through a system of plugins, managed by the Jena Assembler framework. Prior to Eyeball Repair, the only plugins Eyeball supported were Inspectors: instances of the `Inspector` interface, each given the task of inspecting for one type of 'fault' (or inconsistency). Eyeball Repair is designed to extend these plugins to include interfaces for `Analysis` and `Doctor`. There should be a one-to-one mapping of Analyses to Doctors to Inspectors. A useful convention within Eyeball is to name its Inspectors according to the type of issue which it inspects; this convention is to be maintained in writing the Analyses and Doctors. Each Analysis module should parse the relevant portion(s) of the Eyeball report, inspect the section of the model it refers to, and make a suggestion as to how best to fix the error (known as analysing the report). This suggestion can then be appended onto the relevant report item. When the Doctor receives this report, it should simply apply the fix as suggested by the report (called doctoring the model).

End user transparency is vital; one of the powerful aspects of Eyeball (and by extension Eyeball Repair) is that the user does not necessarily need to understand the full set of rules that Eyeball uses to check RDF (although it is both desirable and helpful if they do). This transparency should be extended further by Eyeball Repair, such that the user need not understand *how* a fix was arrived at, provided the fix does not generate inaccurate RDF for the user. Throughout Eyeball Repair, the fixes that are applied should be of a lowest common denominator; that is to say, they should be *as descriptive as possible* while remaining *as accurate as possible*. Furthermore, the output model should give no extra report items if the Eyeball is called to inspect it again with the same configuration. Ideally, there should be no faults reported at all after a repair, but it is important to recognise that this may not always be possible.

Sample Repair Methods

With the overall architecture of Eyeball Repair in mind, the repair it conducts is dependent on the Inspectors that have run. A small sample of these repairs are outlined below, categorised according to the Inspector. They detail both the Analysis and Doctor for the relevant Inspector shipped with Eyeball (see above). A full specification of the full suite of repairs and analyses can be seen in Appendices A and B.

Cardinality

Description

There are three avenues of repair to be chosen from by the Analysis – increase the number of usages of property P on node X; decrease the number of usages of property P on node X; increase the value of the `maxCardinality` statement on class C. The first pertains to the situation when the cardinality of P on X is less than the `minCardinality` of P on C. The second and third pertains to the opposite; the cardinality of P on X is greater than the `maxCardinality` of P on C.

Class

Description

The Analysis will direct the Doctor to add a statement with a minimal framework definition of a Class in the model. It must detect the model profile and define the class accordingly (e.g. `XYZ rdf:type owl:Class` or `XYZ rdf:type rdfs:Class` etc.)

Literal

Description

The LiteralInspector checks primarily for language code and datatype definitions on literals in the model. If the Analysis finds an `eye:badDatatypeURI` it should first analyse the lexical form of that literal. If it conforms to one of the common xsd[7] datatypes, it may instruct the Doctor to alter the datatype URI to that of the relevant xsd datatype. For this version of the code the Analysis should look for valid integers, decimals, dates, times, dateTimes, and boolean (by means of regular expression matching of the datum's literal lexical form). If none of these match, the tool may fall back to either setting a default language code for this literal, or setting it to a default datatype. This behaviour must be user configurable.

Property

Description

This Analysis uses spellcheck functionality. It first builds two dictionaries of known properties from all .rdf files in the mirror/ directory. The first of these contains the full URI of the property, and the second only the localName part. As a first attempt, the Analysis should attempt to find the property in some namespace in that directory. These are compared based on localName (so if, for example, the Analysis encounters `rdf:integer` it can reasonably change it to `xsd:integer`). If this fails, the Analysis should fall back to spellcheck systems. It will first try and find a match in its dictionary of full URIs (i.e. the property might have been misspelled, and still exists in this namespace, or its namespace was misspelled). If it fails to find a match there, it will try and search the localName dictionary (i.e. the property might be both in the wrong namespace, *and* be misspelled). If it cannot find a match, the repair has failed. The user must be notified of this fact.

Analysis Report Output

When the Analysis module has analysed the report and the model, and decided which course of action should be taken, it must append its findings to the relevant report item on the RDF model of the existing report as discussed previously. The content of these additions should follow the definitions described in Appendix B.

Eyeball Repair Configuration

There are a number of user configurable options entailed in the repair process. It is therefore desirable that Eyeball Repair is able to store and retrieve these configuration options; it would be highly undesirable to force a user to set *all* of these options at program execution time. The existing Jena Assembler engine used in Eyeball is configured by an RDF model, stored in `etc/eyeball2-config.n3`, which drives the assembly of components. As this file is plain RDF (in N3), it is the perfect location to store configuration options for Eyeball Repair. As well as adding Assembler directives for the Analysis and Doctor components to this file, it can contain the required configuration options for the various repair modules described in Appendix A. These should all be rooted on the subject `eye:repairConfig`[8] and be statements of the form [`eye:repairConfig`, `eye:`*`configOption`*, { "*Literal Value*" || `eye:`*`configSetting`* || *bNode*}]. If the object is a bNode, then there is more than one datum for each part of that configuration option. For example, the options to configure the mapping of known prefixes and namespaces (`eye:nsPrefix`) needs to record the prefix (`eye:prefix`) and URI of the namespace (`eye:nsURI`) for each option.

User Interface

Command Line

With all of the repair functionality integrated into Eyeball, as Eyeball 2.1 (E2.1), the user interface must also be updated. The implementation of the configuration is such that only two extra options are needed. These are "`-analyse`" and "`-repair`". The former will access each of the Analysis components which are configured, applying updates to the report to suggest the repairs to be carried out (as specified in Appendix B). The "`-repair`" option will direct the Eyeball to both analyse and doctor the model, applying all of the repairs which are specified in the Analysis-generated report. The vocabulary for model output is determined according to the value of the "`-render`" switch. If E2.1 is configured to output its report as N3, the repaired model will also be output as N3. Otherwise (E2.1 is outputting the report as RDF/XML or simple text), the model is output as RDF/XML.

The "`-repair`" switch has an optional data part; if left off, the model will be output to the standard E2.1 output stream before the report. Otherwise, E2.1 will attempt to safely output to the location specified in this data part. If this data part starts in "`jdbc:`", it is assumed to refer to a database; otherwise a file. The database location is specified in the same method as database input ("`jdbc:DB:head:model`"). If it exists already, the model will be backed up in the same database as *`modelName`*`.old-`*`systemTimeInMilliseconds`*. It is vital to note that for very large models or very slow database connections this could take a significant amount of time; the entire model must be re-written to the database irrespective of the number of changes which occur. In a similar vein to database output, if the specified file path exists, it will be backed up as *`fileName`*`.old-`*`systemTimeInMilliseconds`*.

Experimental GUI

Eyeball 2.0 features an experimental graphical user interface based on Swing. E2.1 extends this UI, adding its new functionality. Essentially the E2.0 GUI implements the CLI options in graphical form. E2.1 adds to this its own options for analysing and / or repairing the model. Like on the CLI, if the model is repaired it must be analysed first – this restriction is enforced on the GUI as well as the CLI. Assorted refactoring and improvements are also needed on the GUI for the 2.1 release; the addition of scrollbars to report and model output is vital to make it usable, as well as a means to select output format (text, RDF/XML, N3 etc.) E2.1 also adds the ability for a user to directly save output results (report, output model etc.) to their filesystem from within the Eyeball GUI.

Interactive Model Repair

With the current E2.0 notion of implementing CLI options graphically, it would be up to the user to read and understand in conjunction the report and both input and output models in order to properly understand what has taken place. The E2.1 GUI should be able to make this easier to digest by making the repair procedure interactive. In its simplest form, the GUI can render a plain text version of each of each `eye:Item` in the report, and allow the user to select whether or not to include that repair suggestion in the repairs to be carried out. If the user elects not to, the item is removed from the temporary working report model. Once all items have been configured, Eyeball may run the repair, with this working model as its report input. This mode is not designed to allow the user to choose *how* each repair is carried out (this would violate the *automated* aspect of the RDF repair activity), but simply *whether* each repair is carried out.

Eyeball Web Service

In a similar vein to other HP Labs Semantic Web projects, E2.1 is offered online as a web service. Usage of this service may take one of two forms, depending on the user's need; either a web-based form for the user to configure the Eyeball and upload RDF, or an RPC interface, allowing programmatic access to the Eyeball service. It may be useful to also offer this with a lightweight embedded server (such as Jetty) as a part of the Eyeball package at a later date.

Web Service Implementation

In the interests of demonstration, the web service is constructed in two main sets of components; the service itself, and a service client, written as an interactive web form. The service expects request variables to be passed to it in a particular form in order for it to operate. The following operations are recognised by the service, either as HTTP GET or POST requests. Operations are specified as a request key, with a value expected for some. The service will respond according to the nature of the request (as detailed below)

- *`submitRDF`* - expects an RDF model (in N3, RDF/XML or N-TRIPLE)
- *`getReport`* - returns the report, rendered according to the configuration
- *`getRepairedModel`* - returns the repaired model, in the same language as the input
- *`getValidInspectors`* - returns a model of inspectors, plus their descriptions
- *`getBuiltInAssumes`* - returns a model of the built in assumes
- *`setupConfig`* - expects an RDF/XML model of configuration options. The following options are supported[9] as the predicate of a statement:
  - `eyesvc:doRepair` - can take any value. If a statement is present with this as its predicate, analysis and repair flags will be set
  - `eyesvc:doAnalysis` - can take any value. If a statement is present with this as its predicate, analysis flag will be set
  - `eyesvc:useInspector` - each of the values of statements with this predicate will be added as inspectors
  - `eyesvc:addAssume` - each of the objects (as literal values or URIs) of statements with this predicate will be added as assumes
  - `eyesvc:withBaseUrl` - should specify one URI resource only as the object; specifies the base URI for reading the model
  - `eyesvc:inputModelType` - should specify one literal only as the object (a valid Jena model type, e.g. `N3` / `RDF-XML` or 'Best Guess', in which case the service will attempt to guess the model type)
  - `eyesvc:reportFormat` - should specify one literal only as the object (a valid Eyeball report render format, e.g. `text` / `n3` / `xml`)
  - `eyesvc:collectStatistics` - should specify one value only (`true` / `false`,

yes / no, or on / off. Any case is accepted; however if it is not one of these, it is taken to mean false.) as the object; allows the user of the service to specify whether or not they wish to allow the service to collect usage statistics

If the requested operation is not recognised, or something goes wrong with the service operation, the service will return an error. Otherwise, it will return a single RDF/XML model with its response, depending on the action(s) requested. If no action is requested, there will be no error; simply a blank RDF/XML model. In the absence of a standard for named subgraphs in RDF at the time of writing, an implementation of a similar notion has been used. The E2.1 service adds a statement for each subject in the model, specifying which piece of output it eyesvc:belongsTo. Resources are given for eyesvc:report, eyesvc:outputModel, eyesvc:inspectors, and eyesvc:assumes.

For example, to extract the report from the data stream (assuming the report was requested as RDF rather than a simple text rendering), one could query the graph using a simple SPARQL query as follows:

```
PREFIX  eyesvc: <http://jena.sourceforge.net/EyeballService#>
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?subject ?predicate ?object . }
WHERE
  { ?subject ?predicate        ?object          ;
             eyesvc:belongsTo  eyesvc:report    .
    FILTER ( ?predicate   !=   eyesvc:belongsTo )
  }
```

This query will construct the entire report in its original format. It is important to note that while one may obviously substitute eyesvc:modelOutput (to get the repaired model) for eyesvc:report, if the model includes a statement with predicate eyesvc:belongsTo, that statement will be removed by the FILTER. If this is likely to be a problem, one may need a more complicated SPARQL query to parse the results.

The web form serves as a demonstration of some of the potential of this service. The form itself is, where relevant, dynamically generated; it queries the web service to find the available assumes and inspectors (as well as the rdfs:labels for the inspectors), and constructs HTML accordingly. When submitted, it collects form variables entered by the user, and builds a request accordingly. It submits the RDF/XML configuration model to the service, and collects its response. This response is read into a model, and queried (as detailed above) in order to construct each of the needed models (the report and, if requested, the repaired model), serialise them, and sanitise the serialisation for HTML output.

Collection and Analysis of Statistics

As mentioned in the abstract, the E2.1 web service is capable of collecting and collating statistical data on its usage. The goal of these statistics modules is to answer questions on a number of aspects of Eyeball, to feed back into future repairs. Some of the questions that we hope to glean answers to are;

- What type of issues do real-life RDF instance data seem to have, and how common might these issues be?
- The popularity of the various aspects of Eyeball; e.g. are some inspectors favoured over others? How often does a user ask the Eyeball to analyse or repair these issues?
- How drastic are the changes that the Eyeball Repair functionality typically makes to the input model?
- What are the ontologies that people commonly use? Are certain prefixes favoured over others for these ontologies?
- What are the attributes of a typical RDF model? For example, what is its average size in triples?

It was deemed from an early stage that in order to maximise the amount of user data that is submitted, and minimise privacy concerns, it is vital that no personal data is collected as part of the statistics; no portion of the model may be persisted in the statistical data.

Similarly to the analyses and doctors, the statistics modules were designed to integrate into Eyeball using the Assembler framework. As such, the notion of (and interface for) a `Statistician` has been defined. Each Statistician contains (at minimum) a method to gather the statistical data, and a method to analyse the data, returning a String of its report. The data itself is persisted in a database (configured as part of the configuration file `etc/eyeball2-config.n3`) as a Jena RDF model. The philosophy behind the statistics is wherever possible to collect only raw data (e.g. “these inspectors were used”) at the time it is gathered, so as to minimise any impact on the user experience. This raw data can then be analysed at the time the statistical reports are requested (e.g. “what percentage of the time were these inspectors used?”). The following statistics are currently collected and analysed:

- Model size (number of triples)
- Ontologies and prefixes prefix-mapped
- Issues E2.1 raised
    - Raw number of each issue reported
    - Type of issue
- Configuration Eyeball run with
    - Inspectors used
    - Assumes requested
    - Was the Eyeball asked to Repair or Analyse?

- Percentage delta in model from input to output
    - This analysis *should* be done at gather time, in order to satisfy privacy concerns, as it prevents the statistician having to collect any model data for later analysis
- Date / Time of request

At present, the most accessible way to access these statistics are from the service's web interface. The statisticians may also be run from the command line (the executable class `com.hp.hpl.jena.eyeball.web.statistics.runReports` currently does this). Both of these creates a `statisticalAnalyser` object, which is responsible for coordinating the assembly of the statisticians, connection to the database, as well as calling the analysis of the statistics upon request (it also handles the gathering of statistics from an `eyeballer` object). At present the statistics are not used with the default Eyeball command line, or the Eyeball GUI in the interests of ease of installation for the casual end user. It would, however, only be a small amount of work for a developer to create and integrate a `statisticalAnalyser` for these uses of the Eyeball.

Sample Live Statistics

The web service has, at the time of writing, been live for approximately one week. During this period of operation, 29 sets of statistics have been collected. Their analysis (as provided by the `Statisticians`) can be seen below.

```
Report generated on 14/09/2006 at 09:24:11.

MetaDataStatistician has analysed 38 sets of statistics and found that
requests were made on the following dates:
    On 2006-09-05 there were 8 requests made of the service.
    On 2006-09-06 there were 9 requests made of the service.
    On 2006-09-11 there were 8 requests made of the service.
    On 2006-09-12 there were 4 requests made of the service.
    On 2006-09-14 there were 9 requests made of the service.

ModelStatistician has analysed 38 sets of statistics and acquired the
following results.
The average size of model submitted for eyeballing is 79.5 triples.
The following information was gathered about ontology usage:
    http://schemas.talis.com/2006/bigfoot/configuration# occurred on
    average 0.02 times per model. It was used with the following
    prefixes: bf (1 time).
```

```
http://www.w3.org/1999/02/22-rdf-syntax-ns# occurred on average
0.89 times per model. It was used with the following prefixes:
rdf (34 times).

http://schemas.talis.com/2006/frame/schema# occurred on average
0.02 times per model. It was used with the following prefixes:
frm (1 time).

http://www.w3.org/2000/01/rdf-schema# occurred on average 0.60
times per model. It was used with the following prefixes: rdfs
(23 times).

http://www.w3.org/2001/XMLSchema# occurred on average 0.36 times
per model. It was used with the following prefixes: xsd (2
times), xmls (6 times), j.0 (6 times).

http://purl.org/dc/elements/1.1/ occurred on average 0.28 times
per model. It was used with the following prefixes: dc (31
times).

http://www.w3.org/2002/07/owl# occurred on average 0.55 times per
model. It was used with the following prefixes: owl (21 times).

http://my.domain.com/ occurred on average 1.28 times per model.
It was used with the following prefixes: stillMine (20 times),
mineAlso (15 times), my (14 times).

http://hahah/this/isnt/dc/ occurred on average 0.52 times per
model.

http://eyeball.jena.hpl.hp.com/eg# occurred on average 0.31 times
per model. It was used with the following prefixes: rlybad (6
times), eg (7 times).

http://www.rdfdata.org/ns/ occurred on average 0.02 times per
model. It was used with the following prefixes: rd (1 time).

http://www.eswc2006.org/technologies/ontology# occurred on
average 0.02 times per model. It was used with the following
prefixes: eswc (1 time).

http://xmlns.com/foaf/0.1/ occurred on average 0.02 times per
model. It was used with the following prefixes: foaf (1 time).

http://a9.com/-/spec/opensearch/1.1/ occurred on average 0.02
times per model. It was used with the following prefixes: os (1
time).

http://www.w3.org/2001/vcard-rdf/3.0# occurred on average 0.02
times per model. It was used with the following prefixes: vcard
(1 time).

http://jena.hpl.hp.com/EyeballService# occurred on average 0.02
times per model. It was used with the following prefixes: eyesvc
(1 time).

http://www.daml.org/2001/03/daml+oil# occurred on average 0.02
```

```
    times per model. It was used with the following prefixes: daml (1
    time).

    http://jena.hpl.hp.com/2003/08/jms# occurred on average 0.02
    times per model. It was used with the following prefixes: jms (1
    time).

    http://jena.hpl.hp.com/Eyeball# occurred on average 0.02 times
    per model. It was used with the following prefixes: eye (1 time).

    http://www.example.org/ occurred on average 0.02 times per model.
    It was used with the following prefixes.

    http://purl.org/rss/1.0/ occurred on average 0.02 times per
    model. It was used with the following prefixes: rss (1 time).

    http://jena.hpl.hp.com/2005/11/Assembler# occurred on average
    0.02 times per model. It was used with the following prefixes: ja
    (1 time).
```

```
EyeballingStatistician has analysed 38 sets of statistics and found
the following failures;

    eye:unknownPredicate occurred on average 3.42 times per
    eyeballing.

    eye:unknownClass occurred on average 2.92 times per eyeballing.

    eye:hasNoType occurred on average 4.55 times per eyeballing.

    eye:badURI occurred on average 3.31 times per eyeballing.

    eye:badNamespaceURI occurred on average 0.68 times per
    eyeballing.

    eye:cardinalityFailure occurred on average 0.52 times per
    eyeballing.

    eye:multiplePrefixesForNamespace occurred on average 0.63 times
    per eyeballing.

    eye:badDatatypeURI occurred on average 1.57 times per eyeballing.

    eye:notFromSchema occurred on average 1.44 times per eyeballing.

    eye:jenaPrefixFound occurred on average 0.15 times per
    eyeballing.

On average, models were changed by 41.5% during eyeballing (note: this
includes times the model was not repaired!)
```

```
ConfigStatistician has analysed 38 sets of statistics and found that;

The eyeballer was requested to analyse the model 86.8% of the time.

It was also asked to repair the model 76.3% of the time.

The following input model types were used:
    N3, used in 52.6% of models.
    RDF/XML, used in 47.3% of models.
The following report formats were used:
    Text, used in 81.5% of requests.
    N3, used in 18.4% of requests.
The popularity of Inspectors can be seen below:
    prefix was requested in 100.0% of eyeballings.
    literal was requested in 97.3% of eyeballings.
    cardinality was requested in 100.0% of eyeballings.
    property was requested in 94.7% of eyeballings.
    consistentType was requested in 97.3% of eyeballings.
    class was requested in 97.3% of eyeballings.
    allTyped was requested in 94.7% of eyeballings.
    vocabulary was requested in 94.7% of eyeballings.
    uri was requested in 100.0% of eyeballings.
Finally, the popularity of the assumes:
    dc-all was assumed in 34.2% of eyeballings.
    owl was assumed in 60.5% of eyeballings.
    dc was assumed in 2.63% of eyeballings.
    dcterms was assumed in 2.63% of eyeballings.
```

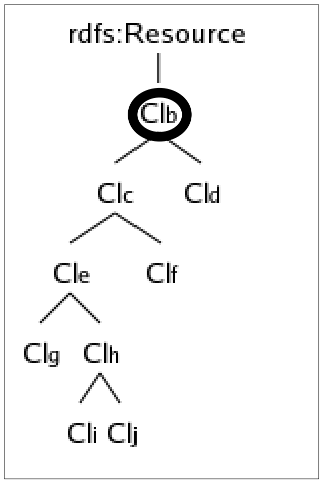
Appendix A - Detailed Repair Methodology

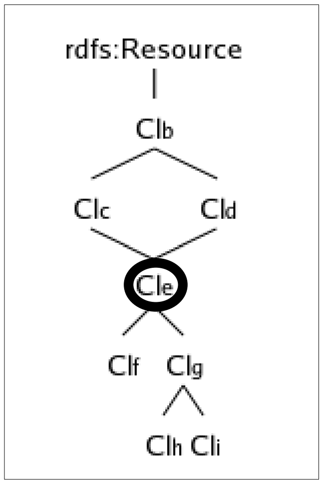
This Appendix details the repair methodology for each of the repairs that E2.1 carries out. For ease of understanding, an example of input and output is given for each. The examples are colour coded, such that statements to be added by the Doctor are **green** on the output, any to be removed are **red** on the input, and those for the Doctor to modify are **blue** on both input and output.

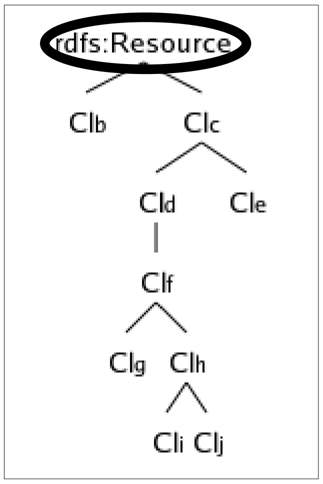
AllTyped

Description

This tool attempts to generate an `rdf:type` statement for each reported node. This is based on what is described loosely as the 'subbiest-subclass'; a class in the class hierarchy which is furthest 'down' from `rdfs:Resource` and has no possibility of being wrong. To put it differently, if the hierarchy of classes is represented as a graph (drawn as tree-like as possible) modelling `rdfs:subClassOf` relationships, the subbiest-subclass is located at the lowest point on the graph where the breadth of the tree is only one and no leaves exist higher up. By way of illustration (for an arbitrary hierarchies of classes Cl_x);







In this way, the Analysis will adhere to the 'maximally descriptive; maximally accurate' design goal. When the subbiest-subclass has been found, a statement can be added to the model setting the `rdf:type` for the reported node to this subclass.

Example

Input

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .
```

```
        :spoo  rdfs:subClassOf rdfs:Datatype ;
               rdfs:subClassOf rdf:List ;
               rdfs:subClassOf rdf:Statement ;
               rdfs:subClassOf rdfs:Literal ;
               rdfs:subClassOf rdfs:ContainerMembershipProperty .

        :flarn rdfs:subClassOf :spoo .
        :splee rdfs:subClassOf :spoo .

        :mine  rdf:value "1829" .
```

Output

```
        @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
        @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
        @prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

        :spoo  rdfs:subClassOf rdfs:Datatype ;
               rdfs:subClassOf rdf:List ;
               rdfs:subClassOf rdf:Statement ;
               rdfs:subClassOf rdfs:Literal ;
               rdfs:subClassOf rdfs:ContainerMembershipProperty .

        :flarn rdfs:subClassOf :spoo .
        :splee rdfs:subClassOf :spoo .

        :mine  a  :spoo ;
               rdf:value "1829" .
```

Cardinality

Description

There are three avenues of repair to be chosen from by the Analysis – increase the number of usages of property P on node X; decrease the number of usages of property P on node X; increase the value of the `owl:maxCardinality` statement on class C. The first pertains to the situation when the cardinality of P on X is less than the `owl:minCardinality` of P on C. The second and third pertains to the opposite; the cardinality of P on X is greater than the `owl:maxCardinality` of P on C.

Adding usages of P on X is a simple affair; the Doctor may add statements with subject X and predicate P. The object that is used should be a blank node, as Eyeball Repair has no way of determining a value, literal or otherwise. The user should be aware of this, and able to check it accordingly (ideally leading them to enter data rather than leave the bNode in place). Removing the usages of P is a slightly more complex problem. For this version of Eyeball Repair, the Analysis will decide which statement(s) to remove statistically. It will attempt to determine the most common datatype of literal values of P. It will then proceed to remove usages of P where its datatype is different to the most common form, until either no more statements exist that match this criterion; or enough statements have been removed to bring the cardinality of P on X down to the `owl:maxCardinality` of P on C. If the Analysis finds there are not enough statements to remove to reduce the cardinality of P on X sufficiently, it will instruct the Doctor to increase the `owl:maxCardinality` of P on C by the minimum amount necessary for correctness.

Example

Input

```
      @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
      @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#>.
      @prefix owl:     <http://www.w3.org/2002/07/owl#>.
      @prefix my:      <http://my.domain.com/> .

      my:Mumps
             a rdfs:Class ;
             rdfs:subClassOf [
                              owl:onProperty my:spoo ;
                              owl:minCardinality 1 ;
                              owl:maxCardinality 2
                             ] .
      my:y   a my:Mumps ;
             my:spoo 17 ;
             my:spoo 42 ;
             my:spoo 93 ;
             my:spoo "132" .
```

Output

```
      @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
      @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#>.
      @prefix owl:     <http://www.w3.org/2002/07/owl#>.
      @prefix my:      <http://my.domain.com/> .

      my:Mumps
             a rdfs:Class ;
             rdfs:subClassOf [
                                         owl:onProperty my:spoo ;
                                         owl:minCardinality 1 ;
                                         owl:maxCardinality 3
                                        ] .
      my:y   a my:Mumps ;
             my:spoo 17 ;
             my:spoo 42 ;
             my:spoo 93 .
```

Class

Description

The Analysis will direct the Doctor to add a statement with a minimal framework definition of a Class in the model. It must detect the model profile and define the class accordingly (e.g. XXX `rdf:type` `owl:Class` or XXX `rdf:type` `rdfs:Class` etc.)

Example

Input

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:Room  a      :NoSuchClass ;
       rdf:value "17" .
```

Output

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:NoSuchClass
       a      owl:Class .
:Room  a      :NoSuchClass ;
       rdf:value "17" .
```

ConsistentType

Description

The Analysis for this must first attempt to discern if one `rdf:type` is more common than others for this node. If it is, then it should instruct the Doctor to remove all other `rdf:types` from the declaration of the node. If it is not, then it may remove *all* `rdf:types` from it, instead setting it to be of `rdf:type` a new class, which is defined as the `rdfs:subClassOf` *all* previously held `rdf:types`.

Example

Input

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:Left    a    rdfs:Class .
:Right   a    rdfs:Class .
:B1      a    :Left ;
       :spoo  56 .
:B1      a    :Right ;
       :flarn 19 .
```

Output

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:Left  a      rdfs:Class
       .
:Right a      rdfs:Class
       .
:B1    a       [ a       rdfs:Class ;
                rdfs:subClassOf :Right ;
                rdfs:subClassOf :Left ] ;
       :flarn  19 ;
       :spoo   56 .
```

Literal

Description

The LiteralInspector checks primarily for language code and datatype definitions on literals in the model. If the Analysis finds an `eye:badDatatypeURI` it should first analyse the lexical form of that literal. If it conforms to one of the common xsd types, it may instruct the Doctor to alter the datatype URI to that of the relevant xsd datatype. For this version of the code the Analysis should look for valid integers, decimals, dates, times, dateTimes, and boolean (by means of regular expression matching of the datum's literal lexical form). If none of these match, the tool may fall back to either setting a default language code for this literal, or setting it to a default datatype. This behaviour must be user configurable.

However, the LiteralInspector will only report one instance of a fault. If the same error occurs more than once, it will not report it. What is more, the bad datatype URI that the Inspector reports may exist on a literal with a different lexical form, and which requires a different datatype (see the example below for datatype `my:bad`). Therefore, when the LiteralDoctor has fixed one instance of the fault, it must invoke the LiteralInspector, LiteralAnalysis, and a new LiteralDoctor to inspect, analyse and doctor the model respectively, in order to ensure that there are no further instances of this fault.

Example

Input

```
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix :         <http://eyeball.jena.hpl.hp.com/eg#> .

:mine
      rdf:value     "1829" ,
                    "chat"@en ,
                    "1066"^^<my:bad> ,
                    "sometext"^^<my:bad> ,
                    "alphabetic"^^<xsd:integer> .
```

Output

```
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix :         <http://eyeball.jena.hpl.hp.com/eg#> .

:mine
      rdf:value     "1829" ,
                    "chat"@en ,
                                1066 ,
                                "sometext"^^<xsd:string> ,
                                "alphabetic"^^<xsd:string> .
```

Prefix

Description

This set of modules will perform one of two tasks. It should either remove duplicate prefixes from a model, or replace a namespace prefix. The former may be done entirely arbitrarily: it will keep one namespace <=> prefix mapping, and delete all others. Once again, the latter is the more complex of the two. It must first be configured (by the user) with a collection of known and desired prefixes and namespaces. The Analysis must then decide;

- Is the prefix specified mapped to a URI in our configuration?
  - If so, change the PrefixMapping, and all affected usages of the URI to that defined in the configuration
  - If not, is the URI which the prefix maps to in our configuration?
    - If so, change the prefix to that which the configuration associates with the URI
    - If not, use the value of `eye:Expected` as the URI for that prefix (change the PrefixMapping, and all affected usages of the URI to that defined in the configuration)

It is important to note here that the Doctor must not *just* alter the PrefixMapping if the URI is to be changed. Because of how Jena handles prefixes, the URI must be altered on *all* nodes on which it is used (otherwise the old URI will remain, without any prefix).

Example

Input

```
      @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
      @prefix eg:      <http://eyeball.jena.hpl.hp.com/eg#> .
      @prefix bad:     <http://eyeball.jena.hpl.hp.com/eg#> .
      @prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

      eg:b1   rdf:value   "bloop" .
      bad:b2  rdf:value   "bleep" .
      :b3     rdf:value   "blarp" .
      bad:b5 rdf:value    "blirp" .
```

Output

```
      @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
      @prefix eg:      <http://www.example.org/> .
```

```
@prefix bad:     <http://eyeball.jena.hpl.hp.com/eg#> .

bad:b1  rdf:value   "bloop" .
bad:b2  rdf:value   "bleep" .
bad:b3  rdf:value   "blarp" .
bad:b5  rdf:value   "blirp" .
```

Property

Description

This Analysis uses spellcheck functionality[10]. It first builds two dictionaries of known properties from all .rdf files in the mirror/ directory. The first of these contains the full URI of the property, and the second only the localName part. As a first attempt, the Analysis should attempt to find the property in some namespace in that directory. These are compared based on localName (so if, for example, the Analysis encounters `rdf:integer` it can reasonably change it to `xsd:integer`). If this fails, the Analysis should fall back to spellcheck systems. It will first try and find a match in its dictionary of full URIs (i.e. the property might have been misspelled, and still exists in this namespace, or its namespace was misspelled). If it fails to find a match there, it will try and search the localName dictionary (i.e. the property might be both in the wrong namespace, *and* be misspelled). If it cannot find a match, the repair has failed. The user must be notified of this fact.

Example

Input

```
prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:mine
       :ID          "blobby" ;
       :srting      "1829" .
```

Output

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

:mine
       xsd:ID  "blobby" ;
       xsd:string "1829" .
```

URI

Description

This module uses spellcheck functionality. The current scope of URIAnalysis specifies five faults to detect and repair. These are as follows;

- *eye:unrecognisedScheme* – First of all, attempt to spellcheck the scheme of the URI against a user-configurable list of valid schemes. If this fails, default to a user-configurable default scheme.
- *eye:uriContainsSpaces* – Attempt to separate the domain and local part of the URI. If they cannot be separated, it is not a URL and for the purposes of this analysis, take 'domain part' to mean the entire URI. Delete any spaces from the domain part. Replace those in the local part with "%20" (without the quotes!).
- *eye:uriNoHttpAuthority* – Attempt to replace the broken start of the URI with "http://".
- *eye:uriHasNoLocalname* – Strip off a trailing '/', '\', ' ', '.', ',', or '!' from the the URI.
- *eye:schemeShouldBeLowercase* – Convert the scheme of the URI to lowercase.

Once the URI repair has been specified, it is up to the Doctor to replace all usages of that URI (in the subject or the object of the statement) with the new, repaired one.

Example

Input

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

<hTtP://nowhere-else/>            a       rdfs:Class .
<unknown:scheme>                  a       rdfs:Class .
<http:no-authority>               a       rdfs:Class .
<http://no where/with spaces>     a       rdfs:Class .
<gobbledegook>                    a       rdfs:Class .
```

Output

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .
```

```
        <http://nowhere-else>            a       rdfs:Class .
        <file:///scheme>                 a       rdfs:Class .
        <http://no-authority>            a       rdfs:Class .
        <http://nowhere/with%20spaces>   a       rdfs:Class .
        <gobbledegook>                   a       rdfs:Class .
```

Vocabulary

Description

The VocabularyAnalysis happens in much the same way as that of PropertyAnalysis. The only major difference is that when VocabularyAnalysis finds a replacement URI, it must check for usages of that URI in the predicate and object of statements in the model and fix them all.

Example

Input

```
        @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
        @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
        @prefix owl:     <http://www.w3.org/2002/07/owl#> .
        @prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

        :mine
               rdfs:type owl:itn
              ; owl:srting "string content"
              .
```

Output

```
        @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
        @prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
        @prefix owl:     <http://www.w3.org/2002/07/owl#> .
        @prefix :        <http://eyeball.jena.hpl.hp.com/eg#> .

        :mine
               rdf:type xsd:int
              ; xsd:string "string content"
              .
```

Appendix B - Analysis Report Output

The content of Eyeball Repair's additions to the report should follow the definitions (described in N3) below, with the instance dependent data highlighted in **bold**:

- AllTypedAnalysis

```
[]    eye:repairType eye:addDefaultType ;
      eye:statementAdded
            [ a    rdf:Statement ;
              rdf:object subbiestSubClass ;
              rdf:predicate rdf:type ;
              rdf:subject nodeURI
            ] ;
      eye:repairConfidence eye:low ;
      eye:checkFix rdf:object .
```

- CardinalityAnalysis

```
[]    eye:repairType eye:increaseNumProperties ;
      eye:statementAdded
              [ a       rdf:Statement ;
                rdf:object [] ;
                rdf:predicate propertyURI ;
                rdf:subject subject
              ] ;
      eye:repairConfidence eye:low ;
      eye:checkFix rdf:object .

[]    eye:repairType eye:increaseCardinality ;
      eye:statementRemoved
              [ a       rdf:Statement ;
                rdf:object object ;
                rdf:predicate propertyURI ;
                rdf:subject subject
              ] ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:Statement .
```

```
[]    eye:repairType eye:decreaseNumProperties ;
      eye:statementRemoved
              [ a       rdf:Statement ;
                rdf:object **object** ;
                rdf:predicate **propertyURI** ;
                rdf:subject **subject**
              ] ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:Statement .
```

- ClassAnalysis

```
[]    eye:repairType eye:defineClass ;
      eye:statementAdded
             [ a       rdf:Statement ;
                rdf:object owl:Class ;
                rdf:predicate rdf:type ;
                rdf:subject **classURI**
              ] ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:Statement .
```

- ConsistentTypeAnalysis

```
[]    eye:repairType eye:defineClass ;
      eye:repairConfidence eye:good ;
      eye:checkFix rdf:Statement .

[]    eye:repairType eye:removeType ;
      eye:statementRemoved
             [ a rdf:Statement ;
               rdf:subject **subject** ;
               rdf:predicate rdf:type ;
               rdf:object **object**
             ] ;
      eye:repairConfidence eye:good ;
      eye:checkFix rdf:Statement .
```

- LiteralAnalysis

```
[]    eye:repairType eye:setDatatype ;
      eye:newValue "newDataType" ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:object .

[]    eye:repairType eye:setLanguage ;
      eye:newValue "newLanguage" ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:object .
```

- PrefixAnalysis

```
[]   eye:repairType eye:removeDuplicatePrefixes ;
     eye:repairConfidence eye:good ;
     eye:checkFix eye:namespacePrefix .

[]   eye:repairType eye:replaceNamespace ;
     eye:repairConfidence eye:low ;
     eye:checkFix eye:namespacePrefix .
```

- PropertyAnalysis

```
[]   eye:repairType eye:replacePredicate ;
     eye:newValue "newPredicateURI" ;
     eye:repairConfidence eye:moderate .
```

- URIAnalysis

```
[]   eye:repairType eye:schemeToLowercase ;
     eye:newValue "fixedURI" ;
     eye:repairConfidence eye:good ;
     eye:checkFix rdf:subject .

[]   eye:repairType eye:removeIllegalChars ;
     eye:newValue "fixedURI" ;
     eye:repairConfidence eye:moderate ;
     eye:checkFix rdf:subject .
```

```
[]    eye:repairType eye:formHttpAuthority ;
      eye:newValue "fixedURI" ;
      eye:repairConfidence eye:low ;
      eye:checkFix rdf:subject .

[]    eye:repairType eye:addDefaultScheme ;
      eye:newValue "fixedURI" ;
      eye:repairConfidence eye:low ;
      eye:checkFix rdf:subject .

[]    eye:repairType eye:removeSpaces ;
      eye:newValue "fixedURI" ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:subject .
```

- VocabularyAnalysis

```
[]     eye:repairType eye:replaceURI ;
      eye:newValue "newURI" ;
      eye:repairConfidence eye:moderate ;
      eye:checkFix rdf:Statement .
```

1 *see also:* http://jena.sourceforge.net/Eyeball/

2 *see also:* http://jena.sourceforge.net/

3 For the purposes of this document, “RDF” refers to the collection of W3C standards around the Resource Definition Framework; including RDF, OWL, RDFS etc.
see also: http://www.w3.org/RDF/
http://www.w3.org/2004/OWL/
http://www.w3.org/TR/rdf-schema/

4 *see also:* http://www.w3.org/2001/sw/

5 *see also:* http://www.w3.org/TR/rdf-syntax-grammar/

6 *see also:* http://www.w3.org/DesignIssues/Notation3

7 *see also:* http://www.w3.org/XML/Schema

8 Throughout this document, the namespace <http://jena.hpl.hp.com/Eyeball#> is given the prefix `eye:`

9 The prefix `eyesvc:` in this document maps to <http://jena.sourceforge.net/EyeballService#>

10 Eyeball 2.1 implements spellcheck functionality using the Jazzy Core engine, a Java implementation of the aspell algorithms.
see also: http://jazzy.sourceforge.net/