MT-Redland: An RDF Storage Backend for Movable Type

Gregory Todd Williams

greg@evilfunhouse.com

Abstract. Existing content management systems and weblogging tools are based on rigid database designs that suffer from a lack of flexibility in the face of an ever increasing need for new types of structured data and metadata. By modifying these systems to use RDF as the native storage format, they may be easily extended in ways unimagined by the original designers. I present just such a system, a combination of the Movable Type weblogging tool and the Redland RDF Application Framework, showing examples of how the system can more easily adapt to diverse semantic annotation needs.

1 Introduction

The use of weblogging tools is continually increasing. Many of these tools allow their data to be exported in structured formats such as RSS[2], thus allowing the data to be used in more contexts than just the weblog's HTML pages. Aggregators and other tools utilize the structured nature of RSS to present weblog data in new and useful ways.

With the use of weblogging tools, a large amount of structured data is being created, with much of it being generated with either RDF-based formats, or with formats that can be mapped to RDF. As a result, weblogs represent a growing source of potential data for the Semantic Web. Despite this, most data on weblogs represents information that is in a format inaccessible to Semantic Web tools – namely in natural language. Only the basic structure of a weblog, including such concepts as authors, entries, and categories, is commonly available in a machine-readable form.

MT-Redland is an attempt to make it easier to add semantic data to weblogs without requiring authors to work directly with RDF. The system replaces the built-in storage facilities of the Movable Type weblogging tool with an RDF backend that allows easy extensibility, and provides user interfaces meant to ease the creation of semantic metadata.

2 Background

Movable Type[1] is one of the most popular weblogging tools aimed at authors wanting to host their own weblog. It is available in both paid and free versions, with both being distributed as Perl source code. Having access to the code, it

is possible to easily customize and integrate Movable Type with other products. Like most tools of its kind however, Movable Type's data storage features suffer from a rigid design that make adding semantic data difficult.

Redland[3] is an RDF application framework consisting of RDF parsing, storage, and querying facilities. Though written in C, bindings for several popular scripting languages, including Perl, are available. This makes Redland a good choice for integration with Movable Type.

Though there has been some work done on integrating semantic web features into Movable Type[11, 7, 14], most of this effort has been in making small additions to Movable Type's template features, and allowing existing data to be exported in new ways. There has been no effort spent on realizing the potential of RDF as a storage mechanism, or on the potential of plugins to allow *creation* of new semantic data.

3 Methodology

Movable Type has a clean, object-oriented design that provides the modularity crucial to easily adding new storage backends. In addition, Movable Type has a rich plugin API for enhancing the features of built-in classes.

MT-Redland is divided into two parts: the storage backend and a set of plugins to manage new data. The storage backend consists of a Redland RDF store, and manages the mapping of classes, objects and attributes to *rdf:type*s, predicates and triples. The plugins supplement the system with interfaces for adding semantic data and functionality to utilize the Redland backend by adding the data to weblog entries.

These plugins aim to use the Movable Type API to make the creation of semantic metadata easy for both weblog authors and developers. MT-Redland includes a framework for adding new user interfaces to Movable Type for collecting metadata. It also includes sample plugins for adding common metadata to weblog entries, such as topics, reviews, and locations. Finally, Movable Type template tags are included for the export of RDF data as RSS 1.0; other syndication formats may be used as normal by construction of appropriate template files.

3.1 Storage Backend

Movable Type's framework maps all database structures to Perl classes and implements the database access with a modular system of Object Drivers. Movable Type ships with Object Drivers for common relational databases (MySQL, PostgreSQL, and SQLite) as well as BerkeleyDB. Classes exist for the basic types of data stored in the system: Authors, Weblogs, Entries, Categories, and so on. Typically (in the relational Object Drivers) these classes all represent database tables with object attributes as columns.

All the code necessary to interface with a particular type of database is contained in a single Object Driver class that implements a common interface

for loading, creating, updating, and removing objects. This design makes the process of adding support for new databases relatively simple.

MT-Redland implements the Object Driver interface by mapping classes and attributes directly to RDF triples. Each object becomes an RDF resource (typically a blank node) with an *rdf:type* of the object's class, and each attribute is attached to the resource with a predicate for that attribute. For example, the MT::Blog class defines an object to represent a single weblog in the system. The class defines attributes such as *name* and *url*. The mapping of the Blog class to RDF would be as follows:

```
<mt:Blog xmlns:mt="http://kasei.us/e/ns/mt/"
         xmlns:blog="http://kasei.us/e/ns/mt/blog#">
  <blog:name>My Weblog</blog:name>
  <blog:url>http://example.com/blog/</blog:url>
  ...
</mt:Blog>
```

The *rdf:type* and predicates are defined by each class with the new methods *get_rdf_class_node* and *get_rdf_column_node*. A default for each method is defined in the Object superclass, allowing classes that have no special needs to inherit the necessary code. Also defined by each Object class is a *get_rdf_column_constructor* method that defines the constructor used to create the Nodes for attribute values. This allows classes to specify the type of each attribute: Resource, Literal, or Blank Node. Using this approach, the MT::Blog class could define the *url* attribute more appropriately as a resource by implementing the column constructor method:

```
sub get_rdf_column_constructor {
  my ($self, $col) = @_;
  return 'new_from_uri' if ($col eq 'url');
  return $self->SUPER::get_rdf_column_constructor( $col );
}
```

This would produce the RDF:

```
<mt:Blog xmlns:mt="http://kasei.us/e/ns/mt/"
         xmlns:blog="http://kasei.us/e/ns/mt/blog#">
  <blog:name>My Weblog</blog:name>
  <blog:url rdf:resource="http://example.com/blog/" />
  ...
</mt:Blog>
```

Classes may define a custom *rdf:type* for their objects with the *get_rdf_type_node* method. Since an Entry in Movable Type can be thought of as equivalent to an RSS Item, the MT::Entry class might define a custom *get_rdf_type_node* method:

```
sub get_rdf_type_node {
```

```
  return RDF::Redland::Node->new_from_uri(
    'http://purl.org/rss/1.0/item' );
}
```

An Entry also shares several attributes with elements from RSS; the column predicates may similarly be mapped with the *get_rdf_column_node* method. Since mapping attributes to existing RDF predicates is a common task, MT-Redland defines *get_rdf_column_node_map* as a shortcut method. It may be used to define a mapping from attributes to predicate URIs:

```
sub get_rdf_column_node_map {
  return ( title      => 'http://purl.org/rss/1.0/title',
           created_on => 'http://purl.org/dc/elements/1.1/date' );
}
```

Finally, a class may define a *get_rdf_node* method if the default blank node representation is not sufficient. An Entry, now defined as an RSS *item*, ought to refer to the entry directly. Thus, the *get_rdf_node* method may override the default to return a Resource node:

```
sub get_rdf_node {
  my $entry = shift;
  return RDF::Redland::Node->new_from_uri( $entry->archive_url );
}
```

An Entry object will now map to a reasonably sensible version of RSS:

```
<rss:item rdf:about="http://example.com/blog/welcome.html"
          xmlns:rss="http://purl.org/rss/1.0/"
          xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rss:title>Welcome to my Blog</rss:title>
  <dc:date>2005-04-01T13:00:00+00:00</dc:date>
  ...
</rss:item>
```

The MT-Redland backend is designed to give as much flexibility as possible to the developer in mapping the Movable Type objects to appropriate RDF. With the methods described above, MT-Redland is able to utilize common RDF vocabularies in mapping the Movable Type objects: Entries use RSS 1.0[2], Authors use FOAF[6], Categories use SKOS[12], and Dublin Core is used for common attributes such as dates. In these mappings, OWL[9] Functional and Inverse Functional Properties are used wherever possible so that the exported RDF data can be meaningfully merged with existing RDF data.

It is hoped that the MT-Redland backend API, combined with Movable Type's callback mechanism, will be able to address any RDF mapping requirement that may be desired, and also provide the ability to integrate with existing vocabularies where appropriate.

3.2 Modifying Movable Type's CMS Editing Interface

Creating a system of allowing plugins to *easily* add metadata has two goals: integrating easily with Movable Type and providing a simple interface for plugin developers to use. The system must integrate with Movable Type in such a way that core Movable Type files are not modified, and doesn't make end users spend time modifying existing templates and code simply to use the plugin. This eases installation of plugins and insures that updates to the Movable Type code do not overwrite the plugin's code. Likewise, creating a simple plugin API allows developers to concentrate on the functionality of their plugins while being able to develop the plugins quickly.

MT-Redland deals with the first goal by allowing plugins to add user interfaces programmatically – without changing any of Movable Type's code or templates. The system exposes two methods for use by a plugin that allow the plugin to add HTML header and body content to the Movable Type editing interface. The HTML headers are useful for adding any styling and scripting to the interface, while the body content allows new HTML form fields to be added.

The HTML content is added to the editing interface by overriding parts of Movable Type's page building code. To avoid the use of fragile regular expressions for inserting the HTML body content directly into the page, a Javascript block is inserted into the HTML header and content is then inserted into the page using an *onload* handler (on the client side) and Javascript's DOM interface.

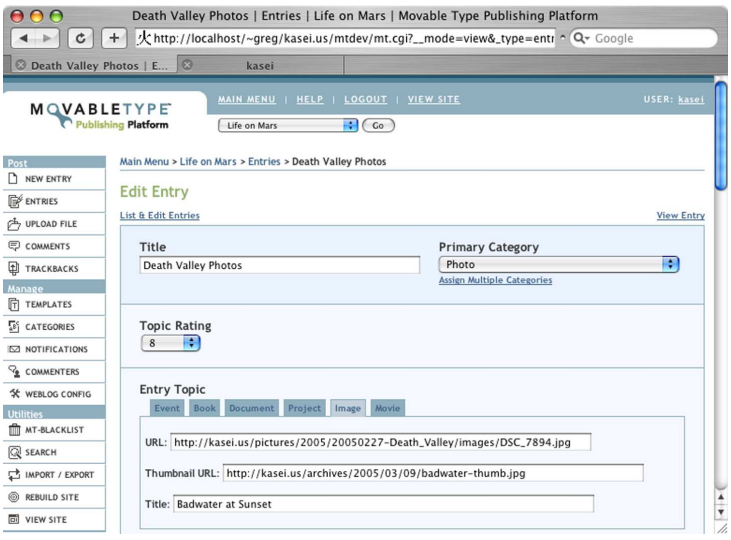

Fig. 1. Movable Type's Editing Interface with fields added for Topic and Review data.

The second goal of providing a simple API for plugin developers is aided greatly by the design of Movable Type's form handling code. During processing

of the form data from the object-editing interface, Movable Type looks up the attributes of the object and uses those values as keys into the form data. As a result, plugin code that adds form fields for custom data must simply add values to the object's list of attributes corresponding to the new form fields and Movable Type will retrieve the form data and place it in the object.

By utilizing the Movable Type callback framework, it is possible to define a *pre_save* handler that intercepts the object just prior to being saved to the database. In this case, the handler removes the custom attributes from the object, and deals with them in an appropriate manner (typically adding statements directly to the Redland model).

MT-Redland currently only supports Movable Type's web-based editing interface; lacking any current support or common approach to extensibility in the weblog-editing application space, MT-Redland does not address the changes that would be required to modify Movable Type's XML-RPC interface. However, using the XML-RPC interface should result in the same functionality as if MT-Redland was not installed. That is, none of the MT-Redland plugins will be available for metadata creation, but Movable Type will otherwise continue to function normally.

3.3 Using Movable Type's Plugin System

MT-Redland includes several plugins that make use of the frameworks described in the previous section. These plugins add useful metadata to weblog entries while being relatively simple to implement. The most significant plugins deal with adding metadata to describe the topic of an entry (using the *foaf:topic* predicate). Individual plugins define specific types of topics that may be described, such as images, books, and events. Each topic plugin defines characteristics of its topic and the appropriate user interface for describing it.

As an example, the Book Topic plugin allows a weblog entry to be described as discussing a specific book. The plugin uses the Reading List Schema[10]. Using MT-Redland's topic API, the code for implementing the book topic, including user interfaces for adding the information, turns out to be rather simple.

First, RDF nodes are defined for the predicates and *rdf:type* that are used:

```
my ($t_book, $p_topic, $p_isbn, $p_title, $p_type) =
  map { RDF::Redland::Node->new_from_uri($_) }
    qw( http://purl.org/net/schemas/book/Book
        http://xmlns.com/foaf/0.1/topic
        http://purl.org/net/schemas/book/isbn
        http://purl.org/dc/elements/1.1/title
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type );
```

Next, the user interface for adding the book's ISBN is defined:

```
MT::Plugins::Redland::CMS::Topic->install_topic('Book', sub {
  my $app      = shift;
```

```
  my $q        = $app->{query};
  my $id       = $q->param('id');
  return unless ($q->param('_type') eq 'entry');
  my $isbn     = '';
  if ($id) {
    my $entry  = MT::Entry->load($id);
    $isbn      = $entry->isbn if ($entry);
  }
  return qq(<p><label for="isbn">ISBN</label>: )
       . qq(<input name="isbn" id="isbn" value="${isbn}" /></p>);
});
```

The MT::Entry object needs to have a new column to hold the ISBN, so the *column_names* method is overloaded:

```
my $orig  = MT::Entry->can('column_names');
*MT::Entry::column_names  = sub {
  return (@{ $orig->(@_) }, 'isbn');
};
```

Finally, object callbacks are defined to load the book data after object construction, and to save the RDF triples for the *foaf:topic* and book data before the object is saved:

```
MT::Entry->add_callback("post_load", 4, $plugin_data, sub {
  my ($cb, $terms, $args, $entry) = @_;
  my $model    = $entry->driver->model;
  my $topic    = $entry->rdf_topic_of_type($t_book);
  return unless ($topic);
  my ($isbn)   = $model->targets($topic, $p_isbn);
  $entry->isbn($isbn->literal_value) if ($isbn);
});

MT::Entry->add_callback("pre_save", 4, $plugin_data, sub {
  my ($cb, $entry)  = @_;
  my $model    = $entry->driver->model;
  if (my $isbn = delete $entry->{'column_values'}{'isbn'}) {
    my $node   = $entry->get_rdf_node;
    my $l_isbn = RDF::Redland::Node->new_literal($isbn);
    $entry->remove_rdf_topics_of_type($t_book);
    my ($book) = $model->sources($p_isbn, $l_isbn);
    unless ($book) {
      $book    = RDF::Redland::Node->new();
      $model->add_statement( $_ ) for
        ( RDF::Redland::Statement->new($book, $p_type, $t_book)
          RDF::Redland::Statement->new($book, $p_isbn, $l_isbn) );
    }
```

```
    $model->add_statement(
      RDF::Redland::Statement->new($node, $p_topic, $book) );
  }
});
```

This results in RDF describing the book as topic:

```
<rss:item rdf:about="http://example.com/blog/hacking.html"
          xmlns:rss="http://purl.org/rss/1.0/"
          xmlns:foaf="http://xmlns.com/foaf/0.1/"
          xmlns:book="http://purl.org/net/schemas/book/">
  <rss:title>Hacking Movable Type</rss:title>
  <foaf:topic><book:Book book:isbn="076457499X" /></foaf:topic>
  ...
</rss:item>
```

By making the process of creating new plugins for MT-Redland simple, it is hoped that developers will be able to easily add plugins for new types of semantic data as new RDF vocabularies emerge.

4 Results and Analysis

The MT-Redland project has been successful in integrating Redland and Movable Type, and making use of the flexibility that an RDF backend provides. Despite this, there are several issues that present challenges to adoption of MT-Redland as a good first-choice for a Movable Type Object Driver.

4.1 Performance Issues

Compared to the relational database Object Drivers, MT-Redland is noticeably slower. The core object classes in Movable Type all map directly to fixed schemas. This makes them perfectly suited for implementation in a relational database where fields are typed and may be indexed efficiently. By using a triple-based storage framework like Redland, this efficiency is lost to flexibility. Since Redland currently lacks typing in its storage facilities, the literals that constitute object attributes are stored as strings, regardless of their actual data type.

The lack of sorting in Redland, and in particular Redland's current implementation of the SPARQL[13] query language, has dramatic speed implications for a system, such as Movable Type, that relies on sorting. Since the data literals that need to be sorted are stored as strings in Redland, MT-Redland must resort to an inefficient sorting routine that attempts to sort string data as expected, regardless of the intended type of the data.

Sorting has been accepted by the W3C Data Access Working Group as an objective of RDF query languages, but has not yet been addressed in the SPARQL query language[8, 13].

4.2 Security and Privacy Concerns

Currently, the RDF MT-Redland stores in the Redland model is not suitable for export to the Web without taking into consideration the security implications. The current implementation of user authentication in Movable Type uses the *crypt* one-way hash function to store user passwords. [5] discusses the security issues with *crypt*-based authentication in the context of UNIX systems. If the RDF containing the *crypt*ed passwords of the weblog were made available on the Web, the problems noted in [5] would be compounded by the availability of the *crypt*ed passwords to the entire population of the Web.

Also of concern when making RDF public is the privacy of the users who's data is represented. In Movable Type, email addresses of the authors of a weblog and people who comment on the weblog are stored in the database. Often, if a URL is provided by a commenter, only the URL and not the email address is made visible in the HTML. By making public the RDF containing these email addresses, the privacy of the users is compromised. To alleviate this problem, a one-way hash of the email addresses may be performed and added to the RDF using the *foaf:mbox_sha1sum* predicate (or similar).

MT-Redland addresses these security and privacy issues by not exporting any RDF triples that use default predicates. In other words, only object attributes that have been mapped onto external predicates are publicly available.

4.3 Uses of MT-Redland's Metadata

There are clear benefits to having Movable Type's data in an RDF database that may be queried using an RDF query language such as SPARQL. By allowing users to access the RDF through exported RSS, and indirectly with SPARQL-based searches, the value of the new semantic data may be seen.

MT-Redland allows objects to be loaded from the database with a SPARQL query. With only minor changes to Movable Type's search interface, weblog entries may be found based on this structured search data. For example, using the included plugins, it is possible to: find all entries about an image that has been rated higher than seven out of ten (Figure 2); find all entries that link to a particular page or site; or find all entries that a specific person has commented on.

The benefit of allowing SPARQL queries to be run against the weblog data may be increased by returning search results in formats other than HTML. Again, with very minor changes to Movable Type's search code, results of a SPARQL query may be returned in any number of formats including RSS or SPARQL's Variable Binding Results XML Format[4].

Systems with the ability to utilize a weblog's semantic data are much more powerful than their regular counter parts, which must rely on keyword-based searches. The availability of new semantic data will allow users to find and make use of relevant data quickly and accurately.

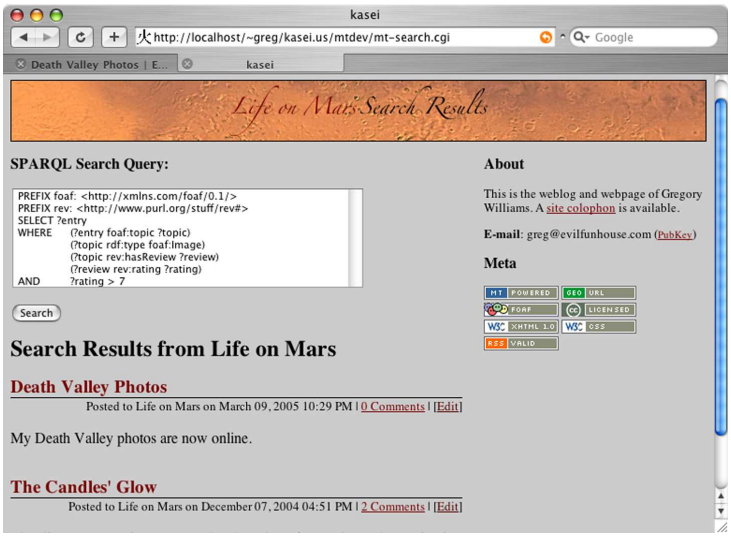

Fig. 2. A weblog search using a SPARQL query.

5 Conclusion and Future Work

MT-Redland is an ongoing project aimed at aiding in the creation of new semantic data and enhancing the extensibility of Movable Type. Despite issues of scaling and privacy, MT-Redland already shows the possibilities of integrating RDF backends with existing weblogging tools. As weblogs and other content management tools become more widespread on the Web, the ability to store and use their data in a structured manner, while remaining extensible, will become increasingly important.

MT-Redland shows how existing tools may be modified to work with the Semantic Web, while retaining their core design and functionality. This approach ought to be available when using triple stores other than Redland and other content management systems than Movable Type. MT-Redland's mapping of relational tables to RDF predicates and statements can be used to easily convert existing relational data to RDF. The higher-level design pattern of implementing all database code in a single Object Driver, however, may not always map easily to an existing code base. In these cases, converting the existing system to use an RDF backend may present more challenges than MT-Redland faced.

To extend the usefulness of MT-Redland, it is desirable to allow the system's RDF data to be exported beyond the boundaries of the system, or to import external RDF that could augment the system's existing RDF. To make this feasible, the security and privacy concerns mentioned above need to be addressed. Potential solutions are to always use hashed values of sensitive object attributes

(email addresses) and to rely on a distributed authentication scheme such as TypeKey[1], which already sees use in some sections of Movable Type.

A second issue that must be addressed before MT-Redland's RDF could be merged with external RDF, and one which might involve more serious structural changes to Movable Type or MT-Redland, is the reliance of Movable Type's classes on integers to uniquely identify objects. This reliance makes sense when using a relational database or BerkeleyDB, where a Movable Type installation can rely on a private database space (BerkeleyDB files or database tables), but poses serious problems to merging RDF from multiple Movable Type installations into one RDF model. The use of statement provenance (using contexts in Redland), may provide a solution to this problem. Other possible solutions involve using predicates that are private to the URL space of the weblog, or modifying the Movable Type code to accept non-integer values as unique identifiers.

References

1. Movable type personal publishing system. http://www.sixapart.com/movabletype/.
2. RDF Site Summary (RSS) 1.0. http://web.resource.org/rss/1.0/spec, May 2001.
3. Dave Beckett. Redland rdf application framework. http://librdf.org/. ILRT, University of Bristol.
4. Dave Beckett. SPARQL variable binding results xml format. http://www.w3.org/TR/rdf-sparql-XMLres/.
5. Walter Belgers. UNIX password security. Technical report, 1993.
6. Dan Brickley and Libby Miller. FOAF vocabulary specification. http://xmlns.com/foaf/0.1/.
7. Riccardo Cambiassi. Mtfriends movable type plugin. http://www.sixapart.com/pronet/plugins/plugin/foaf.html.
8. Kendall Grant Clark. RDF data access use cases and requirements. http://www.w3.org/TR/2005/WD-rdf-dawg-uc-20050325/.
9. Mike Dean and Guus Schreiber. OWL web ontology language reference. http://www.w3.org/TR/owl-ref/.
10. Leigh Dodds. Reading list schema. http://www.ldodds.com/schemas/book/.
11. Seth Ladd. Sha1 movable type plugin. http://www.sixapart.com/pronet/plugins/plugin/sha1.html.
12. Alistair Miles and Dan Brickley. SKOS core guide. http://www.w3.org/2004/02/skos/core/guide/.
13. Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for rdf. http://www.w3.org/TR/rdf-sparql-query/.
14. Gregory Todd Williams. MTCommentIcon. http://kasei.us/code/mtcommenticon/.

[1] http://www.sixapart.com/typekey/