# A Pattern for Domain Specific Editing Interfaces Using Embedded RDFa and HTML Manipulation Tools.

Rob Styles[1], Nadeem Shabir[1], and Jeni Tennison[2]

[1] Talis rob.styles@talis.com
[2] Talis nadeem.shabir@talis.com
[3] Jeni Tennison Consulting jeni@jenitennison.com

**Abstract.** Many applications have the need to provide end users with editing capabilities. Often the nature of the domain and the user's workflow require a specialised editing interface. This paper describes the approach taken to building a specialised editing interface for academic resource lists and extracts the core aspects to allow others to apply the same pattern to their own applications. The solution described uses commonly available HTML manipulation tools and rdfQuery, a javascript RDFa library, to maintain an RDF model embedded in the page. This allows the application to provide a document style editing model over RDF data.

## 1 A Tool for Managing Academic Resource Lists

Talis Aspire is a SaaS (Software as a Service) application for managing lists of academic resources [1]. These resource lists are a key part of course content given to students to help them study more effectively. It provides a reading interface for students as well as a powerful editing interface for the lists' maintainers. This paper focusses on the approach used to implement the editing interface for list maintainers and how that can be seen as a pattern for other domain specific editing interfaces.

The view of the list when reading (Fig. 1) shows the list in the format students are used to receiving, similar in structure to a document. This can be cleanly printed for offline reference and each list, section and item have Cool URIs [10] to allow for bookmarking. Each item then has additional information and functionality behind simple links.

The editing view (Fig. 2) is used by academics and other faculty staff. Common tasks when editing lists are to add or remove particular items, to group items together into sections and to annotate the items to provide additional guidance to the students. The most common mental model of the resource list is that of a document, with the associated mental models that come from editing documents using typical office software.

This lead us to have a number of requirements for the editing interface.

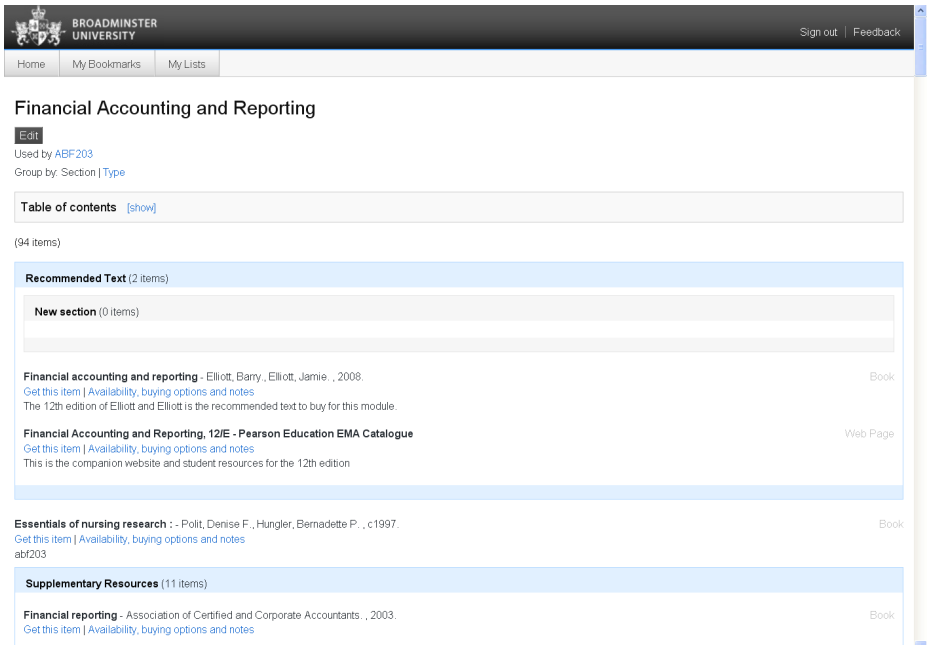


**Fig. 1.** Talis Aspire: Student View of an Academic Resource List

Firstly, the mental model of a document requires that a number of edits can occur before the user saves the document with an explicit 'save' action. This matches the model of typical office software.

Secondly, the document being edited should be visually the same as the document when read, allowing the user to see what the result of their actions will be.

Thirdly, the editing interface should be a dynamic experience, supporting efficient editing with easy-to-use mechanisms for completing the common tasks. In practice this would mean an AJAX interface.

Fourth, the application allows for concurrent editing, something not in most people's mental model of a document, so we wanted to make saves resilient, where possible, to concurrent edits.

Fifth, the underlying data is stored as RDF and may be annotated and augmented by data the application does not know about. Edits should not be destructive to this data.

It is in this context that we set about designing a technical approach that met all five goals and was as simple as possible to implement.

HTML representations of underlying data are usually specialised interfaces designed to show the data in the best way for a human with a particular task to perform. An editing interaction designed for the user must integrate the editing function with this HTML representation and reflect changes the user makes in

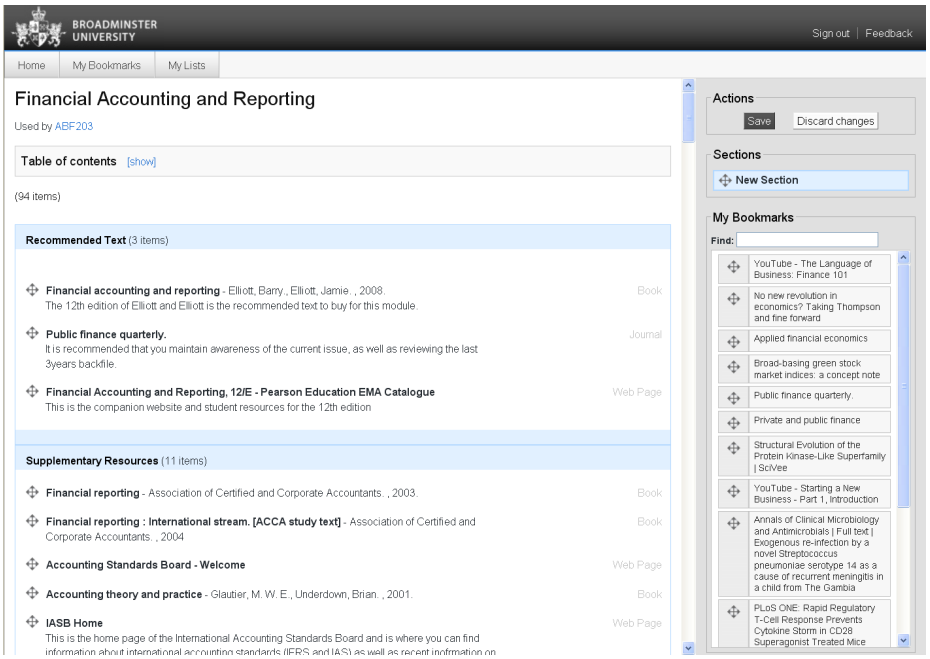


**Fig. 2.** Talis Aspire: Editing View of an Academic Resource List

that representation. In situations where the data is posted back to the server to update the underlying model the server can simply re-render the HTML based on the new model. If a more dynamic editing interface is to be provided using DHTML techniques then the HTML DOM must be manipulated directly and a number of changes are submitted to the server in one go.

There are a number of mature libraries that allow for manipulation of the DOM based on user input, including support for text editing, drag-and-drop, enumerated options and more. Those the team felt familiar with are Prototype[4], JQuery[5], Scriptaculous[6] and YUI[7]. Any one of these libraries could provide the basis of the editing facilities we aimed to provide.

The approach we developed uses client-side javascript to change a read-only page into an editing mode. On switching into edit mode the first thing that happens is that the page is parsed to extract a copy of the initial model, which has been embedded in the page using RDFa [5]. This is kept as the starting point from which to derive a list of changes. After the initial model has been captured we use normal DHTML and AJAX libraries, in our case we used Prototype

[4] http://www.prototypejs.org/
[5] http://jquery.com/
[6] http://script.aculo.us/
[7] http://developer.yahoo.com/yui/

and Scriptaculous, to provide text editing, drag-and-drop re-ordering, removal of elements and drag-and-drop of new items onto a list.

We arrived at our approach independently and later found a great deal of inspiration in the work done on Ontowiki by Dietzold et al [6]. This work attempted to provide generic editing tools for RDFa within a wiki page. The notion of a suite of generic editing tools for RDFa is very appealing, but the user experience requirements for our application called for a specialised editing interface. The intention was to provide editing of structured data while maintaining the list view that users are accustomed to and would be expecting.

## 2 The Solution

The data underpinning Talis Aspire is stored as RDF in a Talis Platform Store [7], making use of several ontologies including AIISO[8], Resource List[9], Bibliontology[10], SIOC[11]and FOAF[12].

Our solution for editing embeds RDFa within the page in such a way that manipulation of the HTML DOM results in consistent and coherent changes to the embedded model.

Server-side the pages are rendered by PHP from data stored natively as RDF in a Talis Platform store. The application does content negotiation to return HTML or rdf/xml to the client. If the response is HTML then RDFa is included in the page markup.

The HTML manipulation during editing uses a combination of custom code for edit dialogs and scriptaculous and prototype to provide drag-and-drop support.

The extraction of the model from RDFa within the page is performed by rdfQuery[13] which also uses JQuery for some page parsing tasks. Other libraries, such as Ubiquity[14] could be used as well, but our familiarity with jQuery and prototyping experience with rdfQuery led us to select that.

Figure 3 shows a very much simplified diagram of the thin layer of application code for Talis Aspire, above a Talis Platform Store, accessed via the internet by both staff and students using standard browsers.

The key to the simplicity of this approach was to recognise that all we needed for an editing session was a pair of before and after models. These models can then be used to generate a changeset[15] that persists the changes to underlying storage. The structure of changesets is such that they also provide an ideal mechanism for creating an auditable change history.

[8] http://purl.org/vocab/aiiso
[9] http://purl.org/vocab/resourcelist
[10] http://bibliontology.com/
[11] http://sioc-project.org/
[12] http://www.foaf-project.org/
[13] http://code.google.com/p/rdfquery/
[14] http://code.google.com/p/ubiquity-rdfa/
[15] http://n2.talis.com/wiki/Changeset_Protocol

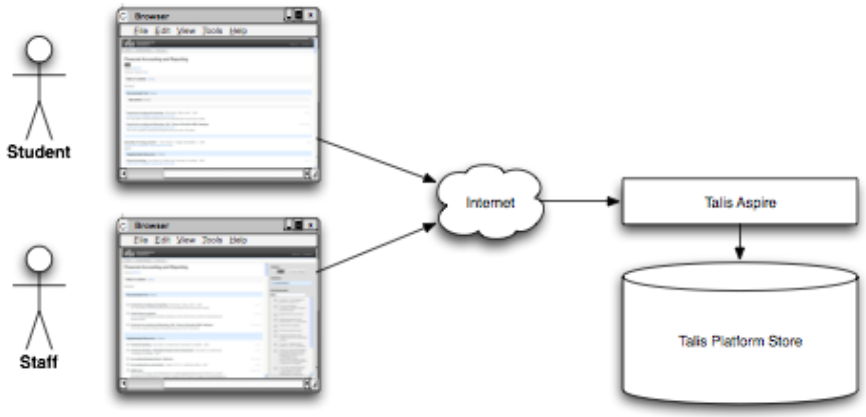


**Fig. 3.** Talis Aspire: Simplified Application Architecture

The changesets provide the equivalent of a Diff in source code configuration management. It is easy to conceive how this could be used to provide a feature to compare versions at some time in the future.

The client side code for saving a list is rather trivial, needing only to extract the final model and package that along with the initial model for submission to the server:

```
function saveListChanges() {
    showModalLayer('Saving list changes - please wait...');
    reindexSequences();
    var newModel = toRDFXML();
    var params = 'oldmodel=' + oldModel + '&newModel=' + newModel;

    new Ajax.Request(listUri,
    {
        method: 'post',
        parameters: {old_model: oldModel, new_model: newModel},
        contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
        onSuccess: function(transport)
        {
            redirect(listUri);
        },
        on409: function(transport)
        {
            alert('Someone else has changed the data...');
            redirect(listUri);
        },
```

```
        onFailure: function(transport)
        {
            alert('There was a problem... ' + transport.responseText);
            redirect(listUri);
        }
    });
}
```

Server-side we have code that uses ARC2[16] and Moriarty[17] to generate the changesets from the before and after models and submit to the underlying store. This code is, again, small and relatively trivial. One of the reasons that the changesets are easy to produce, reconcile and apply is that we do not use blanks nodes. Blank nodes would require more knowledge of the model and use of inverse functional properties. This would increase complexity.

Next we look at how this solution meets our stated goals.

### 2.1 Goal One: Explicit Save

Batching several edits from a single editing session together maintains a mental model for the user that is consistent with that of a document. This was a key factor in our user interaction design. Because edits are all held on the client until the user explicitly submits them the application behaves as expected. The client code is able to do this very simply as the model is embedded in the HTML, the same actions that change the DOM also change the model.

### 2.2 Goal Two: WYSIWYG Editing

The use of the existing HTML, the same HTML as is rendered for the reader view, gives us the same look and feel while editing as while reading. A few subtle changes are made; links disabled and some editing interface elements discreetly added.

By embedding the model within the HTML at read time we negate the need for a separately rendered view for editing, requiring only that the client supports javascript.

### 2.3 Goal Three: Dynamic Interface

We had to worry very little about finding or writing RDFa aware tools because RDFa is embedded using attributes and non-RDFa aware tools simply ignore attributes they don't recognise. The interface is provided by commonly available HTML manipulation libraries. This allowed us to provide a richer interface than we would otherwise have been able to build.

[16] http://arc.semsol.org/
[17] http://code.google.com/p/moriarty/

### 2.4 Goal Four: Concurrent Editing

By having the client maintain both the before and after models for an editing session it becomes possible to detect when different users have made changes concurrently. It is not possible to detect this when submitting the after editing model alone without the introduction of timestamps. This collision detection allows the application to make appropriate choices about how it deals with the conflict.

As the changeset is directly analogous to the diff patches generated by configuration management tools it is clear that non-conflicting changes to the same model can be reconciled and applied, while conflicting changes can be identified and the three states, before editing, edit one and edit two, can be offered to the user for the conflict to be resolved. We currently apply non-conflicting changes and reject conflicting changes with the first set of changes submitted winning.

An alternative approach would have been to implement locking on lists during editing. The benefits of concurrent editing for non-conflicting changes, and the costs of implementing a lock cleanup and reclaim mechanism where lists have been left locked unintentionally mad this unappealing.

### 2.5 Goal Five: Extensible Data

Because the solution uses the changeset protocol to update only the resources and properties it knows about, rather than replacing resources and all of their properties, data that the application doesn't know about is left untouched. This, again, comes from having both the before and after models available to calculate a delta rather than simply performing a replacement.

### 2.6 Complication: RDF Sequences

The ordering and re-ordering of elements in the list posed a particular problem. We model the order of items on lists using RDF Sequences.

```
<http://lists.broadminsteruniversity.org/lists/abf203>
    sioc:name "Financial Accounting and Reporting" ;
    resource:contains <http://lists.broadminsteruniversity.org/items/abf203-1>,
    [...snip...] ;
    a rdf:Seq, resource:List ;
    rdf:_1 <http://lists.broadminsteruniversity.org/sections/abf203-1> ;
    rdf:_2 <http://lists.broadminsteruniversity.org/items/abf203-9> ;
    rdf:_3 <http://lists.broadminsteruniversity.org/sections/abf203-2> ;
    rdf:_4 <http://lists.broadminsteruniversity.org/sections/abf203-3> ;
    rdf:_5 <http://lists.broadminsteruniversity.org/sections/abf203-16> ;
```

We render these in the DOM using the explicit ordering predicates of rdf:_1, rdf:_2 etc. The obvious implication of this is that now the re-ordering of the DOM is no longer enough to create the equivalent re-ordering in the model. We solved this by triggering a function to re-order the rdf:Seq predicates before extracting the post-editing model.

```
function reindexSequences()
{
var containerCuries = new Array();

$$('span[class="sequenceNumber"][rev="rdf:_0"]').each(function(thingToReindex) {
containerCuries[containerCuries.length] = thingToReindex.readAttribute('resource');
});

containerCuries.uniq().each(function(containerToReindex) {
updateSectionSequences(containerToReindex);
});
}
```

One would expect that sequences can be rendered in the RDFa using the sequence dependant rdf:li element as shown by Hausenblas [8]. Unlike the RDF/XML specification [9], however, the RDFa specification [4] does not state that rdf:li should be treated as a special case. As rdfQuery supports the spec accurately it doesn't interpret them specially.

A change to the specification to bring it inline with RDF/XML would allow the model to be re-ordered without changes to the predicates as the sequence is implied by the order of occurrence of the elements within the DOM.

This is one of the complications that results from the immaturity of the domain and the degree to which the specifications have yet to evolve and consolidate.

### 2.7 Complication: Importance of @Rev Attribute

The model uses a number of paired forward and inverse properties. These are used to provide optimisations for querying and convenience when navigating the data as linked data. When an item is removed from a list it is therefore necessary to remove not only the links from the item itself to other elements but also the links from other elements to the item. One example of this is the reference from the list to all of the items it contains, regardless of the level of nesting - this is provided so that the list and all its contents can be retrieved by a single sparql query.

The way we achieved consistency between the DOM editing and the model is through the use of the @rev attribute. This allows us to co-locate statements that reference the item as an object with the item itself in the DOM. This means that removing the item from the DOM also removes the statements that referred to it. This technique removed the need for us to write code to detect statements that had lost their children by the removal of the resource they referenced.

Alternatively we could have made the decision not to render these triples in the RDFa and to use reasoning in the changeset generator or other code to correctly manipulate the model. This would make the RDFa smaller and possibly easier to manipulate at the cost of requiring consuming apps to infer the additional properties from the schema.

As stores become better at reasoning and inference we should see the need for these additional predicates lessen.

### 2.8 Complication: Ignoring Some Changes to the Graph

The final complication we had to deal with is that some aspects of the model are used widely in the graph. The list itself, its sections and its items are a tree structure. Many items can refer to the same underlying academic resource, however. Within the HTML DOM the academic resource will be rendered as child elements of the list item that refers to it. If several list items refer to the same academic resource then the resource will be listed several times.

The problem arises when we wish to delete an item from a list. What we want to delete is the item on the list, i.e. the reference to the underlying academic resource and not the academic resource itself. When the same resource is referenced many times within the same list this does not present an issue as the triples describing the academic resource within the model remain even after one of the items is deleted.

When the academic resource is referenced only once on the list the removal of the item from the HTML DOM also results in the removal of academic resource and would result in a changeset being created to delete the academic resource from the underlying store. As academic resources can be referenced by many lists this deletion would result in other lists referencing a deleted academic resource.

Our solution is to never delete academic resources. This requires introducing knowledge of the model to the code that calculates differences between the before and after models. This is acceptable to us as specialised editing interfaces such as this one are expected to understand the model and limit themselves to changing only those things they are responsible for.

### 2.9 Complication: Performance

The Talis Aspire product works with lists that may contain references to several thousand resources, leading to tens of thousands of triples being embedded within a single HTML+RDFa page. Parsing the RDFa in such a page is a substantial task, and the initial beta version of rdfQuery that we used took tens of seconds to process the very largest of these pages, especially in Microsoft Internet Explorer. This proved to be a useful use-case for the optimisation of rdfQuery, which now parses these pages in just over a second in most cases, which is adequate for our purposes.

### 2.10 Additional Benefit: Re-Use of Data

One of the most attractive things about this approach was that it supported and was supported by our desire to publish RDFa in the pages. With very little out there to consume RDFa and the fact that we already supported content negotiation there was little justification to invest the effort in publishing the RDFa

within the HTML. This approach provided both the simplest implementation for editing as well as a sound reason for including RDFa – which we expect to have additional benefits in future.

### 2.11 Additional Benefit: Re-Use of Code

Very little bespoke code was written for management of editing. Almost all of the bespoke code was written to support specific user interactions during editing. Had RDFQuery supported RDF sequences we would have had to write only the code to post the before and after models to the server.

## 3 Future Work

There are a number of opportunities to simplify the pattern still further as the RDFa spec evolves and as RDF stores and libraries become more capable. We aim to track the situation and make changes as the arise and offer benefit.

There are also opportunities to extend the functionality of the client-side code using the capabilities of rdfQuery more effectively. We currently use it to simply extract the model from the page before and after editing, but it is capable of much much more. It would be possible to use rdfQuery to generate other views of the model from the RDFa, such as a list of sections within the editing interface, counts of items and indicators where the same resource is referenced multiple times. These kinds of interface additions are made much easier by having access to the machine readable model in the client-side code.

## 4 Conclusion

Embedding RDFa within pages provides benefits for the publication of structured data. Beyond that it can also be used to provide a coherent representation of the model that supports editing using tools that are not aware of RDF. The use of these tools can make the provision of specialised, browser-based editing interfaces substantially easier to implement than traditional approaches that involve maintaining a model independently of the HTML DOM.

The client-side, dynamic nature of this approach provides several other benefits beyond implementation simplicity, including a reduction in server interactions and the ability to detect and reconcile concurrent edits.

## References

[1] Clarke, C.: A Resource List Management Tool for Undergraduate Students based on Linked Open Data Principles. In Proceedings of the 6th European Semantic Web Conference, Heraklion, Greece, 2009.

[2] Halb, W., Raimond, Y., Hausenblas, M.: Building Linked Data For Both Humans and Machines Workshop for Linked Data on the Web (2008) http://events.linkeddata.org/ldow2008/papers/06-halb-raimond-building-linked-data.pdf
[3] Tennison, J.: RDFa and HTML5: UK Government Experience http://broadcast.oreilly.com/2008/09/rdfa-and-html5-uk-government-e.html
[4] Adida, B., Birkbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and Processing http://www.w3.org/TR/rdfa-syntax/
[5] Adida, B., Birkbeck, M.: RDFa Primer: Bridging the Human and Data Webs http://www.w3.org/TR/xhtml-rdfa-primer/
[6] Dietzold, S., Hellmann, S., Peklo M.: Using JavaScript RDFa Widgets for model/view separation inside read/write websites In Proceedings of the 4th Workshop on Scripting for the Semantic Web, Tenerife, Spain, 2008.
[7] Leavesley, J. and Davis, I.: Talis Platform: Harnessing Sophisticated Mass Collaboration on a Global Scale. http://www.talis.com/platform/resources/assets/harnessing_sophisticated_mass.pdf
[8] Hausenblas, M.: Writing Functional Code with RDFa http://www.devx.com/semantic/Article/39016
[9] Beckett, D.: RDF/XML Syntax Specification (Revised) http://www.w3.org/TR/rdf-syntax-grammar/
[10] Berners-Lee, T.: Cool URIs don't change http://www.w3.org/Provider/Style/URI