Integration of Metadata in the Web

Jeremy J. Carroll
Hewlett-Packard Labs
Bristol, UK
jjc@hpl.hp.com

ABSTRACT

The metadata module of XHTML2 is a significant improvement on previous approaches to metadata in HTML. A set of metadata attributes can be used throughout the document, and they have a well-defined mapping to RDF, finally permitting the use of RDF within HTML Web pages. This paper shows how to extract the RDF from XHTML2, both with XSLT2 and SAX. It also shows how to add RDF to an XHTML2 document, either into the head, or relevant places in the body. Adding metadata to the body is complicated and computationally difficult; some heuristics are described. These techniques will help bridge the gap between the Web and the Semantic Web.

1. INTRODUCTION

From the beginning of the Web it has been possible to add metadata annotations to the head of HTML documents. However, this has been one of the least successful parts of HTML.

One of the motivations for the Resource Description Framework was to provide improved metadata within HTML documents. However, this goal is still to be achieved, since syntactic problems still remain unfixed after many years of effort. We still have the position where some metadata, in RDF/XML, is added to HTML pages embedded within comments, because otherwise the HTML would become invalid, and the RDF/XML would interfere with the page display.

This paper discusses a new approach to integrating metadata about Web pages into the markup on the page, which is being developed for XHTML2 [2]. This approach is called RDF/A [5], and operates by means of attributes for adding metadata on any XHTML element.

The contribution of this paper is to show how to integrate such XHTML2 metadata with Semantic Web applications, by:

- Extracting RDF from the attributes in XHTML2
- Adding RDF into the head of XHTML2 documents using RDF/A
- Adding RDF into the body of XHTML2 documents using RDF/A

With these functions, when XHTML2 is deployed, it will finally be possible to integrate high quality machine processible and extensible metadata expressed within the Resource Description Framework, into ordinary Web pages on the Web.

2. RDF METADATA IN HTML

Since the very first version of HTML it has been possible to put metadata describing the document within `<meta>` elements within the document head. However, this has been one of the least successful parts of HTML.

There have been a variety of problems, for example:

Clarity of intent It is often not clear what the metadata is meant to mean. Particularly since metadata is often machine processed the informal descriptions in the HTML recommendations have been found wanting.

Extensibility There are a few metadata properties explicitly mentioned in the HTML standards. The mechanisms for using profiles to use new metadata terms are underdeveloped and not deployed. There is a need for a deployed and functioning mechanism which allows new metadata keywords to be added without a standardization process.

'Metacrap' [11] Often the quality of the metadata is very low. Most document authors simply do not bother to add metadata. Worse, they copy-and-paste from an existing document and have incorrect metadata. Yet worse, they deliberately put incorrect metadata, often to improve the chances of the document being found with search engines. Doctorow [11] says: 'People lie. People are lazy. People are stupid.' We can summarize this problem as motivational. There are insufficient incentives to publish correct metadata.

The Resource Description Framework (RDF) is partially motivated to try and address these problems. It does give an extensible framework for metadata with a clear machine processable semantics. It does not directly address the 'metacrap' problem.

2.1 RDF

The very first paragraph of the 1998 RDF recommendation Model and Syntax [22] defines the core use case of RDF as "to describe the data contained on the Web". Since most Web data is HTML, this introduces a *syntactic* problem of how to encode metadata about HTML documents. This in addition to the more general semantic problem, addressed particularly in the RDF Semantics[1] [12], of how to have an extensible formally defined framework for metadata.

Such metadata can be placed in the HTML document, indeed Model and Syntax recommends "simply to insert the RDF [i.e. RDF/XML] in-line", or it can be in a separate document, usually RDF/XML, the recommended XML syntax for RDF [3]. Another common approach is to use Dublin Core [6] metadata within the `<meta>` tags in the head element of the document, see RFC 2731 [21], and [10], e.g.

```
<link
    rel="schema.DC"
    href="http://dublincore.org/qdcmes/1.0/"
```

[1]The general reader will find the treatment in the RDF Primer [23] more accesible.

.

```
 title="DCMES plus DCMI recommended qualifiers">
<meta   name="DC.Identifier"
        scheme="URI"
 content="http://www.ukoln.ac.uk/metadata/
resources/dc/datamodel/WD-dc-rdf/figure1.gif">
<meta   name="DC.Title"
        lang="en"
        content="A simple RDF assertion">
  ...
<meta   name="DC.Creator"
        content="Miller, Paul">
<meta   name="DC.Relation.isVersionOf"
        lang="en"
 content="Figure 1 from RDF Model and Syntax">
```

While it is possible to read such mark-up as RDF, it is only useful for metadata that conforms with the Dublin Core schema, or a schema written with interoperation with Dublin Core in mind. Hence, it lacks the open-endedness of RDF metadata, for which any schema [7],or no schema at all, can be used.

SHOE [14, 13] is an alternative ontology markup which is not directly compatible with RDF. This also can be embedded directly in HTML. SHOE aims to improve on RDF by providing more semantic primitives and better mechanisms for managing schema and ontology evolution. SHOE also, unlike RDF, provides an XML Document Type Definition (DTD) for its mark-up. This helps the use of SHOE within validated HTML.

2.2 Using the Document's own Metadata

Van Harmelen and Fensel [28] show how HTML span elements can be used to pick out of the document body the key data that is the document metadata. An example, is:

```
<HEAD>
  <META NAME="author" CONTENT="#L0">
  <META NAME="location"
    CONTENT="#L1">
  <META NAME="tel" CONTENT="#L2">
  <META NAME="room" CONTENT="#L3">
</HEAD>
<BODY>
 This page is written by
 <SPAN id="L0">Frank van
 Harmelen</SPAN>.
 <SPAN id="L1"> His tel.nr. is <SPAN
 id="L2">47731</SPAN>, room nr.
 <SPAN id="L3">T3.57</SPAN>
 </SPAN>
</BODY>
```

An advantage they pick out for this is that of avoiding duplication: "A basic tenet of information modeling is that redundancy inevitably leads to inconsistency". We note that this practice of including the metadata inside the document is one that goes back millennia, and suggest that supporting such a well-established practice is a must for a solution to web metadata (cf. [15]). this page, and the metadata that is automatically distinguished are the same: thus modifying and author's name to include a middle initial is done once, in one place for both purposes.

2.3 RDF/XML as Metadata markup?

A problem that emerged after the publication of the RDF Model and Syntax Recommendation is that such RDF/XML cannot be embedded in DTD valid HTML 4.0. This and related issues have been considered by the RDF Core working group, as part of their work on updating Model and Syntax. In brief, the response is that it is not possible [25]. Instead, the suggestion is that the metadata should form a separate document, and can be related to the original document using a link element in the head. (Not unlike the way that pictures are included in HTML documents). This can be regarded as the web equivalent of having a separate card in a card index as the primary repository of the document metadata. A long tradition can also be pointed to for this practice.

2.4 Requirements for Metadata markup Syntax

We can summarize this brief survey of the prior art by capturing the requirements and desiderata for Web metadata markup: These are:

- Ability to represent Dublin Core metadata
- Ability to represent other (arbitrary) semantic links
- Compatibility with HTML or XHTML
- Compatibility with validated HTML or XHTML
- Compatibility with XML
- Compatibility with XML validated against some DTD or XML Schema
- Compatibility with RDF
- Ability to refer to the metadata already in-line in the document
- Ability to refer to new metadata in the document head

(See also the list of requirements in [1]). None of the examples already surveyed has all of these properties.

3. OVERVIEW OF RDF

Before proceeding further, we will give a brief summary of RDF, concentrating on its abstract syntax: the RDF graph [20].

RDF describes resources by means of triples. Each triple identifies a subject, the resource being described, and describes one aspect of it, giving the value of some property of that resource. The property is identified by a URI known as the predicate. The value of the property is known as the object of the triple, and may be a literal value, such as a string, or it may be another resource.

So the triple

```
<http://example.org/>
    <http://purl.org/dc/elements/1.1/creator>
    "Jane Doe" .
```

indicates that the resource identified by `<http://example.org/>` has the value `"Jane Doe"` for the property identified by `<http://purl.org/dc/elements/1.1/creator>`. This last property, known as 'Dublin Core creator' indicates the name of one of the creators of the subject resource. To avoid writing long URIs a common convention in RDF is to use qnames to abbreviate URIs by the rule that a qname corresponds to a URI formed by the simple concatenation of the namespace URI with the local name. So this predicate is usually abbreviated as `dc:creator`, with `dc` bound to the namespace URI `<http://purl.org/dc/elements/1.1/>`.

Resources may be identified by means of a URI, or they may be unidentified, and are represented by what is called a *blank node*. A blank node shows that there is a resource, but it is not given a global name. i.e. a blank node acts as an existential quantification.

An RDF graph is a collection of subject/predicate/object triples. The same blank node may appear in multiple triples in a single RDF graph, indicating that there is a resource with the described properties in many triples.

A literal may have an explicit datatype, typically one of the simple types of XML Schema. The datatype too is identified by means of a URI, again often abbreviated to be a qname. A literal without a datatype is known as a plain literal. Plain literals may be qualified by a language identifier. Typed literals do not have language identifiers.

Putting this all together, we can summarize the RDF abstract syntax, as an RDF graph is a set of triples, where a triple is one of the following eight possibilities:

subject	predicate	object
URI	URI	URI
URI	URI	blank
URI	URI	typed literal
URI	URI	plain literal
blank	URI	URI
blank	URI	blank
blank	URI	typed literal
blank	URI	plain literal

Each triple acts as a part of a description of the subject resource, and so an RDF graph acts as a description of many resources. A formal semantics [12] describing this is part of the RDF recommendation.

For working with XHTML, a particular datatype is highly relevant. This is `rdf:XMLLiteral`, defined in [20], which is a datatype for XML content, encoded within RDF graphs.

The other relationship between RDF and XML is that there is an XML syntax defined for RDF, known as RDF/XML [3].

4. THREE FEATURES OF XHTML2

XHTML2 is the next version of HTML currently being developed by the World Wide Web Consortium.

A key goal is to have less presentation and more document structure in XHTML2 than in earlier HTML versions. CSS is now the preferred way of addressing presentational issues.

The deprecated presentational features of XHTML1 have been dropped.

In this paper, we are concerned only with three new features of XHTML2: the `<section>` and `<h>` elements; a generalization of the `<a>` element; and a new approach to metadata.

In earlier versions of HTML, document sections have been implicit, identified by headings at various levels using the `<h1>` through `<h6>` elements. Some authors make the sections more explicit by using `<div>` elements to enclose them.

In XHTML2, while the old heading elements are still supported, a new better structure alternative is also provided. The `<section>` element is used to identify sections and subsections within the document, and the `<h>` element is used for the heading of the section. The level of the heading is implicit by the number of `<section>` ancestors it has. As an example:

```
<html>
 <head><title>An example</title></head>
 <body>
 <section>
  <h>A first level heading</h>
  <section>
    <h>Contents</h>
<ul>
 <li href="#s1">1.</li>
 <li href="#s1.1">1.1</li>
 <li href="#s2">2.</li>
</ul>
  </section>
  <section id="s1">
   <h>1. A second level heading</h>
   <section id="s1.1">
    <h>1.1 A third level heading</h>
   </section>
  </section>
  <section id="s2">
   <h>2. Another second level heading</h>
  </section>
 </section>
 </body>
</html>
```

The table of contents shows that the `<a>` element, while still permitted, is no longer needed in XHTML2. An `@href` attribute is allowed on any element, and a same document reference can be used to refer to an `@id` on any element.

In previous versions of HTML metadata has been placed in the head of the document, using `<link>` and `<meta>` elements. There has been no standard way of aligning such metadata with RDF. In XHTML2 metadata is stored anywhere in the document, using attributes that are permitted on all elements. The metadata is RDF, i.e. the attributes specify RDF triples, and so each XHTML2 document specifies an RDF graph. The next section explains in detail how these attributes work.

5. ATTRIBUTES FOR EXPRESSING RDF

RDF/A [5], is a set of attributes which can be used by an XML vocabulary, such as XHTML2, to carry RDF metadata within the XML vocabulary. The attributes[2] are: `@about` for describing the subject; `@rel`, `@rev`, and `@property` for describing predicates; `@content`, `@datatype`, `@plain` for describing literal objects; `@href` and `@nodeID` for describing objects. In addition an `@id` can be used to indicate a subject, if no `@about` is present.

URIs are resolved against the current in-scope `xml:base` [24].

Qnames are mapped to URIrefs using the RDF convention of concatenating the namespace URI with the local name, URIrefs which do not end with an *NCName* [16] cannot be represented as qnames.

5.1 RDF/A Processing Model

The processing model is declarative. For each element find all predicates, all subjects, and all objects, according to the rules in the next section. Take all combinations, with the constraint that some predicates require literal objects, and others require a resource object, i.e. a URIref or blank node. All such combinations are triples in the RDF graph corresponding to the XHTML document.

5.2 S,P,O Resolution Rules

The detail of the rules is given in the order predicate, object, subject. This is because the predicate rules are simplest, and the subject rules the most complicated.

In order to generate a triple, a subject rule, a predicate rule and an object rule have to be used. Hence, it is possible, for example, for an object rule to match, but no triple to be generated. Or for more than one triple to be generated, using the same object, but different subjects (when more than one than subject rule matches).

5.2.1 Resolution Rules for Predicates with Literal Objects

`@property` attribute If present this specifies a predicate for literal objects by means of a qname.

Implicit predicate If there is an `@content`, `@datatype` or `@plain` attribute, and no `@property` attribute then there is an implicit property for literal objects of `xhtml2:refersToLiteral`.

[2]Note to referees: the exact selection of attributes, and the details of the resolution rules, are still being discussed within the W3C. This paper works from one of the proposals on the table, `http://lists.w3.org/Archives/Public/www-archive/2004Nov/0001`, which is not quite the same as those in the cited paper.

5.2.2 Resolution Rules for Predicates with Resource Objects

`@rel` attribute If present this specifies a predicate for resource objects by means of a qname.

`@rev` attribute If present this specifies a predicate for resource objects by means of a qname. Any resulting triples have subject and object swapped, reversing the direction of the triple in the RDF graph.

Implicit predicate If there is an `@href` or `@nodeID` and an `@about` or `@id` attribute, and no `@rel` or `@rev` attribute then there is an implicit property for resource objects of `xhtml2:refersToResource`.

5.2.3 Literal Object Resolution Rules

Plain literal with `@content` If there is an `@content` attribute and no `@datatype`, then the `@content` value is a plain literal object, with language specified by the in-scope `xml:lang`.

Typed literal with `@content` If there is an `@content` and an `@datatype` attribute, then they specify a typed literal object, with lexical form given by the `@content` and the datatype URI specified as a qname by the `@datatype`.

Plain in-line literal If there is a `@plain` attribute with value `"true"` then a plain literal object is formed with lexical form given by concatenation of text descendent nodes of the element, and with language given by the in-scope `xml:lang`.

Typed in-line literal If there is a `@datatype` attribute and no `@content` attribute, then a typed literal object is formed with lexical form given by the concatenation of text descendent nodes and the datatype URI specified as a qname by the `@datatype`.

In-line `rdf:XMLLiteral` If none of the attributes `@datatype`, `@content` or `@plain` are present, then a typed literal object is formed with lexical form being the Exclusive XML Canonicalization [17] of the element content[3] and with datatype being `rdf:XMLLiteral`.

5.2.4 Resource Object Resolution Rules

`@href` If there is an `@href` attribute present then this gives a URI for the resource object. If it is relative, it is resolved using any in-scope `xml:base` value [24].

`@nodeID` If there is a `@nodeID` attribute present, then this gives a blank node identifier for a resource object.

generated blank node If neither an `@href` nor a `@nodeID` attribute are present then a blank node is generated. This is different from any other blank nodes, but it may participate in more than one triple, formed using the rules for this element, or one of its children.

5.2.5 Subject Resolution Rules

`@about` If present the `@about` attribute gives a URI for a subject. This is resolved using the `xml:base` rules.

`@id` If there is no `@about` attribute present, then the value of an `@id` attribute is used as a fragment ID, to generate a URI for a subject, by resolving the concatenation of `"#"` and the value of the `@id` against the in-scope `xml:base` value[4]

[3]RDF/XML also specifies the formation of rdf:XMLLiteral's using exclusive XML canonicalization.

[4]This is the same as with `rdf:ID` in RDF/XML.

Subject defined by the parent element If the child has neither an `@about` nor an `@id` attribute, then subjects are formed by reference to the parent element.

Explicit object of parent If the parent has an `@href` or a `@nodeID` (or both) then they specify the subject for the child, following 5.2.4.

Parent element's generated object If the parent uses a generated object, because it lacks an explict object, but does have an `@rel` or a `@rev` attribute, then the blank node generated in 5.2.4 is used as the subject for the child element.

Parent element's subject If the parent doesn't match the previous cases, i.e. it does not have an `@href` or a `@nodeID` or a `@rev` or a `@rel` attribute, but it does have an `@about` or an `@id` (but no `@about`) then the `@about` or `@id` is used as the subject of the child triple. This allows idiom like:

```
<head about="">
 <meta property="dc:creator"
       content="Jeremy Carroll"/>
</head>
```

Parent element's generated object If the parent element does not match any of the above cases, i.e. it does not have an `@href`, a `@nodeID`, a `@rev`, a `@rel`, an `@about` or an `@id` attribute then the blank node generated in 5.2.4 is used as the subject for the child element.

6. IMPLEMENTATION GOALS

In this section we consider what a metadata implementation for XHTML2 might do.

From the perspective of an RDF developers toolkit, such as Jena [9], it is useful to be able to read the RDF graph from an XHTML2 document. A further use case within such a framework is serializing an RDF graph as a sequence of `<link>` and `<meta>` elements to be placed in the head of an XHTML2 document.

For HTML authors it is useful if authoring tools allow them to add metadata markup, including the ability to highlight parts of the body text to include in the metadata (as in [18]). This should results in triples being added to the body of the XHTML2 document.

Another scenario which involves adding metadata to the body is where a preexisting XHTML2 document (perhaps created by migrating an HTML4 document) is modified by adding in a separate RDF graph to make a unified document. To get the full benefit of the unified document, each triple should be added to the head or to the body as appropriate. This may be useful for legacy migration. It may also be useful where a backend server generates both an XHTML document and an RDF graph but uses different routes and techniques to generate the two, and wishes to serve a document that is as amenable as possible to further editing. As an example, the metadata may be generated using OWL inferences which user agents may be unaware of, and so performing them server-side will be most effective.

The rest of this paper discusses the extraction of RDF metadata from XHTML2, and the creation of XHTML2 markup from RDF metadata, addressing both the `<head>` and the `<body>` cases.

7. XHTML METADATA WITH XSLT

This section describes an XSLT2 [19] implementation, which is available from `http://lists.w3.org/Archives/Public/www-archive/2004Nov/0001`.

This implementation follows the declarative processing model of section 5.1. In contrast with more procedural implementations, such as the SAX implementation discussed in the next section, this is likely to be slower. (Although that is highly dependent on the optimizations performed by the XSLT engine).

The declarative approach has the following key advantages:

- Clarity. The implementation follows the specification of section 5 in a straightforward fashion.
- Maintainability. As the RDF/A rules are modified during the standardization process it is easier to modify a declarative implementation, since the relationship between the rules in the specification and the rules in the implementation is immediate, rather than implicit.
- Correctness. The clarity of the implementation means that it is more likely to be correct.
- Cost. A correct implementation is achieved more easily using declarative techniques. In particular, the cost of debugging and maintenance is reduced. The source code is a total of 550 lines of XML and XSLT, which is indicative of low-cost.

Briefly, the rules for subjects, predicates and objects from section 5.2 are expressed using XPath [4], in a custom XML format. These are then combined by a first XSLT transform in all possible ways, to give rules for matching triples (with subject, predicate and object). These rules are still in a custom XML format. A second XSLT transform then takes these rules to generate the runtime XSLT transform which implements a mapping from the RDF/A attributes in XHTML2 into RDF/XML.

7.1 Design Overview

7.1.1 S,P,O Rules

Each paragraph specifying a rule for subjects, predicates and objects from section 5.2 is expressed in a custom XML format. For example:

If there is an `@href` attribute present then this gives a URI for the resource object. If it is relative, it is resolved using any in-scope `xml:base` value [24].

is expressed as:

```
<object
    a:object='resource'
    a:match='[@href]'
 rdf:resource='resolve-uri(@href,base-uri(.))'
   />
```

The `@a:match` attribute gives an XPath [4] predicate to be matched against an XHTML2 element. If it applies then the other attributes (`@rdf:resource` in this case) are used in the construction of the RDF/XML output. The value `"resolve-uri(@href, base-uri(.))"` is again expressed in terms of XPath, and implements the `xml:base` rules. The `@a:object` attribute indicates this object is a resource object and not a literal object. The `a:` namespace is used to indicate information that is manipulated during the first transform producing the triple rules.

The predicate rules are similar, for instance:

If an `@rev` attribute is present this specifies a predicate for resource objects by means of a qname. Any resulting triples have subject and object swapped, reversing the direction of the triple in the RDF graph. becomes:

```
<predicate
  a:reversed='true'
  a:object='resource'
  a:match='[@rev]'
  name='@rev'
/>
```

The rules for the `@rel` and `@id` attributes are similar:

```
<predicate
  a:object='resource'
  a:match='[@rel]'
  name='@rel'
/>

<subject
  rdf:about=
 "resolve-uri(concat('#',@id),base-uri(.))"
  a:match=
 '[@id][not(@about or self::xhtml2:head)]'
/>
```

In the `@id` rule we note that both the positive and negative conditions are expressed in the `@a:match` attribute, and that the fact that an xhtml2:head has a default value for `@about` is explicitly coded.

7.1.2 Forming Triple Rules

The rules for subjects, predicates and objects are combined by a first XSLT transform. This takes all combinations of subject and predicate and object rules with the single constraint that the predicate and object rules have the same `@a:object` value. For each combination a new triple oriented rule is generated.

The four rules presented above, hence give rise to the following two triple rules:

```
<match select=
 "xhtml2:*[not(@about or self::xhtml2:head)]
         [@id][@rel][@href]">
 <subject rdf:about=
 "resolve-uri(concat('#',@id),base-uri(.))" />
 <predicate name="@rel"/>
 <object rdf:resource=
      "resolve-uri(@href,base-uri(.))"/>
</match>
<match select=
"xhtml2:*[not(@about or self::xhtml2:head)]
        [@id][@rev][@href]">
 <subject rdf:about=
      "resolve-uri(@href,base-uri(.))"/>
 <predicate name="@rev"/>
 <object rdf:resource=
 "resolve-uri(concat('#',@id),base-uri(.))" />
</match>
```

In the first, we see that the `@a:match` attribute values from the subject, predicate and object have simply been concatenated to give a predicate applying to `xhtml2` elements. When these match, the subject, predicate and object of the triple are formed as shown. In the second rule, we see that the subject and object have been swapped, and correspondingly the `@rdf:resource` attribute has been replaced with an `@rdf:about` and vice versa, corresponding to the different ways of expressing subject and object in RDF/XML.

152 such triple rules are generated.

7.1.3 Creating the XSLT runtime

A second transform takes such triple rules, and creates an XSLT transform implementing them. The basic structure of the output is a transform that processes each element of an XHTML input document in turn, and transforms each one into an (often empty) sequence of `rdf:Description` elements in an RDF/XML document. Each such element is used to specify one triple found in the input, forming by matching a rule against the current element.

The match of a rule against an element is implemented using an `xsl:if` test. The first triple rule above becomes the following XSLT code:

```
<xsl:if test="self::xhtml2:*[@id][@rel]
```

```
 [not(@about or self::xhtml2:head)][@href]">
 <rdf:Description>
  <xsl:attribute name="rdf:about"
     select=
 "resolve-uri(concat('#',@id),base-uri(.))"/>
  <xsl:element name="@rel"
     namespace="namespace-uri-for-prefix(
         substring-before(@rel,':'),.)">
     <xsl:attribute name="rdf:resource"
select="resolve-uri(@href,base-uri(.))"/>
  </xsl:element>
 </rdf:Description>
</xsl:if>
```

While the resulting XSLT file implementing RDF/A is quite long (about 3000 lines), the source code (the first rules file, and the first and second transforms) is relatively short (140 lines for 21 rules; 160 lines for the first transform; 247 lines for the second). Lines of code is a very rough-and-ready measure for XSLT programs.

8. PARSING METADATA WITH SAX

The XSLT approach has some inevitable inefficiencies. It almost certainly requires building a DOM tree or some similar in-memory view of the document, and does not take a streaming approach. It's output is an RDF/XML document, that must be further parsed using an RDF parser if the goal is to manipulate, query, or run inferences over the RDF graph.

In such cases it is more appropriate to build over a streaming XML interface, such as SAX [27].

Each XML element may give rise to one or more triples. The triples depend on the attributes of the element, it's name (since `<head>` has a default value `@about=""`), and its parent (which may provide the subject of the triple). For some triples, with in-line literal content, all the children are potential relevant to the object value: either the text() descendents for plain or typed literals, or all content (elements, comments, processing instructions, and text) for `rdf:XMLLiteral` objects.

This suggests using a stack, similar to how RDFFilter [26] parses RDF/XML, but with the variation that triples are generated with the `endElement` event, after all the children have been processed.

Each time a `startElement` event fires, a new entry is pushed on the stack. This stores:

1. The attributes of the element.
2. The element qualified name.
3. A set of string concatenators for text content.
4. A set of exclusive canonical XML [17] concatenators for `rdf:XMLLiteral` content.

The first two items are initialized from the SAX event. The last two items are inherited from the top entry of the stack with the addition of a text concatenator for this element if there is a `@datatype` or `@plain` attribute or an exclusive canonical XML concatenator if there is a `@property` attribute and neither a `@datatype` nor a `@plain` attribute.

The concatenators are used in combination with other SAX events from the children to build up in-line literal objects, which are finished when all the child nodes have been processed, i.e. when the `endElement` event occurs.

These objects on the stack have lazily evaluated methods implementing the subject, predicate and object rules from section 5.2. Further methods resolve inscope `xml:lang`, `xml:base` and XML namespaces by reference to the attributes or to the parent.

When processing `startElement` events, if the parent, the top element on the stack, has any exclusive canonical XML concatenators then these are updated with the open tag corresponding to the event. Attributes and namespaces in the event are included in the update, sorted as required by [17].

SAX text events are processed by adding the string to all text concatenators and the XML escaped string to all exclusive canonical XML concatenators which are accessed from the top of the stack.

Comment and processing instruction events are also added appropriately to the exclusive canonical XML concatenators of the top element of the stack.

End element events create triples. They do this by invoking the methods corresponding to the subject, predicate and object rules, and generating a triple for every appropriate combination.

In this way, the RDF metadata can be extracted from an XHTML2 document.

9. ADDING METADATA TO XHTML HEAD

In section 3 we identified eight different patterns of triples. Examples of the first, third, sixth and eighth of these, encoded as `<link>` and `<meta>` elements are as follows:

```
<link about="http://example.org/subj"
      rel="eg:prop"
  href="http://example.org/obj"/>
<meta about="http://example.org/subj"
      property="eg:prop"
  datatype="xsd:int"
  content="10"/>
<link nodeID="subj">
   <link rel="eg:prop" nodeID="obj"/>
</link>
<link nodeID="subj">
   <meta property="eg:prop"
  xml:lang="eg"
  content="lexical-form"/>
</link>
```

The other cases can be formed similarly.

It is straightforward to use these examples as a template for an RDF serializer.

Two limitations are:

- Only predicates that can be expressed as a qname can be serialized. This restriction is shared with RDF/XML.
- Only datatype URIs that can be expressed as a qname can be serialized. In particular, the datatype URIs used in XML Schema Component Designators [29] cannot.

9.1 Preexisting Metadata

When adding metadata to an XHTML document it may already have some metadata marked up.

This introduces two issues for adding more metadata. The first is that the blank node identifiers already in the document may clash with the blank node identifiers used by the serializer. This is easy to check for, and the serializer can use other identifiers that have not been used in the original document. However, this does make any API for such a serializer more complicated, because the block of text produced cannot be used in arbitrary XHTML documents. The blank node identifiers depend on the original XHTML document.

A further issue is which of the following is required, when merging an RDF/XML file with an XHTML2 file:

1. Leave the preexisting metadata in the XHTML2 unchanged, and add the new metadata from the RDF/XML.
2. Remove the old metadata in the XHTML2 completely.
3. Do not add metadata from the RDF/XML that is already in the XHTML2 document, but only add 'new' triples, (in the RDF/XML but not the XHTML2) leaving old triples in place.

4. Add ‘new’ triples (in the RDF/XML but not the XHTML2); leave triples which occur in both the RDF/XML and the XHTML2 unchanged; delete triples that occur in the XHTML2 but not the RDF/XML.

Items 2 and 4 update the metadata associated with the XHTML document to be precisely the graph associated with the RDF/XML document. Whereas items 1 and 3 form a merge of the metadata of the two original documents to make a third.

Items 3 and 4 require the ability to compare the metadata in the original XHTML2 document with that in the RDF/XML document. The comparison should return a maximal subgraph found in both, and a remainder. i.e. given two RDF graphs g and h, the algorithm should return a g' and g'' such that: g' is a subgraph of both g and h (and maximal), and that $g = g' \cup g''$.

This problem embeds the RDF subgraph isomorphism problem. This is NP complete, since it is equivalent to the subgraph isomorphism problem for classic undirected graphs (following similar reasoning to that in [8]). Thus we expect to use a non-deterministic algorithm. However, the lesson from RDF graph isomorphism [8], is that good use of the (many)labels within an RDF graph can, in practice, mean that the non-determinism is kept to a minimum.

10. ADDING METADATA TO XHTML BODY

In the previous section we showed that it is straightforward to use RDF/A to add arbitrary RDF metadata to the head of an XHTML2 document. However, when that metadata is about the content of the document then it is more natural to express the metadata within the body of the document. This section discusses this, substantially harder, problem.

We start by looking at a few examples of metadata in the body, exploring the difficulties the problem presents. We then present some heuristics, and an algorithm implementing them.

10.1 Examples

10.1.1

Consider the XHTML document:

```
<html xml:base="http://example.org/doc">
 <head>
  <title>A Document</title>
 </head>
 <body>
  <section>
   <h>A Document</h>
   <p>Editor: Jeremy J. Carroll</p>
   <section id="contents">
    <h>Contents</h>
    <ul>
     <li href="#first">First Section</li>
     <li href="#second">Second Section</li>
    </ul>
   </section>
   <section id="first">
    <h>First Section</h>
    <p>Author: Jane Doe</p>
   </section>
   <section id="second">
    <h>Second Section</h>
    <p>Author: Jeremy J. Carroll</p>
   </section>
  </section>
 </body>
</html>
```

and the RDF metadata:

```
<http://example.org/doc>
          dc:creator "Jeremy J. Carroll" .
<http://example.org/doc>
          dc:title "A Document" .
eg:first  dc:creator "Jane Doe" .
eg:second dc:creator "Jeremy J. Carroll" .
<http://example.org/doc> egs:contents _:s .
_:s rdf:type rdf:Seq .
_:s rdf:_1 eg:first .
_:s rdf:_2 eg:second .
eg:first  dc:title "First Section" .
eg:second dc:title "Second Section" .
```

These can be put together like this:

```
<html xml:base="http://example.org/doc"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:rdf=
 "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:egs="http://example.org/schema#"
>
 <head about="">
  <title property="dc:title"
          plain="true">A Document</title>
 </head>
 <body>
  <section>
   <h>A Document</h>
   <p>Editor: <span about="" property="dc:creator"
   plain="true">Jeremy J. Carroll</span></p>
   <section id="contents">
    <h>Contents</h>
<ul about="" rel="egs:contents">
   <link rel="rdf:type" href="&rdf;Seq" />
     <li rel="rdf:_1" href="#first"
       >First Section</li>
     <li rel="rdf:_2" href="#second"
       >Second Section</li>
    </ul>
   </section>
   <section id="first">
    <h property="dc:title"
      plain="true">First Section</h>
    <p>Author:
    <span about="#first" property="dc:creator"
    plain="true">Jane Doe</span></p>
   </section>
   <section id="second">
    <h property="dc:title"
      plain="true">Second Section</h>
    <p>Author:
   <span about="#second" property="dc:creator"
    plain="true">Jeremy J. Carroll</span></p>
   </section>
  </section>
 </body>
</html>
```

A number of aspects should be noted:

- Most of the metadata is expressed by adding attributes to pre-existing elements.
- We often use literal strings that were already the textual element content in the original document. e.g. “A Document”.
- We often use URIrefs that were already present in the document. e.g. `href="#first"`.
- Sometimes we need to add a `<span>` to enclose a specific textual string. e.g. “Jane Doe”.
- The blank node did not need to have an explicit identifier, because we could arrange all the triples involving it in the same part of the document.
- In one case, we add a `<link>` element to add further metadata. i.e. the first `<link>` child of the `<ul>` element.
- The triple added with the `<link>` and that added on the `<ul>` element could have been swapped around:
  `<ul about="&rdf;Seq" rev="rdf:type">`

```
  <li rel="rdf:_1" href="#first"
        >First Section</li>
```

- Or both of those two triples could have been added with `<link>` elements:
  ```
  <ul>
   <link rel="rdf:type" href="&rdf;Seq" />
   <link rev="egs:contents" href="" />
   <li rel="rdf:_1" href="#first"
         >First Section</li>
  ```
- Some triples with both subject and object occurring in the document could be expressed using that pre-existing markup, for both subject and object: e.g. `eg:first dc:title "First Section"`.
- Some triples with both subject and object occurring in the document could not be expressed using that pre-existing markup. In these cases, we anchored the triple in the body using a literal object, and repeated the subject. e.g. `eg:first dc:creator "Jane Doe"`. This reflects that the occurrence in the document of the subject is too far away from the occurrence of the object.
- The `about=""` on the `<head>` element, is the default value, and could have been omitted.
- Looking at the two occurrences of “Jeremy J. Carroll”, for the first we gave the subject `""`, for the second `"#second"`. This was more natural than the other way around, because both subjects had occurred nearby in the document.

A key to these observations is that some of the nodes in the RDF graph are already present within the XHTML, and we just need to add attributes and maybe some `<link>`, `<meta>`, `<span>` or `<div>` elements to express the appropriate triples. We will call such nodes *realised* nodes.

Realised nodes can occur in a number of ways:

- URIref nodes from `@about` or `@id` or `@href` attributes.
- Literal lexical forms from the text content of nodes. The literals can be plain (as long as the in-scope `@xml:lang` is appropriate) or typed (by the addition of an `@datatype` attribute).
- If the document already has some metadata in it, and we use the subgraph isomorphism techniques described at the end of section 9, then this produces a mapping from (some of) the blank nodes in the XHTML2 document into the blank nodes of the RDF graph being added. Using the inverse of this mapping blank nodes to be added may be realised as `@nodeID` attributes, or as the blank node implicit in the absence of an explicit `@href` or `@nodeID`. The mapping corresponding to a subgraph isomorphism is not uniquely defined in all cases.
- While we are adding triples into the body, we may add a new realisation of the subject or object.

Nodes can be realised in multiple places in the XHTML2 document.

10.1.2 *Tethered Paths*

A short, poorly marked-up, description of the United States could be:

```
<div id="US" xml:lang="en">
 <h2>The United States of America</h2>
 <p>Name: (English) United States of America,
    (Italian)
 <span xml:lang="it">Stati Uniti</span>.</p>
</div>
```

and the RDF metadata:

```
eg:US egs:hasName _:a .
_:a rdfs:label "United States of America"@en .
_:a rdfs:label "Stati Uniti"@it .
```

The metadata has a simple tree structure that can be superimposed on the XML structure like this:

```
<div id="US" xml:lang="en">
 <h2>The United States of America</h2>
 <p rel="egs:hasName">Name: (English)
 <span property="rdfs:label" plain="true"
        >United States of America</span>,
    (Italian) <span xml:lang="it"
property="rdfs:label" plain="true"
>Stati Uniti</span>.</p>
</div>
```

But with a richer use of XHTML mark-up, such as

```
<div id="US" xml:lang="en">
 <h2>The United States of America</h2>
 <table>
 <tr><td>English</td>
     <td>United States of America</td></tr>
 <tr><td>Italian</td>
     <td xml:lang="it">Stati Uniti</td></tr>
 </table>
</div>
```

we find that it is not possible to avoid explicitly labelling the blank node, e.g.

```
<div id="US" xml:lang="en"  rel="egs:hasName"
 nodeID="s">
 <h2>The United States of America</h2>
 <table>
 <tr nodeID="s"><td>English</td>
  <td property="rdfs:label" plain="true"
  >United States of America</td></tr>
 <tr nodeID="s"><td>Italian</td>
  <td xml:lang="it" property="rdfs:label"
      plain="true">Stati Uniti</td></tr>
 </table>
</div>
```

One aspect of the problem is that the path in the RDF graph from `eg:US` to `"Stati Uniti"@it` is of length two; but in the XML document, there are three intervening XML elements.

Thus we introduce the concept of *tethered path* to be a path in the RDF graph whose end-points are both realised in the XHTML document, but whose intermediate points are not. If a tethered path is shorter than the shortest path through XML elements between two nodes corresponding to its end-points, then it cannot be expressed using RDF/A attributes involving those nodes. A tethered path can consist of a single triple. Each triple in the path from `eg:US` to `"Stati Uniti"@it` is directed in the same way: subject, predicate, object. Whereas in the tethered path from `"United States of America"@en` to `"Stati Uniti"@it`, one of the triples is directed object, predicate, subject. We will call a tethered path in which each triple is directed subject, predicate, object a *directed* tethered path.

In many cases realizing a tethered path, by adding appropriate RDF/A attributes to the XHTML, prevents the realization of some other tethered path. Here is an example:

Consider adding

```
eg:p1 egs:prop _:a .
eg:p3 egs:prop _:a .
eg:p2 egs:prop _:b .
eg:p4 egs:prop _:b .
```

to

```
<body>
 <p id="p1">para1</p>
 <p id="p2">para2</p>
 <p id="p3">para3</p>
 <p id="p4">para4</p>
</body>
```

We can add the first two triples, like this:

```
<body>
 <p rev="egs:prop" id="p1">para1</p>
 <p id="p2">para2</p>
 <p rev="egs:prop" id="p3">para3</p>
 <p id="p4">para4</p>
</body>
```

or like this:

```
<body>
 <div>
  <p rev="egs:prop" id="p1">para1</p>
  <p id="p2">para2</p>
  <p rev="egs:prop" id="p3">para3</p>
 </div>
 <p id="p4">para4</p>
</body>
```

But if we do, then that prevents adding the similar markup to include the last two triples. To add all four triples at least one of the blank nodes needs to be given an explicit label in the XHTML. e.g.

```
<body>
 <p rev="egs:prop" id="p1">para1</p>
 <p id="p2" rel="egs:prop" nodeID="b">para2</p>
 <p rev="egs:prop" id="p3">para3</p>
 <p id="p4" rel="egs:prop" nodeID="b">para4</p>
</body>
```

In summary there are lots of choices involved with adding RDF to the body, including:

- For each RDF node which is realised more than once in the XHTML2 document, which of these realisations to use for each triple?
- If there is preexisting metadata some of which is subgraph isomorphic to the required metadata, which of the (potentially multiple) maximal subgraph isomorphisms to use?
- When realising more than one tethered paths which cannot both be realised simultaneously, which to realise and which to not realise?
- For a triple with one realised node, in some cases there is a choice between adding a `@rev` attribute on that node, or a new child with a `@rel` attribute (or vice versa).

An ‘optimum’ solution could perhaps be achieved by having some goodness function that evaluated how clear a particular expression of the RDF in the body is, and then use a nondeterministic approach to explore the choice space and maximise the goodness function. It is highly likely that such an approach would have exponential complexity.

Such an ‘optimum’ solution appears to be very hard to achieve, and too expensive both in terms of runtime complexity and speed, and software complexity and cost, both development cost and maintenance cost.

A better solution is to identify some heuristics which allow the identification of good ways of choosing which triple to add next from the metadata to be added, and where to add it. This heuristics guide adding triples in a deterministic way.

The heuristics suggested from the examples are:

1. prefer adding directed paths
2. prefer adding short tethered paths
3. dislike using the `@rev` attribute
4. dislike adding new elements

These heuristics are used by the following deterministic algorithm, which adds the triples from an RDF graph g to an XHTML2 document d

1. For *lg* from 1 to 3
    (a) Set T to be the set of paths from g of length *lg* tethered in d.
    (b) If $\exists$directed$t \in T$ which can be realised in d by adding only `@rel`, `@property`, `@datatype` and `@plain` attributes: modify d by adding t, and modify g by removing t. Continue outer loop 1. (Note: these paths follow and XML path from an element, to a child element, a grandchild element ...)
    (c) If $\exists$ directed $t \in T$ which can be realised in d by adding only `@rel`, `@rev`, `@property`, `@datatype` and `@plain` attributes: modify d by adding t, and modify g by removing t. Continue outer loop 1.
    (d) If $\exists t \in T$ which can be realised in d by adding only `@rel`, `@property`, `@datatype` and `@plain` attributes: modify d by adding t, and modify g by removing t. Continue outer loop 1.
    (e) If $\exists t \in T$ which can be realised in d: modify d by adding t, and modify g by removing t. Continue outer loop 1.
2. While g contains a triple with a subject or object realised in d
    (a) While $\exists$*triple* $\in g$ which can be added to body of $d$with either subject or object realised, using a `@property` or `@rel` attribute, add *triple* to d, and remove *triple* from g.
    (b) If $\exists$*triple* $\in g$ which can be added to body of $d$with either subject or object realised, using a `@rev` attribute, add *triple* to d, and remove *triple* from g and continue outer loop 2.
    (c) If $\exists$*triple* $\in g$ which can be added to body of $d$with literal object realised, using a new `<span>` element, add *triple* to d, and remove *triple* from g and continue outer loop 2.
    (d) While $\exists$*triple* $\in g$ which can be added to body of $d$with subject realised, using a new `<link>` or `<meta>` element with a `@rel` or `@property` attribute add *triple* to d, and remove *triple* from g. If object of *triple* is a resource, i.e. `@rel` was used, continue outer loop 2.
    (e) If $\exists$*triple* $\in g$ which can be added to body of $d$with subject realised, using a new `<link>` element with a `@rev` attribute add *triple* to d, and remove *triple* from g. Continue outer loop 2.
3. Add all triples in g to head of document, as in section 9.

The ordering of the steps that add various types of paths and triples to add to the body of the document shows a prioritisation between the possibilities.

The algorithm restarts after each addition of a triple that may result in an earlier (higher value) heuristic applying. (These are the various ‘continue outer loop’ instructions) Coding techniques such as memoization can be used to avoid excessive recomputation.

In summary, we have seen that adding RDF to the body is difficult. HTML editing software, which wishes to support such functionality, would also need to use similar heuristics to place metadata around the text it refers to.

11. SOME ISSUES WITH RDF/A

The previous section has shown that there are lots of choices with adding RDF/A to elements in the body.

Experience with RDF/XML suggests that many people get confused with too many options, some of which are rarely used, and hence unfamiliar. For example, the property attribute construction in RDF/XML, is useful in some cases, but does lead to significant confusion.

These two observations suggest that despite simplifications from the initial proposals, RDF/A might still be too complex. Possible further simplications could be:

- Dropping the `@rev` attribute.
- Dropping default predicate rules.

Another aspect of RDF/XML that some people find difficult is to know when to use a qname and when to use a URI. This problem is repeated in RDF/A, with predicates and datatypes expressed by qnames, and subjects and objects with URIs. This might be a mistake which could cause confusion.

A further problem was that adding structure to an XHTML document by putting in container elements such as a `<div>` without any RDF/A attributes may modify the meaning of RDF/A attributes on the child elements. This could be avoided by changing the rules for finding the subject for an RDF/A element without a `about` or `id` to not just look at the parent node, but to search up the XML tree until some ancestor node with an explicit object or subject is found.

12. CONCLUSIONS

RDF/A and its integration into XHTML2 is a big step forward in integrating high quality metadata into Web pages. It allows the combination of the power of Semantic Web techniques with the HTML Web.

Some of the difficulties and choices with adding arbitrary RDF to the body of XHTML2 pages reflects a mismatch between the graph structure of the metadata and the tree structure of the document. In practice, only that part of the metadata that respects the same tree structure as the document, can be added by just using attributes and no additional metadata elements.

The techniques developed in this paper can be used by developers of Web and Semantic Web tools to provide support for this combination by extracting metadata from XHTML2 pages to be combined with other knowledge in Semantic Web applications and knowledge bases, and to take metadata from such Semantic Web sources and add it, in an appropriate way, to XHTML2 pages.

13. REFERENCES

[1] M. Altheim and S. B. Palmer. Augmented Metadata in XHTML. `http://infomesh.net/2002/augmeta/`, 2001.

[2] J. Axelsson, B. Epperson, M. Ishikawa, S. McCarron, A. Navarro, and S. Pemberton. XHTML 2.0. `http://www.w3.org/TR/xhtml2`.

[3] D. Beckett. RDF/XML Syntax Specification (Revised). `http://www.w3.org/TR/rdf-syntax-grammar/`, 2004.

[4] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simon. XML Path Language (XPath) 2.0. `http://www.w3.org/TR/xpath20/`.

[5] M. Birbeck and S. Pemberton. RDF/A Syntax:A collection of attributes for layering RDF on XML languages . `http://www.formsplayer.com/notes/rdf-a.html`, 2004.

[6] D. U. Board. DCMI Metadata Terms. `http://dublincore.org/documents/dcmi-terms/`, 2004.

[7] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0. `http://www.w3.org/TR/rdf-schema/`, 2004.

[8] J. J. Carroll. Matching RDF Graphs. In *ISWC*, LNCS. Springer, 2002.

[9] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the semantic web recommendations. WWW2004, 2004.

[10] S. Cox, E. Miller, and A. Powell. Recording qualified Dublin Core metadata in HTML meta elements, 2000.

[11] C. Doctorow. Metacrap: Putting the torch to seven straw-men of the meta-utopia. `http://www.well.com/~doctorow/metacrap.htm`, 2001.

[12] P. Hayes. RDF Semantics. `http://www.w3.org/TR/rdf-mt/`, 2004.

[13] J. Heflin, J. Hendler, , and S. Luke. SHOE: A Blueprint for the Semantic Web. In *Spinning the Semantic Web*. MIT Press, 2003.

[14] J. Heflin and J. Hendler. Semantic Interoperability on the Web. In *Proceedings of Extreme Markup Languages 2000*, 2000.

[15] M. Hemrich and Schafer. XML-based linking concepts. In *Proceedings of 23rd International Online Information Meeting*, 1999.

[16] D. Hollander, A. Layman, and T. Bray. Namespaces in XML. `http://www.w3.org/TR/1999/REC-xml-names-19990114`, 1999.

[17] J. R. J. Boyer, D.E.Eastlake 3rd. Exclusive XML Canonicalization Version 1.0. `http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/`, 2002.

[18] A. Kalyanpur, J. Golbeck, M. Grove, and J. Hendler. An RDF Editor and Portal for the Semantic Web. In *Semantic Authoring, Annotation & Knowledge Markup Workshop (ECAI)*, 2002.

[19] M. Kay. XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20/`.

[20] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/rdf-concepts/`, 2004.

[21] J. A. Kunze. RFC 2731: Encoding Dublin Core metadata in HTML, 1999.

[22] O. Lassila and R.R.Swick. Resource Description Framework (RDF) Model and Syntax Specification . `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, 1999.

[23] F. Manola and E. Miller. RDF Primer. `http://www.w3.org/TR/rdf-primer/`, 2004.

[24] J. Marsh. XML Base. `http://www.w3.org/TR/2001/REC-xmlbase-20010627/`, 2001.

[25] B. McBride. RDF Issue List, HTML Compliance. `http://www.w3.org/2000/03/rdf-tracking/#faq-html-compliance`, 2002.

[26] D. Megginson. RDFFilter. `http://rdf-filter.sourceforge.net/`.

[27] D. Megginson. Simple api for xml. `http://sax.sourceforge.net/`.

[28] F. van Harmelen and D. Fensel. Practical Knowledge Representation for the Web. In D. Fensel, editor, *Proceedings of the IJCAI'99 Workshop on Intelligent Information Integration*, 1999.

[29] A. S. Vedamuthu and M. Holstege. XML Schema: Component Designators. `http://www.w3.org/TR/2004/WD-xmlschema-ref-20040716/`, 2004.