

hGRDDL: Bridging Microformats and RDFa

Ben Adida^{a,*},

^a*Harvard University, 33 Oxford Street MD-110, Cambridge MA 02118*

Abstract

We propose hGRDDL (pronounced “h-griddle”), a simple mechanism for transforming ad-hoc HTML-embedded structured data, such as microformats, into RDFa. This technique preserves the advantages of the original syntax, notably the correspondence between the rendered HTML and the related structured data, and requires little change on the publisher end. RDFa tool developers can leverage the existing deployments of microformats, while focusing new deployments on RDFa for greater extensibility and consistency, all using the same client-side toolset. We provide a prototype implementation of the hGRDDL processor and of transformations for hCard and hCal, two popular microformats.

Key words: RDFa, microformats, GRDDL

1. Introduction

Since the early days of the Web, there have been efforts to publish semantic data alongside the primarily visual content of HTML. Certain HTML elements, e.g. `` for emphasis, are inherently more semantic and less explicitly visual than others, e.g. `<i>` for italics. Certain HTML attributes, e.g. `rel` on an anchor element ``, are inherently semantic since they generally do not result in any visual effect, rather they indicate a relationship between the current and linked documents. These concepts were the driving force behind the microformats effort [18], which was colloquially called “the small-s semantic web” for its ad-hoc approach to evolving the Web towards more machine-readable data.

1.1. Multiple Approaches

While microformats have grown to provide a number of vocabulary-specific syntaxes, other techniques have emerged to provide more generic semantic web data embedding, with syntax independent of the vocabulary. RDFa [3], the W3C’s work-in-progress in this area, enables the embedding of almost any RDF graph inside XHTML1.1 and XHTML2 [10] using some additional attributes. eRDF [6] can embed a smaller subset of RDF graphs inside plain HTML as long as the publisher controls the document’s `<head>`. Meanwhile, a separate W3C effort, GRDDL [4], aims to extract RDF triples from any XML document, including XHTML, using a document- or namespace-specific transformation. One of the use cases for GRDDL is to extract RDF/XML from an HTML-with-microformats document, though it can be applied to any homegrown syntax, i.e. “unofficial” microformats.

Multiple syntaxes are inevitable. However, variety comes at a cost: each syntax requires its own

* Corresponding author. Tel: +1 617.395.8535
Email address: `ben@eecs.harvard.edu` (Ben Adida).

parser, often its own semantic interpreter, and often its own mechanism for relating rendered screen regions to structured data. Though RDFa provides a consistent syntax and the most expressivity, microformats currently enjoy significantly greater deployment: a number of large publishers, e.g. the social network LinkedIn [15], the online calendar 30Boxes [1], and Apple’s dotMac online email system [9], use microformats. Recently, the total number of microformat-enabled web pages was estimated at “hundreds of millions” [17]. Whatever limitations microformats may have, and however extensible a toolbuilder wishes to make her software, she cannot ignore microformats. How might we give the toolbuilder the best of all worlds while reducing her development effort? How can we enable a world of many structured data syntaxes while reducing the duplication of effort in writing client-side tools?

1.2. Bridging the Syntaxes

We propose hGRDDL¹, an approach that reduces syntax-specific code to a single transformation to RDFa. This solution effectively provides a funnel to RDFa, which becomes a unifying syntax around which all client-side tools can be built. The key point of hGRDDL is that it processes an HTML document and produces *another HTML document that renders identically*. The original HTML document can be entirely replaced by the new HTML document within the user’s browser. All client-side data processing can then be based solely on the RDFa version of the document, even if publishers produced microformats, eRDF, or some other custom syntax for their specific needs.

Note that the specific HTML “host” for RDFa does not change with the transformation. If the original document is XHTML2, then the target document is XHTML2+RDFa, and if the original document is XHTML1.1, then the target document is XHTML1.1+RDFa. As RDFa becomes supported in additional host dialects, we expect hGRDDL to apply accordingly.

¹ Our solution is called hGRDDL because it is meant as a GRDDL-like transformation with a focus on processing microformats, which have a habit of using lowercase ‘h’ in their naming to indicate “HTML”.

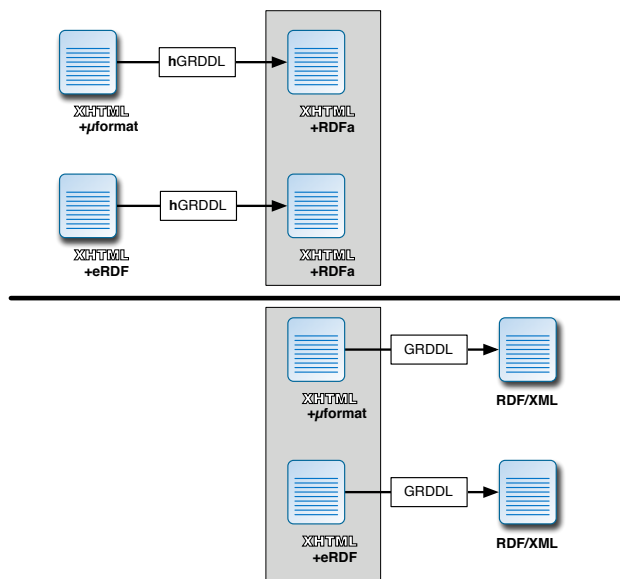


Fig. 1. hGRDDL vs. GRDDL. The hGRDDL transform happens *before* the user sees the rendered content. The point where the user sees the document is represented in the shaded vertical bar. Note how, in the case of hGRDDL, the output of the transformation is *both* the machine-readable data, and the human-readable data.

1.3. Why Not Just GRDDL?

The W3C’s GRDDL effort provides a way to transform microformats, eRDF, and other ad-hoc HTML-embedded syntaxes into RDF/XML. One might wonder what the point of hGRDDL is, then, if plain GRDDL can already extract the RDF. The key point is *human context*: an HTML page may contain a significant amount of structured data, of which the user may only want to select a small portion, typically by visual means, e.g. using a contextual menu on a particular section of the page. A number of tools, e.g. Operator [13], provide just this kind of human-context-driven structured data access.

With GRDDL, once RDF/XML has been extracted from HTML, there is no practical way to associate a point in the visually rendered HTML with its corresponding RDF: the HTML Document Object Model (DOM) [22] has been discarded. hGRDDL preserves the expression of the structured data in HTML form, so that this visual-semantic correspondence is retained.

2. Principles for Embedding Semantics into HTML

There are many methods for embedding semantics in HTML. Here, we outline four guiding principles, which we illustrate by example. Because these principles drove the design of RDFa, it should come as no surprise that RDFa is the only solution that satisfies all four. That said, detailing these principles should clarify the advantages of choosing RDFa as the target of hGRDDL transformations and illustrate why retaining information about the HTML DOM is particularly important.

Independence & Extensibility. When a publisher provides structured data in her HTML, she should determine independently the exact vocabulary she needs. There may be community-defined best practices, but the publisher should not be forced to use a consensus approach, as she knows her requirements better than anyone else. In particular, if she chooses, as a starting point, some best-practice vocabularies, she should be able to extend them, combine them, or use only a subset.

Tools built to recognize the initial vocabularies should still recognize as much of them as remains in the final, customized vocabulary, and be able to perform basic, generic data processing on the additional fields. It should be noted that RDF [16], as an abstract data modeling approach, precisely fits this requirement: any publisher can reuse existing RDF properties and mint new URIs to create new ones, while any consumer can perform HTTP-based discovery on properties it has not yet encountered.

Example 1 (Photography) *A professional photography web site may want to reuse the vocabulary created by a simpler image-sharing web site, including photo caption, size, and date taken. In addition, the professional photography site may wish to describe the lens and exposure settings. A tool that sorts the image-sharing web site photos by “date taken” should continue to function on the professional photography site. This tool is not aware of the new fields, but it can process the old ones. More interestingly, the old tool can still sort, filter, aggregate, and relate photos according to the new fields without knowing their deep semantics.*

DRY (Don’t Repeat Yourself). When human-readable data is the same as the machine-readable

version, a publisher should not have to repeat herself: the HTML should only contain a single copy of the data, which can be both rendered and interpreted as the machine-readable equivalent. In particular, solutions that maintain parallel HTML and RDF/XML files are *not* compliant with this principle.

Example 2 (Creative Commons) *An artist with a simple HTML page licenses his graphic designs under a non-commercial Creative Commons license [5]. He decides to change the license to non-commercial-share-alike, so that other artists who reuse his work must share their new work in the same way. To accomplish this, he simply changes the href in the anchor link to the Creative Commons license. Machine readers should detect this change, too, without the artist having to modify a second portion of his markup.*

Locality. When a user selects a portion of the rendered HTML within his browser, he should be able to access the corresponding structured data, using, for example, a contextual menu. The structured data must thus remain intimately tied to the syntax that expressed it.

Example 3 (Weblog) *A weblog consists of multiple entries, often presented in batches on a single HTML page. A user wishing to extract categorization tags and authorship from a given entry should be able to right-click on the entry and find the related information, independently of other entries. This assumes a properly updated browser but a straight-forward mechanism for locating related structured data.*

Self-Containment. It should be relatively easy to produce a fragment of HTML that is entirely self-contained with respect to the structured data it expresses. At a high level, this enables “copy-and-paste” of semantically marked-up HTML from one document to another. (The source and target documents may require some kind of flag indicating the presence of structured data in the first place, but this flag should not be vocabulary specific.)

Self-containment does not immediately follow from locality: the expression of the structured data may depend on multiple portions of markup sprinkled throughout the page, which weaken self-containment even if the crucial portion of markup still provides locality.

Example 4 (Widgets) *A number of destination web sites, e.g. MySpace [19], provide users with the ability to include HTML fragments from other sources within their page. A self-contained approach to HTML-embedded structured data ensures that these widgets can carry their own structured data with them, independently of the containing page.*

3. Approaches to Embedding Semantics in HTML

Microformats. Microformats were built specifically with the intent to add just enough markup to make existing HTML “semantic.” Using the HTML attributes `class`, `rel`, and `title`, microformats define domain-specific syntaxes, such as hCard for contact information and hCal for calendaring. Microformats are quite widely distributed: a recent estimate gauges deployment at “hundreds of millions of web pages” [17].

Microformats provide *locality* and *DRY*, but they fail to provide *independence & extensibility*. Multiple microformats may conflict with one another, and adding custom fields is not supported, neither technically nor socially within the microformat community [8]. Though it may be technically possible to create one’s own microformat, it is neither recommended nor made easy by the parsing architecture, which tends to assume a fixed set of well-defined vocabularies.

In the current deployments of microformats and their associated parsing tools, *self-containment* is supported, though this assumes that parsers are aware of all possible vocabularies ahead of time. If microformats were deployed according to their specification – using a profile URI – they would not provide true self-containment, as the profile URI needs to be copied into the `<head>` of the target document, an operation which precludes many uses of self-containment, e.g. widgets.

eRDF. eRDF is a syntax that also uses the existing HTML attributes `class`, `rel`, and `title` to enable the embedding of simple, single-layer-depth RDF graphs. It builds on some Dublin Core [7] conventions for embedding attributes in HTML. eRDF provides *DRY* and *locality*, but, because it also requires namespace declarations in the document’s `<head>`, does not fully provide *self-containment*. eRDF has good support for *independence & extensibility*, though not complete support, as any RDF

graph more complex than a single layer of attributes becomes fairly difficult to express.

GRDDL. GRDDL is a technique for declaring and applying transformations to XML documents in order to extract RDF, usually in RDF/XML form. GRDDL can be used specifically to process microformats, eRDF, and RDFa, as long as the HTML document uses a profile URI. GRDDL offers *independence & extensibility* and *DRY*, since anyone can write a GRDDL transformation and the whole point of the transformation is to reuse the existing information in the HTML. However, the end product of a GRDDL transformation no longer contains any HTML DOM information. In addition, the vocabulary-specific transformation is specified in the HTML document’s `head`. Thus, GRDDL does not provide *locality* or *self-containment*.

RDFa. RDFa was built from the ground up to enable the embedding of most RDF graphs in HTML. To accomplish this task in a consistent way:

- new attributes `about`, `resource`, `instanceof`, `property`, and `content` were introduced,
- the existing attributes `rel` and `href` were applied to all elements, not just `<a>`.

As a result, a single syntax can be used for embedding just about any RDF graph, including even blank nodes: RDFa thus supports *independence & extensibility*. Like eRDF, RDFa also supports *DRY* and *locality*. In addition, because RDFa relies on `xmlns` for namespace declarations, it can easily be written so as to support *self-containment*, making it trivial to copy and paste chunks of HTML+RDFa into a page: no vocabulary-specific information is required in the `<head>`.

Selecting a Target for hGRDDL Transforms. Note how the set of data expressible by RDFa is a superset of that expressible by eRDF and microformats (given proper semantic mapping of the microformats). Note also how GRDDL, though it is infinitely flexible in what it can express, does not provide *locality* or *self-containment*. Though the validity of these principles is debatable (and RDFa’s satisfaction of the principles a bit of a tautology), we believe they provide real end-user value. Thus, we choose RDFa as the target for hGRDDL transforms.

4. hGRDDL: Transforming to RDFa

We propose hGRDDL, a simple technique for transforming HTML-embedded ad-hoc semantics into HTML-embedded RDFa. The goal is to provide a single, flexible development path for tool builders, while preserving the existing properties of the original syntax. In particular, the hGRDDL HTML output renders exactly like its input: the embedded semantics remain *local* to the rendered data they pertain to.

Of course, as we are not modifying the syntax of the published document, the limitations remain, too. A microformat or eRDF page, even when hGRDDL-enabled, does not provide self-containment. The point of using RDFa as a target syntax is to unify the client-side toolset and encourage a maximally expressive syntax. However, existing syntaxes cannot gain properties they do not already support.

4.1. GRDDL

In the case of HTML documents, GRDDL uses the `profile` attribute, and takes the following steps:

- (i) detect and retrieve the `profile` indicated in the `head`
- (ii) run a GRDDL processor on the profile document to retrieve a
`grddl:profileTransformation`
triple that indicates the URI of the document-level transformation
- (iii) fetch and apply the transformation to the original HTML document to obtain RDF/XML

4.2. Finding the hGRDDL Transform

hGRDDL, as its name implies, is strongly inspired by GRDDL. Unfortunately, because GRDDL does not provide a way to label transformations according to output format, we cannot use it as-is. We can, however, use a very similar approach without interfering with any existing GRDDL transforms.

hGRDDL bootstraps off the GRDDL `profile`-based process, using the alternate predicate `hgrddl:hProfileTransformation`. This new predicate subclasses `grddl:profileTransformation` so that existing GRDDL processors can use this transformation as a generic way to extract an RDF serialization, which is what the RDFa output of the hGRDDL transformation provides. Note, however,

that the hGRDDL transform is expected to be performed by the user's browser *before* the HTML is rendered, thus the minting of a new predicate. See Figure 1 for an illustration.

4.3. The Transformation

Because the HTML output should render exactly like the input, an hGRDDL transformation should take great care to preserve all of the HTML markup that might be used to style the content or that might in any other way affect it. Specifically, all `class` names should be preserved, and the element structure of the DOM tree should be preserved. The only significant changes should be in the non-CSS attributes.

Preservation of the element structure of the DOM tree may not be possible when there is no one-to-one mapping of the vocabularies: e.g. a full name becomes a first and last name. In such cases, we recommend that an alternate RDF vocabulary be utilized so that the addition of new elements be avoided at all costs. If the transformation must add new elements, it should strive to use as few as possible, though one must acknowledge that the risk of changing the rendering of the page goes up significantly with this approach.

4.4. An Example: hCard

The point of hGRDDL, and its simplicity, are best expressed by example. Consider the following simple, fictitious hCard:

```
...
<head profile="http://www.w3.org/2006/03/hcard">
...
</head>
...
<div class="vcard">
  <a class="url fn" href="http://www.w3.org/People/Berners-Lee/">
    Tim Berners-Lee
  </a>
  <div class="org">W3C</div>
</div>
```

We assume that the hCard profile has been updated to indicate an hGRDDL transformation. The output of the hGRDDL processor should then be:

```
...
<head profile="http://www.w3.org/2006/03/hcard">
...
</head>
...
<div class="vcard" instanceof="card:Card"
  xmlns:card="http://www.w3.org/2001/vcard-rdf/3.0#">
  <a class="url fn" property="card:fn" rel="card:url"
    href="http://www.w3.org/People/Berners-Lee/">
    Tim Berners-Lee
  </a>
  <div class="org" property="card:org">W3C</div>
</div>
```

Note how all existing `class` names were preserved. Only the `instanceof`, `property` and `rel` attributes were used. The output HTML should render exactly like the input. Note also how this syntax-specific transformation is relatively straightforward: it need only add a handful of attributes where it finds the specific microformat properties.

Hard-Wiring some Transforms. Most microformats are published without a profile URI. As a result, most microformat parsers look through the DOM for the top-level `class` values expected for each microformat. Fortunately, this same approach can be used in hGRDDL: look for the top-level microformat `class`, and use it as a flag of microformat presence. When a specific microformat is detected, hGRDDL can simply assume the presence of a profile it already knows about for that microformat, exactly like existing microformat parsers which have hard-wired parsers for each vocabulary.

4.5. Implementation Language

Given that we want a client-side, browser-based, cross-platform implementation, two major routes are immediately worth considering: XSLT or JavaScript+DOM. The XSLT route makes hGRDDL particularly similar to GRDDL. GRDDL already defines ways to handle malformed XHTML and non-XML-based HTML by tidying the input [21], which hGRDDL can mimic. However, this tidying brings problems of its own: it becomes particularly difficult to ensure that the output HTML renders exactly like the input HTML. Because we expect an hGRDDL-aware user agent to perform the hGRDDL processing before rendering the result to the end-user, the XSLT approach is likely unrealistic.

The second approach, JavaScript with DOM access, is far more promising. JavaScript can be tweaked to function seamlessly across browsers, while the DOM can handle less than well formed XHTML. The JavaScript-DOM approach enables us to change the HTML DOM “in place,” tweaking only the attributes we want. The DOM disambiguations performed by a browser when processing malformed input are fully preserved: the hGRDDL JavaScript processor is only tweaking attributes in an existing DOM tree. Thus, we propose an architecture where *the JavaScript transformations are*

run inside a web browser with the input HTML page loaded into the DOM.

5. Implementing hGRDDL

Our prototype implementation is available at <http://ben.adida.net/projects/hgrddl/>

5.1. Deployment Platform

In order to automatically run the hGRDDL transformations, we opted, in our prototype, to deploy the hGRDDL processor as a client-side JavaScript user script. User scripts are typically run automatically by compatible browsers on each page load, with an API that allows for more flexibility, e.g. cross-domain requests, than a normal JavaScript program. The specific API that user script platforms implement was initially defined by Firefox’s GreaseMonkey add-on [11]. We wrote our hGRDDL processor to be as cross-browser-compatible as possible: we tested on Firefox with GreaseMonkey, Safari with CreamMonkey [14], and Opera with its built-in user-script support [20]. We do not yet have an Internet Explorer implementation, as it currently does not provide a complete user-script platform. Firefox 2, Safari 3, and Opera 8 proved successful deployment ground for hGRDDL with very little effort. We hope that IE and add-ons like GreaseMonkey for IE [12] will enable a fully cross-platform approach in the near future. There is no deep reason why this could not be made compatible with IE.

5.2. Components

Profiles and Transforms. We built hGRDDL profiles for two microformats, hCal and hCard. We implemented only the basic features of hCard. Each profile references its own hGRDDL JavaScript transformation built to run against an existing DOM in a browser-independent way. The lion’s share of the code is spent building a cross-platform way to read and write DOM attributes, in particular RDFa attributes that are not yet officially supported by browsers. The transforms themselves are fairly simple, as each microformat field generally maps to a single RDFa field in the appropriate vocabulary.

The hGRDDL Processor. We wrote the hGRDDL processor as a user-script JavaScript program.

Beyond the GRDDL code needed for processing profile documents, our hGRDDL processor required fewer than 25 lines of JavaScript. At a high level, the code looks for a profile attribute in the HTML document's `<head>`, fetches it, looks for the generic hGRDDL profile, and then for an anchor with `rel=hProfileTransformation`. (We effectively hard-wired the GRDDL transformation for the hGRDDL profile.) If an hGRDDL transformation is found, a `<script>` element is added to the document's `<head>` with the transformation's URI as the `src` attribute. This approach is taken to ensure that the hGRDDL transformation executes as unprivileged JavaScript, rather than user-script JavaScript with its greater permissions.

Verifying The Output. We tested our hGRDDL engine's output by invoking the RDFa bookmarklet to extract the N3 syntax for contained triples, once the page was loaded and the hGRDDL engine had done its job. Also interesting is the use of the RDFa Clipboard [2] to verify the preservation of the locality property. Finally, on Firefox, the Operator plugin shows the presence of RDFa, but only after the RDFa bookmarklet is invoked and triggers a DOM change which the plugin then notices. (The asynchronous loading of the hGRDDL transform means Operator does not notice the RDFa change until the DOM tree changes again.)

5.3. Limitations

Our approach is currently a prototype. As such, it has a number of limitations. We highlight the major ones here.

JavaScript and DOM Dependencies. The consumer must have a full JavaScript interpreter with DOM access. If one wants to use hGRDDL for machine-only fetching and parsing of information, this approach may be too heavyweight. We hope that any profile will include both hGRDDL and classic GRDDL transforms: machine-only parsers will choose the GRDDL route.

Delayed Loading. The hGRDDL transformation is fully performed only after the dynamically added `<script>` is loaded and executed. Because this loading and execution is performed asynchronously, client-side tools that trigger on the `window.onload` event may not notice the RDFa immediately. A

more robust implementation of hGRDDL should ensure that the `window.onload` event isn't fired until all transformations have completed.

Dynamic Pages. Performing hGRDDL transformations on the fly when pages dynamically update will likely be slow. Though this limitation is already seen in many browser extensions, including Operator [13], this remains an area of concern come deployment time.

6. Conclusion & Future Work

We have presented hGRDDL, a particularly simple proposal for transforming all HTML-embedded structured data syntaxes into RDFa. The goal is to ensure that tool builders can focus their efforts on RDFa, the most flexible and expressive syntax for embedding RDF in HTML, all the while benefiting from the "hundreds of millions of pages" already deployed with microformats. We have shown how easy it is to implement hGRDDL with cross-browser compatibility.

In the future, it will be particularly interesting to integrate hGRDDL directly into a major RDFa parser. It will also be important to conduct extensive tests to ensure that, as we hypothesize, the rendering of the content is never affected by an hGRDDL transform. We will also need to make sure that hGRDDL is made efficient enough to function inside highly dynamic web pages. It may also be interesting to consider hGRDDL as a publisher-side pre-processing tool, when publishers eventually decide to fully upgrade to RDFa.

7. Acknowledgements

The author wishes to acknowledge Keith Alexander, Dan Connolly, Michael Kaply, and Elias Torres for a number of productive discussions, and the anonymous reviewers for very useful contributions and tweaks. The original idea for hGRDDL came from discussions between the author and Dan Connolly. The very first versions of the hGRDDL code were written in collaboration with Elias Torres.

References

- [1] 30 Boxes. 30boxes.com, last viewed June 28th 2007.
- [2] Ben Adida. RDFa Clipboard. <http://www.w3.org/2006/07/SWD/RDFa/impl/js/rdfa-clipboard/>, last viewed June 28th 2007.
- [3] Ben Adida and Mark Birbeck. RDFa Primer. <http://www.w3.org/TR/xhtml-rdfa-primer/>, last viewed June 28th 2007.
- [4] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). <http://www.w3.org/TR/grddl/> last viewed June 28th 2007.
- [5] Creative Commons. <http://creativecommons.org>, last viewed June 28th 2007.
- [6] Ian Davis. RDF in HTML (eRDF). <http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml>, last viewed June 28th 2007.
- [7] Dublin Core Metadata Initiative. <http://dublincore.org/>, last viewed June 28th 2007.
- [8] Dimitrios Zachariadis. 4d: A geo microformat alternative proposal. <http://microformats.org/wiki/thoughts-on-extending-the-geo-microformat>, last viewed June 28th 2007.
- [9] dot Mac. <http://mac.com>, last viewed June 28th 2007.
- [10] Steven Pemberton et al. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). <http://www.w3.org/TR/xhtml1/>, last viewed June 28th 2007.
- [11] GreaseMonkey Firefox Add-On. <http://www.greasespot.net/>, last viewed June 28th 2007.
- [12] Greasemonkey for IE. <http://www.gm4ie.com/>, last viewed June 28th 2007.
- [13] Michael Kaply. Operator. <http://www.kaply.com/weblog/category/operator/>, last viewed June 28th 2007.
- [14] Kato Kazuyoshi. CreamMonkey. <http://creammonkey.sourceforge.net/>, last viewed June 28th 2007.
- [15] LinkedIn. <http://linkedin.com>, last viewed June 28th 2007.
- [16] Frank Manola and Eric Miller. RDF Primer. <http://www.w3.org/TR/rdf-primer/>, last viewed June 28th 2007.
- [17] microformats.org turns 2. <http://microformats.org/blog/2007/06/21/microformatsorg-turns-2/>, last viewed June 28th 2007.
- [18] Microformats. <http://microformats.org>, last viewed June 28th 2007.
- [19] MySpace. <http://myspace.com>, last viewed June 28th 2007.
- [20] Take Control with User JavaScript. <http://www.opera.com/support/tutorials/userjs/>, last viewed June 28th 2007.
- [21] Dave Raggett. HTML Tidy Library Project. <http://tidy.sourceforge.net/>, last viewed June 28th 2007.
- [22] W3C. Document Object Model (DOM). <http://www.w3.org/DOM/>, last viewed June 28th 2007.