A Model for Open Semantic Hyperwikis

Philip Boulain, Nigel Shadbolt, and Nicholas Gibbins

IAM Group, School of Electronics and Computer Science,
University of Southampton, University Road,
Southampton SO17 1BJ, United Kingdom
{prb,nrs,nmg}@ecs.soton.ac.uk
http://users.ecs.soton.ac.uk/{prb,nrs,nmg}

Abstract. Wiki systems have developed over the past years as lightweight, community-editable, web-based hypertext systems. With the emergence of semantic wikis such as Semantic MediaWiki [6], these collections of interlinked documents have also gained a dual role as ad-hoc RDF [7] graphs. However, their roots lie in the limited hypertext capabilities of the World Wide Web [1]: embedded links, without support for features like composite objects or transclusion. Collaborative editing on wikis has been hampered by redundancy; much of the effort spent on Wikipedia is used keeping content synchronised and organised.[3] We have developed a model for a system, which we have prototyped and are evaluating, which reintroduces ideas from the field of hypertext to help alleviate this burden.

In this paper, we present a model for what we term an 'open semantic hyperwiki' system, drawing from both past hypermedia models, and the informal model of modern semantic wiki systems. An 'open semantic hyperwiki' is a reformulation of the popular semantic wiki technology in terms of the long-standing field of hypermedia, which then highlights and resolves the omissions of hypermedia technology made by the World Wide Web and the applications built around its ideas. In particular, our model supports first-class linking, where links are managed separately from nodes. This is then enhanced by the system's ability to embed links into other nodes and separate them out again, allowing for a user editing experience similiar to HTML-style embedded links, while still gaining the advantages of separate links. We add to this transclusion, which allows for content sharing by including the content of one node into another, and edit-time transclusion, which allows users to edit pages containing shared content without the need to follow a sequence of indirections to find the actual text they wish to modify. Our model supports more advanced linking mechanisms, such as generic links, which allow words in the wiki to be used as link endpoints.

The development of this model has been driven by our prior experimental work on the limitations of existing wikis and user interaction.We have produced a prototype implementation which provides first-class links, transclusion, and generic links.

Key words: Open Hypermedia, Semantic Web, Wiki

1 Introduction

Hypermedia is a long-standing field of research into the ways in which documents can expand beyond the limitations of paper, generally in terms of greater cross-referencing and composition (reuse) capability. Bush's *As We May Think* [4] introduces a hypothetical early hypertext machine, the 'memex', and defines the "essential feature" of it as "the process of tying two items together". This *linking* between documents is the common feature of hypertext systems, upon which other improvements are built.

As well as simple binary (two endpoint) links, hypertext systems have been developed with features including n-ary links (multiple documents linked to multiple others), typed links (links which indicate something about *why* or *how* documents are related), generic links (links whose endpoints are determined by matching criteria of the document content, such as particular words), and composite documents (formed by combining a set of other documents).

Open Hypermedia extends this with *first-class* links (and anchors) which are held external to the documents they connect. These allow links to be made to immutable documents, and to be added and removed in sets, often termed 'linkbases'. One of the earliest projects attempting to implement globally-distributed hypertext was Xanadu [8], a distinctive feature of the design of which was *transclusion*: including (sections of) a document into another by reference.

The World Wide Web, while undeniably successful, only implements a very small subset of these features—binary, embedded links—with more complicated standards such as XLink failing to gain mainstream traction. Since then, applications have been built using the web as an interface, and following its same, limited capabilities. One of these classes of applications is the semantic wiki, an extension of the community-edited-website concept to cover typed nodes and links, such that the wiki graph structure maps to a meaningful RDF graph.

In this paper, we present a model which extends such systems to cover a greater breadth of hypermedia functionality, while maintaining this basic principle of useful graph mapping. We introduce the Open Weerkat system and describe the details of its implementation which relate to the model.

2 Open Weerkat

Open Weerkat is a model and system to provide a richer hypertext wiki. This implementation is built upon our previous Weerkat extensible wiki system [2].

From our experimental work [3], we have identified the need for transclusion for content-sharing, better support for instance property editing, and generic linking, which requires first-class links.

At the same time, we must not prohibit the 'non-strict' nature of wikis, as dangling links are used as part of the authoring process. We also wish to preserve the defining feature of semantic wikis: that there is a simple mapping between nodes and links in the wiki, and RDF resources and statements.

In the rest of this section, we look at the core components of the model, and some of the implementation considerations.

2.1 Atomic Nodes

The core type of the model is that class which is fundamental to all wiki designs: the individual document, page, article, or component. We use here the general and non-domain-specific term *node*.

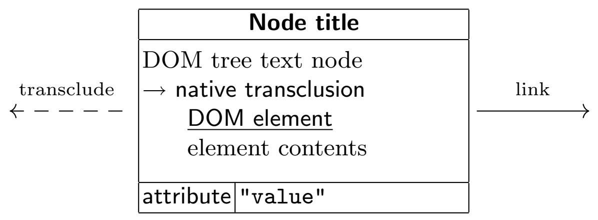

Fig. 1. Model diagram legend

Components As we have said, our model draws from both hypermedia and semantic wiki. In particular, we preserve the notion that wiki nodes are parallel to semantic web resources. Because these resources are atomic (RDF cannot perform within-component addressing on them, as that is only meaningful for an *representation* of a resource), we have carefully designed our wiki model to not rely on link endpoint specifications which go beyond what can be reasonably expressed in application-generic RDF. Anything about which one wishes to make statements, or to which one wishes to link, must have a unique identity in the form of an URI, rather than some form of (URI, within-specifier) pairing.

Figure 1 shows the components of a node in the model. (We use this diagram format, which draws inspiration from UML class diagrams, to depict example hypertexts throughout this paper.)

Every node has a title which serves as an identifier. Titles are namespaced with full stops ('.'), which is useful for creating identities for content which nominally belongs within another node.

Node content is either a DOM tree of wiki markup, or an atomic object (e.g. an image binary). A notable element in the DOM tree is the 'native transclusion', which indicates that another node's content should be inserted into the tree at that point. This is necessary to support the linking behaviour described below, and is distinct from user-level transclusion using normal links.

The bottom of the format shows the attribute-value pairs for the node. The domain of attributes is other nodes, and the domain of values are literals and other nodes. These are effectively very primitive embedded, typed links, and are used to provide a base representation from which to describe first-class links.

Identity, Meta-nodes, and RDF It is a design goal of the model that the hyperstructure of the wiki is isomorphic to a useful RDF graph. That is, typed links between pages are expressible as an RDF statement relating the pages, and attributes on a page are statements relating that page to the associated value. The link in figure 5 should be presented (via RDF export, a SPARQL endpoint, or such) as the triple (Phil, Likes, Perl), with appropriate URIs. (Note that the anchor is not the subject—we follow the native transclusion back into the owning Phil node.) The attribute of the Perl node in the same figure should be presented as the triple (Perl, syntax, elegant). (For 'fat' links with more than two endpoints, there is a triple for each pairing of sources and targets.)

For this, we grant each node a URI, namespaced within the wiki. However, 'the node Perl' and 'Perl the programming language' are separate resources. For example, the node Perl may have an URI of `http://wiki.example.org/node/Perl`. Yet a typed link from Perl is clearly a statement about Perl itself, not a node about Perl. The statements are about the associated resource `http://wiki.example.org/resource/Perl`. (This may be owl:sameAs some external URI representing Perl.)

In order to describe the node itself (e.g. to express that it is in need of copy-editing), a Perl.meta node represents `http://wiki.example.org/nodes/Perl`. This meta node itself has an URI `http://wiki.example.org/nodes/Perl.meta`, and could theoretically have a 'meta meta' node. Effectively, there is an 'offset' of naming, where the wiki identifier Perl is referring to Perl itself semantically, and the Perl node navigationally; the identifier Perl.meta is referring to the Perl node semantically, and the Perl meta-node navigationally.

Versions We must give consideration to the identity of versions, or 'revisions' of a node. We wish to support navigational linking (including transclusion) to old versions of a node. However, we must also consider our semantic/navigational offset: we may wish to write version 3 of the Perl node, but we do not mean to assert things about a third revision of Perl itself. Likewise, a typed link to version 3 of the Perl node is not a statement about version 3 of Perl: it is a statement about Perl which happens to be directed to a third revision of some content about it.

We desire three properties from an identifier scheme for old versions:

Semantic consistency. Considers the version of the content about a resource irrelevant to its semantic identity. All revisions of the Perl node are still about the same Perl.

Navigational identity. Each revision of a node (including meta-nodes) should have distinct identity within the wiki, so that it may be linked to. Intuitively, despite the above, version 3 of the Perl node is Perl_3, not $\mathsf{Perl.meta}_3$.

Semantic identity. Each revision of a node (including meta-nodes) should have a distinct URI, such that people may make statements about them. (Perl_3.meta, writtenBy, Phil) should express that Phil wrote version 3 of the content for the Perl node.

We can achieve this by allowing version number specification both on the node and any meta-levels, and dropping the version specification of the last component to generate the RDF URI. Should somebody wish to make statements about version 4 of the text about version 3 of the text about Perl, they could use the URI `Perl;3/meta;4/meta`. This is consistent with the 'node is resource itself; meta-node is node' approach to converting typed links into statements. Additionally, we have no need to attempt to express that Perl and Perl_3 are the same semantic resource, as this mechanism allocates them the same URI.

It should be stressed that *namespacing* components of the node identifier cannot have versions attached, as any versioned namespace content does not affect the node content. For example, $\text{Languages}_2.\text{Perl}_3$ is not a valid identifier, and would be isomorphic to Languages.Perl_3 if it were.

Representations Each `node` URI should have a representation predicate which specifies a retrievable URL for a representation. (We do not claim to have any authority to provide a representation of Perl itself, merely our node about Perl.) For example, (wiki:node/Perl, representation, `http://wiki.example.org/content/Perl.html`). There may be more than one, in which case each should have a different MIME type. Multiple representations are derived from the content: for example, rendering a DOM tree of markup to an XHTML fragment. Hence, the range of MIME types is a feature of the rendering components available in the wiki software to convert from the node's content.

Should an HTTP client request the `wiki:node/Perl` resource itself, HTTP content negotiation should be used to redirect to the best-matching representation. In the spirit of the '303 convention' [9], if the HTTP client requests RDF, they should be redirected to data about the requested URI: i.e. one meta-level higher. This inconsistency is unfortunately a result of the way the convention assumes that all use of RDF must necessarily be 'meta' in nature, but we have considered it preferable to be consistent with convention than to unexpectedly return data, not metadata, RDF in what is now an ambiguous case. Clients which wish to actually request the Perl node's content itself in an RDF format, should such exist, must find the correct URI for it (e.g. wiki:content/Perl.ttl) via the representation statements.

Requests to resource URIs (e.g. `wiki:resource/Perl`) are only meaningful in terms of the 303 convention, redirecting RDF requests to data about `wiki:node/Perl`. There are no representations available in the wiki for these base resources—only for nodes about them—so any non-RDF type must therefore must be 'Not Found'.

2.2 Complex Nodes

We can build upon this base to model parametric nodes, whose content may be affected by some input state.

Node identity MediaWiki's form of transclusion, 'templates', also provides for arguments to be passed to the template, which can then be substituted in. This

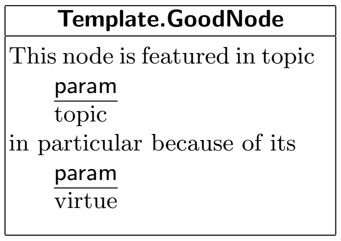

Fig. 2. Exemplary parametric node

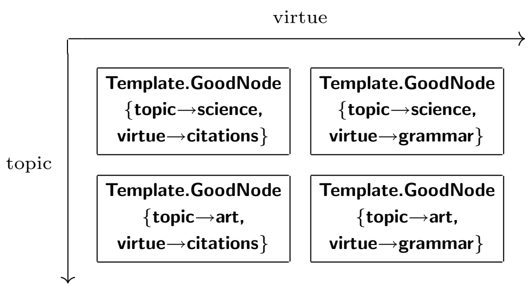
virtue
topic
Template.GoodNode {topic→science, virtue→citations}
Template.GoodNode {topic→science, virtue→grammar}
Template.GoodNode {topic→art, virtue→citations}
Template.GoodNode {topic→art, virtue→grammar}

Fig. 3. Instance-space of a parametric node

is in keeping with the general MediaWiki paradigm that templates are solely for macro processing of pages.

We propose a generalisation, whereby pages may be *instantiated* with arbitrary key/value pairs. The range of our links are node identifiers, so we consider these parameters as part of the identity of an *instantiation* in a (likely infinite) multi-dimensional space of instances. Figure 3 shows a subset of the instance space for a node, figure 2, which has parameters topic and virtue. There is assumed to be an instance at any value of these parameters, although evidently all such instances are 'virtual', with their content generated from evaluating the parametric Template.GoodNode node.

We do not use an $(identifier, parameters)$ pair, as this does not fit the Semantic Web model that any resource worth making statements about should have identity. Granting instances in-system identity is useful, as it encapsulates all necessary context into one handle.

To guarantee that all isomorphic instantiations of a page use the same identifier, parameters must be sorted by key in the identifier. Note that this is orthogonal to user interface concerns—the restriction is upon the identity used by links to refer to 'this instance of this node with these parameters', not upon the display of these parameters when editing such a link. As with revision specifiers, parameters upon namespace components of the identifier are meaningless and forbidden.

Within the node's content, parameters may be used to fill in placeholders in the DOM tree. These placeholders may have default value should the parameter not be provided; and the default-default parameter is to flag an error. For example, a parameter may be used to fill in a word or two of text, or as the target of a link. User interface operations upon Foo {bar→baz}'s content, such as viewing the history, and editing, should map through to Foo, as the instance has no content of its own to operate upon.

Because we model parameterised nodes as a set of static objects with first-class identity which are simply instantiations of a general node, identifiers which do not map to a valid instantiation of a node could be considered non-existent targets. For example, an identifier which specifies a non-existant parameter.

Resource identity We must consider whether such instances are separate Semantic Web resources to each-other, and to the parametric node from which their content is derived. As with version specifiers, parameters affect the *content* of a node, not the resource which it describes. Because the Perl node represents Perl itself, it follows that Perl {bar→baz} still represents Perl. However, as with version specifiers, these node instances still have distinct identity as nodes. As Perl.meta represents the Perl node, so does Perl {bar→baz}.meta represent the Perl {bar→baz} node. Therefore, we can form a URI for a parametric node instance in exactly the same way we form URIs for specific revisions, defined in section 2.1. In brief, the final set of parameters are dropped.

RDF expressions of the hyperstructure should specify that parametric node instances, where used, are derivations of other nodes. For example, remembering that we are making a statement about Perl nodes, not about Perl itself, (Perl {bar→baz}.meta, templatedFrom, Perl.meta).

Eager vs. lazy evaluation The infinite space of non-link parametric node instances can be considered to not exist until they are specified as a link target, as their existence or non-existence in the absence of explicit reference is irrelevant. However, if we also allow parameter values to substitute into the attributes of a node, we can create parametric links. Parametric node instances which are links have the ability to affect parts of the hyperdocument outside of their own content and relations: this is the nature of first-class links. Hence we must consider whether parametric node instantiation, at least for link nodes, is eager (all possible instances are considered to always exist) or lazy (instances only exist if they are explicitly referred to).

Template.FancyLink	
type	Link
source	*param*(from)
target	*param*(to)
decoration	`fancy`

Fig. 4. Free-variable parametric link

Figure 4 highlights a case where this distinction is particularly significant. With lazy evaluation, this template could be used as a macro, in a 'classical' wiki style, to create links. One would have to create links to instances of this link, which would then cause that particular instance to exist and take effect, linking its from and to parameters.

An eager approach to evaluation would, however, treat parametric links as free-variable rules to satisfy. All possible values of from and to would be matched, and linked between. In this case, every node in the hyperdocument would be linked to every other node.

Logically, eager evaluation is more consistent, and potentially more useful: free-variable links are of little utility if one has to explicitly provide them with possible values. It would be better to manually link the nodes, with a type of FancyLink which is then defined to be fancy. If there were some content provided by the Template.FancyLink template, it could still be used, but would simply display this content rather than actually functioning as a link.

This is contrary to common practice on Semantic MediaWiki, which has evolved from practice on Wikipedia, where the templating system works via macro evaluation. We argue that this leads to bad ontology modelling, as class definitions end up embedded within display-oriented templates, such as 'infoboxes'. For example, the common Semantic MediaWiki practice to provide the node about Brazil with a relational link to its capital Brasília would be to include a template in the Brazil node with the parameter capital→Brasília. The template would then contain markup to display a panel of information containing an embedded link of type has capital to the value of the capital parameter.[1]

The problem is that stating that templates have capitals is clearly not correct, and only results in correct information when they are macro-expanded into place.Statements about the template itself must be ignored as they are likely intended to be about whichever nodes use that template. In addition, what could be a statement about the class of countries—that they are the domain of a has capital property—is entangled with the display of this information.

A better approach would be to simply assert the capital on the Brazil page, and then transclude a template whose only role is to typeset the information panel, using only the name of the transcluding page as an implicit parameter. This approach emphasises the use of correct semantics, and using these to inform useful display, rather than 'hijacking' useful display to try to add semantics.

Templating Templating can be achieved through the use of parametric nodes and transclusion. Simple macroing functionality, as in contemporary wiki systems, is possible by transcluding a particular instance of a parametric node which specifies the desired parameter values.

It should be stressed that parametric nodes are *not*, however, a macro preprocessing system. As covered in section 2.2, parametric links are eagerly evaluated:

[1] This example is closely based upon a real case: `http://www.semanticweb.org/wiki/Template:Infobox_Country`

i.e. they are treated as rules, rather than macros which must be manually 'activated' by using them in combination with an existing node. In general, use of macroing for linking and relations is discouraged, as it is better expressed through classes of relation.

2.3 Links

Open Weerkat is an open hypermedia system, so links are first-class: all links are nodes. Nodes which have linking attributes are links. To maintain a normal wiki interface, we present links in an embedded form.

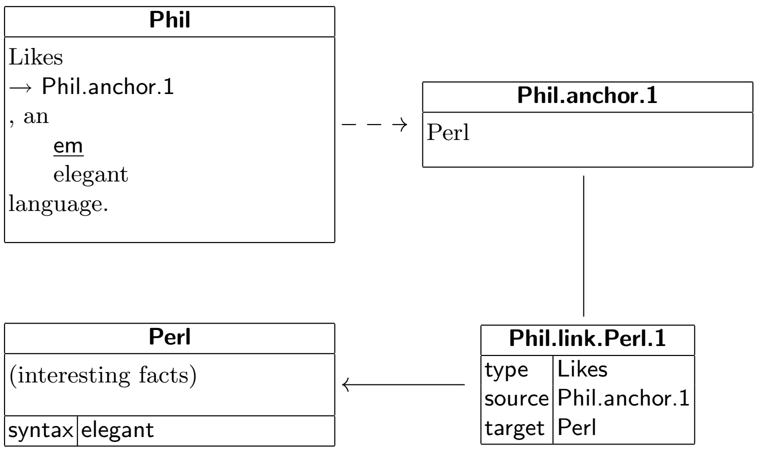

Fig. 5. Linking in the model

Embedded Figure 5 shows user-level linking. As presented to the user in an example plaintext markup, the source for the Phil node would be:

```
Likes [link type=Likes to=Perl Perl], an [em elegant] language.
```

We use edit-time transclusion, where transcluded text is displayed in-line even during the editing of a node, to present the user with the familiar and direct model of embedded linking, but map this into a open hypermedia model. The link element, when written, separates out the link text as a separate, 'anchor' node, and is replaced with native transclusion. A first-class link is then created from the anchor node to the link target. The identity if this link is largely arbitrary, so long as it is unique.

Native transclusion is here an optimisation for creating a named, empty anchor in the DOM, then maintaining a link which transcludes in the node of the same name. It is also considered meronymous: a link involving an anchor is considered to relate the node to which that anchor belongs. Because native transclusion is entirely implicit, only the owning node can natively transclude its anchors.

When editing the node again, the anchor is transcluded back into the node, and converted into a link element with targets from all links from it. (Depending on the exact markup language used to express the DOM for editing, this may require multiple, nested link elements.)

This guarantees that each anchor has a full identity (a node title) in the system. It does not, however, immediately provide a solution to 'the editing problem'—a longstanding issue in hypertext research [5], where changes to a document invalidate external pointers into that document. The anchor names are not here used in the plaintext markup, so ambiguity can arise when they are edited. It should thus be possible to specify the anchor name (as a member of the `*Node*.anchors` namespace) for complicated edits:

```
Likes [link anchor=1 type=Likes to=Scheme Scheme]...
```

A graphical editor could treat the link elements as objects in the document content which store hidden anchor identities, providing this in all cases.

Note that the link's properties in figure 5 are stored as attributes. Theoretically, in the RDF mapping described in section 2.1, an attribute-value pair (source, Phil.anchor.1) in the node Phil.link.Perl.1 is identical to a link of type source from the link to the anchor. However, such an approach would become infinitely recursive, as the source link's source could again be described in terms of links. The attribute-value pairs thus provide a base case with which we can record basic properties needed to describe first-class links.

Transclusive Transclusive links can be used to construct composite nodes. A link is transclusive if its type is a specialisation of Transclusion. A transclusive link replaces the display of its source anchor contents with its target contents. Unlike the 'native transclusion' in section 2.1, user-level transclusive links do not imply a part-of relation. This is because any part-of relation would be between the representations of the nodes, not the resources that the nodes represent. To extend the Brazil example in section 2.2, a country information box is not part of Brazil; instead the Infobox Country node is part of Brazil node.

Edit-time transclusion is user-interface specific, although quite similar to the issues already covered in section 2.3 with the native transclusion performed by embedded anchors. For a simple, text serialisation interface, such as a web form, it is possible to serialise the transcluded content in-place with a small amount of surrounding markup; if the returned text differs, this is an edit of the transcluded node. Again, richer, graphical editors can replace this markup with subtler cues.

Open To realise first-class links while retaining a standard wiki embedded-style editing interface, we have modified Weerkat to work upon document trees, into which links can be embedded, and from which links can be separated. These embedding and separation routines rewrite documents into multiple nodes as is necessary. Transclusion, be it presented at edit-time, or for viewing, is possible via the same mechanism: including the target content within the link's now-embedded anchor.

To embed a link back into a document, including in order to create an XHTML representation of it for display and web navigation, it must be determined which links are applicable to the document being processed. For this, we have defined a new type of module in the system: a link matcher. Link matchers inspect the endpoints of links and determine if the document matches the endpoint criteria. For straightforward, literal links, this is a simple case of identity equality between the endpoint's named document, and the current document.

As part of the storage adaptation for first-class linking, we have introduced an attribute cache, which is fundamentally a triple store whose contents are entirely derived from the attributes of each node. As well as eventually being a useful way to interact with the semantic content of the wiki, this allows us to implement link matching in an efficient way, by querying upon the store.

For example, in the literal endpoint case, assuming suitable prefixes and subtype inference, we can find such links with a set of simple SPARQL queries, selecting $?l$ where:

1. `{ ?l type link . ?l source Scheme . }`
2. `{ ?l type link . ?l source Scheme_5 . }`
3. `{ ?l type link . ?l target Scheme . }`
4. `{ ?l type link . ?l target Scheme_5 . }`

The first two queries find links where this node is a source; the latter two, where it is a target. We must also find links from or to the specific version of the current node, which is provided by queries two and four.

This approach can be extended to deal with endpoints which are not literal, which we consider 'computed'.

Query Query endpoints are handled as SPARQL queries, where the union of all values of the selected variables is the set of matched pages. For example, a query endpoint of `SELECT ?n WHERE { ?n Paradigm Functional . }` would link from or to all functional programming languages. This kind of endpoint can be tested for a specific node via a SPARQL term constraint:

```
SELECT ?n WHERE { ?n Paradigm Functional .
                  FILTER ( ?n = Scheme ) }
```

If multiple variables are selected, the filter should combine each with the logical or operator, so as to retrieve any logically-sound solution to the query, even if some of the variables involved are not the node we are interested in linking with.

Generic Generic endpoints can be implemented as a filtering step on query endpoints.[2] We define a postcondition `CONTAINS ( ?n, "term" )` to filter the

[2] An alternative approach may be to assert triples of the form (Scheme, containsTerm, `term`), but this would put a great load on the triplestore for each content edit.

solutions by those where the node n contains the given term. This postcondition can be implemented efficiently by means of a lexicon cache, from each term used by any generic link, to a set of the nodes using that term. Changes to generic links add or remove items from the lexicon, and changes to any node update the sets for any terms they share with the lexicon. If `CONTAINS` is used alone, n is implied to be the universal set of nodes, so matching is a simple lexicon lookup.

To be useful for generic linking, `CONTAINS` implies an anchor at the point of the term when it is used as a source endpoint. For example, `CONTAINS ( ?n, "Scheme" )` matches the Scheme node, but should link not from the entire node, but from the text "Scheme" within it. For user interface reasons, it is desirable to restrict this only to the first occurrence of the term for non-transclusive links, so that the embedded-link document is not peppered with repeated links. For transclusive links, however, it is more consistent and useful to match all occurrences. While transclusive generic links are a slightly unusual concept, it is possible that users will find innovative applications for them. For example, if it is not possible to filter document sources at a node store level for some reason, a generic, transclusive link could be used to censor certain profane terms.

Multiple `CONTAINS` constraints can be allowed, which require that a node contains all of the terms. Any of the terms are candidates for implicit anchors: i.e. whichever occurs first will be linked, or all will be replaced by transclusion.

Parametric We can use SPARQL variables for parametric links. Every SPARQL variable is bound to the parameter element in the node's DOM tree with the same name: variables and parameters are considered to be in the same namespace. This allows the content to reflect the query result which matched this link. If the query allows `OPTIONAL` clauses which can result in unbound variables, then they could potentially have values provided by defaults from the parameter definitions in the DOM. Default values are meaningless for parameters which appear as compulsory variables in the query, as the query engine will either provide values, or will not match the link.

Parametric links may have interdependent sources and targets, in which case they are simple functional links (the source can be a function of the target, and the target an inverse function of the source). Link matching is performed pairwise for all source and target combinations. For example, consider a link with these select endpoints:

```
source: ?thing WHERE { ?thing Colour Red . }
target: ?img WHERE { ?img Depicts ?thing . }
target: ?img WHERE { ?img Describes ?thing . }
```

This would create links from all nodes about things which are red, to all nodes which depict or describe or those red things. To perform this match, we union each pair of the clauses into a larger query:

```
SELECT ?thing, ?img WHERE {
 ?thing Colour Red . ?img Depicts ?thing .
 FILTER ( ?thing = Scheme || ?img = Scheme ) }
```

A similar query would also be performed for Describes. Note that we may receive values for the variables used as source or target which are not the current node if it matches in the opposite direction. We must still check that any given result for the endpoint direction we are interested in actually binds the variable to the current node. In this example, current node Scheme is not Red, so the query will not match, and no link will be created.

The pairwise matching is to be consistent with the RDF representation presented in section 2.1, and the 'or' nature of matching with static endpoints: a link must only match the current node to be used, and other endpoints may be dangling. An alternative approach would be to create a 'grand union' of all sources and targets, such that all are required to be satisfied. Neither approach is more expressive at an overall level: with a pairwise approach, a single target endpoint can include multiple `WHERE` constraints to require that all are matched; with a union approach, independent targets can be achieved through use of multiple links (although they would no longer share the same identity). The union approach is more consistent with regard to the interdependence of variables; with the pairwise approach, one matching pair of source/target endpoints may have a different variable binding for a variable of the same name to another. However, it loses the RDF and static endpoint consistency. Ultimately, the decision is whether the set of targets is a function of the set of sources (and vica-versa with the inverse), or if it is the *mapping* of a function over each source. In lieu of strong use cases for n-ary, interdependent, parametric links (most are better modelled as separate links), we choose the former for its greater consistency, and ability for a single link to provide both behaviours.

Functional We also give consideration to arbitrarily-functional links. These are computationally expensive to match in reverse (i.e. for target-end linking and backlinks) unless the functions have inverses. We do not currently propose the ability for users to write their own Turing-complete functions, as the complexity and performance implications are widespread.

However, we can potentially provide a small library of 'safe' functions: those with guaranteed characteristics, such as prompt termination. One such example which would be of use is a 'concatenate' function:

```
source: SELECT ?n WHERE { ?n type ProgLang . }
target: CONCAT( "Discuss.", ?n )
```

This would be a link from any programming language to a namespaced node for discussing it.However, it highlights the reversibility problem: the inverse of `CONCAT` has multiple solutions. For example, "ABC" could have been the result of $CONCAT$("A", "BC"), $CONCAT$("AB", "C"), or a permutation with blank strings. Hence, while it is easy to match the source, and then determine the target, it is not practical to start with the target and determine the source.

We suggest that any endpoint which is an *arbitrary* function of others in this manner must therefore only ever be *derived.* Matching is performed against all other endpoints, and then the functional endpoints are calculated based on the results. A link from `CONCAT( ?n, ".meta")` to `CONTAINS( ?n, "lambda" )` would only ever match as a backlink: showing that any node containing 'lambda' would have been linked from its meta-node, without actually showing that link on the meta-node itself. A link with only arbitrarily functional endpoints will never match and is effectively inert.

3 Conclusions

In this paper, we have approached the perceived requirement for a more advanced communually-editable hypertext system. We have presented a solution to this as a model for a "semantic open hyperwiki" system, which blends semantic wiki technology with open hypertext features such as first-class linking. We also offer an approach to implementing the more advanced link types with a mind towards practicality and computational feasibility.

Providing users with stronger linking and translusion capabilities should help improve their efficiency when working on editing wikis such as Wikipedia. Inter-document linking forms a major component of current editing effort, which we hope to help automate with generic links. Content re-use is complicated by surrounding context, but even in cases where texts could be shared, technical usability obstacles with current macro-based mechanisms discourage editors from doing so. We address this with the concept of edit-time transclusion, made possible by the wiki dealing with programatically manipulatable tree structures.

Beyond this, we wish to address other user-study-driven design goals, such as improving versioning support that allows for branching.

References

1. T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2):74–82, 1992.
2. P. Boulain, M. Parker, D. Millard, and G. Wills. Weerkat: An extensible semantic wiki. In *Proceedings of 8th Annual Conference on WWW Applications*, Bloemfontein, Free State Province, South Africa, 2006.
3. P. Boulain, N. Shadbolt, and N. Gibbins. *Weaving Services, Location, and People on the WWW*, chapter Studies on Editing Patterns in Large-scale Wikis, pages 325–349. Springer, 2009. In publication.
4. V. Bush. As We May Think. *The Atlantic Monthly*, 176:101–108, Jul 1945.
5. H. Davis. *Data Integrity Problems in an Open Hypermedia Link Service*. PhD thesis, ECS, University of Southampton, 1995.
6. M. Krötzsch, D. Vrandečić, and M. Völkel. Wikipedia and the semantic web - the missing links. In *Proceedings of the WikiMania2005*, 2005. Online at `http://www.aifb.uni-karlsruhe.de/WBS/mak/pub/wikimania.pdf`.
7. F. Manola and E. Miller. RDF Primer. Technical report, W3C, Feb 2004.
8. T. Nelson. *Literary Machines*. Mindful Press, Sausalito, California, 93.1 edition, 1993.
9. L. Sauermann, R. Cyganiak, and M. Völkel. Cool URIs for the Semantic Web. Technical Report TM-07-01, DFKI, Feb 2007.