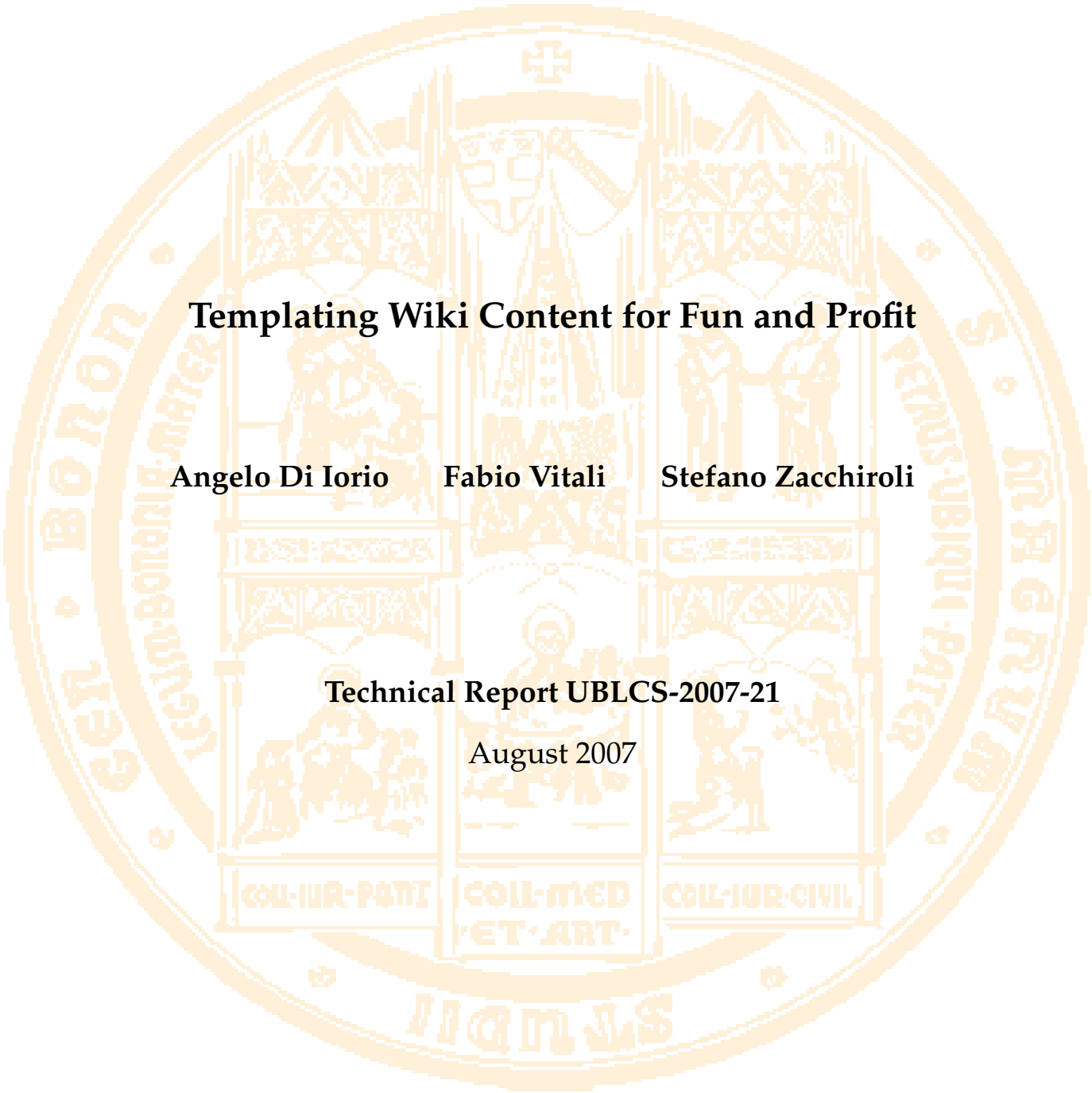

Templating Wiki Content for Fun and Profit

Angelo Di Iorio **Fabio Vitali** **Stefano Zacchiroli**

Technical Report UBLCS-2007-21

August 2007

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in PDF and gzipped PostScript formats via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`.

Recent Titles from the UBLCS Technical Report Series

2006-29 *Motifs in Evolving Cooperative Networks Look Like Protein Structure Networks*, Hales, D., Arteconi, S., December 2006.

2007-01 *Extending the Choquet Integral*, Rossi, G., January 2007.

2007-02 *Towards Cooperative, Self-Organised Replica Management*, Hales, D., Marcozzi, A., Cortese, G., February 2007.

2007-03 *A Model and an Algebra for Semi-Structured and Full-Text Queries (PhD Thesis)*, Buratti, G., March 2007.

2007-04 *Data and Behavioral Contracts for Web Services (PhD Thesis)*, Carpineti, S., March 2007.

2007-05 *Pattern-Based Segmentation of Digital Documents: Model and Implementation (PhD Thesis)*, Di Iorio, A., March 2007.

2007-06 *A Communication Infrastructure to Support Knowledge Level Agents on the Web (PhD Thesis)*, Guidi, D., March 2007.

2007-07 *Formalizing Languages for Service Oriented Computing (PhD Thesis)*, Guidi, C., March 2007.

2007-08 *Secure Gossiping Techniques and Components (PhD Thesis)*, Jesi, G., March 2007.

2007-09 *Rich Media Content Adaptation in E-Learning Systems (PhD Thesis)*, Mirri, S., March 2007.

2007-10 *User Interaction Widgets for Interactive Theorem Proving (PhD Thesis)*, Zacchiroli, S., March 2007.

2007-11 *An Ontology-based Approach to Define and Manage B2B Interoperability (PhD Thesis)*, Gessa, N., March 2007.

2007-12 *Decidable and Computational Properties of Cellular Automata (PhD Thesis)*, Di Lena, P., March 2007.

2007-13 *Patterns for Descriptive Documents: a Formal Analysis*, Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A. A., Vitali, F., April 2007.

2007-14 *BPM + DM = BPDM*, Magnani, M., Montesi, D., May 2007.

2007-15 *A Study on Company Name Matching for Database Integration*, Magnani, M., Montesi, D., May 2007.

2007-16 *Fault Tolerance for Large Scale Protein 3D Reconstruction from Contact Maps*, Vassura, M., Margara, L., di Lena, P., Medri, F., Fariselli, P., Casadio, R., May 2007.

2007-17 *Computing the Cost of BPMN Diagrams*, Magnani, M., Montesi, D., June 2007.

2007-18 *Expressing Priorities, External Probabilities and Time in Process Algebra via Mixed Open/Closed Systems*, Bravetti, M., June 2007.

2007-19 *Design and Evaluation of a Wide-Area Distributed Shared Memory Middleware*, Mazzucco, M., Morgan, G., Panzieri, F., July 2007.

2007-20 *An Object-based Fault-Tolerant Distributed Shared Memory Middleware*, Lodi, G., Ghini, V., Panzieri, F., Carloni, F., July 2007.

2007-21 *Templating Wiki Content for Fun and Profit*, Di Iorio, A., Zacchiroli, S., Vitali, F., August 2007.

Templating Wiki Content for Fun and Profit

Angelo Di Iorio **Fabio Vitali** **Stefano Zacchiroli** [1]

Technical Report UBLCS-2007-21

August 2007

Abstract

Content templating enables reuse of content structures between wiki pages. Such a feature is implemented in several mainstream wiki engine. Systematic study of its conceptual models and comparison of the available implementations are unfortunately missing in the wiki literature. In this paper we aim to fill this gap first analyzing template-related user needs, and then reviewing existing approaches at content templating.

Our investigation shows that two models emerge—functional and creational templating—and that both have weakness failing to properly fit in "The Wiki Way". As a solution, we propose the adoption of creational templates enriched with light constraints, showing that such a solution has a low implementative footprint in state-of-the-art wiki engines, and that it has a synergy with semantic wikis.

1. Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, ITALY

1 Introduction

Template ¡¡

> A gauge, pattern, or mold, commonly a thin plate or board, used as a guide to the form of the work to be executed; as, a mason's or a wheelwright's templet.
> *(1913 Webster)* [4]

In the wiki lingo the term "template" is frequently encountered, but it is used with various meanings. In [10] one of the most widespread interpretation is used, the *presentational template interpretation*: a template is an HTML page sketch with holes that will be substituted by the wiki engine with the rendering of parts (e.g. title and body) of a wiki page content. Using this interpretation, the "work to be executed" of the dictionary definition above is the final page rendering.

In this paper we are concerned instead with the *content template interpretation*: a template is a mechanism to reuse, to some extent, wiki page content across different pages. The "work to be executed" is now the most characteristic action of the wiki workflow: content editing. This justifies our subsequent use of the term "template" to refer to this interpretation of content templating.

(Content) templating is a feature frequently found in state-of-the-art wiki engines, which comes with heterogeneous implementations of (what we believe to be) the same underlying concept. The first aim of this paper is thus to identify the precise meaning, from the point of view of wiki *users*, of templating. We believe this effort to be worthwhile and its outcome to be useful to engine implementors for deciding upon the usefulness of implementing templating as a feature of their engine. As a part of this identification process we recognize two alternative templating models which can be found in state-of-the-art wiki engines: functional templating (in which templates are invoked by name and passed parameters) and creational templating (in which templates are simply copied as new content at the beginning of a page revision history).

Among the two templating models we find no one-size-fits-all model, as both have issues. For example, functional templating fails to preserve the linearity of markup with respect to their final rendered form, possibly diminishing the attractiveness of their markup for newcomers. Creational template on the other hand fails to keep references between original generating templates and final instance pages, thus loosing the ability to spot inconsistencies among pages sharing a common template. The second aim of this paper is therefore to propose a new templating model, called *lightly constrained templating*, which builds on top of creational templating, and overcomes the drawbacks of both other models.

Paper structure As a first step in the identification of content templating, in Section 2 we perform a task analysis considering the wiki user goals that can be mapped to template-related tasks. Then, in Section 3, we detail the characteristics of the functional and creational templating models, characterizing them on the basis of how the APPLYTO conceptual template action is implemented. Section 4 presents lightly constrained templating while Section 5 discusses its deployment in standard and semantic wikis. Section 6 concludes the paper and presents some related works.

2 Task Analysis

The abstract wiki templating concept, which we are in fact trying to corner, is frequently blurred to a specific templating mechanism implementation available in some wiki engine. For example, a WIKIPEDIA [22] user probably thinks that wiki templating as a whole is achieved through the creation of pages like Template:Infobox_software[2] and their invocation from the markup of other pages. Similarly a MoinMoin [13] user hearing "wiki template" probably depicts in her mind the list of pages she can choose from when following a dangling page creation link.

2. `http://en.wikipedia.org/wiki/Template:Infobox_software`

Table 1. Goals and tasks (with roles of users pursuing them) that can be mapped to actions on devices implemented by wiki templating mechanisms.

No.	*Role*	*Task / Goal*
1.	editor	instantiate a predefined boilerplate page; *goal*: (quickly) create a new page that is instance of another one
2.	tailor	create/modify a boilerplate page for future use *goal*: publish a generic page to foster population of the wiki space with its instances, by making easier their creation
3.	editor	copy and modify an existing page; *goal*: (quickly) create a new page that is similar to another one
4.	editor	apply/unapply to a page, a content structure found in other pages *goal*: uniform (distinguish) the content organization of a page with (from) other pages
5.	tailor	create a predefined content organization for future use *goal*: publish a reusable page component to foster population of the wiki space with pages that share content organization at some extent
6.	tailor	change at once the structure of several pages *goal*: (quickly) change at once the content organization of several pages to smooth visitors' experience when reading them
7.	editor	copy and reorganize the content of an existing page *goal*: create an alternative view of some content on the already present in the wiki

To analyze templating in a way as engine-agnostic as possible, we performed a task analysis [16] of all the user activities that can benefit from all kinds of templating support implemented by wiki engines. The results of our task analysis are summarized in Table 1. In the table we stick to some well-established terms [16]: a *goal* is a state of the system a user wishes to achieve, an internal task (or simply a *task*) is a sequence of one or more activities the user thinks are required to achieve a goal.

We will not bother the reader with the hierarchical task analysis of each task, since the more the tasks are decomposed, the tighter their dependencies become on a specific templating mechanism implementation. Yet we observe that all tasks can be eventually decomposed to actions on devices that are specific of some templating mechanisms, such actions will be presented in Section 3.

Each task of Table 1 is specific of a peculiar wiki user role (that notwithstanding the fact that the same user can play different roles at different times). The roles we have considered are the following:

visitor the most common user role, i.e. a user browsing the wiki simply to view pages authored by others. All template-related actions are hidden to her, so she is the actor of none of the tasks shown in Table 1.

Yet visitors are relevant to our discussion, as users in other roles act *for* them, in particular to improve their user experience by uniforming the organization of (parts of) pages;

editor a user that edits the source text of wiki pages either to create new pages from scratch or to change pre-existing pages;

tailor users that are, possibly temporarily, responsible of some wiki section (i.e. page sets), for example of all the pages pertaining to a given subject/category or which share other characteristics such as being co-located in the page namespace. Note that tailors are not the

only authors of these parts of the wiki, but they are simply in charge of the customization of the contained pages [11, 12], since otherwise the wiki workflow would be pointless.

Sometimes tailors have access to privileged technical features that are not available to other users. In such settings, according to folklore, users gain tailor (or "admin", as it is called in some wiki communities) privileges on the basis of reputation.

Task highlights Task 1 and 2 (as we will see they often come in pairs) are both related to the common need of providing representations, in the form of several wiki pages, of objects of the universe that conceptually belong to the same class. For instance, a set of courses offered by computer science departments might need to be described by similar wiki pages, the same can be required for publishing periodically on the wiki reports of some recurring event (software releases, work meetings, football matches, ...), or to "poll" users on a given subject.

Tasks 1 and 2 map to complementary activities pursued by a tailor and an editor. The former wants to foster the creation of several pages describing objects of the same class. Her strategy for achieving so, according to task 2, is preparing a generic page (*boilerplate page*), which represents a generic instance of the class of interest. The generality is expressed as page incompleteness: to obtain from the boilerplate an actual page (i.e. a page describing a class instance), the set of "holes" occurring in the page should be filled with instance-specific data. This strategy is the wiki workflow equivalent of the PROTOTYPE object-oriented design pattern [3]: an instance of a desired class is obtained by first copying an available prototype of the same class, and then filling the missing information.

The issue of how to avoid that future edits on the instance page do not change "fixed" parts of the generic page, therefore breaking the class-instance relationship, is not trivial to achieve in some templating models (see Section 3). Task 1 is the point of view of an editor on the strategy of the tailor of task 2: to create an instance page of some class she chooses the corresponding prototypical page at page creation time (as it happens with MoinMoin templates) and fills its holes.

Task 3 is similar, but acts on page pairs which are not related by a prototype-instance relationships: the editor wants to create a page which is only similar to another and, as it frequent happens, she starts doing so by copying and pasting the markup of the similar page. Obviously, in order to fulfil this task we require no templating mechanism device at all.

A different goal is that of enforcing a common content organization on several pages, in the hope of inducing a common look and feel and smoothing user experiences while browsing pages that share the same organization. Several examples of this need can be found in WIKIPEDIA, where summary/navigational boxes are made available by tailors to be used to describe conceptually similar parts of related pages. To name just a few of them: the software information box we mentioned earlier in this section, the box used to summarize the information of a discography album[3] and to navigate chronologically among several albums, the taxonomy box[4] associated to each animal page, and countless other.

Tasks 5 and 6 correspond respectively to tailor activities needed to setup a piece of reusable content structure, and to perform a batch change to all pages using it. Task 4 is the editor activity required to deal with a reusable content structure: use it in a page (by adding its invocation in the markup, possibly passing it parameters) or get rid of its usage (by removing the corresponding markup, possibly inlining template parts by hand).

Task 7 is related to a broader meaning of templating, which considers the informative content—or the semantics, in the context of semantic wikis—of a page as a template to be applied to (the informative content of) other pages. Exotic as it may appears, scenarios in which this kind of templating can be useful are not infrequent. For example, in WIKIPEDIA several conceptual sets of items do exist and are represented as several lists sorted with varying criteria. To mention a paradigmatic case: from the Lists_of_countries[5] (note the plural) page, several `list of`

3. `http://en.wikipedia.org/wiki/Template:Infobox_album`
4. `http://en.wikipedia.org/wiki/Template:Taxobox`
5. `http://en.wikipedia.org/wiki/Lists_of_countries`

`countries by ...` pages are linked and are supposed to contain all the same countries, yet sorted differently. A way to see such an organization is to consider the List_of_countries[6] (singular) page as an informative content template page, and the other lists as different presentation of the same content.

3 Templating Models

The fact that wikis have increasingly been rising to complex content management systems (CMSs), in different domains and for different uses, should have led implementors to work hardly on templating functionalities. While motivations and user needs behind templating have been discussed in the previous section, a question is still unanswered: "how is wiki templating actually deployed today?". We address it in this section.

A first objective of our research was to feel the pulse of the wiki clones with regard to that issue. Some very interesting surprises came out. First of all, we discovered that a relatively small number of clones actually support content templating (as we will detail later on). In most cases clones use the term "templating" in the presentational template interpretation to characterize customizable and interchangeable layouts and skins, more than the automatic generation of new pages based on existing ones. The good news is that content templating is supported by the most widespread wikis, with some slight differences and peculiarities.

Although different in the implementative details, all the templating model we are aware of can be framed into a single architecture, depicted in Figure 1. In that and subsequents figures, straight arrows represent wiki actions, with the distinction of *external actions* (which are initiated by a user and have bold labels) and *internal actions*; dashed arrows represent generic relationship among wiki entities; the big black arrow represents the mapping of conceptual actions to their actual implementations.

Two kind of pages are involved in template processing:

template (e.g. `FooTemplate` in Figure 1): the master page on which other pages can be based;

instance (e.g. `FooishPage`): the result page obtained by applying a master page to a target one.

The core of templating is the APPLYTO action, which consists of actually generating an instance page from a template (i.e. *applying* a template *to* a target page). Different implementations of that action characterize different templating models.

Before discussing these models, it is worth remarking that templates are normal wiki pages, which can be accessed and edited as any other page of the wiki site. Unlike "real" CMSs, where templates are often special resources, wikis consider templates and instances at the same level, and allow users to manage both kinds in (almost) the same way.

As a consequence, the VIEW action can be applied to both template and instance pages: it produces a rendered page, where content is blended with the overall layout of a wiki site, enriched with user interface elements and hyperlinks. Similarly, a SAVE action can be performed to create new versions of templates and instances. The EDIT action presents some differences: it is a common wiki EDIT for the templates, but can be limited to some extent when applied to the instances, as we will discuss in a while.

Narrowing the analysis to the APPLYTO action, we identified two different templating models, respectively called *functional templating*, in which templates are invoked by name and are possibly passed parameters, and *creational templating*, in which templates are simply copied as new pages at the beginning of instance revision histories.

3.1 Functional Templates

The most common example of functional templating is implemented in MediaWiki [21] and heavily used by WIKIPEDIA users. A functional template is a page including a set of named placeholders (or "holes") which will be substituted for actual values passed as formal parameters, when

6. `http://en.wikipedia.org/wiki/List_of_countries`

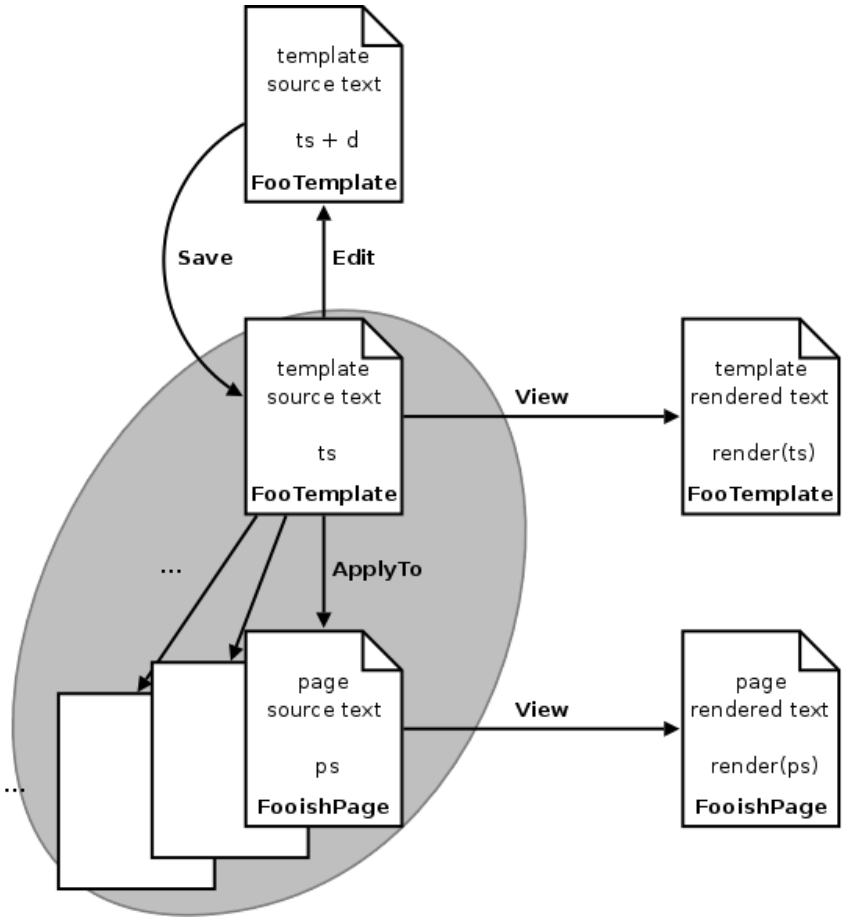

Figure 1. Template relationships with wiki (common and template-specific) wiki actions in a generic templating model. The actual implementation of the ApplyTo action characterizes different templating models.

invoking the template for a specific content. For instance, most WIKIPEDIA information boxes which provide structured information about countries, sports, animals, and plants are created with such technique.

A functional template is applied invoking it by name passing actual parameters. Both invocation and parameter passing are achieved with special purpose syntax in the wiki markup. Figure 2 summarizes this templating model.

The first feature of such templating is a strong and permanent connection between a template and all the instances derived from it: whenever an instance page is displayed, in fact, the template is expanded on-the-fly and the result is in turn passed to the rendering engine of the wiki clone. There is no way to have individual instances differ from the original template other than in filling its named holes.

A second relevant aspect is about the source code of templates and instances. They are quite different: a template contains instructions and structures to organize the final content, while an instance contains the invocation of template by name and parameter passing.

As a consequence, the user experiences while editing templates or instances are sensibly different. It involves dealing with placeholders and (HTML or wiki-syntax based) structures for templates, and with parameters modifications for instances. In the instances, users can simply simply add, remove, or change parameters (which can be arbitrarily complex though) and cannot

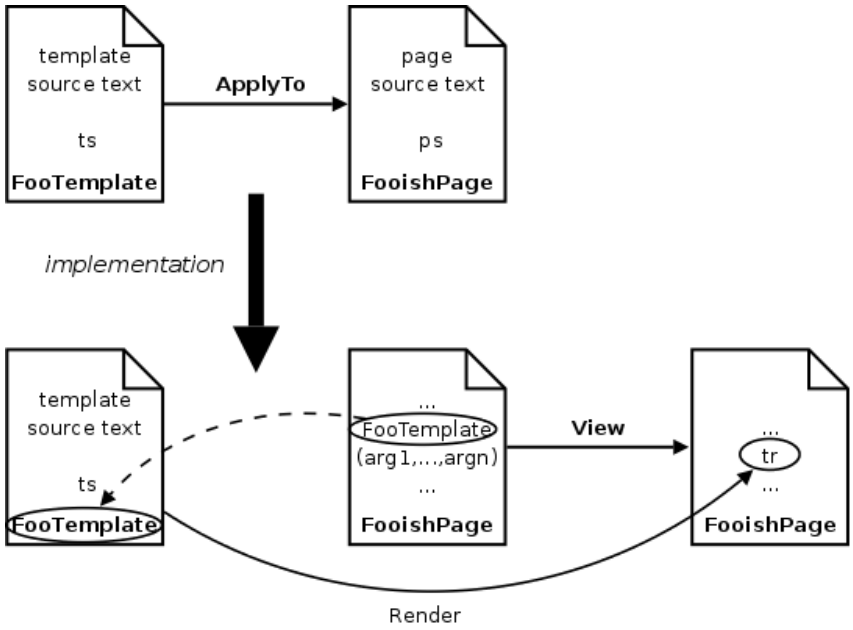

Figure 2. Functional templating: ApplyTo action implementation.

modify the overall structure of the template content, unless they are willing to directly modify the source of the template. That however is a different action to be performed on a different page, possibly requiring different access permissions as it would affect several other pages.

Another related editing problem in instances is the *non-linearity of the markup* with respect to its rendered form. Wiki editors can usually edit source text in wiki engines they do not know by spatial analogy: the desired editing point can be found comparing the rendered text of a page with its companion source text, possibly complementing this with some plain text search. This cognitive process is possible only if the markup is *linear* with respect to the rendered text, i.e. only if a particular change (addition, modification, removal) to the source text maps to a similar change in its rendered form. Functional templates violate this property: if the user try to find its editing point looking at some rendered text that comes from the template, she has no hope to find the corresponding source text in the markup of the instance page.

With respect to the tasks discussed in Section 2 functional templating scores pretty well, as on top of its devices tasks 2, 4, 5, and 6 can be implemented straightforwardly. However, task 1 requires additional syntactic and conceptual knowledge, and produces as a result pages potentially troublesome, due to markup non-linearity, for the average editor.

This brief analysis helps us in highlighting the most relevant pro/con aspects of functional templating. First of all, evident benefits derive from the strong connection between templates and instances: template matching is automatically enforced since pages are actually generated on-the-fly. Deviation from the original template are prevented and pages (or better, template fragments) are automatically and permanently uniform. Moreover changes on a template are automatically spread all over the instances so that managing set of pages with the same template is simple, fast and reliable.

The creation and editing of page instances is another important advantage: authors do not need to master complex structures or deal with the overall organization of page content, but they simply have to provide a value for each parameter. Functional templating is then very useful for input of structured data, like table records.

On the other hand, the fact that the source content of an instance does not correspond to the actual template content arises some issues, in terms of adherence to the wiki philosophy. One

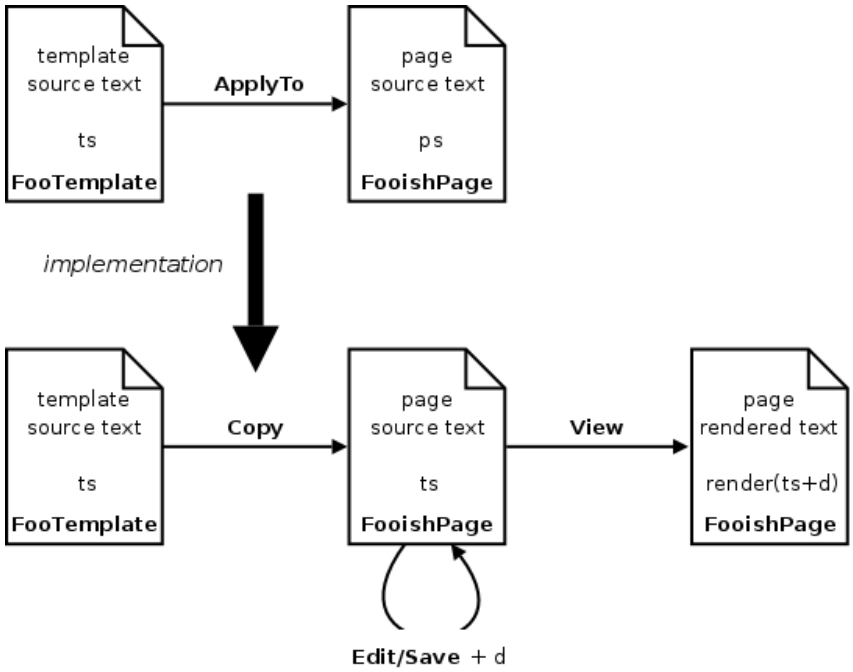

Figure 3. Creational templating: ApplyTo action implementation.

of the most appreciated and comfortable features of a wiki is the full control over the content that is provided to authors, despite the simplicity of the syntax. Wiki users are not required to master complex languages and protocols, yet have fine-grained editing control over their pages. To some extent, such control is lost in the functional model since users pass their parameters to a black box which is in charge of re-organizing their presentation.

Users might be also interested in slight modifications to their pages in places other than the named holes, even if they are generated from a template. Those changes cannot be applied by strictly adopting a functional approach, since the modification on a template would be propagated over all the derived pages.

3.2 Creational Templates

The term "creational template" introduced in this paper indicates those pages used as the starting point for the creation of new ones, with the same structure and initial content. MoinMoin [13] first introduced such templating model, which is now supported by other wiki clones: basically it consists of simply copying the whole source code of a page into a new one, and editing the new page as usual. Creational templates are widely used to create (initially) uniform pages in a wiki site or to quickly generate new content from preexisting content. Figure 3 summarizes this model.

Creational templates are orthogonal to functional templates. First of all, with creational templating, templates and instances are only *weakly* connected: once a page has been derived from a template, it lives as an independent entity within the wiki. It can be modified and twisted, up to become something completely different from the original source. Similarly, two pages derived from the same template can end up having nothing in common. There is no way (although there would be many reasons) to connect a page and a template for the whole life-cycle of the content.

The second point is the fact that the source code of a template and its instances use the same syntax, and that such syntax can be completely templating-agnostic. Modifying an instance means modifying directly the (HTML or wiki-syntax based) structures of the page. The content of an edited instance page is, in fact, directly passed to the rendering engine which transforms it

into the final wiki page; no access to the template page is needed at VIEW-time. Thus, the EDIT action is template-agnostic both for instances and templates.

Editing a template, however, does not imply changes in the instances derived in the past from that template, as it happens in the functional model. Of course such changes do affect instances that will be created in the future.

Regarding the task analysis of Section 2, we observe that tasks 1, 2, 3 are straightforwardly implemented on top of creational templating devices. Task 5 is possible in theory, but it has to be built on top of an input mechanism that permits to choose a creational template while editing *part* of a page. To our best knowledge none of the wiki clones supporting creational templating implements this.[7]

As expected, the orthogonality with the functional approach is mirrored by pros and cons of creational templating. Creational templating does not provide any mechanism to enforce template-matching on future revisions of a page and so it makes possible (and very common) that radical changes happen between the initial instance and subsequent versions. Since the connection between a template and its instances practically disappears after the first COPY action, these templates do not guarantee uniformity among pages. For the same reason, users cannot modify with a single action all the pages derived from a template (i.e. task 6 is not mappable to actions on creational templating devices), and the management of set of pages still requires manual and error-prone interventions.

On the other hand, the fact that the output of the APPLYTO action is the final content (not yet rendered into the layout and formatting of the wiki site, but already organized in tables, lists, ...) is a point of strength for the creational approach. Users, in fact, have fine-grained control over the pages and they can easily customize template instances, saving the straightforwardness and speed of the wikis. In other words, markup linearity with respect to the rendered text is not hindered by creational templating.

3.3 Templates in State-of-the-Art Clones

It is quite evident that our analysis has not ended up with the proclamation of a winner. On the contrary, both functional and creational approaches proved to have benefits and drawbacks, according to the needs (and preferences) of the users. It is no accident that they are differently supported by wiki clones, and that they both are widely used in the community.

In order to evaluate the actual support for wiki content templating, we have studied the TopTen list reported in [1] (including current engines and pending nominations). Although 25 names over the huge amount of existing clones is a very small percentage, such a list should give us a clear picture of the current state-of-the-art clones. Table 2 reports our results.

A first fact is the different support between creational and functional templates: more than a third of the clones adopt a creational approach, while only MediaWiki and few others adopt a functional one. Actually, full-fledged functional templating is supported only by MediaWiki (and Wikia which is MediaWiki-based). We considered in the same class also other wikis, such as WackoWiki, TikiWiki, and OddMuse, which supports functional templating without parameter passing (e.g. as some form of page inclusion). This choice is justified by the observation that pros/cons of functional templating is only marginally affected by parameter passing.

Similarly, XWiki allows designers to define patterns of structured information (stored as a different page, and called by an include-function) which can be re-used in any page. Pages containing such information can be edited and re-edited as simple HTML forms, and will be rendered as the original template. Even if the word "template" is not explicitly used by XWiki, and users do not access the source/text code of the invocation, such feature can undoubtedly be classified as a functional template.

---

7. However, a workflow supporting such *local creational templating* is easily conceivable. Consider for example a wiki engine with support for section (i.e. heading-delimited page part) editing. When the user clicks on the edit link for an empty section, a creational template selection box can be presented to her. Then, the selected template can be used to fill only the section being edited instead of the whole page as it happens in other creational template capable engines. From there on, the templating model is identical to that presented in this section.

Table 2. Templating model support in state-of-the-art wiki engines.

Engine	*Creational*	*Functional*	*URL*
AtlassianConfluence	✓	✗	`http://www.atlassian.com/software/confluence/`
DidiWiki	✗	✗	`http://www.didiwiki.org/`
DokuWiki	✓	✗	`http://wiki.splitbrain.org/wiki:dokuwiki`
EclipseWiki	✗	✗	`http://eclipsewiki.sourceforge.net/index.html`
ErfurtWiki	✗	✗	`http://erfurtwiki.sourceforge.net/`
InstikiWiki	✗	✗	`http://instiki.org/show/HomePage`
JSPWiki	✗	✗	`http://jspwiki.org/`
KwikiKwiki	✗	✗	`http://kwiki.org/`
MediaWiki	✓	✓	`http://www.mediawiki.org/`
MoinMoin	✓	✗	`http://moinmoin.wikiwikiweb.de/`
OddMuseWiki	✗	✓	`http://www.oddmuse.org/`
PerSpective	✓	✗	`http://www.high-beyond.com/`
PhpWiki	✗	✗	`http://phpwiki.sourceforge.net/`
PmWiki	✗	✗	`http://www.pmwiki.org/`
TeleparkWiki	✗	✗	`http://www.telepark.com/`
TikiWiki	✓	✓	`http://tikiwiki.org/`
TWiki	✓	✗	`http://twiki.org/`
UseModWiki	✗	✗	`http://www.usemod.com/cgi-bin/wiki.pl`
VeryQuickWiki	✓	✗	`http://www.vqwiki.org/`
VimKi	✗	✗	`http://www.dausha.net/index.php/Technical/VimKi`
WackoWiki	✗	✓	`http://wackowiki.com/`
WakkaWiki	✓	✗	`http://wikkawiki.org/WakkaWiki`
Wikia	✓	✓	`http://www.wikia.com/`
XwikiWiki	✗	✗	`http://xwikiwiki.xwiki.com/`
ZwiKi	✗	✗	`http://zwiki.org/`

Also the class of creational templates includes some wikis which do not provide such templates in their original form (as MoinMoin does), but implement some differences. For instance DokuWiki provides different namespaces to cluster set of pages sharing a common skeleton. When a page is created, DokuWiki looks up whether there is a template for that namespace and copies it into the edit field for the new page. However, such template file cannot be edited directly through a web interface, but an administrator has to access and change a `.txt` file saved on the server. TWiki implements a creational approach natively for some pages (for instance, the users profiles) and allows users to simulate it for other pages by customizing the installation.

Creational templates are very often supported by wikis which integrate a WYSIWYG editor: for instance Confluence, TikiWiki or PerSpective allow users to select a template, to create a new page based on that template, and to further change its content, without accessing the source code. Even in those cases, however, the template is only used for the first deployment of that page.

Some wiki clones deserve a special mention. First of all, MediaWiki and TikiWiki: to our knowledge, they are the only two wikis which directly support both functional and creational templates. We also found very interesting the templating support of VeryQuickWiki. VeryQuickWiki supports an APPENDTEMPLATE action: basically users can paste pre-defined text fragments (templates) at the end of the editing area. Templates can be then likened to the creational ones since the text is copied into a page (in case with holes to be filled with actual values) and users can further change it; to some extent, they are functional since they can be applied at any time. A limit of these templates is the fixed position of the included fragments, besides the usual weak connection between instances and templates.

Finally, we also found many lists of requested features (for instance, for ZWiki or JSPWiki)

where both creational (referred to as "a la MoinMoin") or functional ("a la MediaWiki") templates hold a relevant position. Our impression is that there is still a strong willingness to support and improve wiki content templating.

4 Towards Improved Creational Templates

Our answer to the question of whether state-of-the-art wiki templating model can be improved is affirmative. In this section we present a novel templating model, as an attempt to create a merger of the positive aspects of the two templating model presented in the previous section. Our model is not radically new, but rather a conceptual "patch" for creational templating.

Our starting point is maintaining access to the markup of the page as a fundamental feature of wikis. This has led us to base our proposal on top of creational rather than functional templating. The main drawback of creational templates is the lack of "retroactivity", induced by the weak connection between templates and instances: no change which breaks the original template can be prevented or notified. Our solution to this issue consists of integrating a non-invasive mechanism to validate *a posteriori* page content, so as to check whether the prototype-instance relationship has been violated. Our proposal is still a model of the architecture of Figure 1 (being a patch of the creational model) and exploits an instantiation of the light constraint framework [9], which is briefly described in the next section.

4.1 Light Constraints in a Nutshell

In [9] we described a framework for dealing with *light constraints* on the content of wiki pages, observing how *de facto* constraints spontaneously appear in wiki communities to encode best authoring practices or to account for the needs of domain-specific engines. Light constraints can be thought as constraints which can be enforced respecting "The Wiki Way" workflow of viewing and editing pages.

The constraints are expressed in our framework as validator functions that can be associated to pages (in a many-to-one relationship) and used to check whether the constraints have been respected, a step possibly involving additional pages (the *validation context*). Failed validation attempts return sets of located error messages.

Since saving cannot be inhibited when constraints are violated without diminishing authors' freedom, our framework uses *conditional saving* to present to the editor validation errors (if any) as a single additional step before a SAVE action can be completed. The editor can then either ignore the errors and save the page anyway or perform additional editing round-trips to fix them.

Similarly, the VIEW action is changed to *annotated viewing*: visitors viewing a page are shown the usual page content as well as validation information such as validators, contexts, and failed validation outcomes. Thus the internal state of the engine is fully disclosed to visitors and the efforts of WikiGnomes [19] can be driven (by tailors) to fix validation errors.

4.2 Lightly Constrained Templates

We use the term *Lightly Constrained Templates* (LC Templates) to indicate creational templates, empowered by light constraints on template matching. The key observation consists of encoding as light constraint the fact that a page should match the template it has been generated from. Whenever a page is displayed or saved, the system verifies if that template is matched and notifies the validation status to the users. Figure 4 shows this approach.

Each page is associated to a validation function which takes in input the content of that page and a validation context. We defined a validation context as a set of pages required for validation: in this case, it is only the original template which generated the current page. The validator produces two possible outcomes: a "ok" message if the template matches the instance, or a list of localized errors (indicating where mismatches occur) when it does not. According to our general architecture the VIEW action becomes an annotated viewing (the page contains both the actual content and the validation report), while the SAVE action becomes a conditional save (the

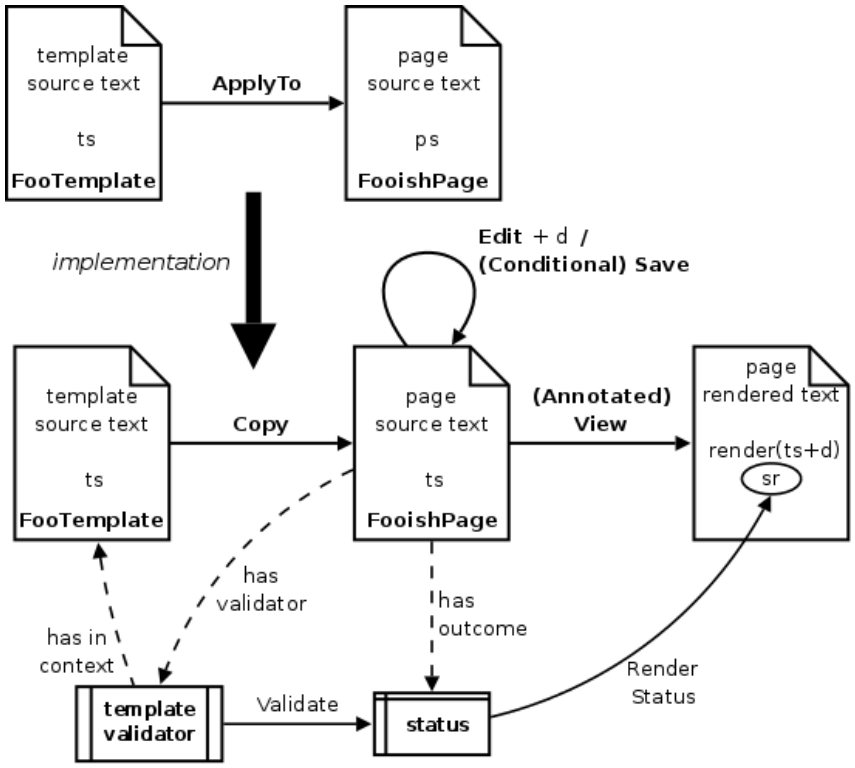

Figure 4. Lightly constrained templating: ApplyTo implementation and overall architecture.

report is displayed before saving, and users can choose to ignore or take it into account, by fixing mismatches).

It is worth observing that such architecture is independent from the granularity and power of the validation process. Different solutions can be implemented: from regular expression matching to search over predefined areas, up to advanced templating on the informative content of a page. In Section 5 we will discuss a possible spectrum of validators. What is relevant here is the fact that such validators improve the templating model without interfering with the open wiki philosophy.

Our model implements a permanent connection between templates and their instances. Such important feature, which plain creational templating was lacking, is obtained by the introduction of light constraints. Although a page is not actually generated from a template on every VIEW (as happens with a functional approach), the validation process guarantees a permanent relation between those pages.

As an additional benefit LC templates enable *template re-targeting*. Since the instance-template relationship is kept in the validation context, and given that the validation context can be changed by tailors, it is possible to change the template applied to a page even after the page's creation time. If the new template still matches the instance page no constraint violation will be reported, otherwise all mismatches will be shown to the tailor and, for those left unfixed, to editors and visitors.

Note also that LC templating allows tailors to modify a template and consequently modify all the derived pages. Our architecture provides for a daemon in charge of notifying changes of a page to all the pages which include it in their validation context. Applied to this domain, it means that a modification on a template is spread all over the wiki. Whenever a page "connected" to that template will be displayed, a notification message will be included in that page. Then users can spontaneously work to fix the mismatch.

There is a subtle but very important difference with the functional approach: while that approach *enforces* a template-matching, at the price of limiting editing freedom, our approach *verifies and notifies* mismatches. Users can ignore that notification, and go on saving their content. At the same time, they have complete freedom and full information on their content. In a sense we are decentralizing batch changes exploiting the primary power of wiki communities: collaboration.

The linearity of markup is another positive aspect of LC templating. Being substantially a patched version of the creational templating, our approach provides a correspondence between the source of a page and its rendered version. Users access the same content, but also receive information about the templates and, if needed, can apply slight changes and customization.

Another interesting property derives from the decoupling of templating validation and rendering. Since the validation is performed outside the wiki (even if a wiki module performs that operation, it remains logically independent), different validators can run (as we discuss later on) without changing the overall architecture of the wiki. Moreover the same validator can be potentially re-used in different wiki clones.

Summarizing, we believe that lightly constrained templates are the best trade-off for wiki content templating in a generic setting. Specific settings, such as strong needs of structured editing, might still consider the adoption of functional templating; we will discuss them in Section 6. The need of templating page parts, as opposed to whole pages, can be easily addressed with LC templates by implementing the local creational templating workflow proposed in Section 3.2.

5 LC-Templates Deployment

Although we are not providing an actual implementation of a wiki engine supporting LC templates, in this section we argument that such feature is straightforward to support by coupling an instantiation of the light constraint framework (as already presented in [9]) and an engine which already supports plain creational templating (that is, the vast majority of template-enabled engines, as discussed in Section 3.3). To support our argument we present a deployment technique for LC templates in such a wiki engine and discuss the relationship among LC templates and semantic wikis.

5.1 Syntactic Template Matching

With the light constraints machinery already in place, the task at hand is brought to implement a single validator whose specification is as follows.[8] The validator can be associated to instance pages and relies on a singleton validation context containing the template that has to match the instance. In fact it is recommended that in a LC deployment the association is added automatically by the engine when a page (or a section, in case of local templating) is generated creationally choosing from a list of the available templates. Of course tailors are free in the future to remove the association or to change it, re-targeting the instance page.

The validator will be invoked on an instance page at (conditional) SAVE-time, or alternatively in a batch fashion (if a batch validator is available) or, as a last resort, at VIEW-time, depending on the details of the light constraint machinery implementation. Its output, the *validation outcome*, is either an assessment that the instance matches its template or a list of error messages containing the string `"template mismatch"` and located where the mismatches happen.

Fulfilling the above specification implementing the corresponding function is straightforward. We only require that a special markup syntax, a marker, is available for denoting where holes occur in the source text of a template page. Such a template can be converted to a non deterministic regular expression by replacing hole markers with blocks of "any character sequence" (that would be `.*` in popular regular expression syntaxes). The obtained regular expression can be applied (with the appropriate flags such as multi-line support and with appropriate string delimiters) to instance pages to check for template matching.

8. We recall that a *validator* is usually a simple function written in the implementation language of the wiki engine and loaded as an external plugin.

If the regular expression matches an instance it is safe to return a validity assessment. If it does not, a naive assumption is that the page is not an instance of the template. In order to be less naive, a few additional steps in the generation of the regular expression can be be taken. For example the template page can be parsed by the legacy wiki engine page parser and "any character sequence" blocks (or more specific blocks like "any blank character") can be inserted where appropriate.

Localization of mismatches can be easily implemented as well. Indeed atomic `.*`-delimited blocks of the generated regular expressions can be wrapped in optional groups `(...)?` (a feature supported by all major regular expression engines) and a posteriori checking for empty groups can be used to identify where an expected markup structure is missing. Yet easier, PCRE (the most widespread regular expression engine) supports *callouts* [6], external functions that can be attached to regular expression markers and are invoked when the engine passes them. Callouts are passed the current position of the underlying automata on the input string and can store that information for further processing.

5.2 Semantic Template Matching

Semantic wikis engines are no longer as exotic as they appeared to be in the recent past. Both the wiki literature and the milieu of real-life implementations is full of examples: Rhizome [18] (in which users can express RDF statements in a simpler plain text language called ZML), SweetWiki [14] (integrating an extended WYSIWYG editor with support for semantic annotations), or SemperWiki [14] and countless others, including a semantic extension of WIKIPEDIA [20] that expresses semantics as typed links between pages.

The question of how LC templating fits with semantic engines is proper. We believe that not only it does fit well, but that it also spins-off an interesting synergy. We already discussed in [9] how validators can benefit of the availability of page semantics to perform validity checks which are more intimately related to the informative content of a page than to its presentational structure alone.

This aspect, reconsidered in the LC templating setting, leads to a scenario in which the content structure of a template page is its semantic, ...what is the "meaning" of a content template if not the very same set of content structures which builds it up?. Similar structural meanings have already been studied both in the document engineering community [7] and for wiki-specific needs [8, 15]. Once such kind of meanings are available and can be extracted from instance pages, the template matching validator would be more reliable, being capable of abstracting over string matching and syntactic ambiguities.

An added benefit of LC templating coupled with semantic annotations on pages would be the ability to implement devices that support task 7 of Table 1, which has been left unsupported by the templating models discussed so far. Indeed we can push forward the above-mentioned idea and consider the (non-structural) meaning of a page, i.e., the set of its semantic annotations, as a novel templating content. We call it *meaning templating* and we like to place it in the far extreme of the following templating spectrum:

$$\begin{array}{c}\text{presentational}\\ \text{templating}\end{array} \longrightarrow \begin{array}{c}\text{content}\\ \text{templating}\end{array} \longrightarrow \begin{array}{c}\text{meaning}\\ \text{templating}\end{array}$$

where being more on the right corresponds to an higher degree of machine understanding of templates. With meaning templating the informative content of a page is encoded as its semantic annotations and semantic verification among the informative contents of two pages can be performed. This can be exploited for example to ensure (to fulfill the WIKIPEDIA needs of Section 2) that two pages contain the same set of items, notwithstanding their order of appearance.

6 Conclusions and Related Work

In this paper we reviewed the models of content templating in state-of-the-art wiki engines: functional templating and creational templating. The former relies on special markup referring to

other pages by name and on parameter passing to fill named holes. The latter is similar to markup copy & paste but happens automatically at the beginning of a page life cycle. As both models have drawbacks we proposed our own—lightly constrained templating—which builds on top of the creational model adding support for light constraints [9]. We showed in which respects our patched model is an improvement over its competitors.

To our best knowledge, the wiki literature is mostly devoid of works on wiki-specific templating needs, with the notable exception of Wiki-Templates [5]. In their work, Haake, Lukosch, and Schümmer propose their own templating mechanism, specially crafted to support editing of structured wiki pages. Our analysis here is more general and all the models we discussed are applicable both to structured and structureless wiki pages. An additional contribution of [5] is the identification of a set of requirements for supporting editing of structured wiki pages. It is interesting to briefly discuss how our templating model scores with respect to such requirements. On the basis of the following considerations we claim that LC templating, in spite of not having being designed for structured editing, does fulfill most of its needs.

As expected LC templating fails to properly address "R1: structure" (i.e. helping users to organize structured data), since it is structure-agnostic. However, if the pages are structured per se, adopting LC templating does not pose any additional problem: structure can be copied when templates are applied, and structure-specific user interfaces can be used for editing the templates themselves (e.g. the editing of the `edit template` of Wiki-Templates) and their instances (e.g. the complex web forms generated from `edit templates` of Wiki-Templates). Structure and LC templates can even exhibit synergism since structure, if expressed as the semantics of a given page, can be exploited to write validators with a deep understanding of page content (see Section 5.2).

With respect to requirement "R2: readability" (i.e. making easier to access content) we believe LC templates perform better than Wiki-Templates due to markup linearity. In Wiki-Templates a user willing to contribute to a page associated to an `edit template` (the template which has generated the input data form) is likely to experience interaction breakdown [17] either if it comes from a wiki engine other than Wiki-Templates or if the `edit template` has been changed. In a sense we are convinced that `edit templates` may hinder habit formation: they are probably worthwhile if structured editing is a major goal, but not in a more generic setting. LC templating on the contrary ensures markup linearity helping the user in finding the desired editing point in the source text by analogy with the rendered text. A similar argument holds for Wiki-Templates users editing `display templates`.

Requirement "R3: safety regarding unintended changes" (i.e. making users aware of the effects of their edits) is fulfilled by LC templates which can enforce safety by the mean of validation (see Section 5). Finally, requirements "R4: tailoring" (i.e. allowing users to customize their content structures) and "R5: sharing of tailorings" (i.e. allowing users to share their templates) are satisfied for free in LC templating since templates live in the same space of ordinary pages. As such they can be published and customized by tailors without the need of implementing new conceptual wiki actions.

This paper achieved two goals: on the one hand, shedding lights on wiki (content) templating, on the other proposing a general and improved model which is applicable to varying domains, engines, and user needs. In particular, we plan to further investigate the validation mechanisms of Section 5, and to integrate them in the light constraint enabled wiki engine we are currently developing. We also plan to extend our analysis to other wiki clones. Our idea is reviewing all the engines included in the WikiMatrix [2] and proposing a new field named "content templating model", since that feature has been surprisingly neglected so far.

References

[1] c2.com wiki. Top ten wiki engines. `http://c2.com/cgi/wiki?TopTenWikiEngines`.

[2] CosmoMode. WikiMatrix: compare them all. `http://www.wikimatrix.org/`.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1995.

[4] The GNU version of The Collaborative International Dictionary of English. `http://www.ibiblio.org/webster/`, May 2007. v.0.48.

[5] Anja Haake, Stephan Lukosch, and Till Schümmer. Wiki-templates: adding structure support to wikis on demand. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pages 41–51, New York, NY, USA, 2005. ACM Press.

[6] Philip Hazel. PCRE - Perl-compatible regular expressions. `http://www.pcre.org/pcre.txt`.

[7] Angelo Di Iorio, Daniele Gubellini, and Fabio Vitali. Design patterns for descriptive document substructures. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2005.

[8] Angelo Di Iorio and Max Völkel. WIF: Wiki interchage format. `http://www.wikisym.org/wiki/index.php/WSR_3`, 2006.

[9] Angelo Di Iorio and Stefano Zacchiroli. Constrained wiki: an oxymoron? In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 89–98, New York, NY, USA, 2006. ACM Press.

[10] Bo Leuf and Ward Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, pap/cdr edition, April 2001.

[11] Wendy E. Mackay. Patterns of sharing customizable software. In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 209–221, New York, NY, USA, 1990. ACM Press.

[12] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. User-tailorable systems: pressing the issues with buttons. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, New York, NY, USA, 1990. ACM Press.

[13] The MoinMoin Wiki Engine. `http://moinmoin.wikiwikiweb.de/`.

[14] Eyal Oren. Semperwiki: a semantic personal wiki. In *Proc. of 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure*, Galway, Ireland, 2005.

[15] Eyal Oren and Max Völkel. Towards a wiki interchange format (wif). In *Proceedings of the First Workshop on Semantic Wikis*, 2006.

[16] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction: Concepts And Design*. Addison Wesley, 1st edition, April 1994.

[17] Yvonne Rogers. Coordinating computer-mediated work. *Computer Supported Cooperative Work (CSCW)*, 1(4), December 1993.

[18] Adam Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5):87–91, 2005.

[19] WIKIPEDIA. WikiGnomes. `http://en.wikipedia.org/wiki/Wikignomes`.

[20] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, MAY 2006.

[21] WikiMedia. MediaWiki. `http://wikipedia.sourceforge.net/`.

[22] WIKIPEDIA, The Free Encyclopedia. `http://www.wikipedia.org/`.