A Novel Methodology Based on Formal Methods for Analysis and Verification of Wikis

Giuseppe De Ruvo, Antonella Santone
Department of Engineering, University of Sannio, Benevento, Italy
e-mail: {gderuvo@unisannio.it, santone@unisannio.it}

***Abstract*—A wiki is a collaborative Web site whose content can be edited by anyone who has access to it. Wikis are becoming a new work tool in enterprises and are widely spreading everywhere. In fact, they are often used as internal documentation for various in-house systems and applications. Understanding and maintaining the structure of a wiki may be a crucial aspect. As well as software grows, decays and needs refactoring, the organic growth of a wiki inevitably leads to its degradation.**

We propose a novel methodology based on formal methods to analyse and verify the architecture of wikis. Formal verification helps to perform refactoring. Each wiki category, a set of wiki pages, is modelled using the Calculus of Communicating Systems (CCS) process algebra in order to verify specific properties. First experiments conducted on a adequate number of categories of Wikipedia, assess the validity of our methodology revealing new directions for future research.

***Index Terms*—Wiki, Temporal Logic, CCS Process Algebra, Refactoring.**

I. Introduction and Related Work

The first wiki was created in 1995 by Ward Cunningham. In the following years the wiki idea was widely adopted and today there are hundreds of wiki engines available[1] which provide a plethora of features. Thus, as occurs in large software systems understanding and maintaining their structure is not a negligible element to consider. Not only software evolves, the content stored in a wiki evolve as well. When articles grow beyond a certain size, it is better to split them into smaller ones. There are many cases in which modifying the structure of a wiki becomes necessary. On huge wikis the continuous degradation of content structure causes substantial maintenance work. Programs are not yet capable to solve above issues automatically. Whether and how to split a wiki page or a wiki category can only be decided by a human.

The benefit of wikis to develop user and software documentation of Free Libre Open Source Software (FLOSS) projects extends when reusing software modules: it is possible to reuse software documentation too with the same advantages i.e. saving time and effort with reduction of costs. Almost all FLOSS projects do not simply add links to the existing modules' wiki documentation. Sometimes it is proper to add one or two sections of text to better explain the concepts. This especially occurs when dealing with commercial/enterprise software projects which reuse FLOSS modules. In fact, commercial projects do not prefer wikis as a form of documentation. In order to reuse wiki documentation in disparate ways we need an adequate set of methodologies and tools.

We propose a novel methodology based on formal methods for refactoring the architecture of wikis. In particular, we use model checking which is a formal technique for proving the correctness of a system with respect to a desired behavior. This is accomplished by checking whether a structure representing the system (typically a labelled transition system) satisfies a temporal logic formula describing the expected behavior. In this paper we will show how to use model checking to analyse and verify wikis transforming them into Calculus of Communicating Systems (CCS) [1] processes and then checking if they satisfy suitably-defined properties. One of the advantages of this procedure based on formal methods is that once the CCS model is generated, formal verification tools such as [2] can be also used for verifying many properties. Properties are expressed using the selective-μ-calculus logic [3]. We exploited the capabilities of model checking in a previously unexplored area with encouraging results. Our methodology has been tested on three categories of Wikipedia. We chose Wikipedia because it has plenty of content.

As far as we know nobody addressed the analysis and verification of wikis using formal methods and model checking. However, there are some works regarding wiki refactoring focused on either methodological or technological point of view. On the other hand, there are some papers which employed formal methods to face issues in Web Application Engineering.

Rosenfeld et al. [4] proposed an approach to detect quality problems in semantic wikis inspired by the bad smell problems in software engineering. The approach mostly focus on annotations in order to incrementally create a structured ontology, while we work on links to maintain the architecture.

Alluvatti et al. [5] also investigated on the quality and evolution of wikis. They proposed to evaluate the quality of wikis basing on the number of edits and contributors per page. Aversano et al. [6] presented a preliminary work for refactoring wiki content. The method is built upon a software refactoring method which exploits the dominance relations on the analysed software system call graph. Due to its preliminary fashion, the authors did not present significant results to quantify the effectiveness of the approach.

Donini et al. [7] applied model checking techniques to perform automated verification of the UML design of web applications. Haydar et al. [8] presented an approach that use execution traces of a web application to automatically generate a communicating automata model to model checking the

[1]http://www.wikimatrix.org

application against predefined properties. Unlike other works, in our opinion, our approach is valuable also in different contexts, apart from refactoring, in that the CCS translation of wikis can be exploited to develop other types of wiki analyzers or for reusing issues.

The remainder of this paper is organized as follows. First, in the next Section we give basic definitions of wikis, CCS and selective-μ-calculus. Section III deals with our methodology for analysis and verification of wikis. In Section IV we discuss achieved results and how to reach particular goals. Finally, in Section V we conclude and we provide new inputs for further research.

II. Preliminaries

In this section, after introducing the basic definitions of wikis, we present the Calculus of Communicating Systems (CCS) [1], the process algebra we have adopted for analysing and verifying wikis, and selective-μ-calculus.

A. Wikis

A wiki (from the Hawaiian wiki, to hurry, swift, quick) is a collaborative Web site whose content can be edited by anyone who has access to it. Ward Cunningham's "WikiWikiWeb"[2] lets software developers create a library of software patterns.

A wiki is generally divided in categories. Each category may be split into subcategories. Categories contain pages and each page belong to different categories. Pages are divided into sections and each section may be divided into subsections. Each page has links to other pages or categories and have external links (i.e. links to web pages). Since wikis are easy to edit, they changed how we construct knowledge repositories on the Web. Wikis allow groups to form around specific topics and they are a great way for a group of people to coordinate and create content, even though that group counts thousands of people in different places [9]. Here are some typical things we can do on a wiki: create an article on a specific topic; make changes to other people's articles, without requiring their permission; create links between articles; group similar articles together into convenient categories; view the history of an article to see all the changes, who made them, and when; see interesting statistics about the articles e.g. which ones are most popular and/or need updating.

There are disparate types of wikis:

- Wiki Mapia, combines Google Maps with a wiki system supporting over 35 languages.
- WikiTravel, a travel guide and LyricWiki, a listing of lyrics by album.
- Flu Wiki intends to help local public health communities coping with a possible (avian) influenza pandemic, and Ganfyd is an online collaborative medical reference that is edited by medical professionals and invited non-medical experts.
- Diplopedia, billed as the Encyclopedia of the USA Department of State. It houses a unique collection of information pertaining to diplomacy, international relations, and Department of State tradecraft.
- IkeWiki [10], a semantic Wiki developed at Salzburg research for collaborative knowledge engineering. While it has been developed primarily as a tool for ontology engineering, it can be used in a variety of application scenarios.

A recent study pointed out the benefits of using wikis in enterprises for knowledge sharing [11]. Moreover, many FLOSS projects use wikis for technical and user documentation, e.g. Eclipse and its Eclipsepedia.

In Section IV we will analyse an adequate set of categories of Wikipedia.

B. Process algebra: CCS

Historically, process algebras have developed as formal descriptions of complex computer systems, and in particular of those involving communicating, concurrently executing components. There are many examples of process algebras. In this paper we use Milner's Calculus of Communicating Systems (CCS) [1]. Readers unfamiliar with CCS are referred to [1] for further details. The syntax of *processes* is the following: $p ::= nil \mid x \mid \alpha.p \mid p+p \mid p|p \mid p\backslash L \mid p[f]$, where α ranges over a finite set of actions $\mathcal{A} = \{\tau, a, \overline{a}, b, \overline{b}, ...\}$. The action $\tau \in \mathcal{A}$ is called the *internal action*. The set of *visible actions*, $\mathcal{V}$, ranged over by $l, l' \ldots$, is defined as $\mathcal{A} - \{\tau\}$. Each action $l \in \mathcal{V}$ (resp. $\overline{l} \in \mathcal{V}$) has a *complementary action* $\overline{l}$ (resp. l). The restriction set L, in the processes of the form $p\backslash L$, is a set of actions such that $L \subseteq \mathcal{V}$. The relabeling function f, in processes of the form $p[f]$, is a total function, $f : \mathcal{A} \to \mathcal{A}$, such that the constraint $f(\tau) = \tau$ is respected. The constant x ranges over a set of constant names: each constant x is defined by a constant definition $x \overset{\text{def}}{=} p$, where p is called the *body* of x. We denote the set of processes by $\mathcal{P}$. The standard *operational semantics* [1] is given by a relation $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$. $\longrightarrow$ is the least relation defined by the rules in Table I.

We now informally explain the semantics for CCS by induction over the structure of processes. nil represents a process that can do nothing. There is no rule for nil since it cannot evolve. The process $\alpha.p$ can perform the action α and thereby become the process p (rule **Act**). The process $p+q$ is a process that non-deterministically behaves either as p or as q (rule **Sum** and symmetric). The operator $\mid$ expresses parallel composition. p and q may act independently: if the process p can perform α and become p', then $p|q$ can perform α and become $p'|q$, and similarly for q (rule **Par** and symmetric). Furthermore, p and q may also together engage in a communication whenever they are able to perform complementary actions. That is, if p can perform a visible action l and become p', and q can perform $\overline{l}$ and become q', then $p|q$ can perform τ and become $p'|q'$ (rule **Com**). If L is a set of visible actions, $p\backslash L$ is a process that behaves as p except that it cannot perform any of the actions (as well as the corresponding complementary actions) lying in L externally, although each pair of these complementary actions can be performed for communication internally (rule **Res**). The operator $[f]$ expresses the relabeling of actions. If p can perform α and become p', then $p[f]$ can perform $f(\alpha)$ and become $p'[f]$ (rule **Rel**). The behavior of the process x ($x \overset{\text{def}}{=} p$) is that of its definition p (rule **Con**).

[2]http://c2.com/cgi-bin/wiki?WikiWikiWeb

TABLE I: Standard operational semantics of CCS

Act	$\dfrac{}{\alpha.p \xrightarrow{\alpha} p}$	**Sum**	$\dfrac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'}$	**Rel**	$\dfrac{p \xrightarrow{\alpha} p'}{p[f] \xrightarrow{f(\alpha)} p'[f]}$	**Par**	$\dfrac{p \xrightarrow{\alpha} p'}{p\vert q \xrightarrow{\alpha} p'\vert q}$
Con	$\dfrac{p \xrightarrow{\alpha} p'}{x \xrightarrow{\alpha} p'}\ x \overset{\text{def}}{=} p$	**Com**	$\dfrac{p \xrightarrow{l} p',\ q \xrightarrow{\bar{l}} q'}{p\vert q \xrightarrow{\tau} p'\vert q'}$	**Res**	$\dfrac{p \xrightarrow{\alpha} p'}{p\backslash L \xrightarrow{\alpha} p'\backslash L}\ \alpha, \bar{\alpha} \notin L$		

C. *Selective-μ-calculus*

The selective-μ-calculus, introduced by the author and others in [3], is a branching temporal logic to express behavioral properties of systems. It is equi-expressive to μ-calculus [12], but it differs from it in the definition of the modal operators.

Given a set $\mathcal{A}$ of actions and a set *Var* of variables, the selective-μ-calculus logic is the set of formulae given by the following inductive definition:

- `tt` and `ff` are selective-μ-calculus formulae;
- Y, for all $Y \in$ *Var*, is a selective-μ-calculus formula;
- if φ_1 and φ_2 are selective-μ-calculus formulae then $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ are selective-μ-calculus formulae;
- if φ is a selective-μ-calculus formula then $\langle K \rangle_R\, \varphi$ and $[K]_R\, \varphi$ are selective-μ-calculus formulae, where $K, R \subseteq \mathcal{A}$;
- if φ is a selective-μ-calculus formula then $\mu X.\varphi$ and $\nu X.\varphi$ are selective-μ-calculus formulae, where $X \in$ *Var*.

The satisfaction of a formula φ by a state s of a transition system, written $s \models \varphi$, is defined as follows:

each state satisfies `tt` and no state satisfies `ff`; a state satisfies $\varphi_1 \vee \varphi_2$ ($\varphi_1 \wedge \varphi_2$) if it satisfies φ_1 or (and) φ_2. $[K]_R\, \varphi$ and $\langle K \rangle_R\, \varphi$ are the selective modal operators:

$[K]_R\, \varphi$	is satisfied by a state which, for every performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K, evolves to a state obeying φ.
$\langle K \rangle_R\, \varphi$	is satisfied by a state which can evolve to a state obeying φ by performing a sequence of actions not belonging to $R \cup K$, followed by an action in K.

The selective modal operators $\langle K \rangle_R\, \varphi$ and $[K]_R\, \varphi$ substitute the standard modal operators $\langle K \rangle\, \varphi$ and $[K]\, \varphi$. The basic characteristic of the selective-μ-calculus is that each formula allows us to immediately point out the parts of the transition system that do not alter the truth value of the formula itself. More precisely, *the only actions relevant for checking a formula are the ones explicitly mentioned by the selective modal operators used in the formula itself.* Thus, the result of checking the formula is independent from all other actions. This information can be exploited to obtain reduced transition systems on which the formula can be equivalently checked (see, for example, [13]). The precise definition of the satisfaction of a closed formula φ by a state of a transition system can be found in [3].

III. The Methodology

In this section we present our methodology to analyse and verify wikis. The main distinctive feature of this methodology is the use of formal methods (to the Authors' knowledge, never used before). In practice, from the wikis we derive CCS processes, which are successively used to perform model checking. The goal of our methodology is to aid software engineers and domain experts during maintenance tasks. Our approach requires a four-step process. The four steps are:

1) CCS model creation
2) Formal verification process (model checking)
3) Synthesis generation
4) Maintenance

In the following subsections the four steps are discussed in detail.

A. *CCS model creation*

We use as internal representation the CCS language. Thus, CCS specifications are generated from wikis. This is obtained by defining a wiki-to-CCS transform operator $\mathcal{T}$. The function $\mathcal{T}$ directly applies to wikis and translates them into CCS process specifications. The aim of $\mathcal{T}$ is to avoid the construction of "expensive" data structures such as Program Dependence Graphs while retaining their accuracy for formal verification. The exploitation of process algebra-based formal methods for analysing and verifying wikis may seem overkill, but we think that it is a viable solution due to the availability of formal verification tools. Furthermore, it opens a wide field of opportunities to researchers for wikis understanding and documentation purposes. The function $\mathcal{T}$ is defined for each object of wikis. All wiki objects have been translated in CCS.

First of all, when analysing a category C, we have to distinguish between different pages belonging to C. A page can belong to the same category C, to a different category or can be an external web page. Thus, in our model we divide pages into:

- INT_inP: the internal page P belongs to the analysed category C;
- INT_outP: the internal page P belongs to a different category;
- $EXTP$: the page P is an external web page.

Wiki category

The wiki category C containing k pages is translated into the following CCS process $CATEGORY$:

$$\mathcal{T}(C) \;=\; CATEGORY \overset{\text{def}}{=} INT\_inP_1 + \cdots + INT\_inP_k$$

The CCS process $CATEGORY$ represents the possibility of reaching one of the k internal pages.

Wiki page

The i-th internal wiki page P_i (either INT_inP_i or

INT_outP_i) with n sections $SP_{i_1}, \dots, SP_{i_n}$ is translated into the following CCS process P_i:

$$\mathcal{T}(P_i) \;=\; P_i \overset{\text{def}}{=} inP_i.(SP_{i_1} + \cdots + SP_{i_n})$$

The CCS process P_i performs the action inP_i, meaning that page P_i has been reached, followed by the summation $SP_{i_1} + \cdots + SP_{i_n}$. This summation means that each section of the page can be non-deterministically reached. Each constant process SP_{i_k}, with $k \in [1..n]$, represents the CCS translation of the i_k-th section of the page P_i.

Wiki section

Let SP_{i_j} be the j-th wiki section of page P_i with k_1 internal pages belonging to the analysed category; k_2 internal pages belonging to a different category and k_3 web page links. SP_{i_j} is translated into the following CCS process SP_{i_j}:

$$\begin{aligned}\mathcal{T}(SP_{i_j}) \;=\; SP_{i_j} \;\overset{\text{def}}{=}\; & INT\_inP_1 + \cdots + INT\_inP_{k_1} + \\ & INT\_outP_1 + \cdots + INT\_outPk_2 + \\ & EXTP1_1 + \cdots + EXTP_{k_3}\end{aligned}$$

The CCS process SP_{i_j} represents the possibility of reaching one of the $k_1 + k_2 + k_3$ pages.

To evaluate the effectiveness of our transformation function $\mathcal{T}$, we have to consider both complexity and automation. From the complexity point of view, it is easy to show that the complexity of the extraction of the CCS model from wikis is linear in the length of the pages. From the automation point of view, it is worth noting that the function $\mathcal{T}$ is syntactically defined and it is completely automatic, thus it does not require user intervention and manual efforts.

B. *Formal verification process (model checking)*

In our approach, we use model checking to verify wikis. Once we have the CCS processes of wikis, we can use selective-μ-calculus logic to specify desired properties. We consider the following properties.

Island Property: $[K]_\emptyset\, [inX]_\emptyset\, \mathtt{ff}$

where $K = \{inP_i \mid i \text{ is a page of the analysed category}\}$

The property states that page X is a island, i.e. page X (action inX) cannot be reached starting from any page (set of actions K) of the analysed category.

Loop Property: $\nu Z.\, \langle inX \rangle_\emptyset\, Z$

The property states that page X belongs to a loop, i.e. page X (action inX) belongs to an infinite path starting from the initial state.

Strong Core Property: $[inP]_\emptyset\, \langle inX \rangle_{\mathcal{A}}\, \mathtt{tt}$

The property states that page X is a strong core page for page P, i.e. page X (action inX) can immediately be reached from page P (action inP).

Weak Core Property: $[inP]_\emptyset\, \langle inX \rangle_\emptyset\, \mathtt{tt}$

The property states that page X is a weak core page for page P, i.e. page X (action inX) can be reached from page P (action inP), even after having crossed other pages.

Other properties

Unlike other bespoke wiki analysis techniques, methods and tools, we can easily add new properties and proceed with new analyses thanks to the power of model checking. In fact, it is sufficient to express the new meaningful properties of interest in the selective-μ-calculus logic. For example, we can add a "dead-link/missing page" property - i.e. articles which do not exist with that exact name, exploiting the popular "safety" property. Although property "missing page" may be easily verified in other ways, we argue that since we have a formal model it is smarter to integrate everything in a unique framework and even pursue other goals.

C. *Synthesis generation*

The IIT Delhi Concurrency Workbench [2] is one of the most popular environments for verifying concurrent systems which supports several different specification languages, among which CCS. The model checker of the IIT Delhi Concurrency Workbench is applied to the products coming from the previous stage. Its outputs are employed by a software module that writes a Comma Separated Values (CSV) file, one per category. Rows represent the wiki pages, whilst columns "Weak Core Property" and "Strong Core Property" contains respectively the number of times each aforementioned properties are satisfied; instead, "Island Property" and "Loop Property" can be TRUE or FALSE. Except for the "Island Property" and "Loop Property" that work using a target page only, the remaining two properties have been applied to all the pages, i.e. to check that X is a weak/strong core page we have to verify the formula for each page P of the category.

D. *Maintenance*

The final stage deals with maintenance tasks. After model creation, automatic model checking and synthesis generation stages, we have all the necessary products to reach our goals, i.e. simplify maintenance tasks of a wiki. We employ the results of the previous formal verification process.

An "Island Property" allows to check whether a page is an "island". The latter does not communicate with any other page: it is an isolated page. This means in term of refactoring that we detected a problem on the category and the page should be moved elsewhere.

A "Loop Property" proves that the page is involved in a loop. The pages composing the loop behave as concepts that span across multiple pages. Thus, when adding a new page that may affect the ones of a loop, we need to split it keeping up the original structure and avoiding new links which would make future additions and modifications harder. Creating unnecessary links may also bewilder the reader.

When a page is referred by another page, it is a candidate to be elected as a "Core" page. A "Weak Core Property" demonstrates that a page is referred by another page of the category a certain amount of times directly or indirectly. A "Strong Core Property" means that a page is directly linked to the one whose property is verified. To elect a page as "Strong Core" (resp. "Weak Core") we have to established a threshold n (resp. m). Thus, we call "Strong Core" (resp. "Weak Core") a page to which are connected at least $n\%$ (resp. $m\%$) pages belonging to the inspected category. As occurs with software modules, it is crucial identifying a core page during refactoring

```
'''Edmund Melson Clarke, Jr.''' (born July 27, 1945) is a [[computer scientist]]
and [[academic]] noted for developing [[model checking]], a method for formally
verifying hardware and software designs. He is the [[FORE Systems]] Professor of
[[Computer Science]] at [[Carnegie Mellon University]].  Clarke, along with
[[E. Allen Emerson]] and [[Joseph Sifakis]], is a recipient of the 2007
[[Association for Computing Machinery]] [[Turing Award|A.M. Turing Award]].

== Biography ==

Clarke received a [[Bachelor of Arts|B.A.]] degree in [[mathematics]] from
the [[University of Virginia]], [[Charlottesville, Virginia|Charlottesville, VA]],
 in 1967, an [[Master of Arts (postgraduate)|M.A.]] degree in [[mathematics]] from
[[Duke University]], [[Durham, North Carolina|Durham NC]], in 1968,
== Work ==
```

Fig. 1: Wiki Page Markup Snippet of “Edmund M. Clarke” - english Wikipedia

tasks because a core module has a lot of responsibilities: it is like all the involved pages (modules) need it to work.

The properties are flexible and may be adapted to various domains. For example, one may change aforementioned thresholds or add new domain oriented properties.

IV. Experimental results

A. Parsing Wikipedia

Wikipedia is powered by Mediawiki[3], a popular free web-based wiki software application, developed by the Wikimedia Foundation.

It is the main engine of most wikis all over the world. It is written in the PHP programming language and it uses a back-end database. Indeed, all the wiki pages are stored in a MySQL database. The Wiki Markup Language is adopted for both pages and relationships (i.e. links). An example of a wiki page markup snippet regarding Edmund M. Clarke is shown in Figure 1. The figure shows sections starting with a double “=“ and internal links surrounded by double square braces (i.e. “[[FORE Systems]]”).

The database dumps of Wikipedia are freely available. They can be downloaded from Wikimedia[4] or using Special:Export function to select one or more categories[5]. The latter is usually available in most wikis even the smaller ones.

Dump data must be parsed in order to start the analysis. “Wiki Dumps” contains pages in wiki markup language, while “MediaWiki Parsing” provide facilities to fetch them and proceed with next “Model Creation”. A MediaWiki parser is capable of locating categories, pages, sections and links.

Wiki categories, wiki pages, wiki sections and links are transformed into CSS processes to realize a suitable model. Aforementioned selective-μ-calculus properties are automatically generated in this stage. Thus, “CCS Model Creation” yields two files - .ccs and .mu files.

Experiments were conducted on three categories of Wikipedia:

- “Fungi found in fairy rings”(Fungi) (composed of 22 pages);
- “Computer science conferences”(CS) (composed of 79 pages); and
- “Naval battles involving Great Britain”(Battles) (composed of 161 pages).

[3]http://www.mediawiki.org/wiki/MediaWiki
[4]http://dumps.wikimedia.org
[5]http://en.wikipedia.org/wiki/Special:Export

TABLE II: Number of Islands and Loops per Category

Category	Total pages	Islands	Loops
Fungi	22	13	0
CS	79	24	9
Battles	161	120	5

As can be noted, the chosen categories are different for size (increasing) and topic.

The execution time was maximum 30 minutes. This is quite high, considering the small scale of the projects: anyway, we found that it is mostly due to the overhead needed to invoke the external command-line IIT Delhi Concurrency Workbench from the Java prototype. We plan to integrate a CCS equivalence checker in the architecture, removing this overhead. Nevertheless, the objective of our work is to focus on the methodology rather that on the efficiency of the tool. In fact, the prototype has been shown to yield good results, even though it takes a long time.

A synthesis of the collected results is shown in Table II. The latter shows only the results obtained by verifying properties “Island“ and “Loop”. “Fungi” is a really small category and it does not contain core pages. We grouped pages in (weak and strong) groups, depending on the number of references per page, as shown in Figures 2a and 2b for the “CS” category. In Figure 2a, “SRef$N$” indicates a group whose pages are directly referred N times. On the one hand, we set the threshold for “Strong Core” to 10%. Hence, in category CS the group “SRef10” contains all strong core pages. One of them is “List of computer science conferences”. In fact, different pages refer to the “List of computer science conferences”. Similarly, in Figure 2b, “WRef$N$” indicates a group whose pages are indirectly referred N times. On the other hand, we set the threshold for “Weak Core” to 25%. Thus, in category CS only group “WRef21” contains weak core pages. One of them is “Symposium on Applied Computing”. Indeed, many pages of the category cross through it.

Moreover, in Battles we found 20 core page candidates, respectively 12 “Weak Core” and 8 “Strong Core” pages.

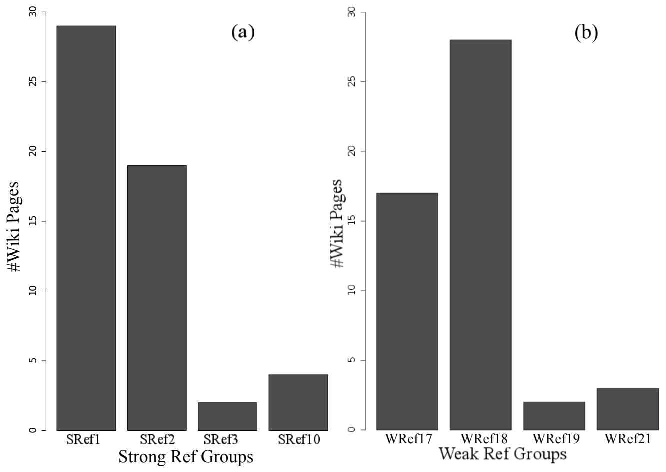

Fig. 2: Number of pages per Strong Ref groups (a) and Weak Ref groups (b) - CS category of English Wikipedia

Note that we applied our methodology on Wikipedia because it has a big architecture with many links and pages. Wikipedia is a case study only and our main purpose is not focus on the free encyclopedia. We want to aim attention at the emerging trend of enterprise wikis [11] exploited as powerful knowledge sharing tools in enterprises.

We work offline and avoid parsing wikis on the fly or do web-crawling. Our methodology can be easily integrated as a part of existing processes because of its simplicity. Refactoring operations can be done on the dump. The latter may be replaced/updated during idle times (e.g. during the night).

Refactoring also assume another fashion if employed to warn the user while editing the wiki. In this meaning we deal with "gardening", but it is just a set of warnings and should not affect the user i.e. he can freely edit and add content for knowledge sharing.

V. Conclusion and Future Work

In this paper we have investigated a novel methodology based on formal methods for analysis and verification of wikis. Starting from a wiki dump, after parsing wiki markup using a MediaWiki parser, we generate a process model for each wiki category employing the Milner's process algebra Calculus of Communicating Systems [1] and a set of specific properties expressed using the selective-μ-calculus [3]. We verified these properties against the model in order to simplify maintenance tasks.

Encouraging results obtained from three categories of Wikipedia motivate us to prosecute our research. Future work will focus on improving and extending the methods outlined in this article. In fact, there still is much work to do. First, our method must be integrated in the wiki design process to provide real benefits. We want to model check a large amount of wiki pages and category and focus on enterprise wikis. Weak and strong equivalence [1] may be applied to compare similar structures of categories of wikis and help a user to add/edit new wiki pages (i.e. "gardening" operations). Equivalence powered methods [14] will also check and improve the quality of a wiki e.g. avoiding to add useless content or helping the user proposing an architecture. Natural Language Processing techniques may be useful tools to add semantics to the methodology improving the accuracy of results. Yet, new properties derived from software bad smells and Opdyke's individual refactoring operations [15] will be verified.

Furthermore, we want to design and develop user-friendly tools to add new properties without knowing the selective-μ-calculus logic and to help both experts and non-experts to set the thresholds, as done in [16]. We even plan to show our results to Wikipedia's maintainers and engineers since we used the free encyclopedia as a first case study.

Each step of our methodology has been translated in a Java prototype to fulfil all the stages except the last. "Maintenance" stage still needs the participation of a domain expert, as occurs in other contexts [17]. We will also going to automate this step and integrate our software in a wiki engine i.e. MediaWiki, demonstrating the powerful of formal methods even in this area. We intend to redesign the tool to face two fundamental challenges: performance, the speed at which the tool can return its results, and scalability, the extent to which the tool can manage increasingly large categories.

Finally, nothing prevents to apply our methodology to other kind of text like DocBook, LateX, even web content e.g. property "Island" may be exploited to improve the security.

References

[1] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[2] R. Cleaveland and S. Sims, "The ncsu concurrency workbench," in *CAV*, ser. Lecture Notes in Computer Science, R. Alur and T. A. Henzinger, Eds., vol. 1102. Springer, 1996, pp. 394–397.

[3] R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini, "Selective mu-calculus and formula-based equivalence of transition systems," *J. Comput. Syst. Sci.*, vol. 59, no. 3, pp. 537–556, 1999.

[4] M. Rosenfeld, A. Fernández, and A. Díaz, "Semantic wiki refactoring. a strategy to assist semantic wiki evolution," in *Proceedings of the Fifth Workshop on Semantic Wikis (SemWiki 2010), co-located with 7th European Semantic Web Conference, ESWC*, 2010.

[5] G. M. Alluvatti, A. Capiluppi, G. De Ruvo, and M. Molfetta, "User generated (web) content: Trash or treasure," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, ser. IWPSE-EVOL'11. New York, NY, USA: ACM, 2011, pp. 81–90.

[6] L. Aversano, G. Canfora, G. De Ruvo, and M. Tortorella, "An approach for restructuring text content," in *ICSE*, 2013, pp. 1225–1228.

[7] F. M. Donini, M. Mongiello, M. Ruta, and R. Totaro, "A model checking-based method for verifying web application design," *Electronic Notes in Theoretical Computer Science*, vol. 151, no. 2, pp. 19–32, 2006.

[8] M. Haydar, A. Petrenko, S. Boroday, and H. Sahraoui, "A formal approach for run-time verification of web applications using scope-extended ltl," *Information and Software Technology*, vol. 55, no. 12, pp. 2191–2208, 2013.

[9] R. Chebil, W. Chaari, S. Cerri, and K. Ghedira, "A causal graph based method to evaluate e-collaboration scenarios," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, June 2013, pp. 225–230.

[10] S. Schaffert, "Ikewiki: A semantic wiki for collaborative knowledge management," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on*, June 2006, pp. 388–396.

[11] A. Stocker, A. Richter, P. Hoefler, and K. Tochtermann, "Exploring appropriation of enterprise wikis," *Computer Supported Cooperative Work (CSCW)*, vol. 21, no. 2-3, pp. 317–356, 2012.

[12] C. Stirling, "An introduction to modal and temporal logics for ccs," in *Concurrency: Theory, Language, And Architecture*, 1989, pp. 2–20.

[13] R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini, "Reduced models for efficient ccs verification," *Formal Methods in System Design*, vol. 26, no. 3, pp. 319–350, 2005.

[14] N. De Francesco, G. Lettieri, A. Santone, and G. Vaglini, "Grease: a tool for efficient "non-equivalence checking," *ACM Trans. Softw. Eng. Methodol.*, to appear.

[15] W. F. Opdyke, "Refactoring: A program restructuring aid in designing object-oriented application frameworks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1992.

[16] G. De Ruvo and A. Santone, "An eclipse-based editor to support lotos newcomers," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2014 IEEE 23nd International Workshop on*, June 2014.

[17] M. L. Bernardi, M. Cimitile, and D. Distante, "Web applications design recovery and evolution with re-uwa," *Journal of Software: Evolution and Process*, vol. 25, no. 8, pp. 789–814, 2013.