Equivalence-based Selection of Best-fit Models to Support Wiki Design

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 the access. Wikis are becoming a new work tool in enterprises and are widely spreading everywhere. Indeed, it is important to consider the design and evolution of a wiki. We present a methodology to help wiki designers, engineers and domain experts. In practice, from the wikis we derive formal models, which are successively used to perform equivalence checking. More precisely, in order to design a wiki p we propose a methodology for the selection of the best-fit wiki model q, among a set of candidate ones. For best fit we mean that p and q have a similar structure. To handle the complexity of finding all possible candidates processes q, a heuristic function can be used to filter the set of significant candidates, and to speed up the search of a successful one, which is the main contribution of the paper. Eventually, the elected model may be exploited to start the design process.**

***Index Terms*—Wiki, Equivalence Checking, Process Algebra, Design.**

I. Introduction

The first wiki (WikiWikiWeb) 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.

In 2006 Ward Cunningham himself presented a set of Wiki Principles [1] that have been driving wikis. Basically, the father of wikis stressed the openess of a wiki, where pages - even if incomplete or rather poorly organized - can be edited by anyone who sees fit (Open Principle). While the community grows up the content of the wiki evolves as well (co-evolution). The activities within the site can be watched and reviewed by any other visitor. Ambiguity and duplication can be removed by finding and citing similar or related content. Thus, wikizens - the users of a wiki - can come and go without changing a wiki's identity. During the last decade the identity of wikis has changed, because it has been exploited in different context and environments. In fact, wikis are becoming a new work tool used as in-house document systems and applications and as a powerful aid for providing knowledge sharing capabilities. A recent research by Standing and Kiniti [2] pointed out that organizational or corporate wikis are sustainable and can be beneficial to organizations particularly in improving work processes, collaboration and knowledge management. Indeed, wikis have been seen as new way to do innovation in organizations for developing new products and processes. Furthermore there are many Free Libre Open Source Software projects (FLOSS) which a are using wikis as main form of documentation. Software versioning repositories (e.g., github[2], bitbucket[3]) provide an integrated wiki.

WikiWikiWeb[4] was developed in order to make the exchange of ideas between programmers easier acting as a library to create design patterns. Thus, the first idea of wiki has its foundation in software engineering, but the principles of a wiki do not rely on software engineering maybe due to their open-editing nature. Notwithstanding, the evolution of wikis must be driven by solid principles to bring real benefits in various disciplines. As well as a large software system cannot be designed simply adding components in disparate ways by anyone who is able to write code, a large wiki may need proper design phase even though "wikizens" can behave as either readers or designers. This does not mean refusing all the starting guidelines: methodologies and tools have to adapt a wiki according to its purpose and target domain, providing the integration with the wiki nature itself.

We argue that we can easily add a proper methodology to the wiki design process without affecting the basics that lead the wiki-model to success. We propose a wiki-design methodology based on formal methods.

In this paper we use Milner's Calculus of Communicating Systems (CCS) [3] as specification formal language. For simplicity we use CCS, but we can equivalently use any other process algebra based specification language, such as CSP and LOTOS. Based on CCS, different methods and tools have been developed for the verification of systems. Examples of such methods are *model checking* [4] and *equivalence checking* [3]. The aim of model checking is to explore the behaviour of a system exhaustively, in an attempt to find errors, while the aim of equivalence checking is to determine whether two systems are equivalent to each other according to some mathematically-defined notion of equivalence. Model checking has been applied to several fields. For instance, it has been used in bioinfomatics to infer gene regulatory networks from time series data [5] or to assess the correctness of JVM implementation [6]. In this paper, we apply equivalence checking to compare a desired wiki artefact with existing ones. The user can express a set of features (i.e., number of links, pages, categories and so on) and it is prompted with a set of similar structures. The latter can be chosen as a starting

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

[2] http://github.com

[3] http://bitbucket.org

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

point. From wikis we derive process specifications and from desired artefact's features we build a possible specification. After that, we compare existing wiki processes speeding up the exploration using a heuristic function. The remainder of this paper is organized as follows. First, in Section II we give basic definitions of wikis and CCS. Section III deals with our methodology for design and verification of wikis. Finally, in Section IV we conclude providing new inputs for further research and we present comparisons with related work.

II. Preliminaries

In this section, after introducing the basic definitions of wikis, we present the Calculus of Communicating Systems (CCS) [3], the process algebra we have adopted for analysing and verifying wikis.

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"[5] 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 [7]. 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 i.e., which ones are most popular and/or need updating. There are disparate types of wikis such as Wiki Mapia that combines Google Maps with a wiki system supporting over 35 languages; WikiTravel, a travel guide; LyricWiki, a listing of lyrics by album; Flu Wiki, intending to help local public health communities coping with a possible (avian) influenza pandemic, and Ganfyd, 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; IkeWiki [8], 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 [9]. Moreover, many FLOSS projects use wikis for technical and user documentation, i.e., Eclipse and its Eclipsepedia.

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

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) [3]. Readers unfamiliar with CCS are referred to [3] 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 $x \stackrel{\text{def}}{=} p$, where p is called the *body* of x. Given a set $L \subseteq \mathcal{V}$, by $\overline{L}$ we denote the set $\{\overline{\alpha} | \alpha \in L\}$. We denote the set of processes by $\mathcal{P}$.

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 $|$ 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 ($x \stackrel{\text{def}}{=} p$) is that of its definition p (rule **Con**).

The standard *operational semantics* [3] is given by a set of conditional rules describing the transition relation of the automaton corresponding to the process p. This automaton is called labelled transition system (LTS) of p.

Equivalence: Given two LTSs, a natural question is whether they describe the same behaviour. To answer this question we must first specify what we mean by "the same". For example, are we satisfied if both LTSs can perform the same sequences of actions (starting from their initial states) or do we want to

impose stricter criteria? In other words, we have to specify when we consider two LTSs to be equivalent.

Various notions of equivalence have been defined in the literature. Some are finer than others, meaning that the criteria that two LTSs should meet for them to be called equivalent, are stronger.

In the following we introduce a well-known notion of behavioural equivalence which describes how processes (i.e., systems) match each other's behaviour: the *strong equivalence* introduced by Milner. Strong equivalence is a kind of invariant relation between process that is preserved by actions as stated by the following definition.

Definition 2.1 (strong equivalence): Let p and q be two CCS processes.

- A strong bisimulation, $\mathcal{B}$, is a binary relation on $\mathcal{P} \times \mathcal{P}$ such that $p\ \mathcal{B}\ q$ implies:
  (i) $p \xrightarrow{\alpha} p'$ implies $q \xrightarrow{\alpha} q'$ with $p'\ \mathcal{B}\ q'$; and
  (ii) $q \xrightarrow{\alpha} q'$ implies $p \xrightarrow{\alpha} p'$ with $p'\ \mathcal{B}\ q'$
- p and q are strongly equivalent ($p \sim q$) iff there exists a strong bisimulation $\mathcal{B}$ containing the pair (p, q).

III. The Methodology

In this section we present our methodology for the selection of the best-fit wiki model, among a set of candidate models, for a given wiki that must be designed. More precisely, a user describes a tentative design and the methodology finds and returns the wiki design with the closest structure characterised by number of sections and links (internal/external pages). In practice, from the wikis we derive CCS processes, which are successively used to perform equivalence checking. The goal of our methodology is to help wiki designers, engineers and domain experts during evolution and re-factoring tasks. As shown in Figure 1, our approach requires a four-step process. The four steps are:

1) CCS model creation
2) Selection of the best-fit wiki model
3) Equivalence checking
4) Wiki design process and verification

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.

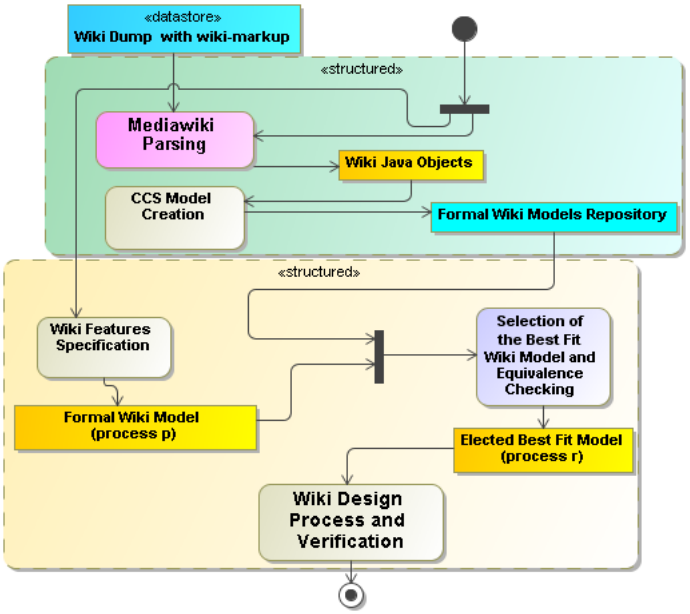

Fig. 1: The Methodology

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:

- internal page belongs to the analysed category C;
- internal page belongs to a different category;
- external web page.

Wiki category, wiki page and wiki section are then translated in CCS process. For the details the reader can refer to [10]. A small CCS specification is shown in Table I. It models a category CAT with two pages, i.e., P_1 and P_2. Page P_1 is composed of one section SP_{1_1} (with a link to P_2), while P_2 is composed of two sections SP_{2_1} and SP_{2_2}. SP_{2_1} has a link to an external page of the category CAT (action pwk), while SP_{2_2} has a link to an external web page (action $pext$).

TABLE I: A CCS specification example

CAT	$\stackrel{\text{def}}{=}$	$P_1 + P_2$	SP_{1_1}	$\stackrel{\text{def}}{=}$	P_2
P_1	$\stackrel{\text{def}}{=}$	$inP_1.SP_{1_1}$	SP_{2_1}	$\stackrel{\text{def}}{=}$	$pwk.nil$
P_2	$\stackrel{\text{def}}{=}$	$inP_2.(SP_{2_1} + SP_{2_2})$	SP_{2_2}	$\stackrel{\text{def}}{=}$	$pext.nil$

Actually, as shown in Figure 1, the "CCS model creation" is obtained analysing wiki databases. The database dumps of Wikipedia are freely available. They can be downloaded from Wikimedia[6] or using Special:Export function to select one or more categories[7]. The latter is usually available in most wikis even the smaller ones. Dump data must be parsed in order to start the analysis. "Wiki Dump" contains pages in

[6]http://dumps.wikimedia.org
[7]http://en.wikipedia.org/wiki/Special:Export

wiki markup language, while "MediaWiki Parsing" provides facilities to fetch them and through the transformation operator $\mathcal{T}$, CCS processes are generated in order to create the "Formal Wiki Models Repository". A MediaWiki parser is capable of locating categories, pages, sections and links.

Using the wiki-to-CCS transform operator $\mathcal{T}$, we can create a repository of CCS models corresponding to existing wiki categories. Afterwards, we have to define the CCS process corresponding to the wiki category that must be designed. For this purpose, we have to take into account the number of internal/external pages. This information can guide the definition of the structure of the CCS process corresponding to p ("Wiki Features Specification" in Figure 1).

B. Selection of the best-fit wiki model

Exploiting the previous step, we can operate on a repository of CCS models corresponding to existing wiki categories. The next step is to use that repository to find wiki-processes q that best fit with the wiki process p that must be designed. For best fit we mean that p and q have a similar structure, independently of the action name of the pages. To handle the complexity of finding all possible candidates processes q, a heuristic function can be used to filter the set of significant candidates, and to speed up the search of a successful one. The heuristic function proposed in this article exploits the process p and essentially takes into account its syntactic structure; the output of the function, namely $\widehat{h}$-value, is an integer value that, roughly speaking, measures the complexity of the process q with respect to the structure of p. The function produces 0 to mean that q cannot be the good candidate for the process p: in this case the structure of q is in contrast with structure of p and the unsuccessful result is returned as fast as possible. In the other cases, greater is the value returned by the heuristic function, greater is the structural similarity with p.

We formally define the function $\widehat{h}$.

Definition 3.1: Let p and q be two CCS processes. The function $\widehat{h}(q,p)$ is inductively defined on the process q as in Table II.

For $q = nil$ (*Rule R1*) $\widehat{h}$ returns 1, when p cannot move, as trivially nil is equivalent to p, otherwise $\widehat{h}$ returns 0.

When applied to $\alpha.q$ (*Rule R2*), if $p = \beta.p'$ the number returned by $\widehat{h}$ is 1 plus the value returned by the recursive application of the function to q and p'. If $p = p'[f]$, we simply iteratively apply the function on $\alpha.q$ and p' as the relabelling function does not modify the structure of p. Otherwise, the function terminates returning 0.

When the choice of two processes is encountered $q = q_1 + q_2$ (*Rule R3*), if also p is a summation the maximum number between the two components is returned, while if $p = p'[f]$, we simply iteratively apply the function on q and p' as the relabelling function does not modify the structure of p. Otherwise, the function $\widehat{h}$ returns 0, since trivially q does not have the same structure of p.

For the parallel composition of processes (*Rule R4*), the function acts as *Rule R3*.

When considering a restricted process $q\backslash L$ (*Rule R5*), if $p = p'\backslash L'$ the number returned by $\widehat{h}$ is 1 plus the value returned by the recursive application of the function to q and p'. If $p = p'[f]$, we simply iteratively apply the function on $q\backslash L$ and p' as the relabelling function does not modify the structure of p. Otherwise, the function $\widehat{h}$ returns 0, since trivially q does not have the same structure of p.

When considering a relabelled process $q[f]$ (*Rule R6*), if $p = p'[f]$, we simply iteratively apply the function on p' as the relabelling function does not modify the structure of p. Otherwise, the function is applied on q and p.

Finally, when both p and q are constants, by (*Rule R7*) the function applies to the bodies of each constant; while when only q is a constant ($q = x$) the function is recursively applied to the body of x and on p itself. The halting of $\widehat{h}$ is not a problem; it is sufficient to expand once the body of each constant x. This can be obtained storing in a set C each constant which has been expanded. Initially, C is equal to the empty set. When we find an already expanded constant the value returned is 0, otherwise we recursively apply the function on the body of the constant.

Note that $\widehat{h}$ is very simple to calculate as it is syntactically defined.

Example 3.1: Consider the following CCS repository:

$$\mathcal{R} = \{q_1, q_2, q_3\}$$

where

$$\begin{aligned} q_1 &\stackrel{\text{def}}{=} a_1.b_1.nil|c_1.d_1.nil \\ q_2 &\stackrel{\text{def}}{=} a_2.b_2.nil + d_2.nil \\ q_3 &\stackrel{\text{def}}{=} a_3.(b_3.nil + c_3.nil) + d_4.nil \end{aligned}$$

Let $p = a.(b.nil + c.nil) + d.nil$ be the CCS process that we have to designed. It turns out that:

$$\widehat{h}(q_1,p) = 0, \qquad \widehat{h}(q_2,p) = 4, \qquad \widehat{h}(q_3,p) = 9$$

The above results suggest that the more suitable process $q \in \mathcal{R}$ is the process q_3, since it has the same structure of the process p. It turns out that $\widehat{h}(q_1,p) = 0$, since q_1 has a structure completely different from p, while $\widehat{h}(q_2,p) = 4$, since q_2 match with p only a summation, two prefixing operators (i.e., the prefix after a_2 and that after d_2) and one nil operator. Finally, $\widehat{h}(q_3,p) = 9$, as all the right operators (four prefixing operators, two summation operators, three nil) occurring in q_3 match with those occurring in p.

C. Equivalence checking

The notion of best-fit model is implemented using equivalence checking. We use the IIT Delhi Concurrency Workbench [11] which is one of the most popular environments for verifying concurrent systems which supports several different specification languages, among which CCS and incorporates an equivalence checker. This phase starts from the processes whose heuristic values are maximal. The processes with bigger heuristic values are the most likely similar to the wiki category that has to be designed. Equivalence checking is applied as the following strategy suggests.

$$\textbf{R1}.\quad \widehat{h}(nil,p) = \begin{cases} 1 & \text{if } p = nil \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{R2}.\quad \widehat{h}(\alpha.q,p) = \begin{cases} 1+\widehat{h}(q,p') & \text{if } p = \beta.p' \\ \widehat{h}(\alpha.q,p') & \text{if } p = p'[f] \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{R3}.\quad \widehat{h}(q_1+q_2,p) = \begin{cases} 1+\max\{(\widehat{h}(q_1,p_1)+\widehat{h}(q_2,p_2)),(\widehat{h}(q_1,p_2)+\widehat{h}(q_2,p_1))\} & \text{if } p = p_1+p_2 \\ \widehat{h}(q_1+q_2,p') & \text{if } p = p'[g] \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{R4}.\quad \widehat{h}(q_1|q_2,p) = \begin{cases} 1+\max\{(\widehat{h}(q_1,p_1)+\widehat{h}(q_2,p_2)),(\widehat{h}(q_1,p_2)+\widehat{h}(q_2,p_1))\} & \text{if } p = p_1|p_2 \\ \widehat{h}(q_1|q_2,p') & \text{if } p = p'[g] \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{R5}.\quad \widehat{h}(q\backslash L,p) = \begin{cases} 1+\widehat{h}(q,p') & \text{if } p = p\backslash L' \\ \widehat{h}(q\backslash L,p') & \text{if } p = p'[f] \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{R6}.\quad \widehat{h}(q[f],p) = \begin{cases} \widehat{h}(q,p') & \text{if } p = p'[g] \\ \widehat{h}(q,p) & \text{otherwise} \end{cases}$$

$$\textbf{R7}.\quad \widehat{h}(x,p) = \begin{cases} \widehat{h}(q,p') & \text{if } x \stackrel{\text{def}}{=} q \text{ and } p = y \text{ and } y \stackrel{\text{def}}{=} p' \\ \widehat{h}(q,p) & \text{if } x \stackrel{\text{def}}{=} q \text{ and } p \neq y \end{cases}$$

TABLE II: The $\widehat{h}$ function.

Let p be the CCS process corresponding to the wiki category that must be designed and $\mathcal{R}$ the following CCS repository:

$$\mathcal{R} = \{q_1, \ldots, q_n\}$$

1) Insert in the set S all the processes $q_i \in \mathcal{R}$, $1 \leq i \leq n$, with the maximum $\widehat{h}$-value;
2) Check in S, for a process r, such that
$$\mathcal{S}(r[f]) \sim \mathcal{S}(p[f])$$
where f ia a relabebling function defined as explained below.
3) if r exists then the best candidate is r, otherwise we can return any process in S.

The strategy operates as follows:

1) First, we construct the set S of CCS processes selecting, from the repository $\mathcal{R}$, all processes with the maximum $\widehat{k}$-value. This is obtained exploiting the $\widehat{h}$ function previously defined. The processes in S are the processes most structural similar to the process p that has to be designed.
2) Then, we formally verify the structural similarity between any process r in S and p. This is obtained using strong bisimilarity equivalence. Since we are not interested in the action name of a single page but only in finding similar structure, all the actions occurring both in r and in p are relabelled without the specific name of the page. This is obtained using the function f that relabels all actions corresponding to specific pages with generic new action names. They must indicate only the type of the page (internal/external). In addition to equivalence checking, in this step we can also apply model checking, if we want to be more accurate on the similarity between two processes.
3) Finally, we elect as best model r, if it exists a process satisfying the conditions of step 2, otherwise we elect as best model any process in S.
The two steps "Selection of the best-fit wiki model" and "Equivalence checking" are grouped in Figure 1.

D. Wiki design process and verification

The final stage deals with design and maintenance tasks. The elected model of the previous step may be exploited to start the design process. Moreover, the CCS model can be also used to verify wikis as done in [10]. After model creation, automatic model checking can be used. Once we have the CCS processes of wikis, we can use temporal logic to specify desired properties. For example, we can consider the property: "Island Property" that allows to check whether a page is an "island". The latter does not communicate with any other page: it is an isolated page. It is important to verify a similar property since it means, in term of refactoring, that we detected a problem on the category and the page should be moved elsewhere. Obviously, the wiki structure can be naturally modelled as a graph. Verifying properties like "Island property" can be done by a simple reachability check or graph connectivity analysis.

Nevertheless, the availability of powerful formal verification tools allow us to easily verify any other property.

IV. Conclusion and Related Work

We proposed an equivalence-based methodology to support the design of wikis. First, from existing wiki databases, we derive wiki formal models, by parsing a wiki dump. Thus, a formal repository is generated. Afterwards, an user can provide

a set of desired features in order to build a tentative process. We proceed with equivalence checking looking for a similar structure between the formal repository of processes and the tentative one. Eventually we chose the process that best-fit the one that must be designed. For best-fit we mean that the processes have a similar structure without considering the name of the actions which activate the pages. In particular we selected best-fit models speeding the election up with the aid of a heuristic function.

Actually, application of formal methods has been highlighted in connection with a variety of disciplines such as design pattern mining [12], biology [5][13], test generation [14], battery management systems [15] among others. Nevertheless, as far as we know nobody addressed the problem of wiki design exploiting the power of formal methods. The first attempt can be found in [10], where the authors propose a novel methodology based on model checking to analyse and verify the architecture of wikis.

Rosenfeld et al. [16] 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 look for similar structures.

Alluvatti et al. [17] 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.

Puente et Diaz [18] defined the semantics of common refactoring operations based on Wikipedia best practices, advocated for the use of mind maps as a visualization of wikis for refactoring and introduced a Domain Specific Language for wiki refactoring built on top of FreeMind, a mind mapping tool.

Aversano et al. [19] 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. In a more recent paper [20], Dohrn and Riehle extended their Wiki Object Model (WOM) to achieve wiki transformations and help users to easily and consistently evolve the content and structure of a wiki. They employed a stored WOM and used an expensive script driven by the eXtensible Stylesheet Language: Transformations (XSLT), modifying the foundations of a wiki.

Future work will focus on improving the heuristic-based function in order to increase the accuracy of the selected models. A step to improve the readability of the selected models may be added [21]. We even plan to apply our methodology in a real context to prove the effectiveness of the equivalence-based selection of best-fit models. An unique framework to design and refactoring [10] wikis can be reached. Finally, we plan to extend our technique, in order to introduce semantic analysis of category names and to model with Markov chains, the navigation among pages especially because some links are more likely to be used than others [22].

REFERENCES

[1] W. Cunningham *et al.*, "Wiki design principles," 2006.

[2] C. Standing and S. Kiniti, "How can organizations use wikis for innovation?" *Technovation*, vol. 31, no. 7, pp. 287–295, 2011.

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

[4] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 2001.

[5] M. Ceccarelli, L. Cerulo, and A. Santone, "Infer gene regulatory networks from time series data with formal methods," in *2013 IEEE International Conference on Bioinformatics and Biomedicine, Shanghai, China, December 18-21, 2013*, 2013, pp. 115–120. [Online]. Available: http://dx.doi.org/10.1109/BIBM.2013.6732473

[6] A. Calvagna, A. Fornaia, and E. Tramontana, "Assessing the correctness of JVM implementations," in *2014 IEEE 23rd International WETICE Conference, WETICE 2014, Parma, Italy, 23-25 June, 2014*, 2014, pp. 390–395. [Online]. Available: http://dx.doi.org/10.1109/WETICE.2014.33

[7] 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.

[8] 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.

[9] 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.

[10] G. De Ruvo and A. Santone, "A novel methodology based on formal methods for analysis and verification of wikis," in *2014 IEEE 23rd International WETICE Conference, WETICE 2014, Parma, Italy, 23-25 June, 2014*, 2014, pp. 411–416. [Online]. Available: http://dx.doi.org/10.1109/WETICE.2014.25

[11] 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.

[12] M. L. Bernardi, M. Cimitile, G. De Ruvo, G. A. Di Lucca, and A. Santone, "Improving design patterns finder precision using a model checking approach," *CAiSE forum*.

[13] M. Ceccarelli, L. Cerulo, G. De Ruvo, V. Nardone, and A. Santone, "Infer gene regulatory networks from time series data with probabilistic model checking," *FormaliSE 2015*.

[14] P. Arcaini, A. Gargantini, and E. Riccobene, "An abstraction technique for testing decomposable systems by model checking," in *Tests and Proofs - 8th International Conference, TAP 2014, Held as Part of STAF 2014, York, UK, July 24-25, 2014. Proceedings*, 2014, pp. 36–52. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-09099-3_3

[15] F. Baronti, C. Bernardeschi, L. Cassano, A. Domenici, R. Roncella, and R. Saletti, "Design and safety verification of a distributed charge equalizer for modular li-ion batteries," *IEEE Trans. Industrial Informatics*, vol. 10, no. 2, pp. 1003–1011, 2014. [Online]. Available: http://dx.doi.org/10.1109/TII.2014.2299236

[16] 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.

[17] 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.

[18] G. Puente and O. Díaz, "Wiki refactoring as mind map reshaping," in *CAiSE*, 2012, pp. 646–661.

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

[20] H. Dohrn and D. Riehle, "Design and implementation of wiki content transformations and refactorings," in *Proceedings of the 9th International Symposium on Open Collaboration*. ACM, 2013.

[21] S. Vajjala and D. Meurers, "Readability assessment for text simplification: From analyzing documents to identifying sentential simplifications," *International Journal of Applied Linguistics, Special Issue on Current Research in Readability and Text Simplification*, 2014.

[22] G. De Ruvo and A. Santone, "Analysing wiki quality using probabilistic model checking," in *2015 IEEE 24th International WETICE Conference, WETICE 2015, Larnaca, Cyprus, 15-17 June, 2015*, p. To appear.