An Enhanced Wiki for Requirements Engineering

David de Almeida Ferreira, and Alberto Rodrigues da Silva
INESC-ID / Instituto Superior Técnico
Rua Alves Redol 9, 1000-029 Lisbon, Portugal
david.ferreira@inesc-id.pt, alberto.silva@acm.org

***Abstract*—The high number of unsuccessful IT projects, due to the inconsistent and ambiguous requirements specifications, justifies the proposal of new socio-technical approaches to overcome these software quality issues. To address these problems it is required a platform that simultaneously enforces Requirements Engineering best practices, fosters stakeholders' participation, and supports requirements negotiation and management. In this paper, we present the rationale and design of an enhanced Wiki for dealing with Requirements Engineering aspects, according to a CSCW approach. Moreover, we discuss the benefits of this Wiki-based solution in the broader context of ProjectIT initiative, and how it can leverage the quality and rigor of requirements specifications through its integration with CASE tools.**

***Keywords*-Software Requirements; Software Engineering Tools and Methods.**

I. Introduction

Requirements Engineering (RE) is crucial for achieving success on IT projects. The RE process consists of a structured approach to the activities of capturing, organizing, documenting, and maintaining the requirements of a target system. The usage of techniques and models throughout this process supports the systematic execution of these activities. Wiegers [1] argues that the process of engineering requirements should be separated into two sub-processes: (1) *development of requirements*, which is dedicated to introducing new requirements during the early stages of the software development process; (2) *requirements management*, which is intended to monitor changes to all requirements already established and to assess the impact of including them into the target system. Regarding the first sub-process of RE, the early activities with which it deals are of paramount importance for both project management and the software development process [2]. The target system's requirements are the starting point to define the scope of the problem to be solved. Mistakes made early in the software development process, such as inadequate, inconsistent, incomplete, or ambiguous requirements, outcome in quality problems and high costs to correct them afterwards [3]. Therefore, requirements specifications influence all the subsequent activities and define the basis for assuring software's quality.

Despite the lack of accuracy, due to inherent characteristics of natural language, the majority of requirements documents are still specified with it. This is mainly due to natural language's expressive power and because non-technical users are familiar with it in their daily activities. To address this problem we created an extensible requirements specification language based on linguistic patterns [4]. Additionally, we developed a supporting CASE tool in the scope of the ProjectIT initiative [5]. This tool, the ProjectIT-Studio/Requirements, assist non-technical stakeholders during the specification activity by providing context-aware feedback with Natural Language Processing (NLP) techniques, thus allowing them to validate their own contributions. However, despite of the results achieved with the desktop-based approach, we realize the importance of social, organizational, and collaborative issues during requirements development, especially within contexts regarding to virtual and distributed teams. These scenarios require an evolutionary discovery of requirements in order to address the associated high level of uncertainty [6].

Given their characteristics, Wikis present themselves as a feasible solution to address the aforementioned problem. They are lightweight authoring systems that support interactive and collaborative work through an online environment. Wikis follow the principles of the constructivist learning theory [7]. By itself, the Wiki concept is simple [8]: it is based upon an open model and easy edition workflow. According to this model, by using a standard web-browser, any reader is simultaneously a potential author and reviser. By comparing and commenting each other's inputs, the participants not only become aware of what their peers know (the community knowledge), but they also benefit from an unconscious self-assessment effect, hence consolidating their believes. Since Wikis support work group processes, they are used in several contexts to record and refine ideas and information. Besides these inherent advantages, Wikis are excellent platforms for supporting project management activities, since they allow users to share ideas, define project's scope, keep logs of project progress, document project plans, and coordinate negotiation and discussions [9].

Because Wikis can easily integrate different stakeholders' perspectives and support software development processes, we consider Wikis not only suitable to become the underlying technology for a RE web-based tool, but also a prominent Web 2.0 [10] technology for integrating into ProjectIT-Enterprise, the existing web-based tool for process definition and project management.

This paper presents a metamodel that conceptualizes the main Wiki concepts. Additionally, it introduces some enhancement features that do not only ease the integration with the tools we developed, but also supports an attribute-based approach to capture requirements metadata. It still promotes reuse through templates of requirements doc-

uments, for capturing the best practices of RE. Section 2 introduces ProjectIT-Enterprise, highlighting its goals, background, and architecture. Section 3 provides a detailed analysis of the proposal for a Wiki-based RE tool. This leads to a discussion, in Section 4, about the feasibility of this enhanced Wiki system, when compared with projects and tools that share some of its goals. Finally, Section 5, draws the key conclusions, which justify our perception that this proposal has innovative contributions for the community.

II. ProjectIT Context

As a result of the experience gathered from previous projects, the Information Systems Group of INESC-ID, started some years ago an initiative, called ProjectIT [5]. Its main goals are to enhance productivity and rigor of Software Engineering activities. To achieve these goals and to assess our research ideas, we began the development of a complete software development workbench, which supports project management, requirements engineering, analysis, design, and code generation activities.

There are two complementary set of tools: ProjectIT-Enterprise and ProjectIT-Studio. The former is a web-application that supports the management of virtual and distributed teams, and presents a strong emphasis on activities related with project management [11]. ProjectIT-Enterprise supports the definition of tailored software development processes. This feature entails the definition of best practices and both project and artifact templates, and enables the instantiation of designed process afterwards as concrete projects. The *a priori* existence of a process has a normative influence and accelerates the project bootstrap within organizations by reducing coordination efforts, due to the clear definition of roles, workflows, and artifacts [11]. The main components of ProjectIT-Enterprise are depicted in Figure 1. On the other hand, the latter is

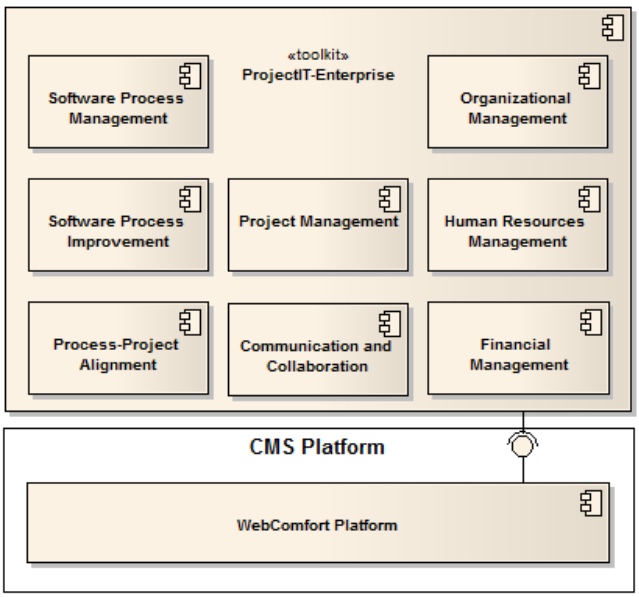

Figure 1. ProjectIT-Enterprise components.

a desktop-based CASE tool whose goal is to provide a set of tools for enhancing productivity of requirements specification and management, design models, automatic code generation, and software development. For dealing with Requirement Engineering issues we designed a model for requirements specification which, by raising their rigor and quality through verification, facilitates the reuse and integration with model-driven development environments, specifically ProjectIT-MDE tools [12]. ProjectIT-Studio/Requirements is the tool that embodies these features, and it is implemented as a plugin for ProjectIT-Studio. It provides a rich text editor that supports ProjectIT-RSL, a controlled natural language designed for requirements specification (for details see [4]). This editor supports the vision of a tool for writing requirements documents, like a word processor with *intellisense* that detects and gives instant feedback of errors violating the requirements language. It uses NLP techniques to capture the underlying requirements model in order to verify it.

Although ProjectIT-Studio/Requirements supports activities from the requirements development process, it lacks support for the RE's sub-process that deals with requirements management. Moreover, there is no support for communication mechanisms and the collaborative environment is insufficient. Therefore, we decided to explore Web 2.0 [13] features and develop a collaborative environment that mimics the ProjectIT-Studio/Requirements features, namely its rich text-editor. As mentioned in Section I, Wikis appear as a more adequate and feasible approach to the aforementioned shortcomings due to the simple philosophy behind the concept and theirs sustained success. We decided to develop a Wiki-based system to endow ProjectIT-Enterprise with collaborative Requirements Engineering support. The tight integration with requirements information leverages the awareness of all the other project management perspectives [2].

Despite the simplicity of the Wiki's philosophy, there are several Wiki engine implementations, each one with its own goals and peculiarities. We considered the possibility of adopting and customizing one from the myriad of available Wikis [14]. However, taking into account the characteristics of the special-purpose Wiki system that we are developing, we decided not to. The process of tailoring the functionalities of an existent Wiki engine to our specific needs appears to be overwhelming, not being worth the effort when compared with a solution from scratch; not to mention the potential problems related with architectural and technological idiosyncrasies. Moreover, according to the insight we gained during this research, the current state-of-the-art Wikis struggle to overcome common problems that could be easily eradicated through the adoption of a web-based platform/framework. There are several commodities, such as user management, role-based access control, and workspace confinement that could be obtained right out-of-the-box from this approach. There are several candidates for such platform/framework, namely the emerging Content Management System (CMS) platforms. Moreover, through the support of a CMS plat-

form, one could achieve content isolation through modularization of components and the possibility of multiple instantiation of the same module.

Following this line of thought, we decided to develop the Wiki system on top of the same platform as the ProjectIT-Enterprise tool, the WebComfort CMS platform [15]. WebComfort presents several advantages when compared with the ad-hoc and manual process of developing web-applications [16]. The relation of the enhanced Wiki with the WebComfort platform and its tight integration with the ProjectIT-Enterprise is illustrated in Figure 2.

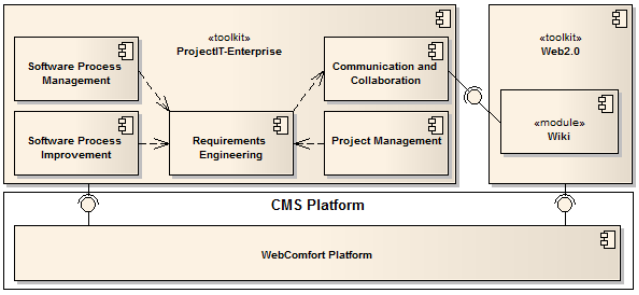

Figure 2. Wiki's integration with ProjectIT-Enterprise.

III. Wiki Enhancements

This section presents the design concerns of an enhanced Wiki system whose rationale is to support both Requirements Engineering's sub-processes. To fully understand the decisions' rationale, one must bear in mind its scope within a CSCW platform for Software Engineering, the ProjectIT-Enterprise tool. Our proposal for a special-purpose Wiki system addresses the following aspects: (1) structural and semantic issues; (2) a dynamic attribute-based mechanism; (3) fine-grain access control through customizable permissions; and (4) reuse through a templating mechanism. Each one of these topics is presented in detail in the following subsections.

A. Wiki Metamodel

Before adding complex features we decided to formalize the main concepts of a Wiki system as a starting point. The idea was to keep the simplicity of this kind of systems as much as possible, while identifying aspects with potential for improvement. The result of this conceptualization is presented in Figure 3. For clarity of the diagrams, the hierarchical labeling mechanism and the thread-like discussion page concepts that support discussions about a specific *WikiPage* are not represented.

WikiNamespace. As previously mentioned the ProjectIT-Enterprise tool deals with project management aspects. Hence, to succeed with the planned integration, we have to address concepts related with work-breakdown structure (WBS), namely: iterations, phases, and work packages. The alignment between the Wiki and the WBS concepts compels to the introduction of a scope mechanism to circumvent the problem of *WikiPage*s' name collision, which may occur when one needs

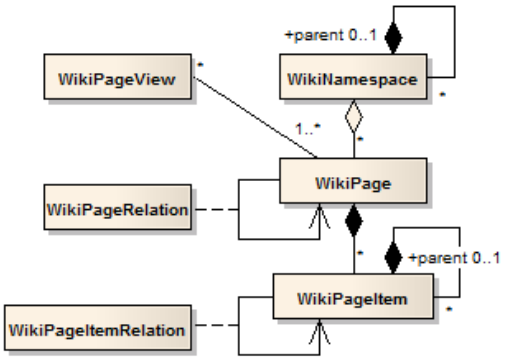

Figure 3. The Wiki metamodel.

to focus on the same subject according to different viewpoints (e.g., Software Engineering phases) or at different times (such as project iterations). The solution is to introduce the *WikiNamespace* concept. This approach avoids name collision without disrupting with the simple Wiki principles [8]. Just few Wikis address the name collision problem [17], and those that deal with it use a simpler concept called *workspace*, which is similar to the single scope provided by a root *WikiNamespace* without any child *WikiNamespace*s. The best analogy is to think about the *WikiNamespace* concept as a folder of a traditional OS's file system. We can create a hierarchy of *WikiNamespace*s like packages to gather related *WikiPage*s, without running the risk of name collision when dealing with different perspectives of the same subject. A good example is thinking of the same software component (i.e., a leaf of the project's WBS) during two different iterations of a project, or even at specification and design phases.

WikiPage. Following the same line of thought, we can think of the *WikiPage* concept as a file within a specific *WikiNamespace* - this is aligned with the implementation of the early Wikis, where wiki pages are stored as different files in the root directory of the Wiki system. Each *WikiPage* has a unique name inside the scope provided by the respective *WikiNamespace* and stores content related with its name. Within this conceptualization *WikiPage*s are containers of information about a specific topic.

WikiPageRelation. To enrich the structural and semantic approach for producing Wiki's content, we introduce the concept of typified relations between *WikiPage*s. The introduction of this concept allows one to link *WikiPage*s and simultaneously express a specific meaning to the relation's purpose. Although it introduces complexity, it also provides flexibility and empowers the Wiki users to establish mode advanced relations, which can be used by search and filtering mechanisms.

WikiPageItem. This concept represents *WikiPage*'s content blocks. The purpose behind the fine-grained

distinction of *WikiPage*'s elements is twofold: semantic and structural. The former allows the Wiki system to evaluate the *WikiPageItem* content according to its type, which is important to distinguish between content that is required to be interpreted and content that is not intended to be analyzed by ProjectIT tools, such as textual descriptions that are only meant to be read by humans. The latter, provides scope context within a *WikiPage* through nesting, going further than simple HTML or traditional Wiki table-based structuring, when allied with the semantic information.

WikiPageItemRelation. The purpose of this concept is similar to *WikiPageRelation*, but it is used instead to create relations between smaller chunks of information. The rationale is to support the establishment of relations between content blocks in a wiki-crosswise manner. Besides the semantic information provided by the relation type, there is the possibility of constraining the *WikiPageItem*s' types of the relation edges. For now this feature is going to be postponed until further results from a proper case study.

WikiPageView. This concept explores both the structural and semantic information provided by the typified relations established among *WikiPage*s and between *WikiPageItem*s, and also the metadata provided by the dynamic attribute-based mechanism presented in the section III-B. This concept works as a virtual *WikiPage* whose content is generated on-the-fly according to a query that combines the previous data sources; thus gathering information from one or more *WikiPage*s. This concept can be compared to an on-demand report or the Mac OS X smart folders. The concept of filtering *WikiPage*s' content can be extended with the Pipes and Filters architectural style [18]. This allow us to follow a divide-and-conquer approach to define *WikiPageView*s; thus enabling query reuse trough composition and avoiding large unmanageable queries.

The purpose underlying the design of this generic Wiki metamodel is to apply its concepts within the context of the ProjectIT-Enterprise tool, in order to support the RE process. Figure 4 depicts a straightforward example of the application of this metamodel's concepts in a concrete IT project. Each project must have an associated root *WikiNamespace*, and within that scope we can have several other *WikiNamespace*s, being one of them dedicated to Requirements Engineering. Getting away from the traditional document view [2], the "requirements document" consists on the set of all *WikiPage*s under the latter *WikiNamespace*'s scope. Each *WikiPage* holds several requirements devoted to a specific component of the target system, and it can relate to other *WikiPage*s. In this scenario, textual descriptions and requirements specifications correspond to *WikiPageItem*s. Figure 4 also exemplifies the match between *WikiPage*s and subsystems. Finally, we can observe the relations among requirements and between requirements and artifacts.

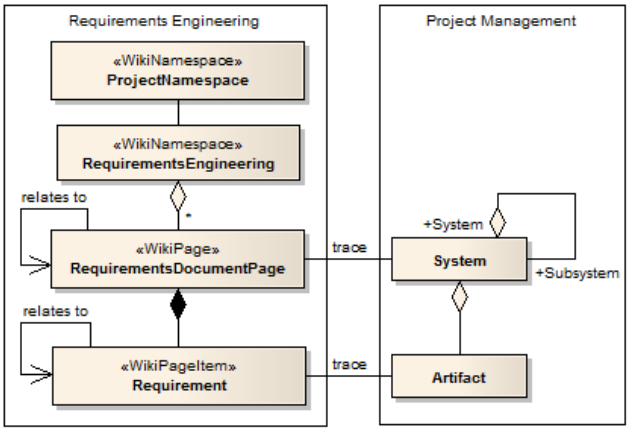

Figure 4. The Wiki metamodel applied to the RE.

B. Wiki Dynamic Attributes

The idea of adopting a dynamic attribute-based mechanism is motivated by the approach presented in [2]. In their book, Hood et al. suggest several attribute sets for each one of the software development process phases and explain how they can be interwoven with requirements in order so provide an up-to-date overview of the target system's scope, which in turn feeds all the other perspectives of project management. As an example, a possible list of attributes to support a requirement's change request is: *ID*, *Short Description*, *Owner*, *Proposer*, *Priority*, *Date of Registration*, *Status*, *Date of Decision*, *Result of Analysis*, and *Rationale*. The model of the dynamic attribute-based mechanism was adapted from an existing Document Management System, the WebC-Docs [19]. Since this system provides an identical mechanism for file system concepts, we decided not "reinventing the wheel". Hence, the design of this mechanism is based upon this system, with some slight changes. The user interaction with WebC-Docs clearly illustrates the flexibility and benefits of the adoption of a similar mechanism for a RE platform.

Attributes are essential to the enhancement of the traditional Wiki concept. They provide metadata to semantically enrich the concepts of the Wiki Metamodel. Since they can be dynamically defined, their potential is virtually unlimited. They not only provide human-readable classifiers, but can also support automatic and machine-computable tasks, such as inference, verification (with user-friendly warnings), and advanced metadata search or even content filtering. Moreover, they are essential for one of the Wiki metamodel's concept, the *WikiPageView*, which is based on an attribute-based query mechanism (explained in the subsection III-A).

The model of the dynamic attribute-based mechanism is presented in Figure 5, which also presents the relations with the Wiki metamodel and highlights its association with the *Role* concept of the WebComfort platform.

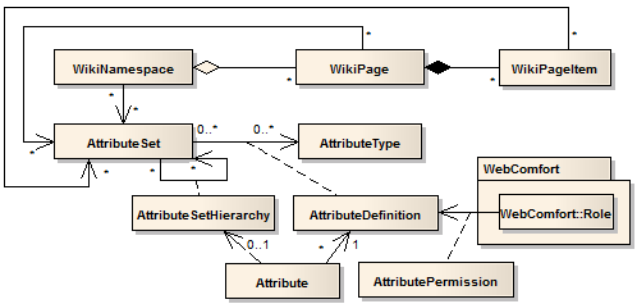

Figure 5. The dynamic attribute-based mechanism.

AttributeType. This is the most basic concept and entails the definition of the data type of the *Attribute* value. It allows one to specify the assembly (to support the addition of specific data types), the programming language type, and a user-friendly designation.

AttributeSet. This concept acts as a form-like aggregation mechanism, providing a friendly name and a purpose's description for a set of related *AttributeDefinitions*. Regarding to a *WikiNamespace* we can specify *AttributeSet*s for both *WikiPage*s and *WikiPageItem*s. These *AttributeSet*s are automatically inherited by all child *WikiNamespace*s and are applied to *WikiPage*s and *WikiPageItem*s within their scope. A *WikiPage* can define their own *AttributeSet*s, which are appended to the inherited *AttributeSet*s of its parent. Additionally, one can define *AttributeSet*s for the *WikiPageItem*s contained within the *WikiPage*, according to their type. Each *WikiPageItem* can have an associated *AttributeSet*, according to its *WikiPageItemType*. For example, a simple *WikiPageItem* of plain text *WikiPageItemType* doesn't require *Attribute*s. However, it is a good practice [2] to associate *Attribute*s to *WikiPageItem*s that contain a requirement specification (another *WikiPageItem* type), which in turn can differ according to the perspective under analysis (this can be achieved with a *WikiPage* under the scope of another *WikiNamespace*).

AttributeDefinition. Within the context of the association between the previous two concepts, the *AttributeType* and the *AttributeSet*, we introduced the *AttributeDefinition* concept, which enriches this association with the following attributes: name; description; mandatory; default value; and default value generator. The latter automatically provides a scope-related meaningful *Attribute* value.

AttributePermission. The design of this concept has some slight improvements when compared with the functionality of WebC-Docs [19]. It supports a fine-grained definition of *Attributes*' visibility and whether the *Attribute* is read-only or not. This way one can define for the same *AttributeDefinition* several configurations of *Attribute* according to the user's *Role*.This avoids the need of defining an *AttributeSet* for each possible arrangement of *Attributes*' visibility and edition state.

AttributeSetHierarchy. The rationale behind this concept is to provide the possibility of reusing *AttributeSet*s to create more complex ones or to allow multiple occurrences of the same *AttributeSet*. It consists of a name and a description for the *AttributeSet* aggregation.

Attribute. This concept corresponds to the "instantiation" of the *AttributeDefinition*. It simply holds a valid string representation of the typed value. From Figure 5 we can observe that *Attributes*' context is provided by not only by *AttributeDefinition*, but also by the association with *AttributeSetHierarchy*. This association is needed to uniquely identify the right parent in cases where there is more than one application of the same *AttributeSet*.

C. Wiki Permissions

This subsection presents the integration of the previous concepts with a role-based access control mechanism (illustrated in Figure 6). Although Wikis are simpler and more permissive regarding the access and edition of wiki pages' content, we decided to constrain this typical behavior, since the integration with the ProjectIT-Enterprise tool requires information confinement and confidentiality.

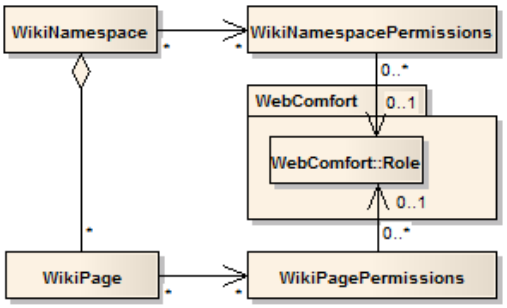

Figure 6. The fine-grained Wiki RBAC.

WikiNamespacePermissions. This concept entails all permissions regarding the *WikiNamespace* concept. The aspects that it addresses can be divided into two main groups: (1) current *WikiNamespace*'s permissions; and (2) child elements' permissions. Within each one of them we can identify concerns regarding CRUD operations, *AttributeSet*s, and permissions. Hence, for the first group we have: (1) *CanAccessWikiNamespace*, which specifies if the *Role* can access to its definitions and to its child elements of the current *WikiNamespace*; (2) *CanDelete* and *CanUpdate*; (3) *CanApplyOrRemoveAttributeSets*, which indicates if the user *Role* allows managing *AttributeSet*s of the *WikiNamespace*; (4) *CanChangeAttributeValues* and *CanViewAttributeValues*, that are important in the practical usage scenario of the dynamic attribute-based feature; and finally (5) *CanChangePermissions* to state if the user can manage permissions for that *WikiNamespace*. The second group of permissions consists on: (1) *CanCreateWikiNamespaces* and *CanDeleteWikiNamespaces*;

(2) *CanCreateWikiPages* and *CanDeleteWikiPages*; and (3) *CanChangeChildAttributeSettings*, to indicate if the *Role* can change the *AttributeSet*s of its children.

WikiPagePermissions. The rationale for this concept is to introduce a layer of permissions at *WikiPage*'s level. The facets that it entails can also be grouped together, following the same criteria used for the aspects of *WikiNamespacePermissions*. Hence, related with the concept itself we have: (1) *CanAccessWikiPage*, *CanDeleteWikiPage*, and *CanEditWikiPage* properties, which are used to support the ordinary CRUD operations on *WikiPage*s; (2) *CanApplyOrRemoveWikiPageAttributeSets*, *CanChangeAttributeWikiPageValues*, and *CanViewWikiPageAttributeValues* that are required to configure the management of *WikiPage*'s attributes; and (3) *CanChangePermissions*, which is needed to specify if the user's *Role* can modify *WikiPagePermissions*. The second group of permissions refers to *AttributeSets*' management of *WikiPage*'s child elements, *WikiPageItems*: (1) *CanApplyOrRemoveWikiPageItemAttributeSets* for management, *CanChangeAttributeWikiPageItemValues* for edition, and *CanViewWikiPageItemAttributeValues* for visibility and authorization concerns.

D. Wiki Reuse through Templates

The last feature that we propose for the enhanced Wiki-based platform is the possibility of capturing Requirements Engineering best practices through a templating mechanism. According to our experience, there should be two ways of defining a Wiki template.

The first one is straightforward, and consists on the definition of a Wiki template from scratch. The isolation and modularity provided by the base CMS platform allows one to define a Wiki as a module's instance and then populate it with the appropriate information associated with the best practices that the template should enforce. Afterwards, one can create several Wiki module instances and import the contents of a previously defined template. The next step is to start using it as a new project by populating the *WikiPage*s of each instance.

The second approach, and likely the most interesting one, consists in the reverse process. The idea is to take a Wiki of an ongoing or finished project and extract the Requirements Engineering best practices from it. With the introduction of the dynamic attribute-based mechanism (previously described), one can capture the best practices in an elegant and clear-cut manner. The idea is to take advantage of this mechanism by defining an *AttributeSet* for template-related metadata at the project's level *WikiNamespace* (the Wiki's root). This *AttributeSet* is automatically applied to all root *WikiNamespace*'s child elements. Afterwards, one can use a Wiki analyzer (similar to the *WikiPageView* concept, but in a Wiki-crosswise scope) that captures information marked with the template-related metadata to extract that content (best practices) to another empty Wiki instance.

IV. Related Work

There are academic and industrial projects that stress the benefits of using a Wiki-based approach to support the active stakeholders' participation during Requirements Engineering processes [20]. Wikis have the potential to leverage stakeholders' collaboration. Moreover, their lightweight edition workflow, allied with the possibility of using them as central documentation repositories, make them easier to use and tailor than proprietary requirements management tools [20].

Towards content's structure, the work presented by Iorio et al. [21] promotes the concept of light constraint to directly encode community best practices and peculiar issues of domain-specific content. Although the concept *per se* seems to contradict the Wiki philosophy, it is useful to enforce well-formedness conditions on specific page contents. The Wiki-based system presented in this paper reflects our alignment with the idea that some constrains should be employed to improve the quality of the requirements development's deliverables.

Regarding the capture of best-practices through templates, Decker et al. [20] propose a basic document structure for Wiki-based Requirements Engineering based on well-known and adopted *use case* templates. They introduce Wikis as general-purpose and flexible documentation platforms, with the ability to be extended in order to capture additional information. According to their perspective, Wikis can be adapted to cover most of the organization-specific needs of an IT project. Their work strengths our belief that Wikis are the most adequate solution for supporting an online collaborative Requirements Engineering approach.

Still about templates, Haake et al. [22] advocate that the tasks of Wiki templates creation and reuse should be considered as a detached process from Wiki's content editing. The rationale is to make structure more conscious and disallowing accidental structural changes. Moreover, they advocate that this detachment improves the source's readability of the Wiki's content, since structural elements do not unintentionally disturb the content editing process with their markups. The templates design is comparable to the process of identifying classes and attributes in object-oriented analysis. However, their Wiki markup for the design of templates is rather complex and it doesn't support nesting of content blocks. Additionally, their *attribute* concept is different from ours, since they represent inputs to be used within a form-based wiki page.

With respect to the benefits of using NLP techniques, Witte et al. [23] introduce an innovative vision of a "self-aware" Wiki, capable of reading, understanding, transforming, and writing its own content. Their goal is to go beyond purely syntactic approaches by bringing the full potential of current natural language technologies to the Wiki platform. Although a completely automated understanding of natural language is still not feasible, there already exist several robust language processing techniques that can strongly improve the user's experience. Witte's work consists on a complementary approach to

Semantic Web: automatically obtaining semantic metadata by applying NLP techniques.

Besides these research projects, there are renowned and full-fledged commercial tools, such as IBM's Rational RequisitePro [24] and DOORS [25], and also Borland's CaliberRM [26]. These tools offer deeply integrated solutions with a strong emphasis on traceability issues and the integration with development and test suites of third parties [27]. Albeit these commercial suites support distributed teams to allow a broader stakeholder's participation, they use specific tools that act as a view of their requirements database (e.g., DOORS Web Access [28]), thus the core of the majority of these tools is still strongly desktop-oriented [29]. However, and according to the trend mentioned in [30], we are assisting to an increasing number of exclusively web-based tools, such as GatherSpace [31], eRequirements [32], and Polarion's Requirements 2.0 [33]. These online tools have a strong focus on project and virtual teams management, but they basically mimic the desktop-based tools with rich Web 2.0 technology. Polarion's Requirements 2.0 innovates by using a built-in Wiki to gather and share requirements [33]. However, this Wiki is solely used as a source of free-form text that can be conveyed into its requirements database, like the textual content of a Word document. The Wiki content can be backtracked throughout the software product's lifecycle through links. Nevertheless, whatever the technology used, the problem remains the same: these tools treat requirements as black-box elements, neglecting its natural language semantics in favor of enhanced traceability. This clearly is in contrast with our approach, which seeks to capture each requirement meaning and verify requirement models accordingly.

Our proposal appears as a natural sequence of our previous work and its feasibility is corroborated by the aforementioned projects. Moreover, the proposed enhanced Wiki system design seeks to comply with Wiki design principles [34] and to address most of Wiki problems [13]. As Whitehead [30] points out, the future directions for collaboration in Software Engineering include the tight integration between web- and desktop-based development environments, fostering the participation of not only domain experts, but also engaging customers and end users during the entire software development process.

V. Conclusions

This paper presents the design of an enhanced Wiki system to support the RE's main processes [1]. We still introduce a conceptualization for the core concepts of Wikis. The goal of this metamodel is to integrate the Wiki system with the ProjectIT tools; namely to provide a collaborative environment to support requirements specification with a controlled natural language, and also to manage requirements in the broader scope of project management. The control activities of project management will not grow to their full strength if they do not make use of RE's information [2]. Additionally, we propose a dynamic attribute-based mechanism to capture metadata, and also its integration with a fine-grained RBAC mechanism. Furthermore, with the flexibility the dynamic attribute-based mechanism, we still propose an elegant approach to define Wiki templates, or even to extract them from ongoing or finished projects. Related work sustains the feasibility and benefits of our proposal, namely Semantic Wikis and Wikis embedded with NLP techniques [23].

As future work we intend to evaluate this proposal's achievements through the participation of third party entities, namely we plan to conduct academic and industrial case studies, with computer science students and software houses, respectively. With the first iteration, within the academic context, we will assess the expressive power of the controlled natural language and the Wiki-based tool's suitability and usability, according to an end-user perspective. During this stage, we expect to encounter the coarse-grain drawbacks of this approach and of its tool's support, regarding both the RE's processes. We want to evaluate to which extent the text-based approach is sufficient and how users will circumvent potential limitations that they might encounter. Another important aspect to assess during these evaluations relates with training time for a user to become proficient in using this approach and the respective tool: we need to check if users are comfortable with the tool's interface; if the controlled natural language and tool's support are intuitive enough; and if, despite being flexible, the dynamic attribute-based mechanism and the permissions associated with it are too complex to grasp. The second iteration, within an industrial context, will provide the feedback regarding the tool's adequacy to real-life projects, namely it will allow us to identifying more linguistic constructs, requirements document templates, and to fine-tune the supported RE's processes. Moreover, this assessment will address the tool's scalability issues, due to the complexity and size of modern information systems. Within this real-life scenario, and regarding social and collaboration issues, we intend to identify the criteria that ensure that a Wiki fits into the organizational culture, which are required to foster and support active online communities. Furthermore, this assessment will provide useful information about to which extend does the up-to-date requirements information can benefit the stakeholders' awareness of the project status and support a reliable decision-making process.

References

[1] K. Wiegers, *Software Requirements*, 2nd ed. Microsoft Press, March 2003, ISBN: 978-0735618794.

[2] C. Hood, S. Wiedemann, S. Fichtinger, and U. Pautz, *Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes*, 1st ed. Springer, December 2007, ISBN: 978-3540476894.

[3] T. Bell and T. Thayer, "Software requirements: Are they really a problem?" in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 1976, pp. 61–68.

[4] C. Videira, D. Ferreira, and A. Silva, "A Linguistic Patterns Approach for Requirements Specification," in *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 302–309, ISBN: 0-7695-2594-6.

[5] A. Silva, "O Programa de Investigação ProjectIT (whitepaper)," October 2004.

[6] D. L. Duarte and N. T. Snyder, *Mastering Virtual Teams*. Jossey-Bass Inc., Publishers, 2000.

[7] M. Notari, "How to use a Wiki in education: Wiki based effective constructive learning," in *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*. New York, NY, USA: ACM, 2006, pp. 131–132.

[8] W. Cunningham, "Wiki Design Principles," March 2008, Retrieved Wednesday 25th March, 2009 from http://c2.com/cgi/wiki?WikiDesignPrinciples.

[9] A. Ebersbach, M. Glaser, and R. Heigl, *Wiki : Web Collaboration*. Springer, November 2005.

[10] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandecic, "The two cultures: mashing up web 2.0 and the semantic web," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 825–834.

[11] P. Martins, "ProPAM - a Software Process Improvement Approach based on Process and Project Alignment," Ph.D. dissertation, Instituto Superior Técnico (IST/UTL), Lisbon, Portugal, September 2008.

[12] A. Silva, J. Saraiva, D. Ferreira, R. Silva, and C. Videira, "Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools," *IET Software Journal*, vol. 1, no. 6, pp. 217–314, December 2007.

[13] S. Murugesan, "Understanding Web 2.0," *IT Professional*, vol. 9, no. 4, pp. 34–41, 2007.

[14] CosmoCode, "WikiMatrix: compare them all," Retrieved Wednesday 25th March, 2009 from http://www.wikimatrix.org/.

[15] SIQuant, "WebComfort.org," Retrieved Wednesday 25th March, 2009 from http://www.webcomfort.org.

[16] J. Saraiva and A. Silva, "The WebComfort Framework: An Extensible Platform for the Development of Web Applications," in *Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2008)*. IEEE Computer Society, September 2008, pp. 19–26, 978-0-7695-3276-9.

[17] P. Thoeny, "TWiki - the Open Source Wiki for the Enterprise," Retrieved Wednesday 25th March, 2009 from http://twiki.org/.

[18] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, 1st ed. Addison-Wesley Professional, October 2003, ISBN: 978-0321200686.

[19] SIQuant, "WebComfort.org - WebC-Docs," Retrieved Wednesday 25th March, 2009 from http://www.webcomfort.org/WebCDocs.

[20] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-Based Stakeholder Participation in Requirements Engineering," *IEEE Software*, vol. 24, no. 2, pp. 28–35, 2007.

[21] A. D. Iorio and S. Zacchiroli, "Constrained Wiki: an Oxymoron?" in *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*. New York, NY, USA: ACM, August 2006, pp. 89–98.

[22] A. Haake, S. Lukosch, and T. Schümmer, "Wiki-templates: adding structure support to wikis on demand," in *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*. New York, NY, USA: ACM, October 2005, pp. 41–51, ISBN: 1-59593-111-2.

[23] R. Witte and T. Gitzinger, "Connecting wikis and natural language processing systems," in *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*. New York, NY, USA: ACM, 2007, pp. 165–176.

[24] IBM, "Rational RequisitePro," Retrieved Monday 8th June, 2009 from http://www-01.ibm.com/software/awdtools/reqpro.

[25] ——, "DOORS," Retrieved Tuesday 9th June, 2009 from http://www.telelogic.com/products/doors.

[26] Borland, "CaliberRM - Enterprise Software Requirements Management System," Retrieved Monday 8th June, 2009 from http://www.borland.com/us/products/caliber/index.html.

[27] A. S. Guild, "Volere - Requirements Tools," Retrieved Monday 8th June, 2009 from http://www.volere.co.uk/tools.htm.

[28] IBM, "DOORS Web Access," Retrieved Tuesday 9th June, 2009 from http://www.telelogic.com/products/doors/doorswebaccess.

[29] INCOSE, "INCOSE Requirements Management Tools Survey," Retrieved Monday 8th June, 2009 from http://www.incose.org/ProductsPubs/products/rmsurvey.aspx.

[30] J. Whitehead, "Collaboration in Software Engineering: A Roadmap," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 214–225.

[31] GatherSpace, "Agile Project Management, Requirements Management - GatherSpace," Retrieved Monday 8th June, 2009 from http://www.gatherspace.com.

[32] S. Robinson, "eRequirements," Retrieved Monday 8th June, 2009 from http://www.erequirements.com.

[33] Polarion, "Polarion Requirements 2.0 Features," Retrieved Wednesday 10th June, 2009 from http://www.polarion.com/products/requirements/features.php.

[34] A. V. Deursen and E. Visser, "The Reengineering Wiki," in *CSMR '02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2002, p. 217.