Semantic Documentation in Requirements Engineering

Ricardo de Almeida Falbo, Carlos Eduardo C. Braga, Bruno Nandolpho Machado

Ontology and Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo, Brazil
falbo@inf.ufes.br, {caducbraga, brunonandolpho}@gmail.com

Abstract. Currently, most requirements documents are prepared using desktop text editors. These documents are intended to be used by human readers. In this paper, we discuss the use of semantic annotations in requirements documents, in order to make information regarding links between requirements and other software artifacts, such as other requirements, use cases, classes and test cases, interpretable by computers. To do that, we extend a semantic document management platform to the requirements domain, and explore the conceptualization established by the Software Requirements Reference Ontology in order to provide features to support some activities of the Requirement Engineering Process, namely: prioritizing requirements, analyzing impacts from requirements changes, tracing requirements through traceability matrices, and verifying requirements using checklists.

Keywords: Requirements engineering, requirements documentation, semantic documentation, semantic annotation, semantic document, ontologies.

1 Introduction

Documents are the dominating format for knowledge communication and storage, and they tend to continue to be for the foreseeable future [1]. Documents provide a rich resource that describes what organizations know about their processes, application domains and products. According Uren et al. [2], documents account for 80-85% of the information stored by many companies.

Document repositories generally are large, and the demand for efficient search and retrieval techniques increases insofar the amount of documents increases. Efficient document management is an important part of the knowledge management strategy in an organization, mainly in the case of software development. However, most knowledge in electronic documents is available only for human interpretation and not for computer systems. This makes tasks related to indexing and retrieving knowledge items complex.

In the Semantic Web community, researchers advocate that ontology-based metadata can be added into the web content so that this content becomes available for machine interpretation [3, 4]. By adding ontology-based annotations into web contents, it is possible to enrich these contents, and to allow semantic computing of them [3]. The act of adding ontology-based metadata into syntactic information resources making them semantic information resources is named semantic annotation. Ontologies play an important role in semantic annotation, because they provide a basis for capturing the intended meaning of a portion of shared knowledge.

The semantic annotation approach can also be used in documents generated by desktop tools, such as Open Office Writer or Microsoft Word. This is an important finding because, despite the advances on electronic documentation, desktop text editors are still the most used solution for documenting software development [5, 6].

Several tools have been developed intending to support semantic annotation. The Infrastructure for Managing Semantic Documents (IMSD) [7], AktiveDoc [8], PDFTab [9], SemanticWord [10] and KIM [11] are some examples. These tools use domain ontologies for semantically annotating documents, and provides a set of general features for managing semantic documents, such as features for annotating, storing, indexing and retrieving documents.

Despite these tools can be used in different domains, only general features are provided, not taking advantage of the particularities of the domain conceptualization. However, in order to provide a more effective support to domain tasks, we advocate that it is useful to explore the specific conceptualization provided by the ontology. In other words, it is useful to explore the ontology elements (concepts, relations and properties), and use them to develop domain-specific functionalities. We achieve this conclusion by applying IMSD in the software requirements domain [12]. As a consequence of this study, we glimpsed an opportunity to extend IMSD to provide specific features supporting the Requirements Engineering (RE) process.

Requirements documents register and formalize the results of the RE process, being the basis for several subsequent activities in the software process, such as design, coding, testing, and maintenance [13, 14]. Thus, we decided to extend IMSD, introducing requirements-specific features, giving rise to IMSD-Req. Using the facilities provided by IMSD, IMSD-Req introduces new features for: (i) supporting requirements change impact analysis; (ii) evaluating consistency of requirement priorities, (iii) generating traceability matrices; and (iv) verifying requirements using checklists.

In this paper we present IMSD-Req, and discuss how it supports requirements prioritization, requirements management and requirements verification. The remainder of this paper is organized as follows. Section 2 regards the theoretical background of the paper, discussing briefly Semantic Documentation, ISDM, and its initial use to support the RE process. Section 3 presents ISDM-Req and its features. Section 4 discusses related works. Finally, Section 5 presents our conclusions and future work.

2 Semantic Documentation

In the context of the Semantic Web, in order to make Web content available to computer systems, metadata are added to Web information resources [3]. In general, metadata concern “data about data”. Metadata are used as a mechanism for expressing the semantics of information, in order to facilitate information seeking, retrieval, understanding, and use [4]. Semantic annotation is about assigning to the entities in the text links the corresponding semantic descriptions using metadata [15].

Since the notion of semantic annotation has arisen in the Semantic Web community, usually it is associated with the act of annotating Web resources with metadata. However, its applicability goes beyond the Web Semantic context and can also be applied to documents generated using desktop text editors, like MS Word or Open Office Writer [2, 7, 16]. By adding semantic annotations to desktop documents, we can reach "intelligent" documents. In sum, a semantic document knows about its own content so that automated processes can know what to do with it [2].

Both in Semantic Web and in Semantic Documents, we need shared representations of knowledge to establish the basic vocabulary from which metadata statements can be asserted [4]. Since ontologies aim at capturing the intended meaning of a portion of shared knowledge, many researchers defend their use to semantically annotate information resources (web pages or desktop documents) [2, 3, 4, 9]. Semantic documents aim at combining documents and ontologies, and allowing users to access their knowledge in multiple ways. The ultimate goal of semantic documents is not merely to provide metadata for documents, but to integrate documentation and knowledge representation in a way that they use a common structure [9]. Based on domain ontologies, semantic annotations can be added to documents. Once annotated, we can extract knowledge and link contents from different documents according to the shared ontology. Merging the content extracted from several documents, we are able to achieve a more holistic view of the knowledge available in the organization [7].

However, the successful use of semantic documents strongly depends on the cost and effort dispended to manipulate them, since document authors cannot daily suffer with the overhead associated with annotating them. Thus, it is necessary to provide tools to reduce the effort needed to create the document contents and to semantically annotate them [2, 10]. Several tools have been developed intending to support semantic document management [7,8,9,10,11]. As discussed in the Introduction of this paper, this work uses the Infrastructure for Managing Semantic Document (IMSD) [7].

IMSD [7] supports creating document templates with semantic annotations. It uses Subversion, a version control system, to store the semantic documents and the content extracted from them (in RDF format). In addition to the Semantic Document Repository, IMSD has three main modules [7]:

- Semantic Annotation Module (SAM): responsible for allowing users to semantically enrich document templates written in the ODF format (Open Office). The annotations should be based on ontologies codified in OWL. The documents instantiated from an annotated template keep the annotations inserted in the last.
- Data Extraction and Versioning Module (DEVM): responsible for extracting the content from a semantic document whenever a new version of it is checked in the Semantic Document Repository (a Subversion repository). After extraction, the semantic content of the new version is stored in OWL in another repository, the so called Data Repository, which is also part of DEVM.
- Search and Traceability Interface Module (STIM): responsible for providing features that allows performing ontology-based queries in SPARQL in the Data Repository of DEVM.

IMSD was initially applied for supporting the Requirements Engineering (RE) process of seven projects developed in NEMO (Ontology and Conceptual Modeling Research Group) [12]. NEMO follows a classical RE process, including activities for requirements elicitation, analysis (conceptual modeling), documentation, verification and validation. In parallel, activities of requirements management are accomplished. Two types of documents are used for documenting requirements: Requirements Document and Requirements Specification. The first is directed to clients and users, and captures user requirements. It is written in natural language, following rules defined for writing requirements statements. The second details the user requirements into system requirements, and serves as basis for further development. It is mainly composed by models (use case diagrams, class diagrams, state diagrams, among others), although there are also textual parts, especially the ones related to use case and class descriptions [12].

A striking feature of IMSD for supporting the RE process is the fact that annotations are added into document templates that, when instantiated, give rise to semantic documents. This is an important feature for the RE process, because the use of templates is pointed out as a practice that contributes to the success of a software project [13, 14, 17]. The use of templates is fundamental to ensure homogeneity and to avoid omission of relevant information related to requirements [14, 18]. Moreover, once annotated the document templates, the corresponding requirements documents prepared using these templates are also annotated, diluting the efforts spent in the annotation process among the various documents that are prepared using them.

Although IMSD was considered useful, some improvement opportunities were identified by the developers that used it, especially concerning some RE activities. These improvement opportunities motivated us to extend IMSD by developing requirements-specific features, giving rise to IMSD-Req.

3 Extending IMSD to support the RE Process

From the preliminary evaluation of IMSD in the RE domain [12], we identified, among others, the following improvement opportunities:

IO1. As the amount of documents and requirements increase, it becomes more difficult to analyze the impact of a requirement change using the general functionalities provided by IMSD. Even though IMSD has information available to help this analysis, it is not easy to use them, since the general features provided are not properly adjusted for this purpose, requiring making several SPARQL queries to the Data Repository.

IO2. IMSD has enough information for generating traceability matrices, but it was not possible to automatically generate them. Traceability matrices should be manually produced by the requirements engineer, using as basis the answers to several SPARQL queries. The requirements engineer has to define the SPARQL queries in the Search and Traceability Interface Module (STIM), submits them to Data Repository, and gets the answers, in a tiresome and error-prone process.

IO3. In several projects, requirements engineers have had difficulties in prioritizing requirements. Requirements depend on others and, many times, those relations are not properly considered during priority establishment, leading to inconsistencies. For example, it was common to find high priority requirements depending on low priority requirements.

IO4. Again, to perform requirements verification, it was necessary to elaborate complex SPARQL queries via STIM. Moreover, IMSD did not store the SPARQL queries, requiring the requirements engineer to enter the questions via STIM every time he/she wants to verify the requirements.

Based on the improvement opportunities identified, IMSD was extended to provide requirements-specific features, giving rise to IMSD-Req. Our premise is that IMSD-Req can better support the RE process by exploring the conceptualization provided by the Software Requirements Reference Ontology (SRRO). Following, we present the fragment of SRRO that is used in this paper.

3.1 Software Requirements Reference Ontology

Figure 1 shows a fragment of the conceptual model of the current version of SRRO written in OntoUML [19], a UML profile that enables modelers to make finer-grained modeling distinctions between different types of classes and relations according to some ontological distinctions put forth by the Unified Foundational Ontology [19]. This version extends the version presented in [20] by including some properties of requirements, and integrating this ontology with the Reference Ontology on Software Testing (ROoST) [21].

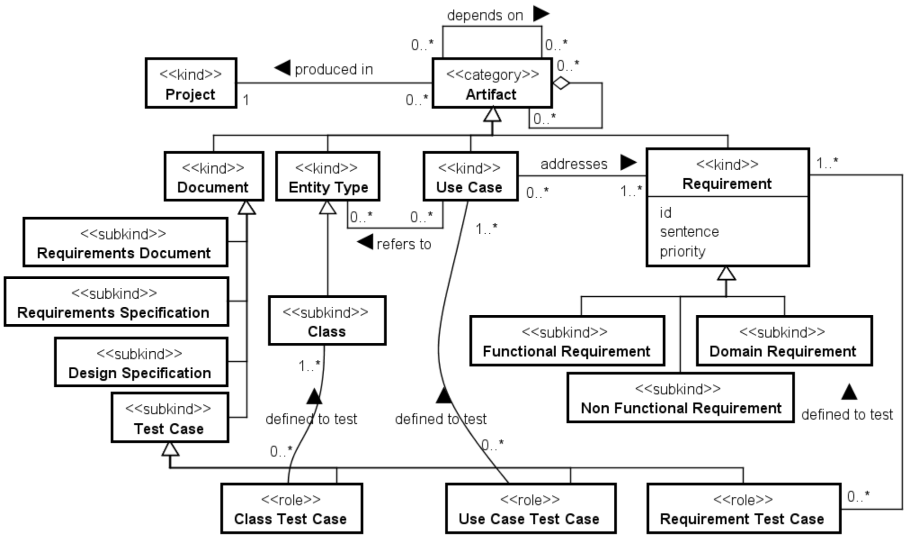


Fig. 1. A fragment of the Software Requirements Ontology

As shown in Figure 1, requirements, use cases, classes and documents are artifacts produced in the context of a software project. Artifacts can be composed by other artifacts. A Requirements Document (or Requirements Definition Document) contains a list of requirements written in a way that is easy for customers to understand [22], and thus the following axiom holds: if an artifact *a* is part of a requirements document *rd*, then *a* should be a requirement: $\forall$ *a: Artifact, rd: RequirementsDocument (partOf(a,rd)* $\rightarrow$ *requirement(a))*. A Requirements Specification, in turn, restates the requirements definition in technical terms appropriate for developers [22]. It typically contains several diagrams, including use case and class diagrams, as well as descriptions of use cases and entity types. Other sub-kinds of documents considered in this work are Design Specifications and Test Cases.

Requirements are features of the system or descriptions of things that the system should be capable of doing in order to fulfill its purpose [22]. Usually, requirements are classified in three main categories: functional, non functional and domain requirements [13, 14, 18, 22]. Functional requirements are statements of services or functions that the system must provide, describing how the system must behave given certain stimuli [18, 22]. Non-functional requirements describe constraints on the functions offered by the system, and overall properties that the system must present [13, 18], and that limit the options for designing a solution to the problem [22]. Finally, domain requirements, also referred as business rules, are derived from the application domain or from the business being supported by the system. They reflect domain or business characteristics and constraints [18].

Requirements must be uniquely identified in order to be referred and traced during requirements management. They are typically written as sentences in structured natural language. Moreover, requirements must be prioritized, in order to establish their relative importance when compared to other requirements [13, 14, 18]. Concerning priority establishment, the following axiom holds: if a requirement *r2* depends on another requirement *r1*, with priority *p1*, then the priority *p2* of *r2* should be less than or equal to *p1*: $\forall$ *(r1, r2: Requirement) (priority(p1,r1)* $\wedge$ *priority(p2,r2)*$\wedge$ *dependsOn(r2, r1)* $\rightarrow$ *p1 >= p2).*

A use case describes a set of actions performed by the system (or by means of interacting with it) that yields an observable result that is typically of value for one or more actors of the system [23]. Use cases are typically defined to address functional requirements. However, they also take domain and non-functional requirements that apply to a specific functionality into account.

When the structure of the system is being conceptually modeled, entity types are represented in a conceptual schema in order to describe domain concepts [23]. Entity types are referred in use cases, since use cases deal with these concepts. Class is a sub-kind of entity type.

Finally, test cases are defined to test the system or small parts of it, including requirements, use cases and classes. A test case aims at testing a portion of code (code to be tested), and specifies the test case inputs and the expected result (see [21] for details concerning the testing domain).

Artifacts can depend on other artifacts. More specifically requirement can depend on other requirements. Use cases depend on the requirements that they address: $\forall$ *(r:*

Requirement, uc: UseCase) (addresses(uc,r) $\rightarrow$ *dependsOn(uc,r)).* And requirement test cases depend on the requirements for which they are defined to test: $\forall$ *(r: Requirement, rtc: RequirementTestCase) (definedToTest(rtc,r)* $\rightarrow$ *dependsOn(rtc,r)).*

The "depends-on" relation is transitive: $\forall$ *(a1, a2, a3: Artifact) (dependsOn(a3,a2)* $\wedge$ *dependsOn(a2,a1)* $\rightarrow$ *dependsOn(a3,a1)).* The "required-by" relation is the inverse of "depends-on": $\forall$ *(a1, a2: Artifact) (dependsOn(a2,a1)* $\rightarrow$ *requiredBy(a1,a2)).* Moreover, if an artifact is composed by other artifacts, then it also depends on them: $\forall$ *(a1, a2: Artifact) (partOf(a2,a1)* $\rightarrow$ *dependsOn(a1,a2)).* For instance, if a requirement *r* is part of the requirements document *rd*, then *rd* depends on *r*.

Considering the transitivity property of the "depends on" relationship, we can establish indirectly dependency relationships between requirements and other artifacts. Since classes and use case test cases depends on the use cases that they refer to, they also depends on the corresponding requirements. This also applies to class test cases, which depends on the classes to which they are defined to test.

SRRO is a reference domain ontology, i.e. a domain ontology constructed with the goal of making a clear and precise description of domain entities for the purposes of communication, learning and problem-solving [24]. A reference ontology is a special kind of conceptual model, representing a consensual model within a community. From the conceptual model of SRRO shown in Figure 1, we designed and coded an operational ontology in OWL, which is in fact used to semantically annotate requirements documents in IMSD-Req.

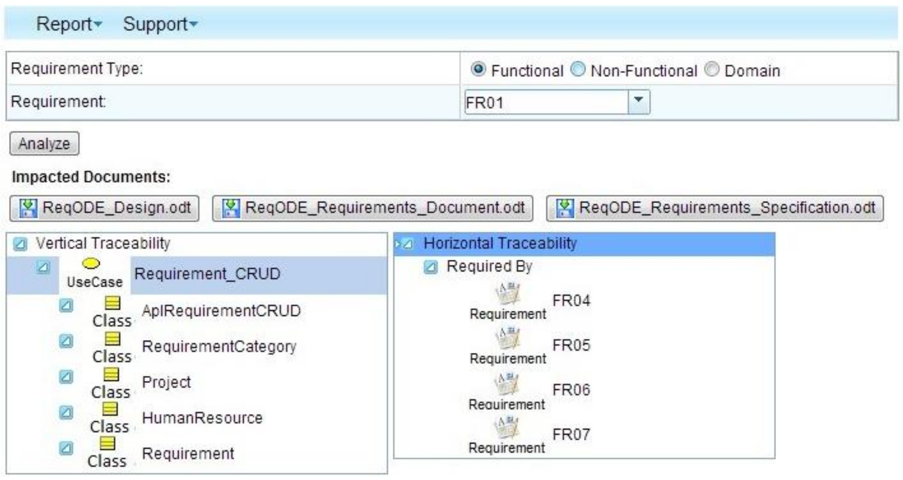
3.2 Domain-specific features of IMSD-Req

In order to address the improvement opportunities (IOs) pointed out in the beginning of this section, ISDM-Req relies on the conceptualization provided by SRRO. Templates for the four sub-types of Document shown in Figure 1 were developed. These templates were semantically annotated with SRRO concepts, relations and properties.

Concerning IO1 (analyzing impacts from changes in requirements), ISDM-Req provides a functionality to support visualizing the impact from a change in a given requirement, based on the "depends on" relationship and the axioms related to it, described in the previous subsection. Figure 2 illustrates this feature.

Based on the requirement id informed by the user, a list of the documents that are impacted by a change in the corresponding requirement is presented. Moreover, two change impact analysis trees are presented: the one on the left concerns with vertical traceability, while the other, on the right, regards horizontal traceability. In the vertical impact analysis tree, artifacts that are directly and indirectly related to the requirement are presented. Directly related artifacts are: use cases and requirement test cases. Indirectly related artifacts are: classes, class test cases and use case test cases. As described by SRRO, classes and use case test cases are related to use cases, which in turn are related to requirements; and class test cases are related to classes.

To address IO2 (automatically generating traceability matrices), we again explored the "depends on" relationship and related axioms to generate several types of traceability matrixes. Requirements x Requirements traceability matrices can be generated for allowing horizontal traceability. Vertical traceability is contemplated by three types of traceability matrices: Requirements x Use Cases, Requirements x Classes, and Requirements x Test Cases matrices. Figure 3 shows a Requirements x Use Cases traceability matrix.



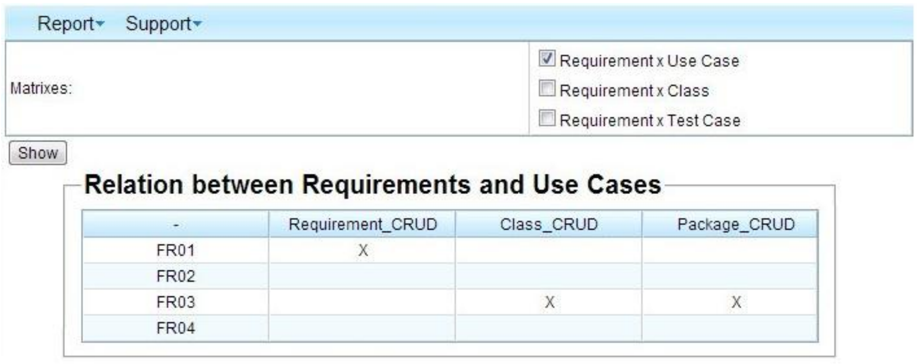
Requirement Type:	[● Functional] ○ Non-Functional ○ Domain
Requirement:	FR01

Impacted Documents:
ReqODE_Design.odt
ReqODE_Requirements_Document.odt
ReqODE_Requirements_Specification.odt

Vertical Traceability
- UseCase Requirement_CRUD
  - Class AplRequirementCRUD
  - Class RequirementCategory
  - Class Project
  - Class HumanResource
  - Class Requirement

Horizontal Traceability
- Required By
  - Requirement FR04
  - Requirement FR05
  - Requirement FR06
  - Requirement FR07

Fig. 2. Supporting Impact Analysis of a Requirement Change



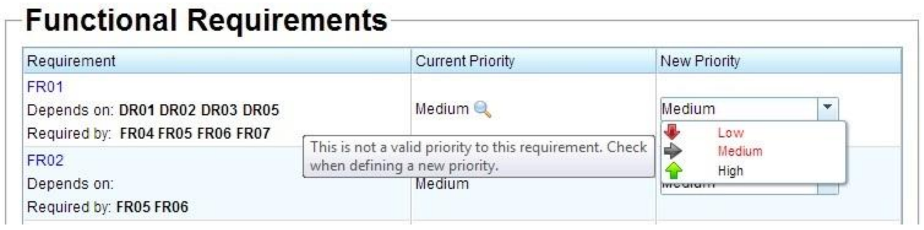
Matrixes: [✓ Requirement x Use Case] Requirement x Class Requirement x Test Case

Relation between Requirements and Use Cases

-	Requirement_CRUD	Class_CRUD	Package_CRUD
FR01	X		
FR02			
FR03		X	X
FR04			

Fig. 3. Requirements x Use Cases Traceability Matrix

To address IO3 (requirements prioritization), we developed a functionality that allows user prioritize requirements taking the priorities of dependent requirements into account. The SRRO axiom constraining how priorities can be established on requirements were enforced. Figure 4 illustrates this functionality. In this example, when the requirements engineer is establishing the priority of the functional requirement FR01, IMSD-Req retrieves the requirements that depend on it (FR01 is required by FR04, FR05, FR06 and FR07), as well as their priorities. In this case, the requirements engineer established a Medium priority to FR01, but one of the four dependent requirements has a High priority. Then a message is presented, and the invalid priority values for this requirements are shown in red (Low and Medium), warning him/her that only a High priority is valid..



Functional Requirements

Requirement	Current Priority	New Priority
FR01 Depends on: **DR01 DR02 DR03 DR05** Required by: **FR04 FR05 FR06 FR07**	Medium	Medium
FR02 Depends on: Required by: **FR05 FR06**	Medium	Medium

This is not a valid priority to this requirement. Check when defining a new priority.

Fig. 4. Supporting Requirements Prioritization

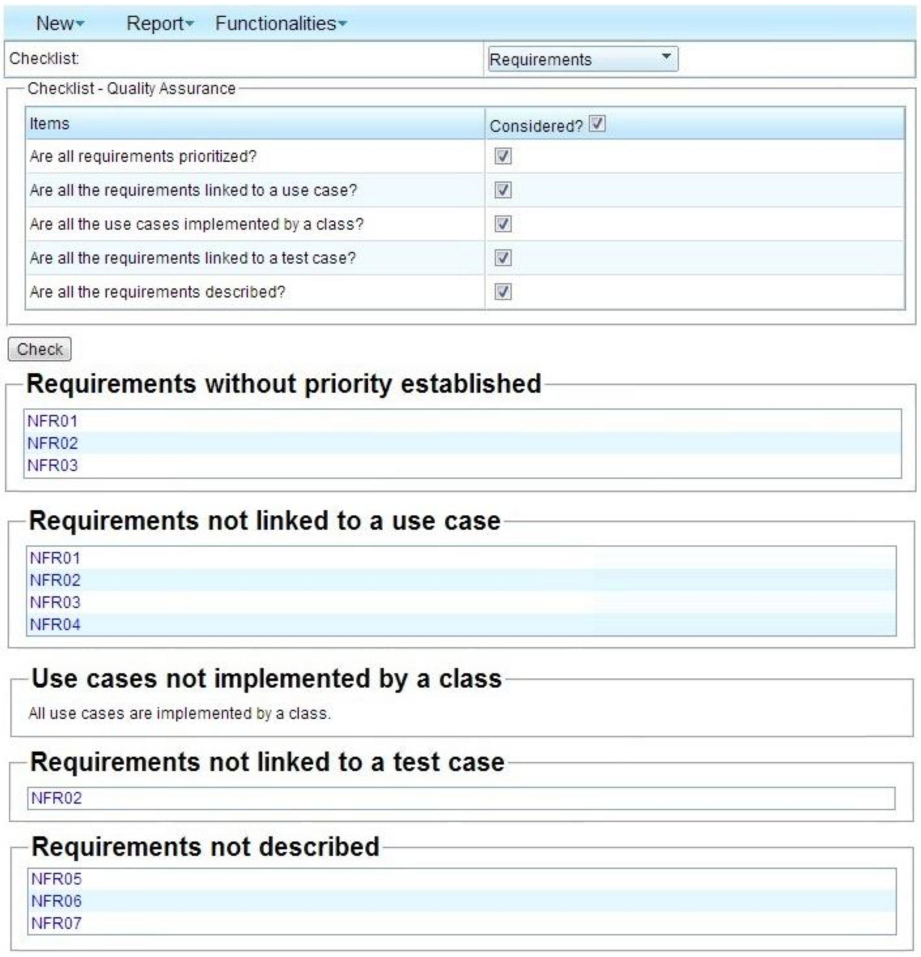
Finally, for treating IO4 (support requirements verification), we developed a functionality for grouping and storing the SPARQL queries to be used in checklists. First, the queries should be defined to be used in the checklists. Once defined the queries, these questions can be used to compose several checklists. Thus, we developed features for both creating the checklist items (the queries) and grouping them in checklists. Figure 5 shows a checklist for verifying Requirements Definition Documents.

When this checklist is applied, if there is no problem in a specific item (e.g., all use cases are implemented by a class), the result is just an informative message. On the other hand, when there are requirements with problems (e.g., requirements without priority established), the corresponding ids are listed.

4 Related Work

As discussed in the Introduction of this paper, many works, such as [7, 8, 9, 10, 11], have been done intending to support the semantic annotation process, providing also general functionalities to retrieve, index and store semantic desktop documents. However, at the best of our knowledge, none of them, except IMSD [7], was applied to support Requirements Engineering (RE). For a general comparison between IMSD and some of these other tools, see [7].

Regarding semantic documentation in RE, most works uses semantic wikis. In general, lots of researchers committed themselves to enhancing wiki capabilities to support RE [25]. In particular, semantic wiki has already proven to be a useful platform for RE [26]. Thus, in this section, we briefly discuss some works that use semantic wikis for supporting RE, namely: WikiReq [27], SOP Wiki [28], ReqWiki [29], and SoftWiki [30]. We should highlight that the first three (WikiReq, SOP Wiki and ReqWiki) are based on the Semantic Mediawiki (SMW) platform [31], and use an extension of it called Semantic Forms, that allows for the creation of forms to add and edit pages that contain templates that themselves hold semantic data. The last is based on OntoWiki [32]. Thus, all of the RE-specific semantic wikis extend general purpose semantic wikis for supporting RE.



Checklist: Requirements

Checklist - Quality Assurance

Items	Considered? ✓
Are all requirements prioritized?	✓
Are all the requirements linked to a use case?	✓
Are all the use cases implemented by a class?	✓
Are all the requirements linked to a test case?	✓
Are all the requirements described?	✓

Check

Requirements without priority established

NFR01
NFR02
NFR03

Requirements not linked to a use case

NFR01
NFR02
NFR03
NFR04

Use cases not implemented by a class

All use cases are implemented by a class.

Requirements not linked to a test case

NFR02

Requirements not described

NFR05
NFR06
NFR07

Fig. 5. Using Checklists to Support Requirements Verification

WikiReq [27] has been developed in order to support stakeholders in acquiring requirements in terms of the concepts defined by Si*, a goal-oriented language. These concepts are: actors, goals, sub-goals, tasks and resources. First, requirements are acquired by means of a set of pre-defined forms, built using the SMW Semantic Forms. Using the annotations incorporated in the Semantic Forms, requirements can be automatically transformed into graphs that can be used by the developers as a basic input in order to define a formalized version of a requirements specification. Another feature of WikiReq is to allow debating about requirements in a specific tab page.

In SOP Wiki [28], users can add properties to pages and define typed links between them. A striking feature of SOP Wiki is the ability to export wiki content (e.g., requirements) to Open Office documents.

ReqWiki [29] extends the SMW platform by means of an ontology and Natural Language Processing (NLP) services. ReqWiki is targeted for use case-driven requirements engineering. Thus, its ontology includes concepts such as actors, goals, use cases, test cases, features, as well as the relationships between them. Semantic Forms are used to add content to the wiki, automatically annotating requirements specifications with elements of the ontology. Using the semantic metadata, in-line queries can also be inserted in wiki pages. The NLP services include services for: (i) finding spelling and grammatical mistakes; (ii) detecting SRS defects, such as weak phrases in a requirement description; (iii) creating a back-of-the-book style index of the wiki content and storing it in the wiki as a page; (iv) automatically extracting named entities, such as persons, organizations or locations.

SoftWiki [30] extends OntoWiki to support requirements engineering according to an ontology for requirements engineering, called SWORE. SWORE defines core concepts of requirement engineering, and the way they are interrelated. For instance, SWORE defines types of relationships between requirements such as *details*, *conflicts*, and *depends on*. Each requirement gets its own URI making it a unique instance on the semantic web. Then, it is linked to other resources using semantic web standards such as RDF and OWL.

IMSD-Req shares several characteristics of the aforementioned wikis. First, IMSD provides a feature for semantically annotating document templates that are similar to Semantic Forms in the SMW platform, in the sense that efforts for annotating a Semantic Form in SMW or a template in IMSD are taken only once. Then, several semantic documents can be produced, as the form/template is filled.

ISMD-Req, ReqWiki and SoftWiki use ontologies for the requirements domain to semantically annotate documents. The underlying ontologies include several common concepts and relations, such as requirement, document, use case, and dependency relationships. Moreover, ontology-based reasoning facilities are available both in ISMD-Req and ReqWiki.

On the other hand, ISMD-Req has two striking features when compared to the aforementioned wikis. First, it deals with desktop semantic documents, instead of semantic wikis. Both desktop documents and wikis provide support for structuring requirements in sections, and are also applicable in other development phases. However, wikis also provide support for collaboration, and support versioning and baselining of requirements [25]. IMSD-Req addresses collaboration only partially, and thus this is a weakness when comparing ISMD-Req with semantic wikis. Since IMSD-Req is integrated to the control version system Subversion, it allows controlling requirement versions and baselines, and allows requirements engineers to edit requirements documents in parallel. Moreover, we should take into account that desktop documents are still the dominant format used by software organizations for documenting software development [5, 6]. Thus, it is important to provide facilities for handling desktop semantic requirements documents. SOP Wiki [28] recognizes the importance of desktop documents, so that it provides a feature for exporting wiki content to Open Office documents.

Another striking feature of IMSD-Req is that it explores the conceptualization established by the Software Requirements Reference Ontology to provide features to support some RE activities, namely: prioritizing requirements, analyzing impacts from requirements changes, tracing requirements through traceability matrices, and verifying requirements using checklists. None of the aforementioned wikis do that. ReqWiki also provides some requirements-specific services, but they do not explore the conceptualization of its ontology (SWORE). They are Natural Language Processing web services.

5 Conclusion

Requirements Engineering (RE) is a critical factor for software projects to succeed [17]. Many problems that occur in software projects arise from shortcomings in the way developers gather, document, agree on, and modify software requirements [14]. In this context, it is essential to provide computational support for RE.

IMSD-Req is an alternative to provide computational support for RE, keeping organizations' culture of using text editors for documenting requirements. It is worthwhile to point out that IMSD-Req is an alternative to requirement CASE tools, since it supports only partially the RE process. This alternative may be useful, once studies related to RE practices and CASE tools adoption, such those conducted by Hoffman and Lehner [17] and Wiegers [14], point out that the use of commercial tools sometimes has negative impacts on the RE process. Besides, in general, the acquisition cost of these tools is prohibitive, especially for small companies [28]. So it is common that companies use general-purpose tools (like text editors) to document requirements. In this sense, such companies do not need to change the way they work. Requirements documents continue to be produced using a text editor, and IMSD-Req features can be used in a complementary way.

IMSD-Req extends IMSD [7], a general propose semantic document management platform, to include RE-specific features that aims at addressing improvement opportunities identified during the use of IMSD for supporting a RE process [12]. For addressing these improvement opportunities, IMSD-Req explores the conceptualization provided by the Software Requirements Reference Ontology, in order to provide features for supporting requirements change impact analysis, evaluating consistency of requirement priorities, generating traceability matrices, and verifying requirements using checklists.

IMSD-Req was used in a pilot project developed in NEMO, and the new features introduced by IMSD-Req showed to be useful. Some adjustments were also done in IMSD-Req, in order to consider the feedback given by this pilot project. We are now planning an experiment using IMSD-Req as a tool for supporting the RE process in a Requirements Engineering course for graduate students. In light of the promising results obtained so far with IMSD-Req, as a future work, we intend to extend IMSD to the software project management domain. Moreover, several features of IMSD can be improved. First, we plan to expand the elements of ODT documents that can be annotated to include also figures. Second, we intend to develop an ontology on documents, and use it to allow exploring annotations in specific parts of a document, such as title, sections, subsections and so on.

Acknowledgments. This research is funded by the Brazilian Research Funding Agency CNPq (Process Number 485368/2013-7).

References

1. H. Eriksson, M. Bang.: Towards Document Repositories Based on Semantic Documents, Proceedings of Sixth International Conference on Knowledge Management and Knowledge Technologies (I-KNOW ’2006), pp. 313-320, Austria (2006)
2. V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, F. Ciravegna.: Semantic Annotation for Knowledge Management: Requirements and a survey of the state of the art, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 4, pp. 14-28 (2006)
3. T. Berners-Lee, J. Hendler, O. Lassila.: The semantic web, Scientific American, 284 (5), pp. 34–43 (2001)
4. M. Sicilia.: Metadata, semantics and ontology: providing meaning to information resources. International Journal of Metadata, Semantics and Ontologies, vol.1, n.1, pp. 83–86 (2006)
5. T. C. Lethbridge, J. Singer, A. Forward.: How Software Engineers Use Documentation: The State of the Practice, IEEE Software, vol. 20, no. 6, pp. 35-39 (2003)
6. A. Forward, T.C. Lethbridge.: The relevance of software documentation, tools and technologies: a survey. Document Engineering (DocEng’2002) (2002)
7. L.O. Arantes, R.A. Falbo.: An Infrastructure for Managing Semantic Document, Proceedings 14th International Enterprise Distributed Object Computing Conference Workshops EDOCW 2010. Los Alamitos : IEEE Computer Society, pp. 235-244 (2010)
8. V. Lanfranchi, F. Ciravegna.: Semantic Web-based document: editing and browsing in AktiveDoc. The Semantic Web: Research and Applications, v. 3532, pp. 93-89 (2005)
9. H. Eriksson.: The semantic-document approach to combining documents and ontologies, International Journal of Human-Computer Studies, vol. 65, Issue 7 (2007)
10. M. Tallis.: Semantic Word Processing for Content Authors, Proceedings of the Knowledge Markup and Semantic Annotation Workshop, Florida, USA (2003)
11. B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, M. Goranov.: KIM - Semantic Annotation Platform, Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Florida, USA, pp. 844-849 (2003)
12. B. N. Machado, L.O. Arantes, R.A. Falbo.: Using Semantic Annotations for Supporting Requirements Evolution, Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE'2011), pp. 185-190, Miami, USA (2011)
13. G. Kotonya, I. Sommerville.: Requirements engineering: processes and techniques. Chichester, England: John Wiley (1998)
14. K.E. Wiegers.: Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Microsoft Press, Second Edition, Redmond, Washington (2003)
15. A. Kiryakov, B. Popov, I. Terziev, D. Manov, D. Ognyanoff.: Semantic annotation, indexing, and retrieval. Web Semantics: Science, Services and Agents on the World Wide Web, v.2, p 49-79 (2004)
16. B. Schandl, B. Haslhofer.: The Sile Model - A Semantic File System Infrastructure for the Desktop. Proceedings of the 6th European Semantic Web Conference, Vienna, pp. 51–65 (2009)

17. H.F. Hofmann, F. Lehner.: Requirements Engineering as a Success Factor in Software Projects, IEEE Software, pp. 58-66 (2001)
18. I. Sommerville, Software Engineering, 9th ed., Addison Wesley (2010)
19. G. Guizzardi.: Ontological Foundations for Structural Conceptual Models, Universal Press, The Netherlands (2005)
20. R. A. Falbo, J. C. Nardi.: Evolving a Software Requirements Ontology, 34th Latin American Informatics Conference (CLEI 2008), Santa Fé, Argentina (2008)
21. E.F. Souza, R.A. Falbo, N.L. Vijaykumar.: Using Ontology Patterns for Building a Reference Software Testing Ontology, 8th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise and Beyond (VORTE 2013), Vancouver, Canada (2013)
22. S.L. Pfleeger, J.M. Atlee.: Software Engineering: Theory and Practice, Prentice Hall, 4th Edition (2009)
23. A. Olivé.: Conceptual Modeling of Information Systems, Springer (2007)
24. G. Guizzardi.: On Ontology, Ontologies, Conceptualizations, Modeling Languages and (Meta) Models. In: Vasilecas, O., Edler, J., Caplinskas, A. (Org.). Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV. IOS Press, Amsterdam (2007)
25. H. Lai, R. Peng, D. Sun, F. Shao, Y. Liu, A Survey of RE-specific Wikis for Distributed Requirements Engineering, 2012 Eighth International Conference on Semantics, Knowledge and Grids, pp. 47 – 55 (2012).
26. B. Hoenderboom, P. Liang, A Survey of Semantic Wikis for Requirements Engineering, University of Groningen, The Netherlands, Tech. Rep. RUG-SEARCH-09-L03 (2009).
27. L. Abeti, P. Ciancarini, R. Moretti. Wiki-based Requirements Management for Business Process Reengineering. In: Proceedings of the 4th International Symposium on Wikis (WikiSym'08), pp. 45–50 (2008).
28. B. Decker, E. Ras, J. Rech, P. Jaubert, M. Rieth. Wiki-based stakeholder participation in requirements engineering. *IEEE Software*, 24(2):28–35 (2007).
29. B. Sateli, S. S. Rajivelu, E. Angius, R. Witte. ReqWiki: a semantic system for collaborative software requirements engineering. In: *Proceedings of the Eighth Annual International Symposium on Wikis and Open Collaboration* (WikiSym '12) (2012).
30. S. Lohmann, P. Heim, S. Auer, S. Dietzold, T. Riechert. Semantifying requirements engineering - the softwiki approach. In *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS)*, pp. 182–185 (2008).
31. M. Krötzsch, D. Vrandecic, M. Völkel, H. Haller, R. Studer. Semantic Wikipedia. In Journal of Web Semantics, Vol. 5, No 4, pp. 251–261 (2007).
32. S. Auer, S. Dietzold, T. Riechert. OntoWiki – A Tool for Social, Semantic Collaboration. Proceedings of the 5th International Semantic Web Conference, pp. 736-749 (2006).