Towards an Ontology-based Software Documentation Management
A Case Study

Anna Goy and Diego Magro
Dipartimento di Informatica, Università di Torino, C. Svizzera 185, Torino, Italy

Keywords: Software Documentation, Documentation Management, Ontology, Semantics.

Abstract: One of the main issues that a company has to face is the generation and maintenance of product documentation. In particular, several software houses have to take into account the frequent need of rapidly updating software applications, and the corresponding technical documentation, as a consequence of changes in the administrative rules and laws. In order to support documentation generation and maintenance processes, we performed an ontological analysis of these processes in a large Italian software house that produces and sells enterprise applications for small-to-medium sized enterprises. The goal of such a domain analysis was to build a conceptual model enabling a formal characterization of the main elements involved in software documentation. Such a formalization represents the "competence" of a system supporting documentation processes, since it enables it to answer competency questions representing the information needs of the documentation writers (e.g., "In which technical sheets/application manuals/operating manuals is used a given concept?"; "Which technical sheets belonging to a given operating manual do mention a given functionality/screenful/form field?"; "Which are the functionalities/screenfuls/technical sheets potentially impacted by the change of a given software module/file?").

1 INTRODUCTION

In this paper we present the outcomes of the ontological analysis we performed on the documentation processes of a large Italian software house that produces and sells enterprise applications for small-to-medium sized enterprises. One of the main problems of the software house is the generation and maintenance of product documentation. In particular, the company has to face the need of modifying software applications, and the corresponding software manuals, as a consequence of the change of administrative rules and laws (e.g., accounting rules). These kinds of updates, concerning both applications and manuals, must be rapid and occur frequently. For example, if a new information item is required within an administrative process, e.g., due to a change in the law, a new field must be added in the application user interface and in the database; moreover, the update must be reported in the proper places within the documentation.

Another process that impacts the generation and maintenance of software documentation is software localization, which requires the translation of the corresponding instruction manuals. Moreover, the different linguistic versions must be kept aligned in order to avoid mismatches between, for instance, the English manual and the Italian one.

Currently, the documentation and maintenance processes are completely manual: company experts write the documentation, update it, and know "by heart" which are the relationships between software modules, functionalities, and document parts, as well as the relationships between different parts of the documentation itself (e.g., the same functionality can be mentioned in different parts of an instruction manual, or in different manuals). The process is quite complex, also because many writers take part in it. A partial automation would help writers and would reduce the introduction of errors within the documentation (e.g., missing or outdated parts).

In order to study the feasibility of a tool supporting the mentioned processes, we performed a domain analysis aimed at identifying the main concepts and relationships that characterize the documentation itself and the software products, considered as subjects of the documentation. The domain analysis was conducted through interviews and informal discussions with the experts from the

Goy A. and Magro D..
Towards an Ontology-based Software Documentation Management - A Case Study.
DOI: 10.5220/0004124001250131
In *Proceedings of the International Conference on Knowledge Management and Information Sharing* (KMIS-2012), pages 125-131
ISBN: 978-989-8565-31-0

company, coupled with a detailed analysis of the existing software documentation.

The goal of this paper is to show that a company documentation generation and maintenance processes can be effectively supported by exploiting a semantic model of the concepts and relationships involved in them. In particular, the explicit and formal representation of such a semantic model represents a "documentation map" which can be browsed by a software tool in order to provide documentation writers with a guide based on the relationships between products and documentation items describing them. In other words, the semantic model represents the system "competence" and enables it to answer competency questions expressing the information needs of the documentation writers. Some examples of such questions are the following: "Which are the concepts used within glossary definitions?"; "In which technical sheets (application manuals/operating manuals) is used a given concept?"; "Which are the concepts (not) defined in the glossary and used within a given technical sheet (application/operating manual)?"; "Which technical sheets do mention a given functionality?"; "Which are the functionalities (screenfuls/technical sheets) potentially impacted by the change of a given software module (file)?"; and so on. A complete list of the competency questions can be found in Appendix.

In the following, we will present the ontological analysis (Section 2), by explaining the role of general and domain-specific concepts and relationships; then we will show how the semantic model can be applied to a concrete case (Section 3), and exploited to model documentation-related concepts/relations and to perform useful inferences. Finally, we briefly survey some related work (Section 5) and conclude the paper (Section 6).

2 ONTOLOGICAL ANALYSIS

Several definitions for the notion of *ontology* in Computer Science have been proposed. Here, following Studer, Benjamins and Fensel (1998), we consider an ontology as a "formal and explicit specification of a shared conceptualization". Thus, an ontology is an artifact represented in a machine-understandable way ("formal" in the terms of Studer and colleagues), which accounts for a set of concepts, relations and other entities providing a - possibly simplified - view of some area of interest ("conceptualization") and which makes explicit the assumptions and constraints on the usage of the above-said concepts, relations, and entities ("explicit"). Moreover, the conceptualization accounted for should be accepted by a group of people ("shared").

The ontological analysis that we conducted on the considered domain was mainly aimed at identifying and explicitly representing the concepts, relations and entities involved in software documentation generation and maintenance processes, as well as at providing an explicit account for their main characteristics.

The derived ontology can be represented in a machine-understandable language and can be used to build and maintain an explicit characterization of the elements involved in the documentation-related processes. Such a characterization and the ontology itself can then be exploited by the software applications that support the documentation-related processes, in order to answer the set of competency questions listed in Appendix and, ultimately, to support people involved in the documentation generation and maintenance processes.

In the following, we will illustrate the outcome of our ontological analysis by discussing the main features of the resulting ontology. We will depict concepts and relations of the ontology by means of Entity/Relationship diagrams, since this approach is well-known within Italian software companies, which are still more familiar with Entity/Relationships and relational databases than with formal ontology languages based on XML.

The concepts and relationships that the domain analysis has identified as relevant for the task at hand can be grouped into two categories. First, there are "general" concepts/relationships, representing more general entities. These notions are directly related to conceptual frameworks provided by *upper level core ontologies* (Oberle, 2006), and are characterized in the upper level of the produced ontology. Second, there are domain "specific" concepts/relationships representing notions relevant to the company products and product documentation. The latter are characterized in a lower (domain) level of the produced ontology.

2.1 General Concepts

Among the concepts belonging to the more general level of the ontology, a major role is played by those representing information items. In order to account for these concepts, and the relationships among them, we exploited the semantic model defined by the Ontology of Information Objects (OIO) (Gangemi et al., 2005), developed at the Laboratory

of Applied Ontology (ISTC-CNR, www.loa-cnr.it), and by O-CREAM (Ontology for Customer Relationship Management) (Magro and Goy, 2012), developed at the Computer Science Department of the University of Torino. In the conceptual framework proposed by these two models (suitably adapted to fit the modeling needs of the considered domain), every Information Item has three aspects: (a) Its meaning, i.e., the Information Content itself; (b) The Language(s) in which the meaning is expressed; (c) The support that physically realizes the information object (Information Physical Realization). Figure 1 shows some of the upper level concepts and their taxonomic relations.

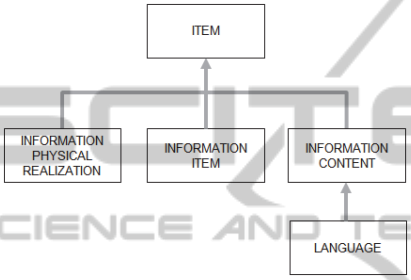

Figure 1: Partial view of the taxonomy of upper level concepts.

2.2 Domain Specific Concepts

Domain specific concepts are those concepts specifically characterizing software documentation or software products.

The software documentation produced by the company can be classified into four main categories: (1) Application manuals (where the main functionality of the software application is described); (2) Operating manuals (where instructions about how to operate on the user interfaces are provided in details); (3) Update notes (explaining software updates); (4) Glossary (explaining the business and technical terminology; currently including more than 1.600 entries).

Figure 2 shows a fragment of the taxonomy related to documentation concepts. A Documentation Resource, which is a top level concept representing a specific type of Information Item, besides being internal (Internal Documentation Resource) or public (Public Documentation Resource), can belong to different typologies (represented by its subclasses): Product Documentation, Glossary, Glossary Entry, Technical Sheet. Moreover, Product Documentation is superclass of Application Manual, Operating Manual, and Update Note, while a Technical Sheet can belong to any of these, thus being an Application Manual Technical Sheet, an Operating Manual Technical Sheet, an Update Note Technical Sheet, or a Glossary Technical Sheet.

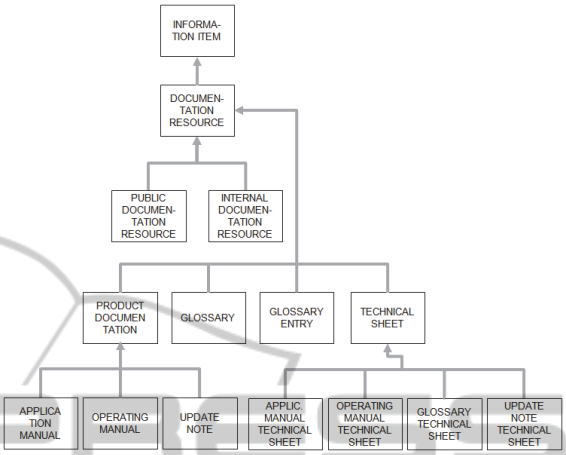

Figure 2: A fragment of the taxonomy of documentation-related concepts.

The main concepts modeling the company products are those related to software, as partially shown in Figure 3. In particular, a Software Element is a specific type of Information Item (according to the notion of *software* specified in Oberle et al, 2006) and can be a Software Module or a Software Module Suite, i.e., an integrated set of software modules. Moreover, both the concepts of Source Code and Executable Code are particular types of Software Elements.

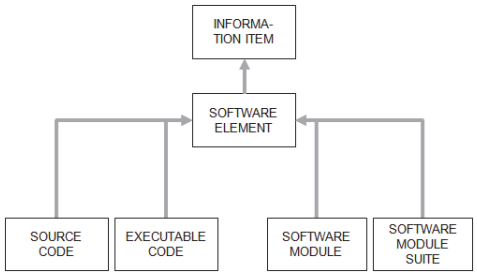

Figure 3: A fragment of the taxonomy of product-related concepts, showing concepts modeling software.

This part of the ontology also includes concepts modeling software functionality and user interfaces, which play a major role in software documentation. Figure 4 shows the main concepts involved in the user interface model, together with their taxonomic relationships. User Interface Element, which is a type of Information Physical Realization, has two subclasses: Screenful, which can be simple (Simple Screenful) or complex (Complex Screenful), and

Screenful Element, which has several subclasses referring to user interface elements, such as form fields (Field).

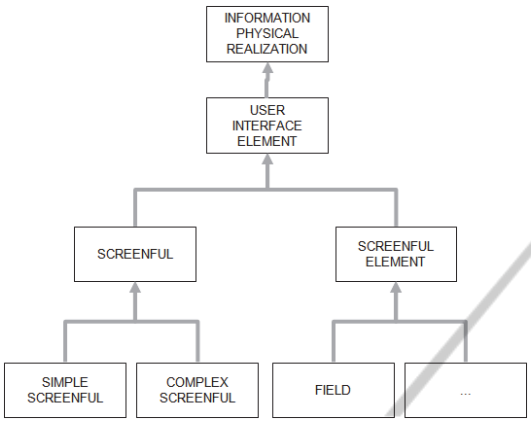

Figure 4: A fragment of the taxonomy of user interface-related concepts.

As far as the functionality is concerned, the most important concept is Functionality, which represents the general functions of a software products (e.g. complaints management) and is distinct from the actual activities involved in the provision of such a functionality. The role of the Functionality concept within some ontological relations can be seen in Figure 5 and 6.

2.3 General Relationships

The most general relationship is the *has_part* relation, modeling the relationship between an object and its parts. Moreover, there are relationships connecting information items to their meanings (*expresses*), to the languages used to express them (*is_encoded, is_completely_encoded*), and to their physical realizations (*is_realized, is_completely_realized*). In addition, there is a relationship linking semantically equivalent information items (*semantically_equivalent*), and there are relationships linking meanings (Information Content) to the elements they are about (*talks_about*), to the elements they identify (*identifies*), and to the concepts they use (*uses*, *defines*, *characterizes*).

These general relations can obviously be used to characterize items belonging to more specific classes; for instance, it is possible to specify the parts of: an application manual, a technical sheet, a glossary entry, a software suite, a single software module, a functionality, a screenful, and so on.

2.4 Domain Specific Relationships

The most relevant specific relationships are those connecting each Software Element to elements of a user interface (User Interface Element) and to the functionality implemented by that software element (*specifies_UI_element*, *implements_functionality*), as shown in Figure 5.

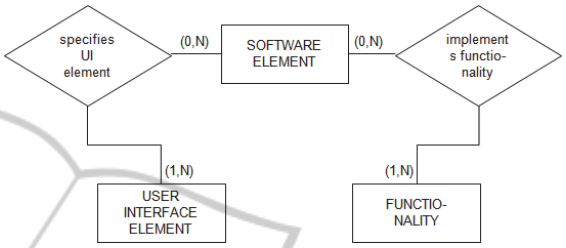

Figure 5: Relationships linking software to functionality and user interface elements.

3 APPLYING THE ONTOLOGICAL MODEL

In this section we present some examples exploiting the ontological conceptualization presented in Section 2, in order to characterize elements belonging to the company documentation system.

3.1 Modeling

The first example we would like to introduce shows how the model supports the distinction between linguistic representations and information contents. This distinction enables us to represent both common aspects and different features characterizing distinct information items.

For example, consider the glossary, which is composed of technical sheets. The ontology enables us to state that a Glossary Technical Sheet is part of a Glossary, which is expressed in some Language (e.g., Italian or English). Now, let's imagine that the company decides to produce an Italian and an English version of the glossary: the Language in which the two Glossary instances are expressed are different (i.e., Italian and English), but all the other involved concepts and relations characterizing information contents are shared. In order to see in some more details these common aspects, let's consider the "Studi di settore" technical sheet. The expression "Studi di settore" (business sector analysis) refers to a methodology used to estimate company and self-employed worker receipts, by

taking into consideration several parameters, including their kind of business. By means of the ontology, we can formally specify that the considered technical sheet contains a Glossary Entry which is composed of two parts, i.e., Glossary Entry Name and Glossary Entry Definition. The Glossary Entry Name *identifies* a specific Concept (i.e., business sector analysis) and the Glossary Entry Definition *expresses* a Concept Definition, which defines the named concept by using other elements defined in the ontology, such as company, self-employed worker, receipt, etc. By separately representing linguistic and content information, the ontology enable us to characterize the two instances (the Italian and the English glossaries) by exploiting the same semantic elements (concepts, relations, and instances).

Another important aspect that is worth pointing out is how the proposed ontology enables us to represent the connections between software modules and functionalities, as well as between software modules and the files containing the corresponding code. These connections enable us to link files to functionalities, and this is an issue of major importance: for example, if some code files are modified, thanks to the mentioned relations writers know which are the impacted functionalities, and thus the manual parts that need to be updated. In the following, we illustrate this case with an example.

In Figure 6, each node represents an instance and is labeled by an instance name (in boldface), followed by the name of the ontology class it belongs to (in uppercase). In particular, the figure shows two software modules, *ma_base_mod* and *ma_cost_mod*; both are instances of the Management Accounting Module class: the first one represents the basic management accounting software module, while the second one represents the cost accounting module. Analogously, the figure shows two functionality instances, *ma_base_func* and *ma_cost_func*; both are instances of the Management Accounting Functionality class, and represent, respectively, the basic management accounting and the cost accounting functionalities. The relationship linking the software modules with the implemented functionalities are instances of the *implements_functionality* relation. Moreover, the figure shows two file blocks (*file_block_01* and *file_block_02*), each one composed of some files; the relationship linking the software modules with the corresponding file blocks are instances of the *is_completely_realized* relation, i.e., the relation that links each information item to the entities providing complete physical realizations for them.

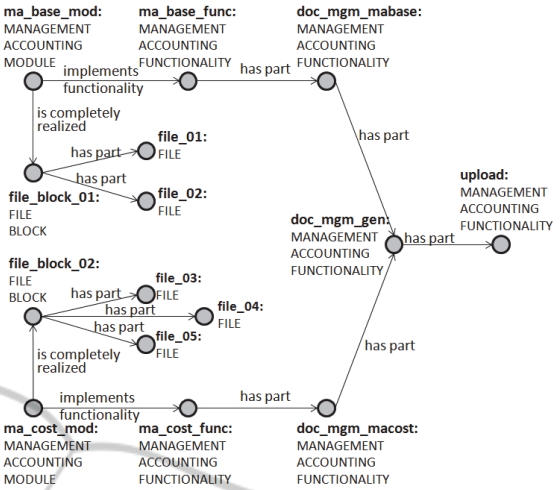

Figure 6: Characterization of the relationships between software modules, functionalities and files.

The semantic model described in Section 2 enables us to provide both a "physical" characterization (which files realize the mentioned software modules) and a "logical" characterization (which functionalities are provided by the mentioned modules). From both these perspectives, we can specify sub-parts, i.e., parts of the file block, and parts of the overall functionality.

The main functionality implemented by the module *ma_base_mod* is the *ma_base_func* functionality (see Figure 6), which is composed of some sub-parts, including, for instance, *doc_mgm_mabase*, which is an instance of Management Accounting Functionality, and represents the management of documents related to the basic functions of management accounting; analogously, the *ma_cost_func* functionality is composed of some sub-parts, including, for instance, *doc_mgm_macost*, which is an instance of Management Accounting Functionality, and represents the management of documents related to the cost accounting functions.

Both *doc_mgm_mabase* and *doc_mgm_macost* have, in turn, sub-parts (sub-functionalities), among which *doc_mgm_gen* (referring to those aspects of *doc_mgm_mabase* and *doc_mgm_macost* which can be considered generic document management functions). Finally, *doc_mgm_gen* includes, as a part, the *upload* functionality.

As we will see in the next section, this characterization enables us to answer competency questions like, for example, "Which are the functionalities implemented by the x module?", "In which software modules is implemented the y functionality?", "Which are the functionalities

potentially impacted by a change in the z file?". Moreover, the ontology enables us to express the links between the documentation resources and the functionalities they are about, therefore the system can also answer questions such as "Which are the technical sheets potentially impacted by a change in the z file?". These questions (and others, all listed in Appendix) are of major relevance in generation and maintenance documentation processes.

3.2 Inferences

Typically, only a part of an information system knowledge is explicitly represented: other knowledge, in fact, can usually be inferred, and made explicit, by applying reasoning mechanisms to the explicit knowledge. In the following, we will describe, with the help of some examples, the role that such inferential mechanisms can play within a documentation system.

First of all, for each individual in the characterization, a reasoning process can infer all the classes that individual is instance of. Moreover, some important inferences concern the part-of relationships between elements. For example, from the characterization of the concepts and relationships concerning the glossary (see Section 2) and from the specific features of the *has_part* relation itself (in particular, the fact that it is reflexive and transitive) the reasoner can infer that a Glossary Entry (a direct part of a Glossary Technical Sheet, which is a direct part of a Glossary in its turn), a Glossary Entry Name, and a Glossary Entry Definition (both direct parts of the considered Glossary Entry) are also parts of the mentioned Glossary.

Another example of useful inference supported by the semantic model presented is the one enabling us to answer competency questions like the following (see Figure 6):

(a) "Which are the functionalities implemented by the module *ma_cost_mod*?". Given the characterization of the *implements_functionality* relation, if a module implements a functionality *f*, then it implements also all *f* sub-parts. Thus, from the presented representation, a reasoner can infer the answer: all those directly linked to the mentioned module by the *implements_functionality* relation, together with all the sub-parts of it, including, for instance, the *upload* function.

(b) "In which software modules is implemented the *upload* functionality?". Again, from the ontology and the characterization of the items in terms of the ontology, a reasoner can infer the answer, which will include the module *ma_cost_mod*.

Inferences like these are very important. For instance, if I am writing an update note concerning a given functionality, I need to know in which modules it is implemented, in order to update all the corresponding manual parts. Analogously, if developers modify a given file, I need to know which functionalities are potentially involved, in order to properly document the changes.

4 RELATED WORK

The idea of using semantic technologies to support software documentation processes has been investigated by focusing on different aspects of the problem. For instance, Kleiber, Sabol, Kern, Muhr and Granitzer (2009) face the problem of keeping the software documentation up to date, given the reduced time-to-market development cycles, and propose a support to the software documentation process based on an underlying ontology modeling software documentation activities and the structure of the software itself. Hepp and Wechselberger (2008) exploit ontologies modeling SAP-related business and technical concepts in order to improve the accessibility of ERP documentation search systems.

There is some work on (semi)automatically deriving ontologies from software documentation; see (Sabou, 2004). Moreover, some approaches aim at supporting software maintenance by exploiting semantic technologies. For instance, Witte, Zhang and Rilling (2007) propose to use ontologies in order to connect and integrate knowledge about source code and software documentation; the resulting knowledge can be exploited to support software maintenance. Similarly, Ambrósio, Santos, Lucena and Silva (2004) present a tool that exploits ontologies to integrate domain and software engineering knowledge in order to help keeping software documentation, up to date.

5 CONCLUSIONS

In this paper we presented an ontological analysis aimed at developing a semantic model useful to support the generation and maintenance of software documentation. In particular, by describing examples from a concrete use case, the paper shows that providing a company documentation system with an explicit semantic model of the concepts and relationships involved in documentation generation

and maintenance processes means increasing the system "competence" about those processes; this competence, in turn, enables the system to satisfy the information needs of the documentation writers. Such needs can be represented by the competency questions in Appendix.

ACKNOWLEDGEMENTS

This work has been partially funded by CELI s.r.l. (www.celi.it).

REFERENCES

Ambrósio, A.P., Santos, D.C.d., Lucena, F.N.d., Silva, J.C.d. (2004). Software Engineering Documentation: an Ontology-based Approach. *WebMedia & LAWeb Joint Conference*. Washington: IEEE Press, 38-40.

Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J. (2005). *Task Taxonomies for Knowledge Content*. Metokis, Deliverable D07.

Hepp, M. and Wechselberger, A. (2008). OntoNaviERP: Ontology-Supported Navigation in ERP Software Documentation. *International Semantic Web Conference - ISWC2008*, LNCS 5318. Heidelberg: Springer, 764-776.

Kleiber, W., Sabol, V., Kern, R., Muhr, M., Granitzer, M. (2009). Using Ontologies For Software Documentation. *Malaysian Joint Conference on Artificial Intelligence - MJCAI2009*. Kuala Lumpur, Malaysia.

Magro, D., Goy A. (2012). A core reference ontology for the customer relationship domain. *Applied Ontology*, 7(1), 1-48.

Oberle, D. (2006). *Semantic Management of Middleware*. Heidelberg: Springer.

Oberle, D., Lamparter, S., Grimm, S., Vrandečić, D., Staab, S., Gangemi, A. (2006). Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems. *Applied Ontology*, 1(2), 163-202.

Sabou, M. (2004). Extracting Ontologies from Software Documentation: a Semi-Automatic Method and its Evaluation. In *Workshop on Ontology Learning and Population at ECAI 2004*, Valencia, Spain.

Studer, R., Benjamins, V. R., Fensel D. (1998). Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25(1-2), 161-197 .

Witte, R., Zhang, Y., Rilling, J. (2007). Empowering Software Maintainers with Semantic Web Technologies. *European Semantic Web Conference - ESWC2007*, LNCS 4519. Heidelberg. Springer, 37-52.

APPENDIX

A system based on a knowledge base as the one described in this paper can support the processes devoted to the production and maintenance of the company product documentation, by answering the following competency questions:

- Which are the glossary elements?
- Which are the concepts defined within the glossary?
- Which are the concepts used within glossary definitions?
- Among the concepts used in glossary definitions, which of them are (not) defined in the glossary?
- Which are the glossary entries which refer to other glossary entries?
- Which are the synonyms in the glossary?
- In which technical sheets (application manuals/operating manuals) is used a given concept?
- Which are the concepts a given technical sheet (manual) is about?
- Which are the concepts (not) defined in the glossary and used within a given technical sheet (application/operating) manual?
- Which are the documentation resources (partially/completely) represented in Italian (English/…)?
- Which technical sheets do mention a given functionality?
- Which are the functionalities never mentioned in any manual?
- Which technical sheets belonging to a given operating manual do mention a given screenful (form field)?
- Which are the screenfuls of a given software module that are never mentioned in any operating manual?
- Which are the functionalities (screenfuls/ technical sheets) potentially impacted by the change of a given software module (file)?
- Which are the software modules which could be involved in the change of a given technical sheet?