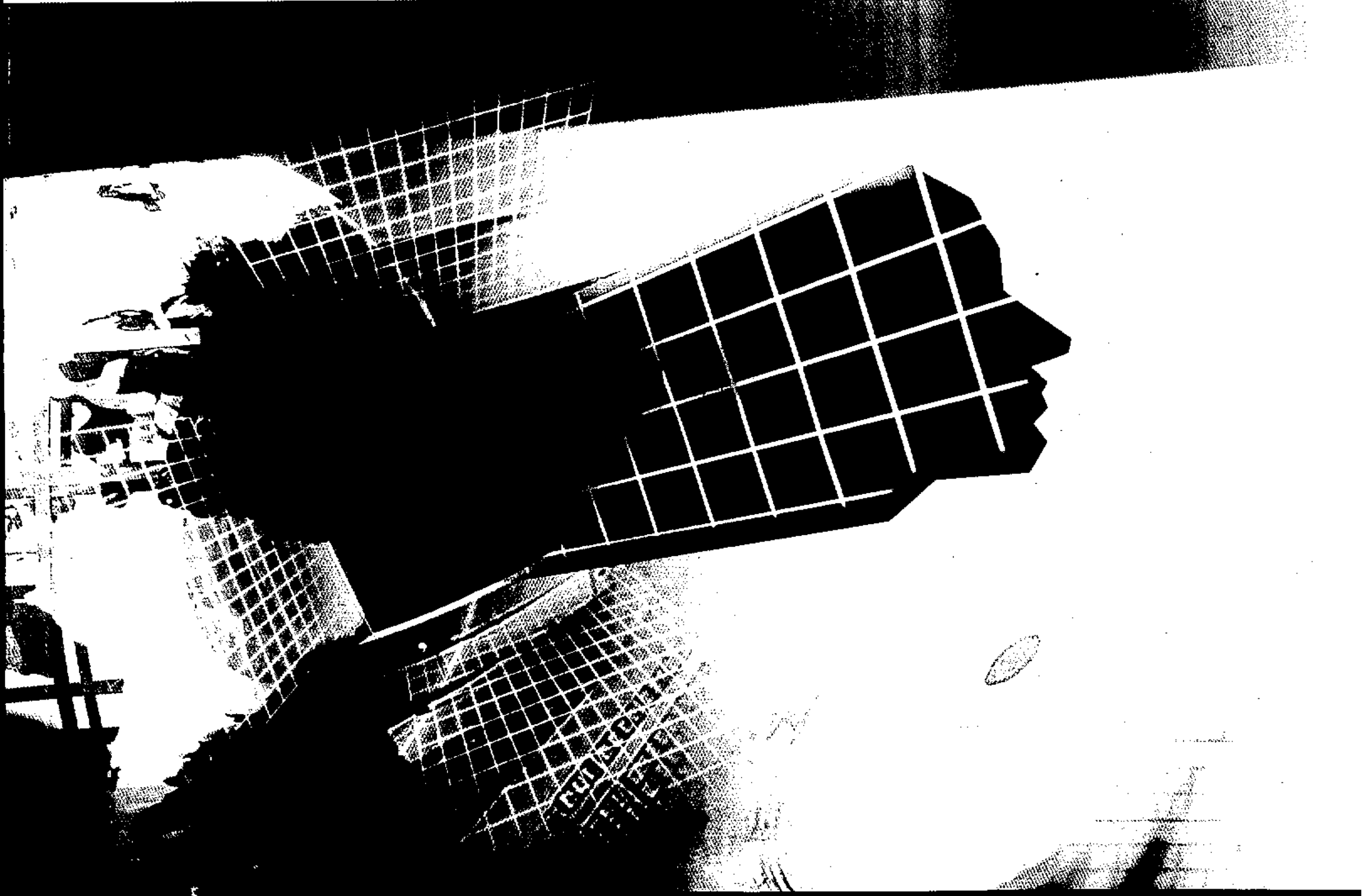
JIISIC'07

VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento

DEL 31 DE ENERO AL 2 DE FEBRERO
LIMA - PERÚ



DEPARTAMENTO
DE **INGENIERÍA**
SECCIÓN INGENIERÍA INFORMÁTICA

90 AÑOS

PONTIFICIA
UNIVERSIDAD CATÓLICA
DEL PERÚ

DIRECCIÓN DE COMUNICACIÓN -NS1

ISBN 978-9972-2885-1-7

ISBN 9972-2885-1-X
9 789972 288517

JIISIC'07

VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento

Lima-Perú
31 de enero al 2 de febrero de 2007

Editado y Compilado por:

Facultad de Ciencias e Ingeniería
Departamento de Ingeniería
Maynard Kong
José Antonio Pow-Sang
Manuel Francisco Tupia
Luis Alberto Flores

90 AÑOS PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento-JIISIC'07

Compilado por:
Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú
Departamento de Ingeniería de la Pontificia Universidad Católica del Perú

Editado por:
Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú
Departamento de Ingeniería de la Pontificia Universidad Católica del Perú
Maynard Kong Wong, José Antonio Pow-Sang Portillo, Manuel Francisco Tupia Anticona y Luis Alberto Flores García.

Primera edición: enero de 2007

Hecho el Depósito Legal en la Biblioteca Nacional del Perú N°2007-00571

ISBN N° 978-9972-2885-1-7

Comité Permanente

Silvia Teresita Acuña, Universidad Autónoma de Madrid, España
Manoel Mendonça, Universidade Salvador, Brasil
Oscar Dieste, Universidad Complutense de Madrid, España

Comité Organizador

José Antonio Pow-Sang, Pontificia Universidad Católica del Perú **(chair)**
Manuel Tupia, Pontificia Universidad Católica del Perú
Luis Flores, Pontificia Universidad Católica del Perú
Felipe Solari, Pontificia Universidad Católica del Perú

Comité de Programa

Maynard Kong, Pontificia Universidad Católica del Perú, Perú **(chair)**
Raul Aguilar, Universidad Autonoma de Yucatán, México
Idoia Alarcon, Universidad Autónoma de Madrid, España
Luis Alberto Alvarez, Universidad Austral, Chile
Marco Alvarez, Utah State University, EEUU
Pedro Antunes, Universidade de Lisboa, Portugal
Joao Araujo, Universidade Nova de Lisboa, Portugal
Marianela Aveledo, Universidad Simon Bolivar, Venezuela
Pere Botella, Universitat Politècnica de Catalunya, España
David Camacho, Universidad Autonoma de Madrid, España
Francisco Camargo, ITESM, México
Zalatiel Carranza, Universidad de Lima, Perú
Dante Carrizo, Universidad Complutense de Madrid, España
Luca Cernuzzi, Univ. Católica Ntra. Señora de la Asunción, Paraguay
Sergio Coronado, University of Luxembourg, Luxemburgo
Ernesto Cuadros-Vargas, Universidad Católica San Pablo, Perú
Angelica de Antonio, Universidad Politécnica de Madrid, España
Amador Duran, Universidad de Sevilla, España
Juan Vicente Echagüe, Universidad de la República, Uruguay
Yadran Eterovic, Pontificia Universidad Catolica de Chile, Chile
Mariano Fernandez, Universidad CEU San Pablo, España
Xavier Ferre, Universidad Politécnica de Madrid, España
Ramon Garcia, Instituto Tecnológico de Buenos Aires, Argentina
Francisco Jose Garcia, Universidad de Salamanca, España
Luis Guerrero, Universidad de Chile, Chile
Ricardo Imbert, Universidad Politécnica de Madrid, España
Mario Jino, Universidade Estadual de Campinas, Brasil
Nora La Serna, Universidad Nacional Mayor de San Marcos, Perú
Guillermo Licea, Universidad Autónoma de Baja California, México
Marta Lopez, Universidad Complutense de Madrid, España
Jose Antonio Macias, Universidad Autónoma de Madrid, España
Esperanza Marcos, Universidad Rey Juan Carlos, España
Victor Hugo Medina, Universidad Distrital Fco. José Caldas, Colombia
Nelson Medinilla, Universidad Politécnica de Madrid, España
Ana María Moreno, Universidad Politécnica de Madrid, España
Jaime Muñoz, Universidad Autónoma de Aguascalientes, México
Melvin Perez, CAM Informatica, República Dominicana
Claudia Pons, Universidad Nacional de la Plata, Argentina
Angel Puerta, Redwhale Software , EEUU
Isidro Ramos, Universitat Politècnica de Valencia, España
Gustavo Rodriguez, INAOE, México
Maria Isabel Sanchez Segura, Universidad Carlos III de Madrid, España

Comité de Programa (continuación)

Colaboradores en el Proceso de Revisión

Prólogo

Este volumen contiene los trabajos aceptados y presentados en las VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'07) celebradas en Lima, Perú, del 31 de enero al 2 de febrero de 2007. Desde su edición inicial en 2001, las JIISIC han demostrado ser el foro de reunión más importante, a nivel Iberoamericano, de investigadores y profesionales interesados en ambas disciplinas.

El evento actual es la continuación de la labor iniciada en las JIISIC'01, celebrada en en Buenos Aires (Argentina), JIISIC'02 en Salvador de Bahía (Brasil), JIISIC'03 en Valdivia (Chile), JIISIC'04 en Madrid (España) y JIISIC'06 en Puebla (México).

En la presente convocatoria se han recibido 88 artículos de calidad científica para su evaluación. Cada trabajo ha sido evaluado por al menos 2 revisores y se ha contemplado la resolución de divergencias, que por cierto han sido muy pocas. Finalmente fueron aceptados 54 artículos de autores procedentes de Argentina, Brasil, Colombia, Corea del Sur, Cuba, Chile, Ecuador, España, Estados Unidos de América, Mexico, Perú y Uruguay. Además de la sesiones técnicas, se aceptaron cuatro tutoriales.

Es preciso indicar que todo esto no hubiera sido posible sin la colaboración de muchas personas. Por ello queremos agradecer especialmente a los miembros del Comité de Programa por su excelente y desinteresada labor, necesaria para renovar la calidad y prestigio ganado. También queremos destacar el enorme esfuerzo de Manuel Tupia, Luis Flores y Felipe Solari, miembros del Comité Organizador, sin cuyo trabajo no hubieran podido celebrarse estas Jornadas. Nuestro agradecimiento al Ing. Eduardo Ismodes, decano de la Facultad de Ciencias e Ingeniería, y al Ing. Kurt Paulsen, jefe del Departamento de Ingeniería, por el gran apoyo que nos han brindado. Por último, pero no al final, expresamos nuestro sincero agradecimiento a todos los autores que aportaron sus contribuciones al evento.

Maynard Kong
Presidente del Comité de Programa

José Antonio Pow-Sang
Presidente del Comité Organizador

ÍNDICE

Asynchronous Merging of Software Ontologies: An Experience

Nicolas Anquetil[1], Aurora Vizcaíno[2], Francisco Ruiz[2], Kathia Oliveira[1] and Mario Piattini[2]
[1] GES Research Group. Catholic University of Brasilia, Brazil
{anquetil, kathia}@ucb.br
[2] Alarcos Research Group. University of Castilla-La Mancha, Spain
{aurora.vizcaino, francisco.ruiz, mario.piattini}@uclm.es

Abstract

Different methodologies exits to merge ontologies. However, most of them need the source ontologies to be defined in a particular and formal way. Moreover, tools to help during the merging process have been developed, but these are thought for synchronous settings (where the knowledge engineers can exchange ideas in real time) and very specific conditions. In this paper we describe the merging process that two research groups have used to merge two maintenance ontologies in an asynchronous way. We also describe the problems that we faced since each research group was in a different country, with different time zone and also different mother languages. The usage of a systematic methodology helped us to tackle these problems as will be explained in this paper.

1. Introduction

Onotologies capture consensual knowledge of a specific domain in a generic and formal way, to allow it to be reused and shared among groups of people. Despite requiring consensus between different experts, there is no single possible ontology to model a particular domain, thus domain-specific ontologies are modeled by multiple authors in multiple settings [24]. For example, in the case of software engineering where ontologies can play important roles, and more concretely in the software maintenance domain, there are several published ontologies [3, 4, 14, 22], each one dealing with maintenance activity from a different point of view. In an attempt to achieve a better result we decided to merge two ontologies, those of Dias *et al.* [4] and Ruiz *et al.* [22] that seemed to be most complementary and which moreover, were based on a third, that of Kitchenham *et al.* [14]. Our goal was to construct a more general ontology by taking into account the most important concepts related to software maintenance.

A great difficulty in this work was the geographic distance between the two teams of authors of the ontologies. As a result, the merging could not be conducted in a typical way where the knowledge engineers could meet and discuss together what concepts to include, what restrictions to apply to these concepts, etc. What is more, we had some extra challenges. For instance, one ontology was developed by a Brazilian University, and the other by a Spanish University and although both ontologies were defined in English this was not the mother tongue of any of the developers of the ontologies. Because of this, misunderstandings might arise. We attempted to counter balance these difficulties by defining a merging process that would take the specificity of our situation into account.

In this paper we present the process followed to merge the two ontologies and we report on the difficulties found and the lessons learned from this experiment.

The remainder of the paper is structured as follows. Section 2 describes the merging process and some methodologies and tools developed for this purpose. Section 3 explains the process that we followed to merge the ontologies. Section 4 presents the benefits and limitations of the approach used. Finally in section 5 conclusions are outlined.

2. Merging ontologies

It is first necessary to clarify the difference between two related words: merging and alignment. Merging ontologies means to create a single coherent ontology from two sources. Aligning ontologies means to establish links between them and allow them to reuse information from one another [16]. Alignment does not aim to create a new ontology. Merging two ontologies implies some kind of alignment as one must map one ontology on to the other to find out the commonalities, synonyms, etc.

There are different methods by which to merge ontologies and many of them provide a tool with which to automatically identify potential matchings or provide an environment to manually find and define these matchings. Mapping tools and algorithms are: ONIONS [23] which allows the creation of a library of ontologies originating from different sources; the Chimaera system [15] provides support to merge ontological terms from different sources, to check the coverage and correctness of ontologies and to maintain ontologies over a period of time; OntoMorph [2] provides two kinds of mechanisms for merging ontologies. One is a syntactic rewriting support that allows translation between two different representation languages, and the other is a semantic rewriting tool that allows inference-based transformations; GLUE [5] uses machine learning techniques, to provide pairs of related concepts with some certainty factor associated to each pair. Another approach is FCA-Merge [24] which takes as input two ontologies to be merged and a set of documents on the domain of the ontologies. The merging is performed by extracting instances that belong to concepts of both ontologies from the documents. Finally, PROMPT is an algorithm embedded in Protégé 2000, that proposes first to elaborate a list with the operations to be performed in order to merge two ontologies [17]. This activity is carried out automatically by a PROMPT plug-in. Then, an iterative process is performed. For each iteration the ontology developer selects an operation of the list and executes it. After that, a list of conflicts is generated and the list of possible operations for the following iterations is updated.

Most previous techniques need the source ontologies to be defined in a particular and formal way and some, such as OntoMorph and Chimarea, use a description logics based approach. Moreover, only FCA-MERGE offers a structural description of the global merging process [24]. These facts, and other difficulties that will be detailed in the next section, led us to define our own merging approach. Contrary to the existing approaches, we did not seek automation of the merging process and will not propose any tool to help. In our experience, very few activities can be automated and when this is possible, they do not represent a significant work load. We will therefore focus on presenting and discussing our methodology which has given good results and proved to be useful.

3. Merging two software maintenance ontologies

This work started as a result of two teams (the Alarcos group from University of Castilla-La Mancha, Spain; and GES from Catholoc University of Brasilia, Brazil) wanting to collaborate on the definition of an ontology for software maintenance. Each research group had already published an ontology on software maintenance separately Ruiz *et al*. [22] and Días *et al.* [4] as a support for their respective ongoing research, but we perceived that each ontology bore the mark of its maker. Our goal in merging the ontologies was to obtain a more general maintenance ontology. Although we work under very strong restrictions, we also perceived positive factors that suggested that the work could be done.

What separates us:

- The Atlantic ocean (geographical distance);
- Five hours (different time zones);
- Two languages (Spanish and Portuguese although very close each other do not allow the easy discussion of such complex issues raised by an ontology merging).

The positive points:

- The domain, software maintenance, is relatively well defined.
- The researchers are all domain experts to some degree.
- Both ontologies are based on the same sources, the main ones being (see also Figure 1) an ontology for software maintenance [14] and another ontology for the software process [6]. Also used as a source by [14]. There are also a number of other minor sources in common such as international standards, significant publications, etc.

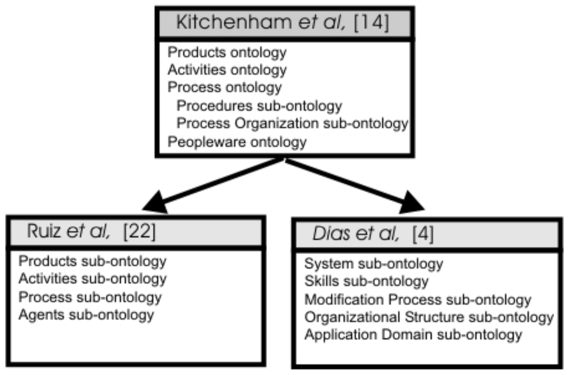


Figure 1. Schematic representation of three software maintenance ontologies

Before going on describing difficulties found and the process that we followed to merge the ontologies in more detail, we will describe the two source ontologies.

3.1. The two software maintenance ontologies

Table 1 and 3 characterize the two ontologies. Table 2 and 4 list some of the references used to build the two ontologies, including the common references highlighted in gray.

Table 1. Details of Ruiz *et al.*' ontology

Concept	Value
Domain	Management of Software Maintenance Projects
Author	Alarcos Research Group (UCLM)
	Ontology to enable information to be interchanged among engineers, managers and users fo maintenance projects
Level of formality	Semi-formal (REFSENO and UML)
Scope	List of concepts: This is classified(for reasons of clarity) into partial ontologies and subontologies: - Maintenance Ontology Products Subontology Activities Subontology Process Organization Subontology Procedures Requests Management Problems Agents Subontology - Workflow Ontology - Measurement Ontology
Source of knowledge	See table 2

Table 2. Sources of Knowledge used Ruiz *et al.*'s ontology

Informal ontology for SMP proposed by Kitchenham *et al.* in [14]
Conceptual model for corrective maintenance by Kajko-Mattson in [12,13]
Ontology for the software development process proposed by Falbo *et al.* In [6]
Conceptual model for software process and software measurement proposed by [1]
Documents which define the MANTIS processes system:
- Model of ISO 12207 life cycle
- Process reference model ISO15504-2 [9]
- ISO 14764 about SMP model [10]
- Model of activities and tasks of the MANTEMA methodology [20]

Ruiz *et al.*'s [22] ontology was focused on the concepts related to software maintenance projects from a static and dynamic point of view. Because of this the ontology also considers workflow and measurement issues. However, in this paper we only focus on the maintenance ontology from a static point of view, since these two issues were perceived as a specificity of Ruiz *et al.*'s ontology that Dias *et al.* did not consider in their work.

Dias et *al.*'s ontology was developed to describe the knowledge used in software maintenance. Therefore, the two ontologies, although focusing on software maintenance, have different goals, scope, organization (sub-ontologies). Note that Kitchenham *et al.*'s ontology, used as a source in both cases considered here, also has a different focus since it is aimed at classifying research in software maintenance.

Both source ontologies were modeled with UML. [7] state that UML may be used as a technique for modeling ontologies since it is easy to understand and use for people outside the AI community. Moreover, there is a standard graphical representation for UML models, and many CASE tools are available to manipulate these representations.

Table 3. Details of *Dias et al.' ontology*

Concept	Value
Domain	Practice of Software Maintenance Projects
Author	GES Research Group (Catholic University of Brasilia)
Purpose	Ontology to identify and organize all the knowledge needed when performing maintenance.
Level of formality	Formal (UML, dictionary of concepts, definition of restrictions in first order logic)
Scope	List of concepts: There are five subontologies: System Subontology Maintenance Process Subontology Computer Science Skills Subontology Organization Subontology Application Domain Subontology
Source of knowledge	More than 30 references, see Table 4 for principle sources

Table 4. Sources of Knowledge used Dias *et al.*'s ontology

Informal ontology for SMP proposed by [14]
Conceptual model for corrective maintenance by [13]
Ontology for the software development process proposed by [6]
Book on software Maintenance [19]
Books on Software Engineering [18, 21]
Standard on Software Maintenance IEEE-1219 [8], ISO 12207, and ISO 14764

3.2. The merging constraints

The work we undertook was to merge these two existing software maintenance ontologies into a new one. Contrary to the ideas proposed by [16] we did not adopt one "stable" (preferred) ontology on which to map the other. In this merge, the two source ontologies have exactly the same "weight". One reason for this is that both source ontologies are approximately of the same age and were too recent at the time we started the merging to have been used in other works. Therefore "changing" one or the other would have exactly the same (small) impact.

We started the work with each group studying the other's ontology. In this way we were able to get a better idea of what difficulties we would have, and what differences existed between this specific work and the merging process proposals found in literature which were of a more theoretical nature (meaning that the goal of these other works was not always to actually merge ontologies, but to propose processes or tools that would help in merging):

- The first conclusion was that the structure of the ontologies, although different overall had some commonalities: description (sub-ontology) of the software system maintained, description of the maintenance process, description of the organizational structure.
- A brief look at the concepts in the two ontologies showed that very few of them have similar names [16]: 11 concepts, which is about 15% of Ruiz *et al.*, and 10% of Dias *et al.*
- As already highlighted, an important constraint is that the two teams are geographically separated. This in itself ruled out most of the proposed merging tools which assume that all work is done at a "central" location.
- One ontology is described in a semi-formal way (see Tables 1 and 3) and the other more formally, but both use UML for graphical representation and textual description of the concepts (dictionary of concept).
- Each group has a biased understanding of the source material. Concretely, each group understands its own ontology and the rational for its design well, and the other ontology much less. There was no central authority which would have a balanced view of the sources ontologies.

We felt, and continue to believe, that the worst difficulty was the geographical dispersion. Merging two ontologies is very much like designing a new one[1]. It is essentially an exchange activity where experts confront their views to reach a common understanding. Doing so at distance proved a major difficulty.

3.3. The merging process

Because we were not able to exchange ideas quickly, we felt the necessity to have a well defined process to follow. As software engineers (primarily), we based this process upon some well known principles in our field, mainly from the USDP (Unified Software Development Process) [11], also known as RUP – Rational Unified Process). The USDP is a process for developing software, an activity that bears some resemblance to our situation:

- There is a highly conceptual initial part to software development (domain modeling) that is very similar to ontology modeling.
- Software development projects nowadays require the participation of many different people often in different places and speaking different languages (off-shore development).
- UML is the representation language of choice.

The positive practices that we wanted to incorporate from the USDP are:

- Iterative and incremental process: don't try to do everything at once but slowly (and iteratively) work toward the solution.
- Manage the project risks so that they don't suddenly jeopardize the project.

In the USDP, the iterative and incremental approach implies that all the functionalities are not implemented at once but spread out over various iterations, and that each functionality will itself be typically developed in several iterations (first the core functionality, then its alternative scenario). This is different from the iterative process proposed by [16], because in PROMPT, the iterations simply repeat the same activity whereas in USDP, each iteration is a small (sequential) process itself, including all activities from the most abstract (requirement elicitation, requirement analysis) to the most detailed (implementation and testing). In the case of ontology merging, we do not have the same abstraction span. However, one may still have different ”activities” such as considering the subontologies, the concepts, the associations between them and finally the restrictions (for formal ontologies).

In the USDP, the iterations are a way of dealing with risks in software development: by developing the riskiest functionalities (only them and only their core feature) in the first iterations, one can evaluate more rapidly if one is able to implement them as planned. This allows better control over the whole project.

We attempted to apply these principles to the merging of ontologies by following the idea of an initial **core** (similar to the core functionalities of the USDP and their core scenario) that we could progressively expand to a complete ontology. This idea was applied on two different levels. First, at the ontology level, we started with a "core" sub-ontology which happened to be common to both ontologies to be merged (we will come back to this later). When we were satisfied that we would be able to merge this sub-ontology, even if the merging was not yet completed, we started to look for the next sub-ontology.

We applied a similar process at the level of concepts. When working on a sub-ontology, we focused first on a core concept (or a small group of core concepts) that we then expanded with related concepts, enlarging the scope to the point that we were satisfied that the sub-ontology considered was completely described. Thus in one iteration, we might be dealing with the core concepts of one sub-ontology

[1] Noy and Musen in [17] suggest that merging is actually a sub-case of designing a new ontology.

while still resolving some pending issues (of relatively little importance) of another sub-ontology.

The associations between the concepts proved to be easy to deal with once the concepts themselves had been agreed upon. Clearly, changing the meaning of concepts or adding new concepts had an impact on the associations, but this is normal and expected. Still we feel that focusing on the concepts first allowed to minimize the amount of re-work.

Finally in our case the restrictions are not an issue as one of the source ontology is semi-formal (therefore we do not need to merge restrictions).

3.4. Merging example: applying the iterative process

To better illustrate how we conducted the merging, we will now describe in some detail how we applied our process. Due to the geographical dispersion of the ontology designers, we adopted the following procedure: First, we agreed on a core sub-ontology to work on, then one team started working on the merging of this portion of the ontology. This team sent a proposal by e-mail to the other team which analyzed it, commented on it (accepting and/or counter arguing) and sent back its comments. This corresponds to one iteration where, as explained previously, we would work at various levels of detail (concept and/or associations). The proposal went back and forth between the two teams, until only minor issues remained pending. Then we started to look for the core concept of a second sub-ontology while the pending issues for the first were resolved ``in the background''. We did this for each of the three sub-ontologies that we required.

The first sub-ontology we considered was the system ontology. When one talks about software maintenance, it seems clear to us that the software system to be maintained should be considered. Indeed, this was the only common sub-ontology that the two source ontologies had (in one ontology it was called Product sub-ontology instead of System sub-ontology) (see Figure 1). Although the two source ontologies agree on this, their respective software system sub-ontologies present significant differences: their names are different and they are at different levels of abstraction. In Ruiz *et al*. [22] the Product sub-ontology has only three concepts whereas there are 27 in Dias *et al.'s* [4] System sub-ontology. The two source sub-ontologies are presented in Figures 2 and 3.

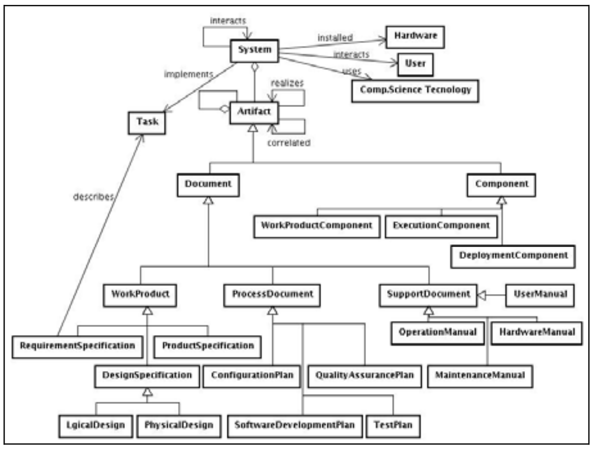


Figure 2. System sub-ontology [4]

To resolve this conflict, we applied the principle of finding the core concepts of this sub-ontology and expanded this core to the entire sub-ontology. Again these core concepts were what the two sub-ontologies had in common. This part proved to be similar to the initial step of the SMART algorithm which recommends the creation of a list of the concepts considered in each ontology [16], looking for concepts with identical names or with linguistically similar names.

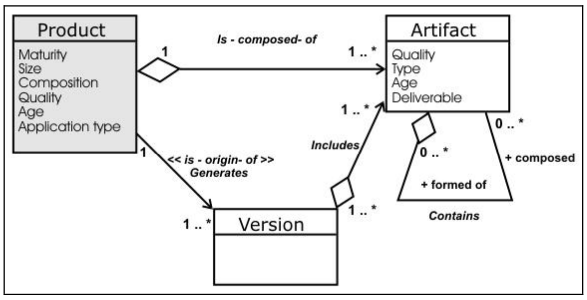


Figure 3. Products sub-ontology [22]

At this point we realized that, an automated tool would have had difficulties in identifying how to map the sub-ontologies as they used two different names (system and product), and their core concepts used different names too (again system and product).

From the three concepts in Ruiz *et al.* Product sub-ontology (Figure 3), two were found in Dias *et al.* System sub-ontology (Figure 2). These are the “software system” maintained and its “artifacts” (a software system is composed of artifacts). It seems clear to us that these two concepts are indeed core concepts of a System sub-ontology and could be used as a base to cultivate the entire sub-ontology. For this reason, we applied the “merge-class” operator, as described in [16]: the two “artifact” concepts were merged in the new sub-ontology, and, the “system” and “product” concepts were merged as a “software product” concept. From this base, we expanded our initial sub-ontology. We decided to include the “version” concept present in one sub-ontology (Figure 3) and not the other (Figure 2), and the rest consisted of deciding what concepts from the System sub-ontology (Figure 2) would reach the merged sub-ontology (less important issues).

This is another point where an automated tool would have been of little use since the commonalities between the two sub-ontologies were very few (two concepts) and most of the work was discussed between the two groups to decide at what level of abstraction we wanted to work and what concepts of the System sub-ontology would be rejected or merged in the final sub-ontology.

The resulting sub-ontology is presented in Figure 4. We needed only one iteration (i.e. one round-trip: proposal from one group and answer/comment from the other) to agree on the core concepts of this first sub-ontology. The remaining (minor) issues were closed in another iteration (focused on the second sub-ontology).

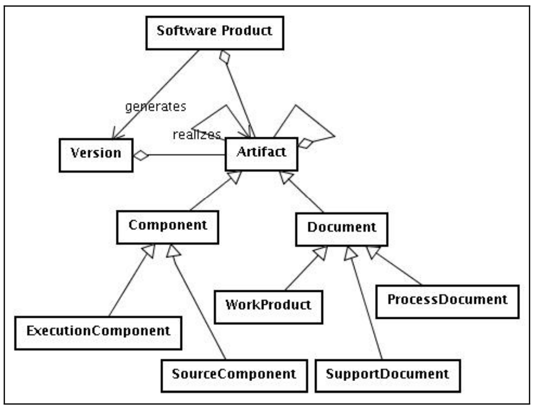


Figure 4. Merged Software Product sub-ontology

After having solved the main problems of the first sub-ontology, we continued. For the other sub-ontologies, we did not have a one to one correspondence between the two sources (one source sub-ontology would map to two, or more, in the other source), but having already defined a core sub-ontology and its concepts helped in merging the rest, because we had a common base to work from.

The next sub-ontology we considered was that of maintenance activity. This seemed to be the next logical sub-ontology to consider after that of system, first because it is also central to the idea of software maintenance, second because it is closely related to the system sub-ontology, and third because the two source ontologies had a sub-ontology relating to the maintenance process. This time, the two source ontologies had more differences since Dias *et al.* consider only one such sub-ontology whereas Ruiz *et al*. have two (see Figure 1). The idea of working iteratively in this case is important because it helps focus on a smaller part of the whole ontology. This is where the process allows better control on the whole project.

Existing tools are deficient in this regard since they appear to consider either the entire source ontologies to be merged (losing the focus we have just described) or would work on two given sub-ontologies (as in our first iteration), thus loosing some information since in this case the mapping is from two sub-ontologies (Ruiz *et al.*) to one (Dias *et al.*).

To merge the Process sub-ontologies, we again started from core concepts which we defined to be the “maintenance process” and its “activities”. Interestingly, the very activity of looking for core concepts showed that if both source ontologies defined the “activity” concept neither had thought above the “process” concept. We agreed that this was an important concept to add. From these two concepts, we progressively added related concepts (for example a decomposition of the activities). This was more difficult than with the first sub-ontologies as we had to select concepts coming from both source ontologies, or either of them, or reconcile differing views on concepts. Things were even more difficult as we found that some interesting concepts were actually members of a fourth sub-ontology (e.g. the technology concept in Dias *et al.*'s Skills sub-ontology). We needed two iterations to settle the core of this sub-ontology, and one more for the minor issues.

The final sub-ontology to be merged is that of Organization which includes concepts on the roles needed to perform the activities, positions in the organization, etc. It is similar to the second one in that it consists of merging two sub-ontologies from one source with one sub-ontology from the other. It is also made more difficult by the fact that some concepts of these sub-ontologies have already been merged in the Process sub-ontology. Again, two iterations were necessary to agree on the core issues. Finally one more iteration was necessary to complete the work and solve the minor issues. In total, we needed six iterations.

4. Lessons learned

We have drawn the following conclusions from this merging experiment and how we dealt with the difficulties identified in Section 3.2:

- Core elements: As one would expect, in most cases the core elements (either sub-ontologies or concepts) we identified were the things the two source ontologies had in common. This is actually an approach adopted by most merging technologies. However, we found at least one example of a core concept (Process sub-ontology) that was not present in either source ontologies.
- Length: Some sub-ontologies required two iterations before we reached an agreement on the core issues and one more for the minor ones. Each iteration in this model corresponds to the analysis of a sub-ontology by one team, and the analysis of the resulting proposal by the other team (round-trip). Each of these analyses by one team could require weeks depending on the work load at that particular time. Typically, one month or more could pass between one group's proposal and the answer to this proposal. This was inevitable given the communication medium we chose (e-mail) and the geographical dispersion.
- Rework: The long intervals between each participation in the merging ultimately increased our work load as the first thing a team would have to do when commenting on a proposal would be to re-analyze the entire process to remind themselves of what had already been discussed, what had been proposed (and why) and what argument had been exchanged. In short, to reconstruct the entire discussion up to that point. A tool to help record and then reconstruct the discussion would be a great help in this sense.
- Communication channel: An ontology captures consensual knowledge in a domain. Reaching this consensus implies sharing ideas, confronting opinions, arguments and counterarguments. This is an activity that requires (a lot of) communication between the participants. Geographical dispersion is a huge obstacle to this communication. Because our objective was to actually merge the ontologies, we have not had time to research and develop a methodology that would alleviate this communication problem. Our method was a simple transposition to e-mail of what could have happened if we had been able to discuss the problem "eye-to-eye". This is definitely not enough, although it does offer some advantages as will soon be discussed.
- Iterative progress: Because the merging spanned a long time frame, it was important to have a structuring framework for discussion. One does not conduct a slow, lengthy, e-mail discussion in the same way as one would carry out an "eye-to-eye" exchange. It was important to have a clear sense of what we were doing at any given point, where we were going and how much was still needed to get there. This is one of the benefits of the iterative approach we used.
- Incremental progress: Usually, as one team was working on a piece of the ontology, the other would simply wait ("idly") for the next round of discussion. We did not concurrently start discussions on various sub-ontologies. This would not fit the incremental approach we chose. It is not clear whether this was a good decision or not. However the iterative and incremental approach allowed us to deal with the risk of making a wrong decision at one point that would imply re-working an entire sub-ontology. It is impossible from our limited experience to say if working concurrently on two sub-ontologies would not have increased such a risk.
- Process formality: Although we presented rather strict definitions of our process, its iterations and how they happened, in reality things are not so clear cut. For example we defined one iteration as a round trip of comments between the two groups. However, the real unit of activity was one analysis by a group ("half an iteration"). For example, a group would start a new iteration (discussion of the core issue for a sub-ontology) and at the same time (in the same e-mail) it would close the iteration of the preceding sub-ontology.
- Asynchronous communication: E-mail communication has the well known advantage of being asynchronous. One does not need to set appointments or wait for others. Given that the two teams are in different time zones this was a very good thing and proved useful.
- Historical record: E-mail communication and written communication in general also offers the advantage of being easily archived. This is important when one needs to go back to past decisions and remember how they were arrived at.

5. Conclusions

Merging methodologies is useful to guide the merging activity and to carry it out in a systematic and ordered way. Some automatic tools have been proposed with the goal of making that activity easier. However, the merging process is very similar to developing a new ontology [7] since it is necessary to understand the source ontologies clearly, to decide the level of granularity of the final ontology and to make a lot of design decisions.

In this paper we have reported our experience in merging two ontologies on software maintenance in real life conditions. These conditions included: geographical dispersion of the participants, semi-formal definition of one ontology, two ontologies which were organized differently with few concepts clearly in common. In these conditions, we have found the use of automated merging tools to be of little value as most of the work consisted of discussion between experts to define what concepts to keep from either source ontology, what the exact definition of some concepts was, or at what level of granularity we wanted to work.

To carry out the merging, we defined an iterative and incremental process where the ontologies are merged iteratively by sub-ontologies and each iteration consists of an incremental approach from some core concepts to the entire sub-ontology.

We closed the paper with a discussion of our experience, highlighting some benefits and drawbacks of our approach.

6. References

[1] Becker-Kornstaedt, U., Webby, R.: A Comprehensive Schema Integrating Software Process Modelling and Software Measurement. . Fraunhofer IESE-Report Nº 047.99 http://www.iese.fhg.de/Publications/Iese_reports/ Fraunhofer IESE., (1999).

[2] Chalupsky, H.: OntoMorph: A Translation System for Symbolic Knowledge. KR'00, USA (2002).

[3] Deridder, D.: A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. Workshop on Knowledge-Based Object-Oriented Software Engineering at 16th European Conference on Object-Oriented Programming (ECOOP 2002), Málaga Spain (2002).

[4] Dias, M.G., Anquetil, N., et al.: Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science, Vol. 9 (2003) 641-658.

[5] Doan, A., Madhavan, J., et al.: Learning to Map Between Ontologies on the Semantic Web. Eleventh Intenational WWW Conference, Hawaii USA (2002).

[6] Falbo, R.A., Menezes, C.S., et al.: Using Ontologies to Improve Knowledge Integration in Software Engineering Environments. 4th International Conference on Information Systems Analysis and Synthesis (ISAS'98), Oraldo Florida (1998).

[7]Gómez-Pérez, A., Fernández-López, M., et al.: Ontological Engineering. (2004).

[8] IEEE: 1219 - Standard for Software Maintenance. IEEE - Institute of Electrical and Electronics Engineers (1998).

[9] ISO/IEC: 15504-2: Information Technology - Software Process Assessment - Part 2: A Reference Model for Processes and Process Capability. (1998).

[10]ISO/IEC: FDIS 14764: Software Engineering Maintenance (draft), Dec-1998. (1998).

[11]Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)

[12]Kajko-Mattsson, M.: Common Concept Apparatus within Corrective Software Maintenance. IEEE International Conference on Software Maintenance (ICSM'99), Oxford UK (1999).

[13]Kajko-Mattsson, M.: Towards a Business Maintenance Model. IEEE International Conference on Software Maintenance (ICSM), Florence Italy (2001).

[14]Kitchenham, B.A., Travassos, G. H., et al.: Towards an Ontology of Software Maintenance. Journal of Software Maintenance: Research and Practice, Vol.11 (1999) 365-389

[15]McGuinness, D.L., Fikes, R., et al.: An Environment for Merging and Testing Large Ontologies. (2000)

[16]Noy, N., Musen, M.: An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support. Workshop on Ontology Management, WS-99-13. AAAI Press, (1999)

[17]Noy, N., Musen, M.: PROMPT: Algotithm and Tools for Automated Ontology Merging and Alignment. Workshop on Ontologies and Information Sharing, Seattle Washington (2000).

[18]Pfleeger, S.L.: Software Engineering: Theory and Practice. Prentice-Hall (2001).

[19]Pigoski, T.M.: Practical Software Maintenance: Best Practice for Managing Your Investment. Jhon Wiley & Sons, New York USA (1997).

[20]Polo, M., Piattini, M., et al.: MANTEMA: A Complete Rigorous Methodology for Supporting Maintenance Based on the ISO/IEC 12207 Standard. Third Euromicro Conference on Software Maintenance and Reengineering (CSMR'99). IEEE Computer Society, Amsterdam Netherlands (1999).

[21]Pressman, R.S.: Software Engineering: A Practitioner's Approach, 5th edition. (2001).

[22]Ruiz, F., Vizcaíno, A., et al.: An Ontology for the Management of Software Maintenance Projects. International Journal of Software Engineering and Knowledge Engineering, Vol.14 (2004) 323-349.

[23]Steve, G., Gangemi, A., et al.: Integrating Medical Terminologies with ONIONS Methodology. Information Modeling and Knowledge Bases VIII. IOS Press (1998).

[24]Stumme, G., Meadche, A: FCA-MERGE: Bottom-Up Merging of Ontologies. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001). Morgan Kaufmann Publishers, Seattle, Washington (2001).