Developing *Graduation Screen Ontology* based on the METHONTOLOGY Approach

Jinsoo Park, Kimoon Sung, Sewon Moon
College of Business Administration, Seoul National University
{jinsoo, majsung, sece80}@snu.ac.kr

Abstract

A clearly-defined and well-structured methodology can reduce ontology development time and increase the possibility of the project success. However, there exists no mature knowledge engineering methodology for ontology developments. Nevertheless, METHONTOLOGY, known as one of the best methodologies, has been adopted in several ontology developments due to its domain-independent characteristics. In this paper, we develop the Graduation Screen Ontology (GSO) based on the guidelines suggested by METHONTOLOGY to analyze the ontology development life cycle suggested in METHONTOLOGY and to identify the issues related to the METHONTOLOGY approach, OWL-DL, and Protégé-OWL. Based on our GSO development experience, we found several drawbacks of the METHONTOLOGY approach. We also address some issues related with Protégé-OWL and OWL-DL. We finally discuss contributions and practical implications of this study.

1. Introduction

Ontologies are now central to many applications such as scientific knowledge portal, information management systems, electronic commerce, and semantic web services. In spite of the increasing importance of ontologies, ontology construction is still a craft rather than an engineering activity (Fernández-López et al., 1999).

A characteristic of engineering is that it provides methodologies, languages, and tools to perform tasks required in some areas (Corcho et al., 2003). Ontological engineering refers to the set of activities, the ontology life cycle, methods and methodologies for constructing ontologies, and tool suites and languages that support them (Gómez-Pérez et al., 2004).

A clearly-defined and well-structured methodology can reduce ontology development time and increase the possibility of the project success. Unfortunately, however, there exists no mature knowledge engineering methodology for ontology developments (Fernández-López et al., 1999; Gómez-Pérez et al., 2004). METHONTOLOGY, nevertheless, has been adopted in several ontology developments due to its domain-independent characteristics (Fernández-López et al., 1997; Fernández-López et al., 1999; Corcho et al., 2003).

In our study, criteria that we considered for selecting an ontology development methodology were the degree of adoptability by non-experts, the levels of details of the methodology, whether to support iterative development tasks, recommendations for formalization, tool supports, recommended techniques, and so on. Based on the criteria, we found that the most appropriate methodology was METHONTOLOGY.

We developed an ontology to represent college graduation screening process. We named it Graduation Screen Ontology (GSO). To implement our GSO, we chose OWL-DL as an ontology development language and Protégé-OWL as an implementation tool. We opt for the graduation screen domain with the following reasons. First, the graduation screen process is a complex task because it sometimes requires a complicated reasoning process. Second, since the graduation screen process for most of the universities is similar, we could reuse GSO in other universities. Finally, because the size of the selected domain is reasonable, we could build GSO within a given period of time.

The objective of this study is, based on our development experience, to analyze the ontology development life cycle suggested in METHONTOLOGY and to identify issues related with the METHONTOLOGY approach, OWL-DL, and Protégé-OWL.

This paper is organized as follows. Section 2 presents a brief overview of METHONTOLOGY, OWL-DL, and Protégé-OWL. Section 3 describes the entire ontology development life cycle of GSO. Section 4 suggests lessons learned from this project. Finally, Section 5 summaries our study and suggests future works.

978-0-7695-3322-3/08 $25.00 © 2008 IEEE
DOI 10.1109/NCM.2008.215

2. GSO Development Background

2.1. METHONTOLOGY

METHONTOLOGY was originally derived from the experience of developing Chemical Ontology at Polytechnic University of Madrid (Fernández-López et al., 1999). It was first introduced in 1999. Then, in 2004, it was expanded and classified into three broad processes. Each process contains specific activities: (1) the management process includes scheduling, control and quality assurance, (2) the development process is divided into specification, conceptualization, formalization, implementation, and maintenance, and (3) the support process is composed of five activities, i.e., knowledge acquisition, evaluation, documentation, configuration management, and integration. METHONTOLOGY adopts some suggestions from the IEEE standard for software development (IEEE, 1996). In this paper, we mainly focus on the development process because that is the most important process in METHONTOLOGY.

2.2 OWL-DL

OWL is an ontology language for the Semantic Web, developed by W3C Web Ontology Working Group. OWL is classified into three sub-languages: OWL-Lite, OWL-DL and OWL-Full. OWL-Lite is the syntactically simplest sub-language. OWL-DL is much more expressive than OWL-Lite and based on Description Logics (DL). DL is a decidable fragment of the first order logic, and therefore, amenable to automated reasoning. OWL-Full is the most expressive sub-language. However, because it is impossible to perform automated reasoning on OWL-Full sub-language, we selected OWL-DL as our ontology development language.

2.3 Protégé-OWL

Although METHONTOLOGY recommends WebODE as a technological framework, we chose Protégé-OWL with the following reasons. First of all, Protégé-OWL is more harmonized with OWL-DL than other tools. Second, it is platform-independent so that developers can use it on any platform. Third, because Protégé-OWL is a free and open ontology editor, this tool has high accessibility compared with other ontology development tools. Moreover, Protégé-OWL supports diverse plug-ins. The most useful plug-in for our project is OntovizTab which displays an ontology with graphical views. Although the development of Protégé-OWL has been historically driven by biomedical applications (Gennari et al., 2003), the system is domain-independent and has been successfully adopted for many other application areas as well.

3. An Ontology Building Process

This section presents the detailed descriptions of our GSO development process. The GSO process was mainly guided by the METHONTOLOGY approach. The life cycle of GSO development includes a pre-development process (i.e., scheduling activity), a full development process (i.e., specification, conceptualization, formalization, and implementation), and a post-development process (i.e., evaluation activity). We will now explain how we have developed GSO based on the guidelines suggested in the METHONTOLOGY approach.

3.1 Pre-Development Process: Scheduling

As recommended by METHONTOLOGY, we first performed a scheduling activity in order to plan the main tasks for the GSO development. The objective of the scheduling activity is to identify the essential tasks, to organize those tasks, and to allocate time and resources (e.g., people, a development tool, an ontology language, etc.). We chose OWL-DL as an ontology language and Protégé-OWL as a development tool.

3.2 Development Process

3.2.1 Specification. In this phase, the domain and the scope of ontology are identified, which include the purpose of the ontology, the intended users, etc. (Fernández-López et al., 1999). Accordingly, we stepped into the specification phase to perform domain analysis, to consider reusing existing ontologies, and to acquire knowledge from the domain. This allowed us to better understand the domain and the existing graduation systems. To obtain an ontology specification document, we carried out several sub-activities, such as domain analysis, search for existing ontologies and knowledge acquisition.

The target domain of our GSO project is a graduation screening process of the largest university in Korea. GSO needs to capture all the information relevant to college graduation. The graduation screen process is so complex because various graduation requirements should be checked. These graduation requirements are**:** the list of required courses, total credits to be taken, a minimum grade point average, the number of enrollments, and so on. The ultimate purpose of GSO is to determine whether students can graduate and to inform which requirements need to be fulfilled according to their graduation requirements.

Target users for GSO are those students, faculty, and administration staffs responsible for the graduation screening process. They should be able to identify the entire educational history of a student precisely whenever they want to. Furthermore, they should be able to detect what other qualifications should be fulfilled to meet the graduation requirements.

We also checked whether similar ontologies or systems exist in this domain that can be reused. Unfortunately, however, there exists no such an ontology**,** but we found an academic portal serving for education administration. This portal provides information, such as grade point average (GPA), course credit hours, and the list of courses the student has taken or is currently enrolled in. The portal, however, does not provide information about graduation requirements. If students want to know about their academic status before graduation, they should visit the administration office to check their academic status or verify their educational history by themselves.

We acquired most of the knowledge during conceptualization. Information was obtained from interviews with persons-in-charge and various sources such as the university Websites, administration offices, libraries, and so on. The most challenging issue was that graduation requirements were varied by different majors and colleges. For example, different graduation requirements existed for different departments even in the same college. In the case of the College of Business Administration, the number of enrollments should be a minimum of eight semesters and a maximum of sixteen semesters. Total credit hours for graduation are 130 credits or more and at least 48 credits should be taken for major courses. On the other hand, the College of Social Sciences requires students to take only 39 credits for their majors. Based on the information we gathered, we found that the university has 16 colleges with more than 100 majors and 1,000 courses.

3.2.2 Conceptualization. The objective of this activity is to organize and structure the knowledge acquired from the external representations that are independent of the knowledge representation and implementation. The conceptualization stage begins with converting informal data into semi-formal specifications using a set of intermediate representations (IRs) based on tabular and graph notations provided by METHONTOLOGY. These IRs (i.e., concepts, attributes, relations, axioms and rules) are valuable because they are easily understood by both domain experts and ontology developers. The difference between the ontology expert and the ontology developer is that an ontology developer is a person who is highly experienced in ontology related tasks (e.g., ontology modeling and ontology development languages), while a domain expert is a person who is specialized in domain knowledge but does not necessarily have sufficient ontology development experience. IRs bridged the gap between people's domain perception and ontology implementation.

Corcho et al. (2005) suggested eleven specific tasks during the conceptualization stage to develop a conceptual ontology as shown in Figure 1. We adopted these eleven tasks as well as tabular forms and graphs which are core expressions for conceptualization.

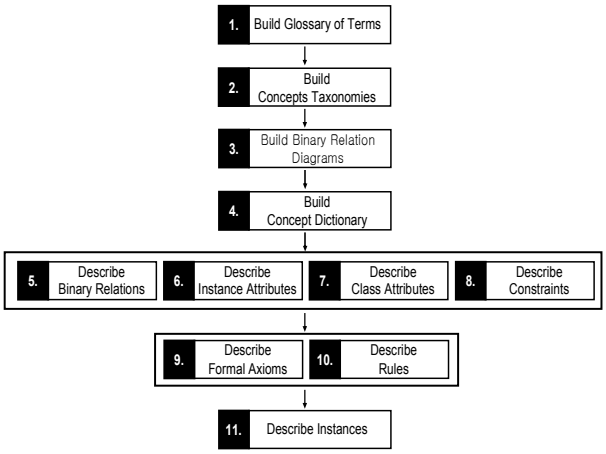

Figure 1. Sub-tasks for conceptualization

In order to build a consistent and complete conceptual ontology, the conceptualization activity must define a set of tasks that should be executed in succession. These tasks increase, step by step, the complexity of the IRs used to build a conceptual ontology. Accordingly, we developed a concept dictionary, a concept taxonomy graph, a binary relations diagram, a binary relation table, a class attribute table, a constant table, an axiom table, and so on.

3.2.3 Formalization and Implementation. IRs developed for GSO during the conceptualization were finally formalized and implemented. We used Protégé-OWL to convert our formal model into an OWL-DL.

GSO consists of seven classes: *Course*, *Credit*, *Enrollment*, *FinalTest_Or_Thesis*, *Grade_Point_ Average*, *Graduation_Requirement*, and *Student*. Figure 2 shows the hierarchy of GSO. Protégé-OWL supports graphical representation of a class hierarchy though OWLVIZ plug-in. This visualization function helps developers and users understand the structure of the ontology more easily than merely showing a text-based ontology structure.

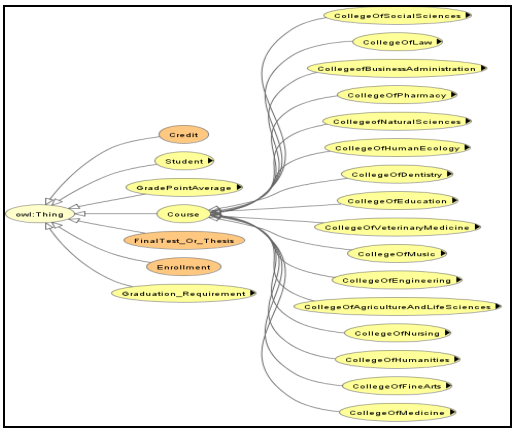

Figure 2. A hierarchy of GSO

In order to define each course, we specified the graduation requirements for each college based on the necessary and sufficient condition included in the "closure axiom" notion. It was intended to express the definition and the axiom of a specific class at the same time. Therefore, generating a closure axiom of a class means defining a class with several restrictions that consist of properties and axiom. For example, the closure axiom of the property *takeRequiredCourse* for the required courses is a universal restriction that acts along the property *takeRequiredCourse*. Also, we used a union restriction to specify the class *Graduation_Requirement*. In the property of *getGPA,* all the subclasses of the class *Grade_Point_Average* were represented with the union restriction.

Since OWL-DL does not provide any method to represent numbers, we used the annotation property and the cardinality restriction to represent numeric expressions in GSO. For example, one of the graduation requirements was that each student had to acquire a GPA of at least 2.0 which is equivalent to a letter grade C0. Since GPA is one of numeric types, we are forced to define GPA with the annotation property. Each subclass of the class *Grade_Point_Average* was represented by defining letters such as *GPA_Aplus*, *GPA_Aplus GPA_AZero*, and *GPA_AMinus* instead of 4.3, 4.0, 3.8, etc. GPA and the range of GPA were described by annotation property.

Properties connect instances from the domain to instances from the range. In GSO, after defining properties, we specified domain and range. Because the core issue for GSO was how to define the graduation requirements for each college, all of the properties had the class *Graduation_Requirement* as their domain. However, in case of the property *hasCredit*, since each course had its own credit, it was related to the class *Credit*, and the property *takeCourse* had inverse properties whose sub-properties consisted of a hierarchy structure.

3.3 Post-Development Process: Evaluation

Ontologies should be evaluated before they are used or reused. Ontology evaluation is composed of three steps: ontology verification, ontology validation, and ontology assessment (Gómez-Pérez and Pazos, 1995). Ontology verification is to ensure whether the ontology is implemented correctly according to all the expected requirements. Ontology validation refers to whether the ontology concepts really represent the real world for which the ontology was created. Ontology assessment focuses on judging the ontology from the user's point of view. There are two kinds of ontology evaluation: the technical evaluation and the user evaluation.

3.3.1 Technical Evaluation (Verification and Validation). The technical evaluation is performed by developers. The criteria of the technical evaluation are consistency, completeness and conciseness (Gómez-Pérez, 1994). Once the formal GSO was developed, we checked consistency, completeness, and conciseness of GSO in order to verify constructs of GSO and validate functions of GSO.

First of all, we checked whether all individual concepts were consistent and ensured no contradictory concepts could be inferred from other concepts and axioms. Then, we ascertained the classification hierarchy and disjoint knowledge to compute completeness. We found several completeness errors (e.g., incomplete concept classification and disjoint concept omission, etc.) in GSO and rectified those errors. Finally, we checked unnecessary concepts and explicit redundancies between concepts to verify the conciseness of GSO. We assured that there were no redundant subclasses, no redundant instances of relations, and no duplicate formal concepts of classes or instance.

3.3.2 Case Analysis Evaluation (Assessment). Since most users cannot evaluate the logical correctness of an ontology, we evaluated GSO qualitatively by user cases. To assess GSO from the user's point of view, we developed four user cases i.e., each case represents one graduation candidate. The two cases (named 'A' and 'B', majoring in Science of Earth Education and Korean Language and Literature, respectively) are eligible for graduation, which means both students fulfilled all their graduation requirements. While the third case (named 'C') and the fourth case (named 'D') majoring in Business and Economics respectively are not eligible for graduation.

We used Racer to perform reasoning on the four user cases. As expected, only cases A and B were correctly inferred. Figure 3 shows the inferred hierarchy

which is automatically computed. We can see two new arrows generated by Racer. This indicates that the case A was inferred as a subclass of the class *Major_Of_Earthscienceeducation* and the case B was also correctly reasoned as a subclass of the class *Major_Of_Koreanlanguageliterature*. Based on this result, we confirmed that the cases A and B accurately refer to the corresponding classes that meet graduation requirements. Our team also found that the cases C and D failed to meet the requirements. Because we originally expected that the user case A and B would be successfully inferred, while the user case C and D would result in no change after reasoning, we concluded that GSO has been successfully implemented.

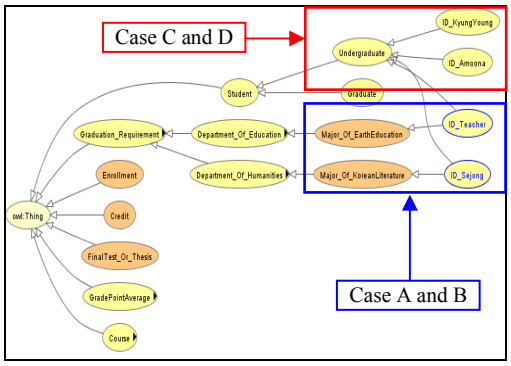

Figure 3. User cases analysis (Inferred hierarchy)

4. Discussion

In this section, we analyze METHONTOLOGY with critical view based on our GSO development experience in order to address some issues of METHONTOLOGY. We also discuss some problems of OWL-DL and Protégé-OWL.

First of all, we present several drawbacks of the METHONTOLOGY approach. First, METHONTOLOGY does not provide specific guidelines for assigning individual developers to certain tasks. METHONTOLOGY consists of three different processes (i.e., management, development, support process). According to METHONTOLOGY, those processes are carried out simultaneously and each of them is performed by various developer groups (e.g., domain experts, ontology experts, etc). However, it does not explain how to allocate those groups to specific tasks, how to combine those tasks together once divided works are completed, and so on. Second, one of the important motivations to adopt METHONTOLOGY is that even a domain expert is supposed to build an ontology easily without help from ontology experts. However, according to our experience, that is not the case. Even though METHONTOLOGY could be fully understood by a domain expert, it is not easy to develop an ontology by the domain expert alone without help from an ontology specialist. Third, METHONTOLOGY focuses heavily on conceptualization and this could be a problem. We found that the IRs developed during the conceptualization phase are sometimes not seamlessly codified into an ontology language. This can cause delays during the formalization and implementation stages. Fourth, METHONTOLOGY does not provide sufficient explanations about methods and techniques applied in each stage, such as a detailed technique on how to extract concepts and methods on identifying relations among concepts. Fifth, METHONTOLOGY allows continuous knowledge acquisition while working on ontology development process. This could be problematic for developers because whenever new knowledge arrives, the update process should be repeated throughout the entire development process, not just in the relevant activity. Sixth, even though METHONTOLOGY emphasizes the role of evaluation which is a part of support process, it fails to provide specific validation and verification method for user evaluation. Seventh, METHONTOLOGY asks developers to fill out many documents (e.g., ontology requirement specification documentation during specification stage and various IRs during conceptualization stage, and so on), which are not always necessary.

We also found out that there are some shortcomings in OWL-DL and Protégé-OWL. The major problem encountered when constructing GSO was that OWL-DL could not express numbers. We defined concepts representing numbers during the conceptualization stage, but numbers could not be transformed into OWL-DL codes. Thus, those concepts were alternatively represented as cardinality or annotation, especially for number comparison (e.g., total credit hours) and floating data-type representation (e.g., GPA). Another issue related with OWL-DL is a *separation* problem. Since the roles of classes and instances are strictly differentiated in OWL-DL, instances used at individual level cannot be used as concepts at the class level. This inevitable problem can beget another problem, i.e., students who entered the school at different years might be assigned different required courses, which are hard to handle. For example, Kim and Park who entered the school in 2005 and 2006 respectively are juniors majoring in business. Although they are both juniors, “E-Commerce” became a required course in 2006 due to the change in the curriculum. Thus, Park must take the course that was elective for Kim. Therefore, even though “E-Commerce” is a single course, we had to create two different instances: *E-Commerce_required* and *E-Commerce_elective*. during first iteration. To resolve this issue, we redefined each course as a class during the second iteration so that

new course requirements for graduation can be more easily handled.

One of the critical problems of using Protégé-OWL is that when the size of the ontology gets bigger, reasoning and storing the ontology take much longer time and the tool is often ended up with unexpected shutdown due to the tool's instability. This instability leads developers to repeat same tasks for several times.

5. Conclusion

In this study, we empirically examined the utility of METHONTOLOGY by actually developing an ontology for real-world use since we would like to analyze the ontology development methodology, the ontology development language, and the development tool.

There are several contributions in this research. First, although previous studies usually focused on the use of rather than the analysis of METHONTOLOGY, for the first time, we attempted to analyze the methodology with critical view in order to find its strengths and weaknesses. Second, based on a variety of experiences, this study discusses lessons learned from the development experience and presents useful suggestions to practitioners who want to develop an ontology based on the METHONTOLOGY approach on their projects. Our study also gives some insight to ontology methodology researchers who want to build up a more advanced ontology development methodology. Third, GSO can be reused by other universities or similar domains since it is developed based on general concepts commonly used in the domain.

However, there are some limitations of the GSO project. First, although METHONTOLOGY encouraged developers to use WebODE as a building tool, we had to adopt Protégé-OWL because WebODE was not available. Even though Protégé-OWL is highly available and one of the most widely used ontology development tools, the adoption of Protégé-OWL instead of WebODE might affect our conclusion on the issues of METHONTOLOGY. Second, since none of previous research tried to develop an ontology for the graduation screen process, the quality of GSO cannot be compared and evaluated.

In conclusion, this study provides several contributions to the ontology methodology and practical implications in the ontology development research field. Therefore, we believe that our study will promote further research on this area. In the future, a comparison between WebODE and Protégé-OWL should be made to examine their differences when adopting the METHONTOLOGY.

References

[1] Corcho, O., M. Fernández-López, A. Gómez-Pérez, and A. López-Cima, *Building legal ontologies with METHONTOLOGY and WebODE* in Law and the Semantic Web. Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications, Benjamins, V. R., Casanovas, P., Breuker, J., and Gangemi, A. (Eds.), March 2005, Springer-Verlag, LNAI 3369. pp. 142-157.

[2] O. Corcho, M. F. López and A. Gómez-Pérez, "Methodologies, tools and languages for building ontologies. Where is their meeting point?," *Data and Knowledge Engineering*, vol. 46, issue 1, July 2003, pp. 41-64.

[3] Fernández-López, M., A. Gómez-Pérez, and N. Juristo, *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*, Symposium on Ontological Engineering of AAAI, Stanford (California), March. , 1997.

[4] M. Fernández-López, A. Gómez-Pérez, and A. Pazos-Sierra, "Building a Chemical Ontology Using Methontology and the Ontology Design Environment," *IEEE Intelligent Systems*, Vol. 1(January/February), 1999, pp. 37-46.

[5] Gómez-Pérez, A, *Some Ideas and Examples to Evaluate Ontologies*, tech. report KSL-94-65, Knowledge System Laboratory, Stanford Univ, 1994.

[6] A. Gómez-Pérez and J. Pazos, "Evaluation and assessment of knowledge sharing technology," In Mars, N.J. (ed.): "Towards Very Large Knowledge Bases," *IOS Press,* 1995, pp. 289-296.

[7] Gómez-Pérez, A., M. Fernández-López, and O. Corcho, Ontological Engineering: with examples from the areas of knowledge management, London: Springer-Verlan, 2004.

[8] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M. Crub´ezy, H. Eriksson, N. Noy, and S. Tu, "The evolution of Protégé-2000: An environment for knowledge-based systems development," *International Journal of Human-Computer Studies*, Vol. 58(1), 2003, pp. 89–123.

[9] IEEE, *IEEE Standard for Developing Software Life Cycle Processes*. IEEE Computer Society. New York (USA). IEEE Std 1074-1995, 1996.