Ontology Engineering Assignment 5

Adriaan Louw (53031377)

September 1, 2018

1 Question 1

1.1 Methontology

Methontology (Fernández, Gómez-Pérez, & Juristo, 1997) is a methodology to describe the steps involved in building an ontology. This methodology can broadly be describe by looking at the stages it prescribes the ontology designers to follow. The first stage is the specification state. In this stage a specification for the ontology is developed which includes things like the purpose of the ontology and its scope. Next is the conceptualisation state, where a conceptual model of the domain in question is developed. This includes a Glossary of Term that defines all the concepts, instances, verbs ect. This is followed by the Integration stage. Here the developers of the ontology need to look at existing ontologies and attempt to reuse as much as possible from those ontologies. Additionally meta-ontologies should be investigated to ensure that the concepts, in the new conceptual model from the previous stage, are compatible with the meta-ontologies. Knowledge acquisition is a stage that runs in parallel with the other stages. In this stage various external sources of information like books or experts are consulted and added to the conceptual model. Following on from the integration stage is the implementation stage. Here the ontology is actually implemented using the chosen technology eg. Prolog. The final stage is the evaluation stage where it is determined wether the ontology has satisfied the initial specification and if not how to correct the ontology (Fernández et al., 1997).

1.2 NeOn

NeOn (Gómez-Pérez & Suárez-Figueroa, 2009) is also a methodology for the creation of ontologies. According to Gómez-Pérez and Suárez-Figueroa (2009) one of the problems with exiting methodologies (including Methontology) is that they do not cater for large ontologies or even address the problem of having distributed teams working in collaboration from different locations. NeOn attempts to address these problems. NeOn breaks the development process up into a set of 9 scenarios. With additional steps to follow within each scenario. In this section we will only briefly describe some of the scenarios. The first scenario is called 'Form specification to implementation' and should be followed when a new ontology is being created without utilizing any existing ontologies. Scenario 3 describes how to reuse ontological resources whether only parts of other ontologies are used or the ontology as a whole. Scenario 7 details how to reuse ontology design patters and scenario 9 describes how to localise an ontology to a new language or culture Gómez-Pérez and Suárez-Figueroa (2009).

1.3 Comparing Methontology and NeOn

The approaches of Methontology and NeOn are quite different. Methontology start with the specification phase. Whereas NeOn forces the designer to determine which scenario their particular problem fits. Forinstance is the ontology using new resources or some form of existing resource (Fernández et al., 1997) (Gómez-Pérez & Suárez-Figueroa, 2009).

NeOn also uses instructs users to use other methodologies (Gómez-Pérez & Suárez-Figueroa, 2009). For instance, in the first scenario dealing with creating ontologies from scratch. After creating the ontology requirements specification document and completing the scheduling task, NeOn prescribes using the conceptualization, formalization and implementation stages of other methodologies like Methontology.

NeOn includes 2 types life cycle models (Suárez-Figueroa, Gómez-Pérez, & Fernández-López, 2012). The waterfall life cycle models are models where the process is broken up into discrete stages and each stage has to be completed before the next stage can begin. The different waterfall life cycles vary in length from 4 to 6 stages. Depending on the amount of reuse of other ontologies. The iterative life cycle differs from the waterfall life cycles in that in each iteration only a subset of the requirements are defined, implemented and evaluated. the life cycle in Methontology resembles a waterfall life cycle. Waterfall life cycles are best for ontology projects where the scope of the project is closed and it has a short duration. Whereas iterative life cycles are better suited to projects with a large amount of developers and the scope i.e. requirements are not completely known (Suárez-Figueroa et al., 2012).

References

Fernández, M., Gómez-Pérez, A., & Juristo, N. (1997). Methontology: from ontological art towards ontological engineering.

Gómez-Pérez, A., & Suárez-Figueroa, M. C. (2009). Neon methodology for building ontology networks: a scenario-based methodology.

Suárez-Figueroa, M. C., Gómez-Pérez, A., & Fernández-López, M. (2012). The neon methodology for ontology engineering. In *Ontology engineering in a networked world* (pp. 9–34). Springer.

1 Question 2

There are various techniques to debug an ontology. Here follows a discussion of various techniques used to debug ontologies in the field.

As Parsia, Sirin, and Kalyanpur (2005) describes, improving the way the ontology is visually represented and how readable it is, can make it easier to spot mistakes in the ontology. For example, when Swoop is in debug mode, any inferred relationships for an entity definition are italicised. Also named classes that are unsatisfiable are marked with a red icon.

Backjumping is a technique used to determine which clashes between axioms in the ontology is causing an inconsistency in the ontology (Parsia et al., 2005). There are various types of clashes for instance when an individual is a member of a class an that class' complement. This is called an *Atomic* clash. Or a *Cardinality* clash where the individual is related to more individuals than its max cardinality permits. A reasoner can find multiple clashes in an ontology, including clashes that are not to be regarded as errors in the ontology but clashes that help the tableau rules to find the correct model. Backjumping adds additional labels to the various types and property assertions of entities in the ontology. This allows the reasoner, as it follows the branches by utilizing the tableau rules, to track the various branches that created the clashes. The reasoner will stop when a clash that does not depend on a non-deterministic branch arises. This is a popular technique used in many well known reasoners. Additionally reasoners need to keep track from which assertions these problematic axioms are derived. In order for the reasoner to be able to tell the user which assertions have caused the issue (Parsia et al., 2005).

Versioning of the an ontology can also be of benefit. Between each version of the ontology a log is created. This log can help the person who is debugging the ontology to find the error by listing all the areas that were updated in the previous versions. This can help to narrow down where the potential problem or problems are (Parsia et al., 2005).

Another approach, described in Guarino and Welty (2009), is instead of trying to debug or find an error in the ontology after the fact, rather to approach the creation and maintenance of an ontology through a methodology like OntoClean. OntoClean attempts to classify entities to help the creator of the ontology to realise the logical consequences of certain design decisions. The properties of these entities can be classified in various ways. Firstly, is the property of the entity in question essential to that entity. In other words which properties have to hold true in "every possible world" that entity can be in. Those properties that have to hold true are called rigid while those that can lose instances due to other external influences, like time, are called non-rigid. While those that properties that are not essential to their instances at all are called anti-rigid properties.

Other way to classify entity properties in the OntoClean methodology are via their identity or unity criteria. To determine the identity is to determine whether entities are the same or different entities and which properties determine that. These properties that determine identity is then marked for further analysis. The unity criterion is also of importance. The unity criterion describes which parts of an entity form an entity (Guarino & Welty, 2009).

As ontologies become larger and more complex, debugging techniques like these become even more essential.

References

Guarino, N., & Welty, C. A. (2009). An Overview of OntoClean. In *Handbook on ontologies* (pp. 201–220). doi: 10.1007/978-3-540-92673-3_9

Parsia, B., Sirin, E., & Kalyanpur, A. (2005). Debugging OWL Ontologies. In *Proceedings of the 14th international conference on world wide web* (pp. 633–640). New York, NY, USA: ACM. doi: 10.1145/1060745.1060837

1 Question 3

Description Logics (DL) is the name of a family of knowledge representation formalisms (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003). It describes the problem domain by defining the domain concepts. Then properties of objects and individuals are defined based on these concepts. Axioms (also called sentences) are used to define concepts and roles. For instance if C and D are concepts and concept C is subsumend D then we denote it as

$$C \sqsubseteq D \tag{1}$$

The key inference type in DL is the subsumption relationship (Baader et al., 2003). The collection of axioms for a specific DL implementation forms a knowledge base (KB).

OWL is based on Description Logics for various reasons. Firstly, OWL needs to be based on some form of logic so that it is able to reason about these concepts and deduce implicit knowledge from them. The most expressive form of logic is First Order Logic. While first order logic is capable of the kind of knowledge representation and inference that is needed for systems like OWL, a lot of the machinery of first order logic are not required leading no unnecessary bloat (Baader et al., 2003).

Inference problems like class satisfiability and subsumption need to be decidable (Horrocks, 2005). In First Order logic these are not necessarily decidable.

Also inference problems needed to be solvable in a reasonable amount of time. Even in worse case scenarios. Baader et al. (2003) showed that if a Description Logics based system is "highly optimised" then this is achievable.

Descriptive logics allows reasoners to reach conclusions within an appropriate amount of time without burdening the reasoner or ontology designer with unnecessary expressive power.

References

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Retrieved from https://www.researchgate.net/profile/Deborah_Mcguinness/publication/230745455_The_Description_Logic_Handbook_Theory_Implementation_and_Applications/links/0deec51cfb6d8ae9d3000000.pdf doi: 10.2277/0521781760

Horrocks, I. (2005). Owl: A description logic based ontology language. In P. van Beek (Ed.), *Principles and practice of constraint programming - cp 2005* (pp. 5–8). Berlin, Heidelberg: Springer Berlin Heidelberg.