Conto: A Prototype Tool for the Generation and Utilization of a Configured Ontology

CONTO: A PROTOTYPE TOOL FOR THE GENERATION AND UTILIZATION OF A CONFIGURED ONTOLOGY

BY

ANDREW LECLAIR[1], B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2015)
(Software Engineering)

McMaster University
Hamilton, Ontario, Canada

TITLE:	Conto: A Prototype Tool for the Generation and Utilization of a Configured Ontology
AUTHOR:	Andrew LeClair[3] B.Eng. (Software Engineering) McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. Ridha Khedri
NUMBER OF PAGES:	x, 132

To my family and loved ones

Abstract

With the massive deluges of data that several domains of study experience, referred to as Big Data, the need to efficiently process and analyze data has risen. Ontologies have been employed to structure these domains with the ultimate purpose of generating new knowledge about the respective domain. Currently, when creating or examining an ontology, the concepts of the domain are limited to being statically defined in relation to other concepts. This limitation on the concept's definition affects the reasoning process by omitting or not properly representing all information that may exist in the domain such as how the constantly evolving environment changes how the data can be understood.

The resulting tool of this research, Conto, allows for the interpretation of a concept as an abstract data type. When coupled to knowledge generation process, these interpretations allow the obtention of new knowledge that would traditionally be unobtainable. The different types of knowledge that can be obtained via the multiple interpretations are explored in this work using examples of ontologies.

Conto is a Protégé plugin that uses Ontograf to display the ontologies.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Khedri for the continuous support of my M. A. Sc. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.
I thank my fellow student, Alicia Marinache, for the stimulating discussions, the constant revisions, and the seemingly endless picture ideas. It seems that we were able to make it out of the forest afterall.
Last but not least, I would like to thank my family and loved ones: my parents and to my brothers and sisters for supporting me throughout writing this thesis, and to my loved for inspiring me to constantly look forward.

Contents

List of Figures

Chapter 1

Introduction

Within the last decade there has been an explosive surge in data being collected.This deluge of data is commonly referred to as Big Data [Run13]. The diversity in domains and vastness of data is demonstrated within the medical field with electronic health records [MD13], in the astronomy field with cataloging interstellar bodies [FB12], and in the business field with collecting and combining business records and statistics [GSLG12]. The extreme amount of data that is being accumulated is only one of the problems of Big Data, the other being the rate that this data is being accumulated. To put the rate into perspective consider the following two existing scenarios: Feigelson et al. [FB12] state that the Synoptic Survey Telescope will output between 5-10 terabytes of data each night, and Gopalkrishnan et al. [GSLG12] estimate that enterprise servers processed 9.57 zettabytes of data globally in 2008. All this data is stored within billions or trillions of rows in databases, and all this data needs to be processed, which is a task that is not possible for humans. Due to the volume and rate of data being collected, it has been a significant area of research to be able to efficiently organize and analyze it [HQX12, CJB99].

To efficiently organize and analyze the data, researchers have begun to better understand and apply the philosophical ideas of *ontology.* An ontology is defined by Gruber in [Gru93] as a formal representation of knowledge, where the knowledge is represented as a hierarchy of concepts within a domain. Defining knowledge itself is a philosophical task that is out of scope of this work. However, we will use the definition presented in [FPSS96], where they claim an interesting pattern to be a sufficient embodiment of knowledge. Fayyad et al. further refine their definition of knowledge by creating ways to distinguish the knowledge by measures of validity, novelty, usefulness, and simplicity. These qualities together compose a notion of interestingness – an encompassing metric for acquired knowledge. Assessing the worth of knowledge with a metric of interestingness is explained to be both subjective and objective: qualities such as novelty are relative to the domain expert, but the simplicity can be measured, such as via the size of its digital representation. The goal of creating the ontology is to represent the knowledge of a domain in a compact, organized form that is easily readable by computers. The domain itself that the ontology is being created for can be of any granularity. For example, the domain that the GO was created for is biological processes, functions, and cellular components [Gen14]. As one can imagine, this is a very broad domain that is composed of numerous concepts. This contrasts the more specific ontology of wines in [NM01]. In essence, an ontology is designed to represent a domain through explicit conceptualizations and relations, which correspond to the objects in the domain and how they relate [Gru93]. Depending on the scope chosen for the domain, the concepts that are created will relate to this granularity. A more general and wide-scoped ontology such as GO will require more concepts, and more general concepts than that of a specific one such as the wine ontology. This leads to the problem of multiple ontologies being created for the same domains, attributing the variances to differences in scopes or interpretations of the domain. To help illustrate the variances in scope granularity, we can compare the GO [Gen14] and the The Music Ontology (MO) [RASG07] to the Ontology of General Medical Science [fBO15] and the Drug Ontology [SWR+99]. The first two ontologies, the GO and the MO are designed to conceptualize entire fields such as the entire field of biology terms and processes in GO. In comparison, The Ontology of General Medical Science and the Drug Ontology conceptualize domains which are more specific and particular. Regardless of their domain, the ontologies are designed to meet Gruber's definition, and so are hierarchies of concepts that are connected through relations.

Due to the fact that ontologies conform to this Gruberian definition, they all share the qualities of being hierarchical and built off concepts and relations. Whether it be the MO, or the wine ontology, the ontology is a hierarchy of concepts connected through relations. A *concept* is any object within the domain. For example, within the domain of animals, the concept of *dog* and *cat* may exist. The concept of dog represents the physical notion of dog within the domain, and will contain specific attributes that the concept embodies. The concepts of an ontology are connected through binary relations. Multiple relations may exist within an ontology, a common type being mereological relations. Mereology is defined by Leśniewski [EH95, oP15] as the theory of parthood; the relations of part to whole and of part to part within a whole. These mereological relations often form the hierarchical structure that is a defining characteristic of ontologies. This is because the concepts become more specialized and specific towards the bottom of the hierarchy. Expanding our example of the animal domain, the two concepts of *dog* and *cat* can be joined with a common ancestral concept of *mammal*. We can see that the concept *mammal* is composed of, and more general than *dog* or *cat*. Although mereological relations are what often compose the hierarchical structure of the ontology, as was mentioned, other relations often exist within the ontology which relate different concepts together. Finally, the idea of an individual can be realized by instantiating a concept. For example, by defining *Fido* as a dog, we are creating a specific instance of the concept *dog*. An individual is different from a concept because where a concept embodies the attributes that make up the *thing* in the domain, whereas the individual is the actual instantiation.

The structural components that build an ontology are analogous to those found in Object Oriented (OO) Design [CK94]. The goal of an ontology is to provide meaning and definition of representations of the world, which is the same goal shared in OO Design [GHJV94]. In [CK94], Chidamber et al. state that the ontological principles laid out by philosophers such as Mario Bunge can be applied to objects defined in OO design. Although Bunge did not state specific ontological definitions for objects, researchers such as Chidamber have employed Bunge's definitions for the OO domain. This application of Bunge's generalized concepts to the OO domain has been demonstrated in works such as [Wan89, WW90]. In an ontology, concepts are defined independent of implementation and encompass the notions of encapsulation, independence, and inheritance. Substantial concepts have properties, features that are inherent to the individuals of said concept. Other OO Design paradigms, such as minimizing coupling and maximizing cohesion, are defined ontologically by Bunge [Bun12] through notions such as *similarity*. The notions of OO Design can be easily related to those of ontological ideas laid out by philosophers such as Bunge.

The languages and methods for the formalization and implementation of ontologies is a wide-area of research, which can be seen in [Gru93, KCS13, LNST08, Gua95, Stu03, WZGP04, WTLH05, NH97]. Several existing formalizations will be discussed in Chapter 2, however, because of the hierarchical nature of ontologies, Description Logic (DL) was the natural formalization of ontologies [Obi07], and so will be a focus of the chapter. To formalize an ontology, the individual components each need to be formalized: the concepts, the relations, and the individuals. DL does this through its two components: the Terminological Box (TBox) which defines the concepts, and the Assertional Box (ABox) which contains the individuals [KSH12]. The concepts of *dog*, *cat*, *mammal*, and how they are mereologically related would be defined within the TBox, and the declaration that *Fido* is a dog would be within the ABox.

The field of organizing the data is an important field of research, but as we have discussed earlier, it is only one part of the problem of Big Data. The other being the analysis and comprehension of the data. DL allows for basic reasoning, and several techniques are explored within it [DCTK11, FT12, HQX12, PGS06, TSDM04, WD10]. In [BL04], reasoning is defined as "the formal manipulation of the symbols representing a collection of believed propositions to produce representations of new ones". There are two types of reasoning: deductive and inductive [Uni15, Orl85]. Deductive reasoning takes existing knowledge and transforms it into new knowledge through axiomatic definitions or other rules. This is contrasting to inductive reasoning which takes existing knowledge and generates new knowledge through generalization and broadening what the existing knowledge claims. In other words, deductive reasoning is the act of supporting a hypothesis or theory with existing facts and knowledge, whereas inductive reasoning is observing patterns or trends in the existing knowledge and creating a general conclusion based on them [Uni15, Orl85].

Often times, inductive reasoning is thought of as *bottom-up*, where a hypothesis – or conjecture – is generated. A conjecture is defined by the Merriam Webster dictionary as "an inference from defective or presumptive evidence". The conjecture that is generated is true to the universe within some confidence, meaning it may be proven wrong when new data is added to the system. This contrasts the results of deductive reasoning, which is that the result is always true in the universe, regardless of what data is added to the system.

The focus of our work is to support future possibilities with inductive reasoning. Inductive reasoning generates new hypotheses, and attempt to generalize data or find patterns within the data. Inductive reasoning is ideal in scenarios that are filled with ill-definedness, or gaps in knowledge, because it attempts to create hypotheses [Art94]. For these reasons, the results of inductive reasoning are significantly more novel and dynamic than those generated by deductive reasoning.

To illustrate a conjecture, we will use a system that contains the knowledge of the temperatures for the days of the months in a year as an example. If we assume our domain of knowledge to be the temperatures within Canada, it may be observed that the month January has low temperatures recorded for the first twenty days. A resulting conjecture could be that if the day of the year belongs to January, then it will be a cold day. The two distinguishing features of a conjecture that separates it from deductive reasoning shall be shown: future data can contradict the conjecture, and the conjecture may only be true within the domain of origin. To show the first, we can see that because the first twenty days of January are cold does not guarantee that the twenty-first day will also be cold. The possible future observation that any of the final eleven days of January are warm would defy the conjecture of January being a month of only cold days. To demonstrate that the conjecture may only be true in the domain of origin, we can compare our conjecture to hypothetical knowledge outside of the domain of Canadian climate. If we were to compare this conjecture to the recorded data of a country in the southern hemisphere, we would likely observe the opposite – that if the day of the year belongs to January, then it will be a warm day!

1.1 Knowledge Discovery Using Ontologies

Up to this point, we have discussed the two topics of ontologies and reasoning as two disjoint topics, however, often times the two are discussed together. Knowledge Discovery using ontologies is a popular and widely researched topic, as demonstrated in papers such as [AIS93, BL04, FPSS96, FT12, KCS13, Stu03]. The goal of creating the ontology is to organize the knowledge and data into a format that is machine readable, so that a reasoner can be applied and new knowledge may be discovered about the domain. Thus, the potential knowledge that can be discovered is entirely dependent on the ontology. As was discussed, an ontology is composed of concepts in a hierarchical structure that is connected through relations. The goal of this work was to discover a way that allows us to expand how we understand a concept. We wished to be able to imbue concepts with data types such that the result is a concept that contains implicit functions associated with its imposed data type, which can ultimately strengthen the conjectures we can generate.

To help illustrate the purpose of this research, in this Section we further explore the structure of ontologies and the mechanics behind inductive reasoning. It should be of note that although a multitude of ontologies exist, the similarities in structure between them allows us to generalize the notion of an ontology to that of the Gruberian definition. The analysis of the structure helps illuminate us to the limitations that currently exist, as well as how these limits propagate to the reasoning process.

Within an ontology, the concepts are organized so that each subconcept is a disjoint decomposition of a common concept. As we have already discussed, the relations which relate the subconcepts to superconcepts are based on mereology – the study of parthood. These relations form that decomposition – and the hierarchical structure of the ontology – by further decomposing each concept into its parts. For example, the concepts of *table* and *chair* are disjoint decompositions of the concept *furniture*. Neither a table is a chair, or a chair a table. Other relations may exist beyond the mereological relations. For example, within the MO [RASG07], there exists the 'compose' relation, which describes how a music artist may be related to a composition. It is evident that the relation 'compose' is not mereological in nature, as a composition is not 'part of' the music artist. Since the relation is not mereological, the guarantee that the structure will remain a hierarchy is no longer there. We may end up with a structure that is no longer ontological, such as the GO [Gen14], a structure which shares more similarity to a controlled vocabulary than an ontology. For this reason, we are limiting our scope of the research to mereological relations.

1.1.1 Introduction to Description Logic and Conjecturing

At the heart of the ontology are the relations and the concepts they relate. Together they describe the domain; the concepts are the realization of each *thing* that may exist in the domain and the relations relate the *things*. To properly conceptualize a domain, the concepts and relations require a formalization to be employed, a popular choice being DL due to its correspondence of the TBox to concepts and the ABox to individuals [FT12]. Within DL, a concept is defined by a terminological assertion which states how the concept is defined within the domain. For example, the concept *Mother* may be defined in the declaration:

$$Mother \equiv Woman \sqcap \exists hasChild.Person \tag{1.1}$$

This states that a mother is someone that is a woman, and has a child that is another person. As we can see, the concept *Mother* is defined by using other concepts and relations to other concepts, in particular, the concept *Woman* and the relation *hasChild*. These terminological assertions populate the TBox, which is the component that several reasoners currently utilize. The knowledge that is acquired through these reasoners varies depending on the algorithm employed, and the goal of the user. For example, subsumption rules can be acquired using the tableau method, and the knowledge acquired will help decide the satisfiability of the ontology [HS01].

Conjectures can also be acquired from the ontology in the form of association rules, as demonstrated in [BFGR07, FT12]. An association rule is of the form of *If-Then* rules, and are written as $X \implies Y$. The rules can be understood as "If X, then with confidence c, Y " where X and Y are individuals within the ontology, and c is the proportion of cases where this rule is exhibited. In [BFGR07] and [FT12], association rules are extracted from the information within an ontology. The data may initially be structured as a database, as is the case of [BFGR07], and the ontology is created with the desire to reason over it to discover rules that allow a better understanding how the data is related. For example, in our earlier example with the ontology of the weather in Canada, we conjectured "If the day of the year belongs to January, then it will be a cold day". We notice it follows the syntax of the rule we just described, and can be re-written as "*Day in January* $\Longrightarrow$ *cold*". Since in our example all prior twenty days exhibited this rule, our confidence would be 1. The extent of techniques that exist for reasoning, and the varying well-established ontologies will be detailed in Chapter 2.

1.1.2 Concept Interpretation

It can be seen that the ability to reason knowledge is dependent on the concepts and the relations that connect them. In the current literature, knowledge is often formalized with DL. This means that concepts are axiomatically defined in the TBox by relating it to other concepts. In other words, the concept is a static conceptualization of a thing in the domain, and the reasoning process can only utilize how it is related to other concepts. For example, we may have the concept *Temperature Reading* which is related to the concept *Weather Forecast* by the *is-A* relationship. Expanding how we define concepts is the intended purpose of this research. We wish to imbue concepts with interpretations of abstract data types and their accompanying theory. Continuing the example, we could expand the concept of *Temperature Reading* in a way so that it is not only a refinement of the Weather Forecast concept, but also a *List*. This would change how the ontology and specifically the concept is understood. *Temperature Reading* has not changed its relationship to *Weather Forecast*, but it does now have the underlying structure of a List. This specific interpretation lends an ordering to the individuals that populate *Weather Forecast*, as well as the functions of a List such as Head and Tail.

By interpreting *Temperature Reading*, we change what is available for the reasoner during the reasoning process. The TBox of the ontology has been augmented with the properties and operations of a list. Initially, the reasoner had the initial TBox definitions: mereological relations between the concepts; that a Temperature Reading is a refinement of a Weather Forecast. With our example interpretation, the reasoner will have access to an augmented TBox: the mereological relations with the addition list properties and operations. Alternatively, we can interpret the concept as a Set, providing the operations that a Set offers, as well as the properties such as containing individuals with cardinality of maximum one (i.e., no duplicate entries). The idea of interpretations allows concepts to be imbued with varying theories and data types, and arms the reasoner with more than just the DL concept definitions. However, the idea of interpretations also extends how we think of an ontology. Each time we interpret a concept, we create an ontology which is understood differently than before the interpreting process by augmenting the TBox in different ways. This understanding of the ontology is called the *configuration*. A configuration of an ontology is the set of interpretations applied to the existing concepts.

1.2 Motivation

The ability to configure an ontology with interpretations is an extremely important area of research for a multitude of reasons. Firstly, as mentioned when first discussing ontologies, the current understanding of concepts in an ontology is static. In the current literature, a concept is a *thing* that is defined by its relation to other concepts.

This insufficient understanding extends into reasoning techniques allowing for only simple conjectures and deductive results. Having the capability to interpret concepts in different ways will allow a more in-depth reasoning process, generating novel and significant conjectures. As it currently is, the reasoning methods process the data in a single static way: the way they are defined in the terminological axioms. Interpretations of concepts allows reasoning methods to process the data in various ways, with each interpretation allowing a specific set of functions available. Each configuration can be imagined as a way of seeing and understanding the domain – a world. Currently, we are limited to a single static world which we can reason on. By creating a configuration, we create a new way to understand the data – a new world – in which we can interact with by reasoning over. It allows for data types and their respective functions to be injected into the reasoning process over ontologies. In today's age of Big Data, the improved mechanisms for reasoning allows industries to condense and utilize their data by taking advantage of the interpretations and functions that are provided. For example, clusters of data that are interpreted as list data structures and organized by the heads of their respective lists. The notion of configurations can be applied to existing ontologies and datasets, enhancing their reasoning processes so common issues can be addressed. Common issues existing in popular domains such as marketing agencies using the existing data to learn new qualities and trends about the market, and the medical field learning new interactions between medicines and illnesses [AIS93, WTLH05]. Being able to make conjectures from this new data would allow industries to effectively take advantage of their large deposits of data, as well as allow efficient ways to store the vast amounts of data.

In [DCTK11] the popular reasoners of the Semantic Web are compared, and in [CP13] it is demonstrated that it is possible to inject *collections* that encompass set theory into the Semantic Web language OWL2 DL. Knowing this, it is a matter of making a tool for Protégé that allows the interpretation of concepts, where the interpretations are modeled after the Collections Ontology defined in [CP13].

1.3 Problem Statement

The area of interpreting the concepts of an ontology as a data type is an open area of research. The different interpretations that can be performed on the concepts give rise to new configurations of the ontology which capture specific understandings of the domain. Currently, when creating or examining an ontology, the concepts are limited to being defined in relation to other concepts. When reasoned on, this limitation on the concept's definition affects the reasoning process by omitting or not properly representing all information that may exist in the domain.

With the large volumes of data being accumulated, the need to efficiently acquire as much useful knowledge has risen. The knowledge that could be derived from the data is related to our understanding of the world from which the data is collected. Therefore, each configuration of an ontology leads to a new set of knowledge that might not be obtained from another configuration. By examining all configurations, we can efficiently and effectively acquire the maximal amount of knowledge in the domain.

The work done extends the Protégé tool to allow for the interpretation of concepts within the ontology, and ultimately the configuration of an ontology, removing this limitation on the concepts. The respective functions of the interpretation can then be applied to the interpreted concepts to learn new knowledge related to the concepts and its individuals.

1.4 Main Contributions

The main contributions to the ontology field include:

(i) An extension to the Protégé editor that allows for the interpretation of concepts.

(ii) The ability to configure and load the ontology with the desired interpretations.

(iii) The ability to apply functions respective to the interpretation on the concept (i.e., union of sets, maximum value of adjacent elements of a list, etc.)

1.5 Structure of the Thesis

The remainder of this thesis is organized as follows:

Chapter 2 provides a survey of the current literature related to ontologies and reasoning.

Chapter 3 introduces Conto which is a result of our work and describes the elements of design that were put into it.

Chapter 4 is a series of examples that we study to examine the usefulness of Conto.

Chapter 5 discusses the impact of Conto in expanding how we think of ontologies, as well as critically evaluating the strengths and weaknesses.

Chapter 6 draws conclusions and suggests future work.

Chapter 2

Survey

In this section, we survey the literature of the current state of ontology formalisms, conjecturing knowledge from ontologies, and tool assistance with these tasks. In Section 2.1, we look at some existing ontologies that exist in the field of domain knowledge. In Section 2.2, we explore why ontologies are used for reasoning, and methods that can be used to conjecture new knowledge, and the domains they can be applied on. In Section 2.3, we investigate the tool support for creating and editing ontologies. Finally, in Section 2.4, we evaluate the current research into fields that relate to the notions of interpretation and configuration.

2.1 Existing Ontological Research

We have stated that DL [KSH12] is a popular formalism for ontologies, and due to its popularity and natural ability to represent ontologies, it is the focus of this research. We have also mentioned the Semantic Web and the languages associated with it (such as OIL+DAML [H^{+}02]), which popular ontology tools such as Protégé [fBIR15] use. In this section , we will explore research done with DL with respect to ontologies, as well as the languages used for implementation. We will explore topics such as the Semantic Web, as well as reasons to why DL is so popular and ideal for our research. In addition to this, we will touch on other formalisms outside of DL for the sake of completeness and comparison. To end this section of the Literature Survey, we will explore popular ontologies to examine the formalisms they employed and the languages that implement them.

The Semantic Web

In [H$^+$02], Horrocks describes the language that is known as OIL+DAML which is intended to design the Semantic Web. The Semantic Web is an extension of the Web, with goals of providing the framework which will allow the better reuse and sharing of the information on the Web. In [H$^+$02], many parallels are drawn between OIL+DAML and DL, claiming that OIL+DAML corresponds to the $\mathcal{SHIQ}$ DL. This means that it includes the following constructors: the basic $\mathcal{ALC}$ DL extended by the transitive roles from the $\mathcal{S}$, role hierarchies from the $\mathcal{H}$, inverse properties from $\mathcal{I}$, and finally the qualified cardinality restrictions from the $\mathcal{Q}$. Horrocks continues to specify that OIL+DAML is similar to the TBox of DL. This similarity implies that the structure of DL, as well as the already existing body of research in DL, could be applied to the Semantic Web. OIL+DAML was one of the initial formalizations of DL using RDF syntax, and it later involved into what is now the family of Web Ontology Language (OWL) languages. This family includes OWL DL, OWL Lite, and OWL Full.

OWL was the next iteration of the language that structured the Semantic Web, as described in [Hor05]. The multiple varieties of OWL are built off the $\mathcal{SH}$ DL. For example, OWL DL is based on the $\mathcal{SHOIN}(\mathbf{D})$ DL, which extends OIL+DAML with nominals ($\mathcal{O}$), replaces qualified cardinality restrictions with just cardinality restrictions ($\mathcal{N}$), and adds data types ($(\mathbf{D})$). With this new DL that the Semantic Web is built on, it became possible to better conceptualize a domain [Hor05, HPsaCAW]. The OWL family ultimately became an important formalization of DL and can be seen in tools such as Protégé which is built off of OWL DL.

We have investigated the Semantic Web, which is a formalization of DL, but there exists several other languages that are used for the creations of ontologies that we will investigate. Each language we will examine has its own advantages and limitations. These niche unique properties have resulted in the spawning of a wide variety of languages, each built to meet a specific goal. Examples of these language properties include the expressive power, the syntax and semantics, and the inference engine. Determining which languages are the most widely-used is a difficult task due to the magnitude of existing languages, and opinions differ depending on the researcher. For example, in [SI02], CycL [MCWD06], Ontolingua [FFR97], FLogic [KL89], CML [SWA+94], and OCML [Mot98] are considered the most used and studied traditional ontology languages, whereas in [CGP00], Ontolingua, OKBC [CFF+98], OCML, FLogic, and LOOM [Cha07] are the most prevalent. Upon observation, overlap between the languages can be seen, such as FLogic, Ontolingua, and OCML. Upon noticing this overlap, we will focus our attention on these languages and formalisms.

Frame Logic

Frame Logic (FLogic), mentioned in [KL89, CGP00], is a first-order predicate calculus that integrates frame-based languages. In [KL89], it is mentioned that FLogic was first conceived to allow object-orientated principles in a database language by supporting object identity, and allows for properties such as inheritance. It should be noted that as we have discussed earlier, OO Design principles are prevalent in ontology studies, and this is a primary motivator behind FLogic. FLogic is composed of a set $\mathcal{O}$ of basic objects, a set of object constructors $\mathcal{F}$, an infinite set of variables $\mathcal{V}$, and the standard logic connectives and quantifiers. The way an object is thought of in FLogic is that it is both an instance and a concept; there is always two interpretations. For example, the object *student* can be thought of as the class of students, and it can also be thought of as the instance of the class *person*. This is to embody the idea of inheritance, where each class can be thought of as an instance of a superclass. Due to this nature of inheritance, in FLogic, the concepts and individuals are organized into a lattice structure. There exists a maximal concept, the meaningless object that contains no instances, and the minimal element which is the biggest class, or the unknown object. The lattice structure itself represents the transitive closure of the IS-A relationship, a topic that is regarded to be controversial (as described earlier in this paper). Within FLogic, the semantics are defined through what Kifer calls its interpretation, I. In essence, the Interpretation is a mapping between the objects in $\mathcal{O}*$ by the elements of U, where U is the universe of all objects that have a lattice structure, and $\mathcal{O}*$ is the *names* of the objects. There exists the difference between DL and FLogic by the definition of a class. In DL, a class is defined through its relations to other concepts, which contrasts FLogic where the class is defined through its interpretation (or frame specification) [dBH08]. FLogic has proved to be one of the more popular formalisms compared to DL, and prior Protégé version 4.0, FLogic was accepted as one of the two types of ontology. The reasons to why DL is favored over FLogic is discussed in [dBH08], and includes reasons such as that FLogic has a Closed-World Assumption, and is undecidable. Both these qualities are in contrast to DL, and inhibit the power of ontologies, where the information is dynamic and constantly growing, and an important related field of research is reasoning which requires decidability.

Ontolingua

Ontolingua is an interesting ontology formalism due to the fact that Ontolingua itself is just a server, and hosts a repository full of ontologies that allow for collaboration among domain experts. In [FFR97], Ontolingua is described as being designed to "...support the design and specification of ontologies with clear logical semantics.". Ontolingua is built off of two ontology formalisms – Knowledge Interchange Format (KIF) and Frame Ontology (FO) – to combine their abilities and cover their limitations. KIF is a monotonic first-order logic that has a simple syntax to support reasoning about relations. KIF allows for the definition of objects, functions, and relations, however, KIF is an interchange format, and so is tedious to make a full specification of an ontology. FO, which is built on top of KIF, is a knowledge representation ontology, and allows for the conceptualization of classes, instances, subclass-of, and instance-of. However, FO does not natively allow for the expression of axioms. The combination of KIF and FO allows for the creation of a graphical ontology that is powered by KIF-defined axioms. The ontologies themselves are stored on the Ontolingua repository allowing for all sorts of collaborators to add, refine, or query the ontology. Despite the massive expressive power provided by KIF, there is a lack of tool support for reasoning. In [FHVH+00], it is claimed that the lack of tool support comes from that Ontolingua does not provide any way for the user to "control" the expressive power. For this reason, Ontolingua has proven to be useful for purposes of querying data, but not to create an ontology with the ultimate purpose of reasoning.

Formal Concept Analysis

Another formalism that is worth of note is Formal Concept Analysis (FCA). Wille describes FCA in [Wil05] as a subfield of *Applied Mathematics* based on the mathematization of concepts and concept hierarchies. He describes the aim of FCA as "... to support the rational communication of humans by mathematically developing appropriate conceptual structures which can be logically activated.". A core part of FCA is answering what a concept is, and further, how they may be mathematically defined. Wille defines a concept as "the basic units of thought formed in dynamic processes within social and cultural environments.". He further illustrates that there are several characteristics behind concepts, such as being domain specific and independent of language. A concept can be constituted by the *extension* – all objects which belong to the object – and the *intension* – all attributes which apply to the objects of the extension. This means that the *subconcept-superconcept-relation* plays a heavy role within the formulation of FCA. A formal context can be defined as $\mathbb{K} \overset{\text{def}}{=} (G, M, I)$, where G and M are sets and I is a binary relation between G and M. Specfically, the elements of G and M are *formal objects* and *formal attributes*,

respectively. A *formal concept* of a formal context is a pair (A, B), where A and B are an element within G and M, respectively. Wille also defines the *subconcept-superconcept-relation*, which is used to create a concept lattice. The concept lattice is the visualization and ultimate representation of the formal concepts. Within the concept lattice, each node may represent one or many formal concepts. The extension of a node consists of all the objects that can be reached by descending the path from the node, whereas the intension consists of the attributes that can be reached by ascending the path from the node. In this way, FCA can represent the binary relations often found in databases in a way that can be reasoned on, and easily understood by humans. However, as [WVV^{+}01] points out, the major drawback to FCA is the limited expressiveness that can be compared with a simple database table. For this reason, FCA is a representation that is ideal for simple binary relations between concepts and attributes.
Several of the other languages, such as LOOM and OCML, are often rooted in one of the previously mentioned formalisms (such as LOOM being based in DL, and OCML being an extension of Ontolingua) [CGP00]. There exists a massive amount of languages, many not listed in this paper, but they all seem to grow from one of the formalisms described in this section. In the next subsection, we will explore some of the ontologies that are used in the public domain. We will investigate the formalism that has been employed, if one exists, and ultimately discuss if what is claimed to be an ontology, is in fact an ontology.

2.1.1 Ontology Examples

Even though there exists a large collection of ontologies available for public use, a few have become the central focus for domains. In this subsection, we will quickly evaluate some of these ontologies to demonstrate how they are used, and how they have been formalized. We will also further discuss the question of what an ontology is through examples, and if what is claimed to be an ontology is actually an ontology. The ontologies we will look at is the GO and the MO.

The Gene Ontology

GO was formed when scientists began to notice and acknowledge that there is likely a single limited universe of genes and proteins [ABB+00]. The goal of the GO was to be this unification of biology, that allows for the interoperability of genomic data. The GO is composed of three disjoint ontologies: the cellular component, the molecular function, and the biological process ontology [Gen14]. As shown in Figure 2.1, the structure of the GO is a graph, and is described in [Gen14] as "loosely hierarchical". Each child term is a specialization of the parent, but there may exist more than one parent.

In Figure 2.1, the root nodes of the three disjoints can be seen of cellular component, molecular function, and biological process. They are considered three disjoint ontologies because there does not exist a parent concept between the three of them, and thus they are separate. To unify these three ontologies, an arbitrary parent concept *thing* can be made that holds no significance other than providing a single top concept. Also shown in the figure are the three types of relations that compose the GO: *is_a*, *part_of*, and *regulates*. They are denoted on the graph by the letters *I*, *P*, and

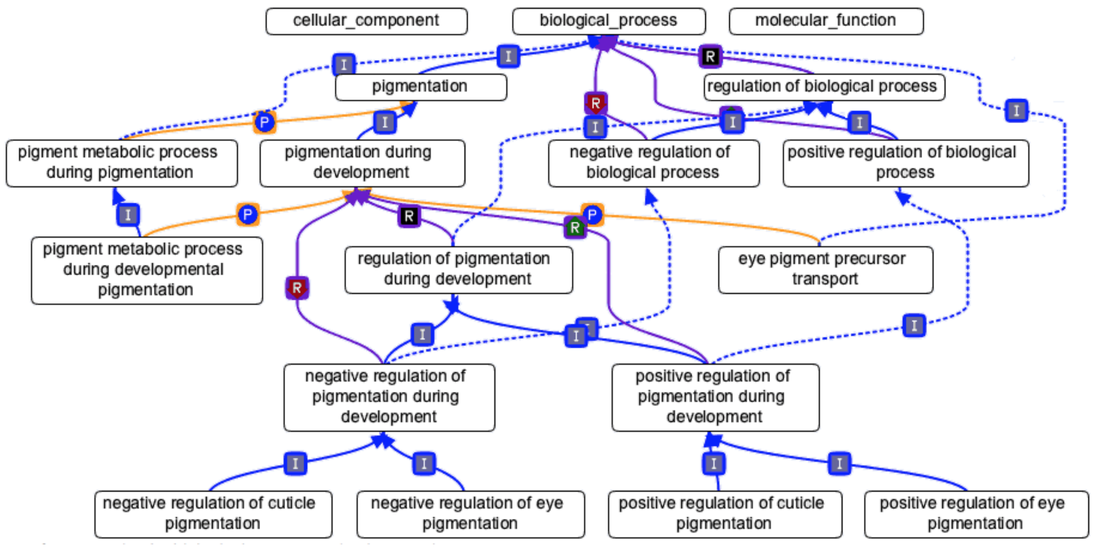

cellular_component
biological_process
molecular_function
pigmentation
regulation of biological process
pigment metabolic process during pigmentation
pigmentation during development
negative regulation of biological process
positive regulation of biological process
pigment metabolic process during developmental pigmentation
regulation of pigmentation during development
eye pigment precursor transport
negative regulation of pigmentation during development
positive regulation of pigmentation during development
negative regulation of cuticle pigmentation
negative regulation of eye pigmentation
positive regulation of cuticle pigmentation
positive regulation of eye pigmentation
I
P
R

Figure 2.1: A small set of the terms from the GO [Gen14].

R, for *is_a*, *part_of*, and *regulates*, respectively. However, as we have discussed in the previous section, an ontology should be formalized through some type of logic and have defined axioms. When investigating the GO, such a formalization does not exist. In [SWS03] this is further investigated, and ultimately discusses that although the GO contains ontology in its title and is built from mostly mereological relations, it is not an ontology according to the definition of an information scientist or philospher. Instead, it is best considered a 'controlled vocabulary', as opposed to an ontology which is composed of terminologies with axioms and definitions. As described in [SWS03], the primary goal of the GO was not ontological either. The authors did not focus on software expression nor the logical expression of the theory encompassing the terms. Instead, the focus was directed toward providing a useful framework for keeping track of the biological annotations that are applied to gene products.

The Music Ontology

The MO is an ontology that is built on top of several ontologies, such as the Timeline Ontology and the Event Ontology [RASG07]. The reason for building ontop of the ontologies is to access information or knowledge that they wish to express when dealing with music-related knowledge. For example, to provide the description of 'this performance happened the 9th of March, 1984' requires temporal information. The MO itself is built on RDF, and is specified using OWL, implying that the formalism behind the MO is DL. This allows the reasoning processes already existing for DL to be involved in anything made using the MO, such as checking for inconsistencies. An example of a description of a music production flow using the MO is provided in Figure 2.2.

One thing that can be observed in the MO is that it allows for relations that exist beyond just mereological. For example, in Figure 2.2, relations such as *produced_work* and *performance_of* exists. Allowing for relations that extend beyond just mereological may result in an ontology that is not strictly hierarchical, which can be observed in the MO. Despite the relations that exists within the MO, we can see that it is based in DL, and moreso, uses OWL to implement the ontology. It can be combined with other ontologies available on the Semantic Web to allow for the expression of detailed musical 'metadata' which can be reasoned on using DL reasoners.

Remarks on GO and MO

From the examples that have been given, we examined the GO and the MO. Upon investigation, we discovered that although the GO has the word ontology within its name, it is not true to it. It has no real formalism, and is not implemented into

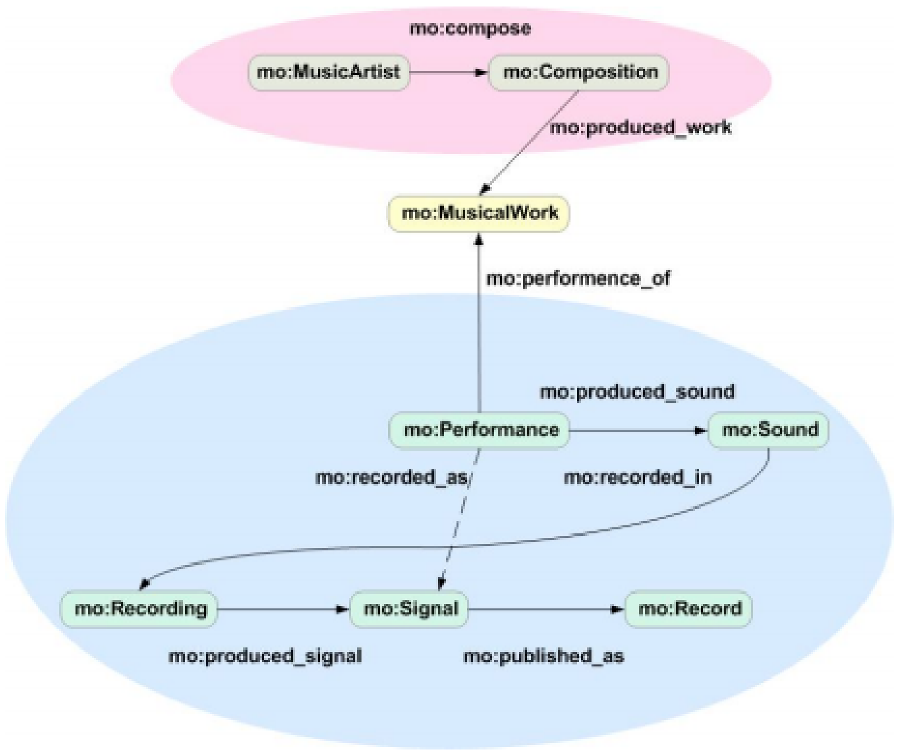


Figure 2.2: An example of using the MO [RASG07].

software, nor do the researchers have the intention to do so in the future. The MO however, is formalized using DL and is written using OWL so that it may interact with other ontologies that exists on the Semantic Web, and utilize the currently existing DL reasoners. We have also encountered the problem with the ontology field stated earlier in this paper: although the GO and MO are both called ontologies, they have stark differences that can be observed within their formalization and implementation. Arguments could also be made to defend the title of ontology for the GO because it better follows the Gruberian definition of an ontology. GO is a hierarchy that is decomposed using mereological relations, whereas the MO has several non-mereological relations. Depending on the specific definition for ontology used, a different verdict on the status of the GO and MO could be made.

2.2 Existing Reasoning Techniques

In the previous section we have evaluated several popular formalisms of ontologies, such as DL and FLogic. In this section of the Literature Survey, we will first explore literature to determine why researchers choose to use ontologies for reasoning. We will investigate the strengths and weaknesses to ontologies. We will then explore research with reasoning methods and techniques on the various ontology formalisms, with an emphasis on DL due to its popularity in literature.

2.2.1 Motivation to Reasoning with Ontologies

In the previous section, we have seen that ontologies are used for structuring a domain using concepts and the relationships between them. They provide a structure that is easily understandable for humans, and is formalized in a way that allows software to take advantage of the structure for reasoning purposes. The specific reasoning processes will be explored shortly, but first we wish to understand the motivation for using ontologies to reason.

In [FEAC02], Fonseca et al. create an architecture which can integrate geographic information from the numerous sensors on Earth seamlessly. They propose an ontology-driven geographic information system, which allows users to view information about the domain. Using ontologies, they are able to better define their domain with richer concepts, such as the concept Lake. The concepts may not necessarily exist in the databases the information is pulled from, but rather, created by experts. These experts may take the data in the databases, and use them as attributes in concepts,

for example, attributes for a lake may be that it has a pH value, and a total volume of water. With these concepts and attributes, Fonseca et al. discuss that users can browse the ontology to retrieve information about specific areas (e.g., a specific lake), as well as classify images. The image classification utilizes the knowledge base of the ontology (as well as some assumptions provided by the domain expert) to drive the image classification algorithm. From the work of Fonseca et al., we can see that the ontology is an essential part of the research. The ontology provides a structure to the massive amount of data they have, and the notion of concepts embodying these attributes allows the relation between the data i.e., the data of fresh water and volume of water can be related together through the concept of Lake.
Another example using ontologies is the work of Rubin et al. in [RDB+06]. In this paper, Rubin et al. develop a methodology for determining injuries based on images and ontologies. They utilize two ontologies for the reasoning. The first is a comprehensive ontology of anatomy which contains organ identities, adjacencies, and other information for anatomic reasoning. They also use an ontology of regional perfusion which contains formal definitions of arterial anatomy. The combination of the geometric model (provided by the images) as well as the ontologies, the consequences of the injury can be deduced. The results of Rubin's research includes the ability to determine which organs are injured given the trajectories of projectiles, whether vital structures are injured, and the ability to predict the propagation of an injury. The ontology provides a rigorous structure composed of the domain knowledge for the reasoning process. Interestingly, whereas the ontology used in [FEAC02] by Fonseca et al. which changes as information streams in from sensors, the ontologies used by Rubin et al. does not change. The ontologies used by Rubin et al. act as an expert,

which consults the image of the injury to deduce the extent of damage, whereas the ontology used by Fonseca et al. is consulted by an expert for its ability to represent the vast amount of data.

Jurisica et al. explore ontologies being used for knowledge management in [JMY99]. They discuss that although ontologies may be constructed for different purposes, a common goal is to enable sharing and reuse among the information of the domain. To achieve this goal, an ontological commitment must be made, which is an agreement to consistently use a vocabulary. This means that the ontology must define the entities and relationships in the domain – it must conceptualize them – and so the information in the domain must be based on these conceptualizations.

Using the result of Jurisica et al., we can re-evaluate the two cases of Fonseca and Rubin in terms of this ontological commitment. Fonseca et al. wished to unify the data of all the sensors, to be able to share the data with the end-users for the ultimate purpose of reasoning and image classification. This required the ontological commitment to create concepts which used this data. Rubin et al. also made the ontological commitment to the concepts so that they could reuse the data pertaining to organs and be applicable to deduce the severity of injury based on the image.

We can see that regardless of the domain – whether it be geography or medical sciences – ontologies are used for their ability to grant the reuse and sharing of the data. These traits are provided by the ontological commitment of a consistent vocabulary, which are utilized by the act of reasoning. Using the ontology for the act of reasoning takes advantage of the structure which is present, and allows for the access of the vast amount of data that has been classified.

2.2.2 Reasoning with Description Logic

The first formalism that will be investigated with respect to its reasoning processes is DL. This is because of the popularity of DL in the literature, and the wide-range of research put into the field of reasoning with DL. When reasoning with DL, one of the first features that must be checked is the type of DL that is being employed, for example, the $\mathcal{SHIQ}$ DL that OIL uses. Each component of the DL provides operations that can be used for reasoning, and may assist or hinder the process. In [HS01], Horrocks et al. describes the $\mathcal{SHIQ}$ DL as insufficient for reasoning due to the fact that it does not contain any concrete data types. Horrocks et al. suggested the $\mathcal{SHOQ}(\mathbf{D})$ in [HS01] due to its abilities to extend individuals, provide concrete data types, and it removed the inverse roles which proved to be cumbersome for reasoners when combined with concrete data types or named individuals. With the $\mathcal{SHOQ}(\mathbf{D})$ DL, he provided an example of reasoning using the tableau algorithm. A tableau that is formed from the algorithm seeks out concept satisfiability, and works with the TBox. However, in [HST00], Horrocks et al. demonstrate that reasoning over the ABox in a DL Knowledge Base is possible. ABox reasoning is valuable because typically in a reasoning process the ABox is assumed to be empty, when in practice that is not the case. Instead, the TBox can be thought of as representing a schema, and the ABox being the populating (and possibly incomplete) data. With this thinking, it can also be imagined that the size of the ABox can be of almost unlimited size, versus the limited size of the TBox, leading to tractability issues of reasoners that utilize the ABox. Using the ABox is of interest due to it being able to be used to decide the problem of conjunctive query containment with respect to a schema – the problem of deciding if one query is contained another query with respect to the database – and with the algorithm proposed in [HST00], the tableau method is extended to incorporate the ABox.

Ontology Classification

Another field of reasoning that is prevalent with relation to ontologies is ontology classification [GHMS10, GHM$^+$12, KK13]. Ontology classification is defined by Glimm et al. in [GHM$^+$12] as the computation of subsumption hierarchies for classes and properties. In essence, ontology classification is the actual construction of an ontology in the way that it 'connects' the axioms and assertions together to create the visual structure humans are familiar with. As Glimm et al. mention, it is a core service that is provided by OWL reasoners, and the resulting hierarchies are used in ontology engineering, where errors can be identified, be used to explain the knowledge base, or answer queries. The methods that are studied for ontology classification are devoted to optimizing the subsumption tests for efficiency and decidability. However, the difficulty comes from that many ontologies are extremely large in size, making even even very efficient subsumption tests expensive. Glimm et al. challenge these issues in [GHMS10, GHM$^+$12] by proposing alternative algorithms, such as 'KP'. Kazakov et al. also discuss ontology classification in [KK13], but instead of researching how to efficiently create the structure, they focus on how to efficiently update one. The issue of frequent re-classification of ontologies is an area of research that is tackled by incremental reasoning procedures. With incremental reasoning procedures in place, the knowledge base will support the addition or deletion of axioms, and not require deep modification to the base reasoning procedure.

Defeasible Reasoning

The reasoning that has been explored by Horrocks et al. in [HS01, HST00] typically produce subsumption rules through the tableau method. However, there is research that tries to extend the types of reasoning that is performed on DL. For example, in [MMS14], Moodley et al. explore defeasible reasoning for DL. In [MMS14], Moodley et al. explore the relations called *defeasible subsumption* ($C_1 \precsim D_1$), which can be compared to its classical counterpart ($C_1 \sqsubseteq D_1$). The difference is exemplified by how the axioms are read: in standard DL subsumption ($C \sqsubseteq D$) is read as "all C's are D's", whereas the corresponding defeasible subsumption ($C \precsim D$) is read "the most *typical* C's are D's". When this relation is reasoned on, the knowledge that is gained is defeasible, which means that it is not absolute like a deductive theorem. As more knowledge is acquired, or existing knowledge is modified, contradictions may arise between rules generated and the present data. Moodley et al. are not the only researchers who are exploring defeasible reasoning with DL, [CMMV13, GGOP13] are two other examples of researchers who are exploring knowledge reasoning that does not have the assumption of certainty. Defeasible reasoning is of interest to us because it produces arguments which are compelling, but not necessarily deductively valid. It is a form of inductive reasoning, and thus a focus of this work.

The Addition of Temporal Operators

Beyond relations being created to improve the reasoning that can be performed, as demonstrated through the nonmonotonic systems in [MMS14, CMMV13, GGOP13], researchers Baader et al. extend $\mathcal{ALC}$ DL with Linear Temporal Logic (LTL) where temporal operators can be applied to TBox axioms and ABox assertions [BGL12]. An example of time changing how the ontology is understood is through the postulated case using the concept "Concussion with no loss of consciousness", an existing concept in the medical ontology SNOMED CT. The way the concept is currently understood could be argued to be incorrect, when a correct representation of the concept would be that after the concussion, the patient remained conscious until the examination. This allows for reasoning to include time as a factor with the operations that LTL introduces, such as the U operator which formalizes 'Until', meaning the duration of one event until another event occurs.

We have examined several approaches to reasoning with DL in this subsection. We have examined the most popular method, subsumption reasoning, and research that has gone into optimizing it with papers such as [HST00, HS01]. We also investigated the uses of subsumption reasoning, the vital ontology classification, and looked at how researchers are improving it with papers such as [GHMS10, GHM^{+}12]. Finally, we explored differing types of reasoning, such as defeasible reasoning that is explored by Moodley et al. in [MMS14], or temporally infused reasoning that is explored by Baader et al. in [BGL12]. We can see that the reasoning field with DL is a field that is constantly trying to adapt to the massive ontologies that have been created, as well as new reasoning methods are being explored to try to learn knowledge that is not found through conventional methods.

2.2.3 Reasoning with non-DL Formalisms

In [BL12], Baral et al. highlight that outside of DL, there does not exist much research into knowledge representation or the reasoning thereof. They instead focus on frame based representations, and try to discover better ways to reason knowledge from them, and what kind of questions this reasoned knowledge can answer. Baral et al. describe a *property acquiring* process in hopes to answer the question *What is X?*, where X is an object in the domain. The process that is described is composed of three steps: Obtaining generalizations of the instance, obtaining what an instance clones from, and the unification process. In the first step, all classes that belong to the object are gathered, to determine all inheriting classes and classes that may be an instance of X. The second step aims to determine the clones of object X from these instances found – a class that is spawned by reusing the existing knowledge frames of the already existing class with the purpose of avoiding repeating encodings of the same set of knowledge entries. The final step, unification, aims to acquire the information now that we know what the instance clones from (from the previous steps). With the information that is found, questions such as "What is X?", "What are the similarities between X and Y?", "What are the differences between X and Y?", and "What is the p of X?" (where p is some property) can be queried from the knowledge base. The resulting knowledge that is reasoned parallels the DL subsumption reasoning, which itself aims to better answer "What is X?" by determining the ordering relation between the concepts within. With the frame based system, queries may be more insightful due to the embedded frame properties that is not found in DL. However, as it was mentioned, the research and development into this type of representation (and other representations outside of DL) are lacking.

The overwhelming amount of research that is put into optimizing and extending the reasoning process over DL absolutely overshadows research put into other formalisms' reasoning techniques, as indicated in [BL12]. However, there does exist reasoning techniques outside of those for DL, as evidenced by Baral et al. however, they seem to parallel reasoning techniques existing in DL, and they lack the plethora of tool support that DL has.

2.3 Existing Ontological Tools

In the previous section, we discovered that there exists a dwarfing amount of research into reasoning for DL compared to other formalisms for ontologies. In a continuation of that topic, for this section of the Literature Survey, we will explore the tools that perform the reasoning algorithms, restricting ourselves to the tools that are based in DL. First we will discuss the tools that can be used for formalizing and representing ontologies, followed by the tools that are used for reasoning so that we may discuss the compatibility between the reasoners and the ontology tools.

2.3.1 Ontology Formalism and Representation Tools

Currently in the field of creating tools for the creation and modification of ontologies, it can be found that there exist several ontology related plugins that are compatible with the Protégé tool [fBIR15, KFNM04, Ala03, HCC^{+}12]. The functionality of the plugin varies, ranging from general functionality like ontology visualization to a specific task like the optimization of the ontology creation. For example, in [KFNM04], an OWL Plugin is introduced for Protégé. The plugin extends the functionality of Protégé so that ontologies that are written in OWL can be edited and used. In [Ala03], a plugin is introduced that is based on TouchGraph technology so that ontologies can be visualized, and in [HCC^{+}12], a suite of plugins is introduced that implements crucial parts of the MIREOT (Minimum Information to Reference an External Ontology Term) specification which improves the ontology creation and

editing process by improving the capabilities of reusing artifacts. The point that the number of plugins that exist for Protégé numerous and their uses are wide does not need to belabored. Protégé is currently one of the most popularly used ontology creation tools for DL ontologies – if not the most popular – and has the support and research into developing new plugins that makes it appealing.
Among Protégé there exists other editors: NeOn Toolkit [HLS+08], SWOOP [KPS+06], and OWLGrEd [LCS12].

NeOn Toolkit

The NeOn Toolkit is a project that is aimed at large-scale ontologies [HLS+08]. Like Protégé, the NeOn Toolkit works with ontologies built from the Semantic Web, and are open-sourced projects open to the community. The combination of being open-sourced, and having a modular design has allowed for plugin development, much like Protégé. The result is a repository of plugins for the tool, such as OntoModel, a visual modeler for the ontology, Text2Onto, a plugin for ontology reasoning, and RaDON, a plugin for ontology diagnosis and repair. The focus of NeOn differentiates itself from Protégé by utilizing the NeOn Toolkit which is the core of the editor; the NeOn Toolkit features methods and tools for managing knowledge that is distributed, heterogeneous, contextualized, and developed collaboratively. The resulting editor is one that specializes in heavy-weight projects, such as multi-modular ontologies, or ontology integration, but proves to be cumbersome for smaller projects that involve smaller ontologies.

SWOOP

SWOOP [KPS+06] is an editor that is also built for OWL ontologies, and differentiates itself from other editors by being an editor that is built for catering wholly towards OWL. It natively contains a basic reasoner that evaluates the structure of the ontology, as well as the reasoners RDFS-like and Pellet. SWOOP prides itself in being designed in a way that is similar to the Web itself: it is open (with plugin support), it is scalable, and it is distributed (collaborative annotation support via Annotea) [KPS+06]. Overall, SWOOP is a niche editor that is ideal for OWL ontologies, and it built in a way that is very reminiscent to a web-based program. It is a simple and intuitive tool that also provides reasoning functions through plugins such as Pellet and RDFS-like. However, it is extremely specific in that it only handles OWL ontologies, and due to this, does not boast the assortment of plugins that other editors have.

OWLGrEd

The final editor we will discuss is OWLGrEd [LCS12]. OWLGrEd is an ontology editor that allows graphical visualization of OWL 2.0 ontologies using UML class diagram notation. The aspects of an OWL ontology are mapped to the UML counterpart, such as OWL class to UML class, data property to class attribute, object property to association, etc. The UML class diagrams were enriched with new extension notations, such as fields in classes for equivalent class, superclass, and disjoint class. The resulting tool is one that can visualize an ontology as a UML class diagram, garnering all the benefits of the OWL ontology relationships and knowledge representation, and the UML class diagrams ability to illustrate the structural part of the system. However, this comes at the cost of losing the simple hierarchical visualization of the ontology. OWLGrEd has tried to minimize this problem with ontology fragment visualization: an algorithm that displays only a fragment of the ontology that meets the users filtering criteria.

We have looked at several editors in this section, such as Protégé, NeOn, SWOOP, and OWLGrEd. Each editor has its niche uses, and excels in one aspect or another, whether it be Protégé's popularity, and large plugin repertoire, NeOn's specialization in large ontologies, SWOOP's optimization to OWL ontologies, or OWLGrEd's ability to visualize as UML class diagrams.

2.3.2 Reasoning Tools

Now that we have an understanding of the editors that exist for DL ontologies, we will investigate the reasoning tools that these editors utilize. We will be specifically investigating the reasoning algorithms the tools employ, and their operability with the editors that have been mentioned.

The reasoners that are being investigated are BaseVISor [MBK06], FaCT++ [TH06], Pellet [SPG+07], RacerPro [HHMW12], and HermiT [SMH08]. The listed reasoners can be ported as a plugin to an OWL-based ontology editor.

BaseVISor

The reasoner BaseVISor is the response to the criticism of the limited construction of composite properties in OWL [MBK06]. A composite property is informally defined by Matheus et al. as "properties composed of other properties, related to the notion of joins in relational databases", and is demonstrated through the example that an “uncle” is composed of the properties “parent” and “brother”. BaseVISor is a forward-chaining inference engine which translates RuleML rules and facts with n-ary predicates to-and-from BaseVISor rules and facts with binary predicates. BaseVISor uses an XML syntax to define facts, create rules, and issue queries. A fact is a triple defined by subject, predicate, and object elements. In essence, BaseVISor takes the facts given, and determines rules within the ontology that satisfy the *If-Then* predicate, where the *If* is the facts given [MM08]. The resulting rules may be of little significance, and require a human to filter the useful rules from the mundane, but may also be incomprehensible for a human to decode due to the size of the rule.

Pellet

Pellet [SPG+07] was the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals, user-defined datatypes, and debugging support. Pellet was built to support the $\mathcal{SHOIN}(\mathbf{D})$ DL. The features of Pellet are plenty and include conjunctive ABox querying, classification, data type reasoning, axiom pinpointing and debugging, integration with rules formalism, multi-ontology reasoning using $\mathcal{E}$-Connections, and non-monotonic reasoning. For example, in [Fen10], the classification ability of Pellet is used: Pellet is used to classify the security ontology and determine a compliance status of each OWL individual representing a resource. Despite Pellet being one of the initial reasoners, it has not fallen behind the many reasoners that spawned after it such as HermiT or FaCT++. When evaluated based on classification time and inferred axioms, it performed on par with HermiT, and in some experiments, outperformed FacT++ and RacerPro [HLY08].

RacerPro

The RacerPro [HHMW12] system is built off of a DL reasoner that is based on the $\mathcal{SHIQ}$ DL, meaning it omits nominals. Much like its brethren reasoners Pellet, HermiT, and FaCT++, it is based on the tableaux method. RacerPro, when evaluated in [HLY08], often perform the classification and reasoning in slower time than the compared reasoners, but as found in [LLBN09], often produces more correct results.

FaCT++

The FaCT++ [TH06] reasoner implements a tableaux decision procedure for the $\mathcal{SHOIQ}$ DL. Through several iterations and improvements, the current incarnation employs a wide range of performance enhancing optimizations such as absorption and model merging. FaCT++ is both sound and complete, and is designed for experimenting with new tableaux algorithms and optimization techniques. When reasoning over a knowledge base, FaCT++ proceeds in three stages: the first being *preprocessing*. In this stage, the knowledge base is loaded into the reasoner where it is normalize and transformed into an internal representation. Secondly, the reasoner performed *classification*, where it computes and caches the subsumption partial ordering of named concepts. Finally, the knowledge base is checked for *satisfiability*, to decide subsumption problems for given pairs of concepts. The third stage, the satisfiability checker, is regarded as the core component of the system. When evaluating the efficiency of FaCT++, in [HLY08] Huang et al. discovered that with large knowledge bases, FaCT++ requires more time for classifying concepts and generating the class hierarchy.

HermiT

HermiT [SMH08] is an OWL reasoner that is based on a hyper-tableau calculus. Whereas reasoners such as FaCT++, Pellet, RacerPro and other reasoners based on the tableaux method, HermiT does not suffer from a performance problem due to non-determinism and model size. The methods behind HermiT strongly emphasize the performance priorities: there exists an "anywhere blocking" strategy which limits the sizes of models that are constructed, and the hypertableau calculus which greatly reduces the number of possible models which must be considered. The results of the experiments show that HermiT is as fast as other DL reasoners when classifying simple ontologies, and usually much faster when classifying more difficult ontologies [SMH08, HLY08].

There exists several reasoners for DL, and we have only touched on some of the most popular ones. Although all the reasoners appear to be similar – with a great many employing the tableau method – they have niche uses. The minor intricacies innate to each reasoner makes them more suitable for one task or another as demonstrated through the experiments in [GBJR+13]. Depending on the metric that is most prioritised, one reasoner may be more advantageous than another.

2.4 Concept Interpretation and Ontological Configuration

In the previous sections we have evaluated the current state of ontology formalisms, how we reason with DL, and tools that implement the reasoning methods and ontology formalisms into software. In this final section of the Literature Survey, we will assess the current state of interpreting concepts with data types and configuring ontologies by investigating related research.

2.4.1 Contextual Ontologies

Contextual Ontologies are first conceived by Benslimane et al. in [BAF+06]. A contextual ontology is an ontological idea where the ontology may have multiple interpretations. In a traditional DL-based ontology, there exists the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The interpretation maps the respective domain, Δ, to the concepts, individuals, and relations defined through the interpretation function, $\cdot$. Contextual ontologies extends this by allowing for a range of interpretations, such that the contextual interpretation is defined as $\mathcal{I} = (\mathcal{I}_0, \mathcal{I}_1, ..., \mathcal{I}_t)$. Each interpretation within this set is a non-contextual interpretation, which consists of the interpretation domain and function as stated earlier. What this contextual interpretation allows for is the definition of concepts in multiple contexts. The example in Example 3.1 would be impossible for a traditional DL-ontology, but is simple and straight-forward in contextual ontologies.

Example 2.4.1. *An employee can be defined in one context (s_1) as anyone who has an employee number, or in another context (s_2) as anyone who works for a company. The assertion for an employee in contextual ontologies would be:*

$$Employee = (\exists EmployeeNumber.Number)[s_1] \sqcup (\exists WorksFor.Company)[s_2]$$

The notion of having contexts for the concepts within an ontology calls for a rethinking of how we perceive ontologies. Traditionally, ontologies could be thought of as contextual ontologies with one static context. Contextual ontologies abstracts this by allowing for the ontology to be interpreted by several contexts. The critical idea of contextual ontologies is the ability to define the same concept through multiple assertions. However, these assertions must all be written using DL, meaning that despite being able to define a concept with different contexts, each interpretation is still limited to the limitations of DL, and none will have data types in their definition. A contextual ontology is a critical evaluation of how researchers critically evaluate traditional ontological concepts. The idea behind contextual ontologies is to extend how we think of a concept from being a single static interpretation to one that can be one of many interpretations. This idea is being explored in other fields, such as Distributed Description Logic [BS03], which aims to handle complex mappings between domains, by unifying concepts between multiple ontologies. However, contextual ontologies and distributed description logic focus on the concept level of the ontology.

2.4.2 Upper Ontologies

Contextual ontologies offers a way to understand concepts in more than one way, and provides us a specific understanding through a single context. Another approach to understanding ontologies in multiple ways is the notion of upper ontologies. As described in [Hoe10], an upper ontology is designed to provide semantic interoperability of domains across multiple domains. In essence, an upper ontology is a unifying agent between multiple domain ontologies with the goals of allowing interoperability between these ontologies. Upper ontologies provide these general concepts that are common to all domains so that they may be used as foundation for the domain ontologies. Each domain ontology that pertains to the upper ontology it belongs to can be thought of as a configuration of that upper ontology – it is a way of understanding those general concepts.

Although we have discussed the notions of what an upper ontology is, it is important to investigate the types of upper ontologies that have been developed. First we will investigate Suggest Upper Merged Ontology (SUMO), proposed by Niles et al. as a unifying force between all the existing ontologies [NP01]. It would be a single, comprehensive, and cohesive structure consisting of the ontologies available on the Ontolingua server, Sowa's upper ontology, various mereotopological theories, ontologies developed by ITBM-CNR, and other sources. SUMO is not the only upper ontology – there exist several upper ontologies such as MASON [LSDS06], an upper ontology for the manufacturing domain, or BIOTOP [BSSH08], an upper ontology for the life sciences. To help better understand upper ontologies, we further investigate BIOTOP. As discussed in [BSSH08], BIOTOP has the purpose of providing an ontologically sound layer for linking and integrating various specific domain ontologies from the life sciences domain. Alongside BIOTOP, there exist several other Upper Ontologies in development, such as the Ontology of Biomedical Reality, which attempts to integrate ontologies from anatomy, physiology, and pathology. BIOTOP provides a hierarchy of biological processes, a hierarchy of biological functions, as well as several qualities and roles. These concepts of BIOTOP hope to integrate the existing domain ontologies by superseding the top concept of the respective domain ontologies. For example, the top concept of the Cell Ontology – a Cell – matches the Cell concept within BIOTOP, and so the Cell Ontology can be found within BIOTOP. BIOTOP achieves this with various other ontologies of the domain, and is thus a unifying ontology between these specific domain ontologies. In essence, an upper ontology aims to describe very general concepts that are the same across all knowledge domains, as we have seen with BIOTOP. In this sense, an upper ontology does not aim to describe and conceptualize one single domain, but rather, all of them.

2.5 Conclusion

In this survey we have explored the current formalisations of ontologies, the tools that formalise ontologies, reasoning methods, and techniques researchers have investigated to perform interpretations to allow for multiple understandings of a single ontology. Although DL are expressive and have a plethora of tools to assist in the creation and edition of ontologies, they do not have a tool or current technique that allows for the interpretation in the way we desire to ultimately allow for configurations of ontologies. The research into contextual ontologies is the closest research to the notion of configuring an ontology. Within a contextual ontology, a concept may be defined using DL with multiple contexts, where each context is a separate interpretation that may provides a different understanding of the concept. However, the contextual ontologies do not have a tool to support their research, and although they call for a new way of thinking about concepts in an ontology, they are still limited by the DL. The concepts defined in a contextual ontology are still strictly defined through relations to other concepts. There also is a lack of reasoning methods that would utilize configurations in the reasoning process.

Chapter 3

The Design of Conto

In the previous chapter we investigated the tools in the current literature that are used to create and edit ontologies in various ontology formalisms. Despite the number of existing tools, and the versatility provided by each, the issue of being able to interpret concepts with a data type remains unaddressed. To remedy this issue we introduce Conto: an extension to the Ontograf plugin for the Protégé editor. Conto allows the user to interpret a concept with different abstract data types, and ultimately compile the configuration to produce the new configured ontology with the desired interpretations.

In this chapter, we develop Conto. In Section 3.1, we list our assumptions in creating the tool. In Section 3.2, we list the requirements and objectives relating to Conto. In Section 3.3, we formally introduce Conto, and detail the design and architecture choices we made to help achieve these requirements and objectives. Finally, in Section 3.4, we provide the expected benefits Conto will bring to the community.

3.1 Assumptions

When designing Conto, we make the following assumptions:

(i) The ontology has been formalized using the OWL or OWL2 languages.

(ii) Identical individuals within the ontology are denoted using the Protégé relation "same individual as".

(iii) An individual or concept belongs to at most one concept. The representation of an individual belonging to multiple concepts is accomplished by having multiple copies of the individual that are related to each other through the "same individual as" relationship, and each copy belonging to one of the desired concepts.

The list of assumptions delineate the primary restrictions of Conto, but were put in place for specific reasons. As we have mentioned, there exists numerous ways for an ontology to be formalized, each potentially varying on drastic properties such as the formalism used, such as FLogic or DL. From this, we had to restrict the formalism that Conto operates on. Due to the fact that it is a plugin for Protégé and the popularity of ontologies written in OWL or OWL2, the first assumption was drawn. Within Protégé, relations can be made to fit a users need, and so it is possible to detail that two individuals are identical in several ways. We created the second assumption to create a built-in standard for this definition of identical individuals. The final assumption was described to ensure the hierarchical structure of the ontology that Conto operates on. If an individual (or by extension, a concept) belongs to multiple concepts, this contradicts our definition of sub-concepts being disjoint from one another, and thus, does not meet our criteria of being an ontology. These assumptions and constraints were made to ensure that Conto operates as intended. Conto does not check if these assumptions were met due to the infeasibility of checking all possible situations.

3.2 Requirements and Objectives

This section describes the requirements that assisted in forming the tool and its functionality, as well as the objectives that the creation of the tool aims to accomplish. The requirement goals for Conto are as follows:

(i) The tool must be an extension of the already existing Ontograf plugin for the Protégé editor.

(ii) The tool must allow the user to be able to select a concept from the UI and apply an interpretation to it.

(iii) The tool must contain a repository of interpretations that are readily available to the user.

(iv) The tool must be designed such that adding new interpretations is simple and stream-lined.

(v) The tool must be able to compile the interpretations the user has selected for the concepts into a new ontology.

(vi) The tool must allow the user to upload an already-made ontology written in OWL and interpret one of the existing concepts.

(vii) The tool shall only allow the user to interpret a concept in one way per configuration, i.e., it will be impossible to 'stack' interpretations on the same concept within a single configuration.

(viii) The tool must not allow the user to apply an interpretation to a concept that is not compatible with the interpretation.

From the listed requirement goals we were able to determine the major objectives of Conto. We desired a tool that was easy for the user to use by allowing for the interaction with the graphical representation of the ontology. Conto should also be simple for developers to add new interpretation methods to the tool. This will allow for the tool to continuously grow, and adapt for user's needs – if they desire their concepts to be interpreted a specific way that does not exist within the interpretation repository, they should be able to develop and integrate the interpretation method with the tool. Conto should therefore be a lightweight tool that allows for the interpretation of concepts.

We also have the objective of starting the interpretation repository with a default list of available interpretations including Set, Bag, and List. We chose these three interpretations to be the beginning default interpretations because of their versatility in being applicable to a large variety of domains, as well as their ability to impose an abstract data type on the concepts. We also believe that the abstract data types we have chosen to be the default have inherit functions to them which will be useful in a wide range of domains. As we mentioned, since interpretations can be developed and added to the repository, creating more specific interpretations that are applicable to more specialized domains is a trivial matter.

The notions of a unique name assumption and consistency within the ontology were evaluated and ultimately deemed to be out of the scope of Conto. With the ability to interpret concepts, it is possible to create duplicate concepts which conceptualize the same part of the domain (through a series of interpretations and operations), defying the unique name assumption. In addition to this, it is possible these two concepts are not consistent with each other; the concepts may contradict each other by have dissimilar instantiations. Although the concepts conceptualize identical parts of the domain, one concept may have individuals that the other concept (which is supposed to be identical) does not. It is left for future work to create a means of interpretation that produces consistent ontology models.

3.3 The Design of Conto

This section provides a layout of the purpose and architecture of Conto. The design process will be discussed to relate it to the requirements and objectives for Conto. Figures will be included which help explain the architecture and designs. In Section 3.3.1, the architecture of Conto will be detailed, describing the components that compose the tool and how they interact with the Protégé tool. Finally, in Section 3.3.2, the justifications behind the design choices will be provided, as well as how the requirements have been met by the architecture of Conto.

3.3.1 Architecture

Investigating a Component-Based Architecture

The architecture of Conto describes how the components interact with each other. Conto is required to be an extension of the already existing plugin Ontograf for the Protégé editor. Conto itself consists of two main components: the main body which applies interpretations to the concepts and holds the configuration for compilation,

and the repository which holds all the methods behind the interpretations. As detailed in the requirements, the repository must be able to have a “plug-and-play” interaction with the user. Meeting this requirement would allow the user to add any new interpretations they have created or downloaded and include it into the repository and immediately apply them to their ontology. This also means that the main body of the tool must view the repository as a black box. Regardless of which interpretations exist within the repository, they must all be applicable to the concepts, or notify the user if the concept and interpretation are not compatible.
To meet all this requirement of the repository, the overarching architecture of the tool is a Component-Based Architecture. Figure 3.1 illustrates how Conto behaves like this architecture, and how Conto interacts with the Ontograf and Protégé environments. The key ideas behind a Component-Based Architecture is that the system is decomposed into reusable, cohesive, and encapsulated component units which can be swapped with another unit if the new unit provides the same base functionality.

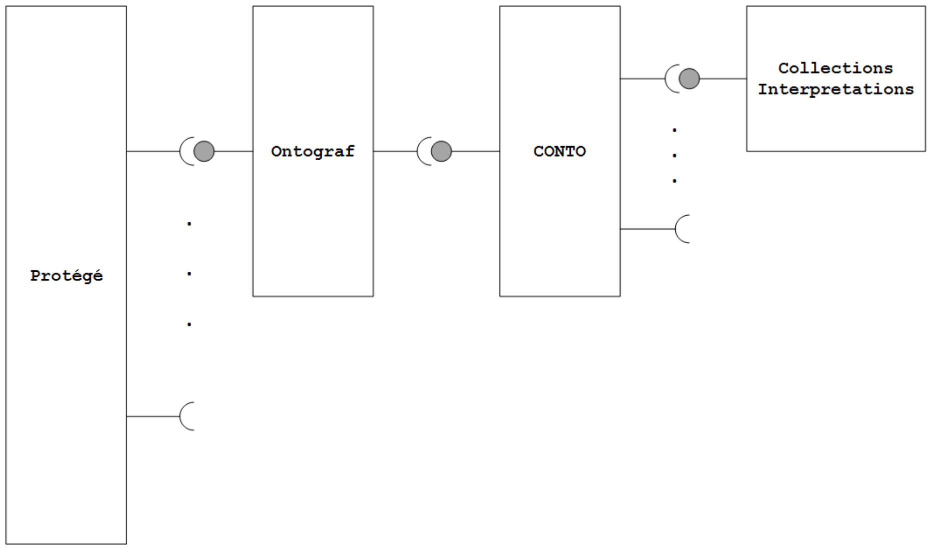


Figure 3.1: The architecture that demonstrates how Conto interacts with Protégé

From the description of Conto, we can see that the system can be decomposed into the following components: the abstract concept that consists of the Protégé environment and Ontograf, the main body of Conto, and the repository of interpretations. The Component-Based Architecture style allows for the repository to have that "plug-and-play" interaction that we desire. The user is able to plug in a new interpretation if they so desire, or they may replace one interpretation with an entirely new one that meets their needs.

Refining as a Broker Architecture

The components that are described by the Component-Based Architecture are abstract in nature: they do not lend themselves to describing what the internal classes may be. The component that is Conto is too abstract; it neglects to describe what is contained within that component. In essence, the abstract idea of Conto is a black box that offers all functionality that we have discussed but does not contain the actual interpretation methods. By evaluating and understanding the role of Conto, we can deduce that there exists a second architecture that exists within the Conto component, and what that architecture is. Figure 3.2 demonstrates the behavior of Conto, in particular, that the tool mediates communication between Ontograf and the interpretation repository. If Ontograf and Protégé were to be viewed as a pseudo-client and the repository of interpretations a server, then Conto can be though of as a broker, resulting in a Broker Architecture. It handles information from Protégé and Ontograf (e.g., a concept within the ontology and a desired interpretation to be applied), and recovers said information about the interpretation within the repository. From here, it applies the interpretation and returns the interpreted concept.

Thinking of Ontograf and Protégé as a pseudo-client is not an incorrect assumption because of the nature of the component-based architecture. The physical user will be interacting with the Protégé client, but as described by the components, Protégé then communicates with Ontograf, and ultimately Conto. If we reduce these steps into one abstract boundary component, we result in the pseudo-client that is Protégé and Ontograf.

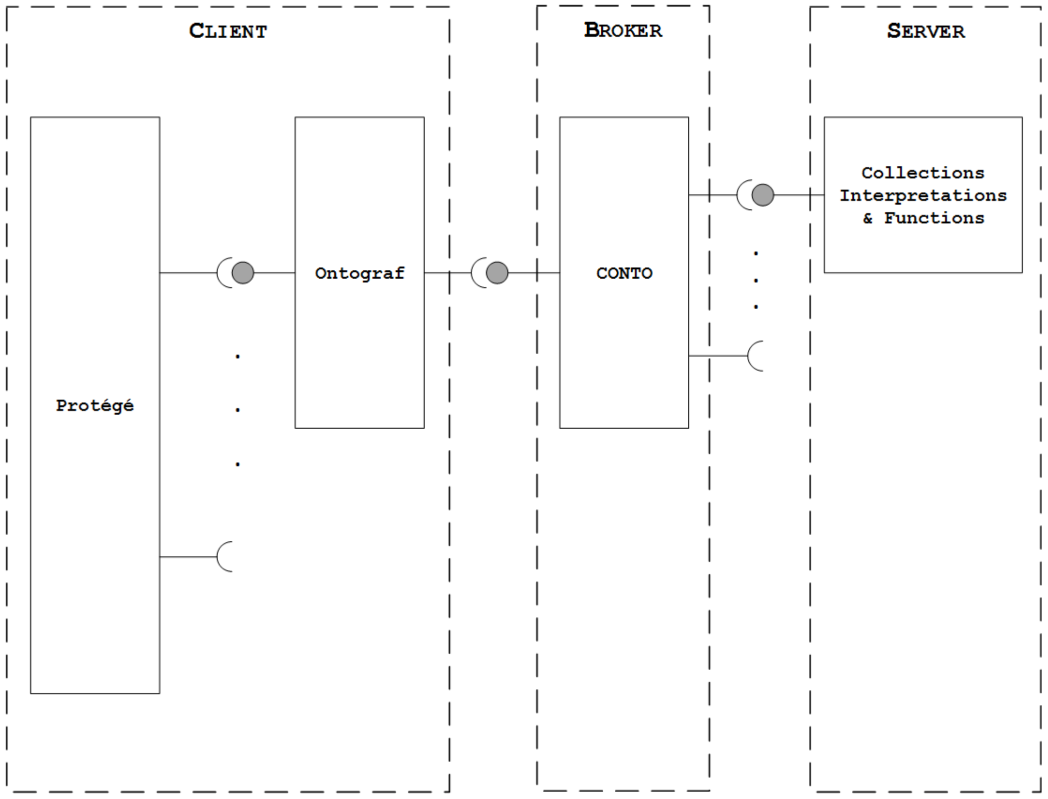


Figure 3.2: How Conto is a Broker between the repository and Protégé

Identifying the Classes

Through the description of Conto, the classes can begin to be formulated. In Figure 3.3, we see the basic classes that compose Conto. The core classes that exist is the

Interpreter, the Ontology Holder, the Uploader, and the Assembler. There also exists the Repository which consists of the interpretations and interpretation functions. The Interpreter and Uploader communicates with this repository as a sort-of boundary class: the classes do not directly contact the interpretation functions, instead they contact the boundary known simply as the Repository. It is how the two components communicate with each other. When the user wishes to apply an interpretation to a concept, the Interpreter class handles this by taking the concept and contacting the Repository for the desired interpretation. This interpretation is placed into the Ontology Holder. The Ontology Holder is an entity which stores the interpreted concepts for compilation. When the user wishes to compile their ontology with the configuration, the Assembler takes every interpretation that has been selected from the Ontology Holder, and compiles the ontology to be displayed using the Ontograf visualizer. If the user instead wishes to upload an OWL ontology, then the Uploader is contacted, which processes the OWL file, and interprets the concept (if the user desires an interpretation) using the interpretation function provided by the Repository. The Repository itself consists of two classes, the interpretations and their respective functions. The Interpretations entity contains the functions behind the actual interpretation – how to interpret the concept. The Functions entity are the associated functions with the interpretations, for example, the union and intersection functions for the set interpretation.

With this structure, we can see that we have met our objectives and requirements. We desired a light-weight tool that was an extension of the Ontograf plugin for Protégé, as well as be built in a way to allow for new interpretation functions to be easily added. We extend the Ontograf tool by using their graphical editor to display our configured

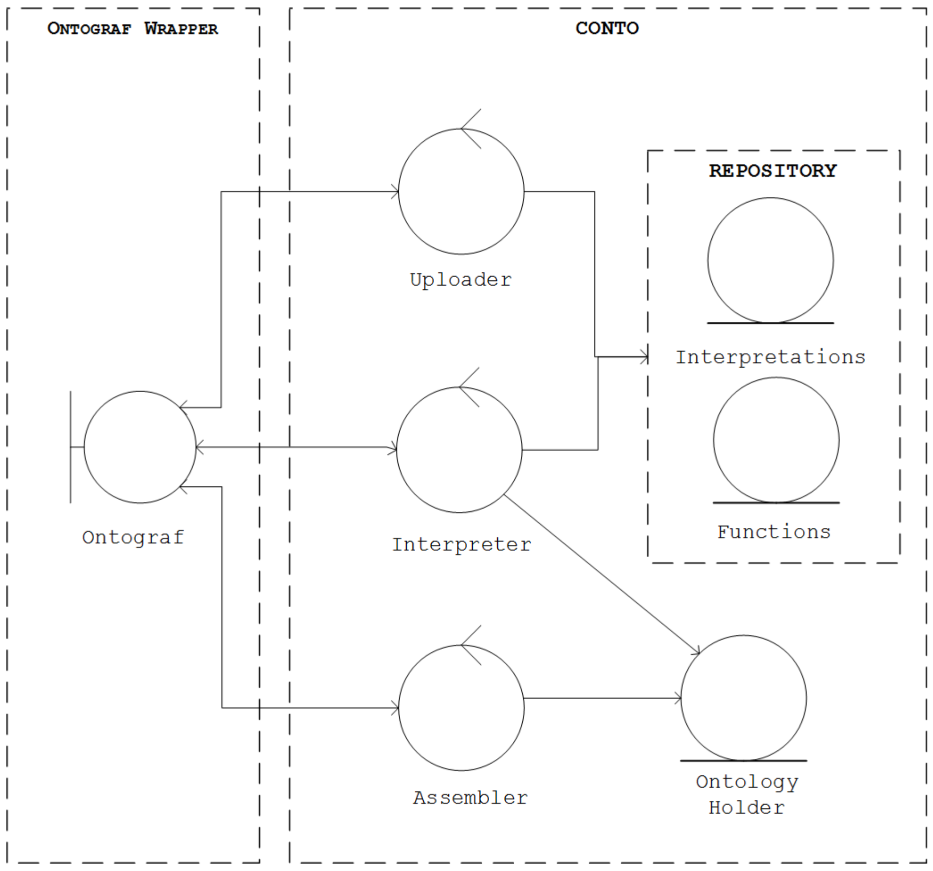


Figure 3.3: The analysis class diagram of Conto

ontology, as well as providing Conto with a method of input. The interpretation, compilation, and upload is done entirely within the tool, and thus is an extension of Ontograf that can be removed if desired. Finally, the Repository that contains all interpretation functions allows for the development of new functions with minimal changes to the interpreter itself. By adding the interpretations and their associated functions (if any) to the Repository, the Interpreter and Uploader would only need to be updated with the names of the interpretations.

3.3.2 Design Choices

In the earlier subsections we have described the architecture of Conto, as well as the classes the compose it. We concluded that Conto is a Component Based Architecture with an internal Broker Architecture. In this subsection, we will evaluate other architectures that were considered, and why they were ultimately discarded. By doing this we will resolve the choices that were made about the classes and architecture.

One of the first architecture styles that was considered when designing Conto was the Layered Architecture. This architecture is often used when creating a tool that is embedded within another greater tool. However, there is a slight nuance to why Conto cannot be layered, and that is because of how the user interacts with Conto. In a layered architecture, the user interacts with the greater encapsulating tool, which in turn communicates with the lower layers (i.e., the user has no direct contact with the lower layers). This contradicts one of our crucial desires: that the user directly interacts with our tool to determine which interpretation they wish to use. In fact, the user may have minimal contact with the actual Protégé functionality. The only communication that Conto has with Protégé and Ontograf is to collect the concepts that have been created, and to display said concepts.

Another popular architecture we considered is Model-View-Controller (MVC), which is a popular architecture for interactive tools. Referring to Figure 3.2, we can see a very close similarity to that of the MVC architecture. The pseudo-client, broker, and server respectively parallel the view, controller, and model. However, in an MVC architecture, the model is static. This means that although the information stored within the model may change, the physical component of the model cannot. One of the primary functional requirements we had was that Conto could accommodate the swapping and removal of interpretations. A strict MVC architecture does not allow for this kind of dynamic behavior.

The final alternative architecture that we considered was the Service-Oriented Architecture (SOA). In a SOA, services exist that have a well-defined functionality, are self-contained, and they are independent from other services. These services are available for the users, and provide their respective functionality to the user. A service directory also exists to assist the user in finding a service. Although the tools functionality shares several similarities to that of a SOA, there are nuances that prevent it from being truly SOA. For instance, if one were to understand the Interpreter as the service directory and each of the interpretations the services offered, then we would see the first problem: In a SOA, the user directly accesses and utilizes the service, whereas with Conto, the user only ever interacts with the Interpreter. The Interpreter is a middle-man for the user to see and use the available interpretations; they are not actually interacting with the interpretation functions. Another conflict between SOA and the design of Conto is that the functionalities of Conto are not independent of each other – a requirement for the services within a SOA. For example, the upload function depends on the interpretation and compile functions. Although the service directory which is a core component for SOA shares many similarities with that of the Interpreter, the nuances leads us to a structure that is more similar to the broker than a directory.

3.4 Expected Benefits

The immediate benefits from Conto are best witnessed through the previous example, however, the expected benefits with the current understanding of ontologies and respect to future research is there. The insight provided by interpreting concepts is invaluable due to it allowing us to represent knowledge that was otherwise impossible. This means that ontologies, which were traditionally thought of as only structures which related concepts within a domain, can be instead thought of a device which provides structure as well as understanding to the data. We can understand concepts as abstract data types and can apply their respective functions. Adding this depth of understanding an ontology as not only the structure of the concepts, but also what the concepts are semantically, calls for a change in how we understand the domains the ontologies conceptualize.

The trajectory of future research is also impacted by this understanding of interpreting concepts. The current reasoning processes do not account for interpreted concepts, and their associated functions. Future research into reasoning must take into account and utilize these interpreted concepts to maximize the knowledge acquired. The interpretations that are created in this work are the first steps – several more interpretations can be developed that provide different insights into a domain.

3.5 Conclusion

In this chapter, we introduced the prototype tool Conto. We provided the requirements and objectives which illustrated the first steps into the design of Conto, and what we wanted to accomplish with the construction of the tool. We described and detailed the architecture of the tool, justifying the decisions by contrasting our chosen Component-Based Architecture hybridized with a Broker Architecture with other popular architectures. By describing our architecture we also demonstrated how the chosen architecture and design choices met the requirements we set out to achieve.

We ended this chapter by providing expected benefits of this tool, emphasizing the changes in how we understand ontologies, and the future research potential into reasoning engines.

Chapter 4

Examples for the Usage of the Tool

In this chapter, we use Conto to investigate different ontologies to demonstrate the process of interpreting concepts and what knowledge can be learned. Through a series of examples, we will see the versatility of configuring ontologies and the extent of applicable domains. The interpretation methods that we will be discussing are the default interpretations that were mentioned in Chapter 3: Set, Bag, and List. As we earlier discussed, the reason for these interpretations being the default is due to their versatility in being applicable to a large variety of domains, and their ability to impose a data type on the concepts.

The three examples we will examine are as follows: the first will introduce the notion of interpreting a concept, the second infers knowledge between concepts, and the final demonstrates how new concepts can be made by interpretations. In these examples, we will utilize all three of our interpretations: the Set, Bag, and List. We will also utilize functions that the interpretations provide, such as the union and intersection functions of a bag or set.

4.1 Example 1 - Interpreting a Concept

For the first example, we will examine a simplified form of the Travel Ontology[1] [CCC+09]. The travel ontology consists of concepts that pertain to travel destinations, such as activities or accommodations. As Figure 4.1 shows, the ontology has been simplified to fit our assumptions (i.e., strictly mereological relations) by the removal of certain relations, as well as the removal of concepts to reduce the size of the ontology for illustrative purposes.

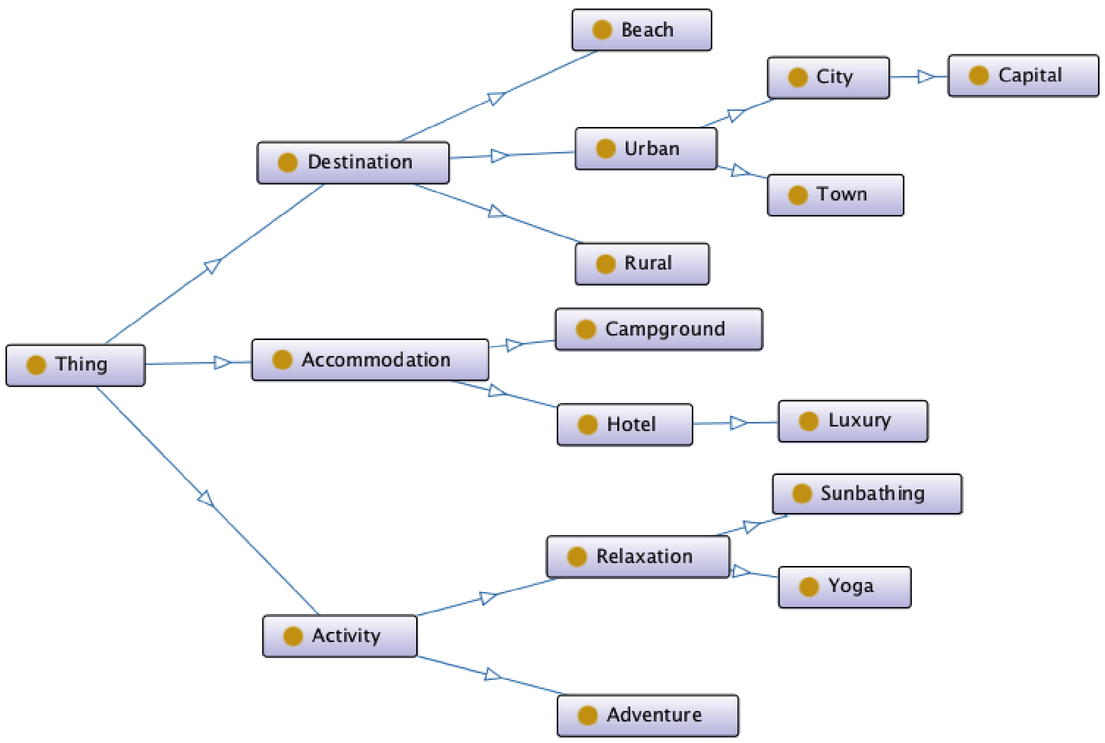


Figure 4.1: The simplified Travel Ontology

[1] `http://protege.cim3.net/file/pub/ontologies/travel/travel.owl`

As we can see from the ontology, we have conceptualized a domain that is populated by data that pertains to tourist travel destinations. The ontology may be populated by data that is input from local databases as well as domain experts (such as travel agents or locals). Focusing on a single concept within the ontology, we can imagine domain experts populating this concept with data, resulting in Figure 4.2.

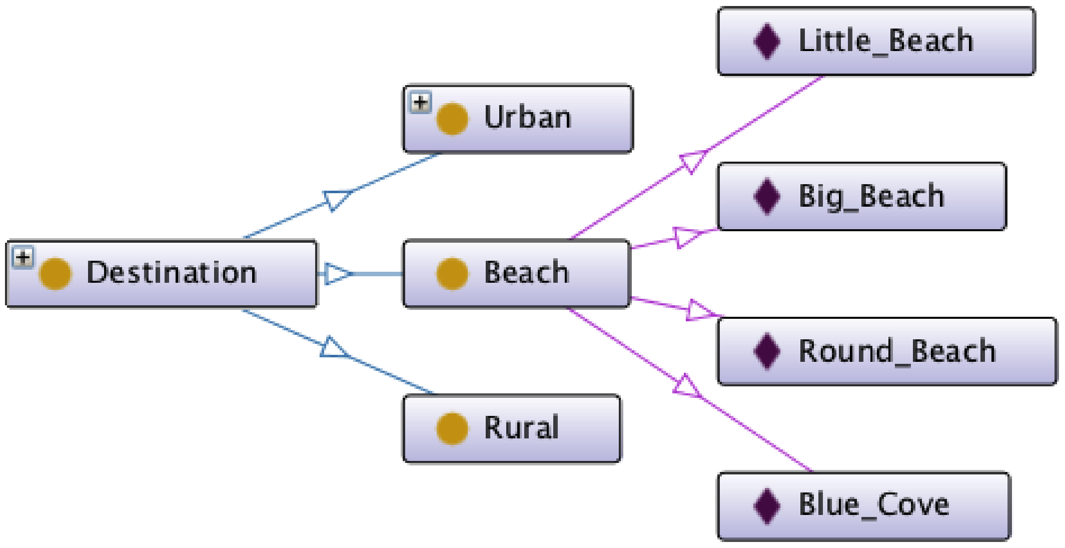


Figure 4.2: The data that was collected for the concept "Beach"

As Figure 4.2 shows, the information that was collected pertains to the concept "Beach". There are four beaches known to the domain experts: Big Beach, Little Beach, Round Beach, and Blue Cove. However, because the data was input by various domain experts, we are worried that perhaps there is an overlap in information: perhaps a travel agent who knows the area knows the beach as another name from another travel agent or what is already recorded in the database. To investigate this, we interpret the concept as a 'Set', resulting in Figure 4.3.

What we see here is that the number of individuals has been reduced to two: only

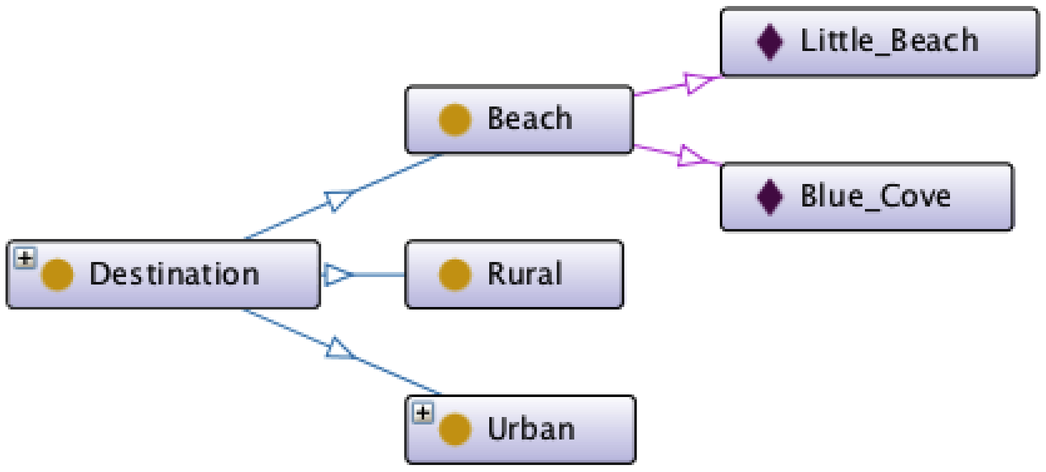
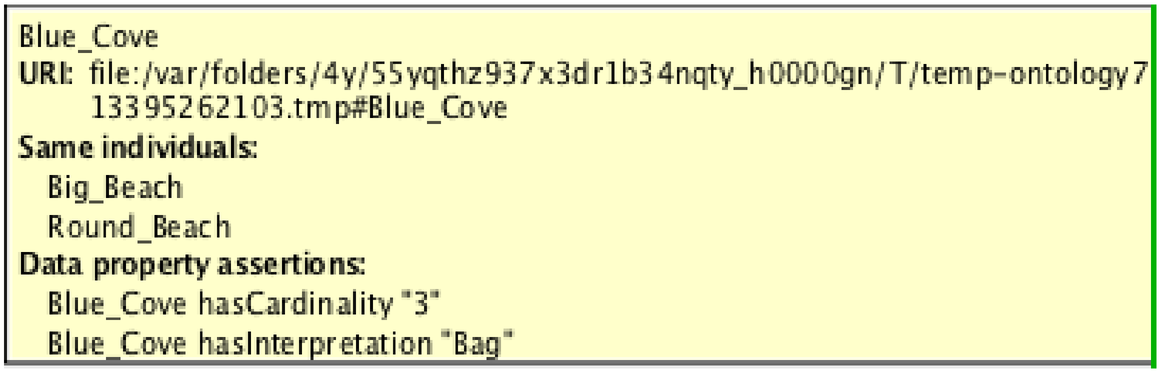


Figure 4.3: The concept "Beach" interpreted as a Set

Blue Cove and Little Beach. By hovering our cursor over the individuals, we can identify which were the duplicates. However, we wish to know how many duplicates existed among the beaches, so we interpret the original concept as a 'Bag'. Now when hovering our cursor over the concept 'Blue Cove', we can see the cardinality as shown in Figure 4.4. As we can see, the individual Blue Cove has been interpreted as a Bag, and it has cardinality 3. Specifically, the other individuals that were identical to Blue Cove were Big Beach and Round Beach.

By interpreting this concept of 'Beach' as a Set or Bag, we have discovered that there were duplicate entries of data put in by the domain experts. The reason for duplicates is unknown and may be accidental or due to error, but by the interpretation of the concept, we have avoided accounting for 4 beaches when in fact there were only 2.



```
Blue_Cove
URI: file:/var/folders/4y/55yqthz937x3dr1b34nqty_h0000gn/T/temp-ontology7
     1339526 2103.tmp#Blue_Cove
Same individuals:
  Big_Beach
  Round_Beach
Data property assertions:
  Blue_Cove hasCardinality "3"
  Blue_Cove hasInterpretation "Bag"
```

Figure 4.4: The information within the individual Blue Cove when Beach is interpreted as a Bag.

4.2 Example 2 - Using Interpretation Functions

For the second example, we will investigate a specific part of an ontology made-up for the purposes of this work which can be found in Figure 4.5. The theoretical ontology conceptualizes a domain related to a company, and the portion given specifically relates to two departments found within the company and the salaries of workers in those departments.

As we can see in Figure 4.5, there exists two departments within the company: the Color-Analysis department, and the Candy-Testing department. The department of Color-Analysis has salaries of \$40,000, \$45,000, and \$50,000, and the department of Candy-Testing has salaries of \$10,000, \$60,000, and \$90,000. With human investigation, it is obvious the variance of the Candy-Testing salaries is much higher than that of the Color-Analysis department.

This example highlights the reasoning potential of interpretations: by interpreting the two departments as numerical lists, we can attain the information in Figures 4.6 and 4.7 by using Conto. The matter of attaining the information is reduced to a trivial process since Conto contains the functions respective to each interpretation.

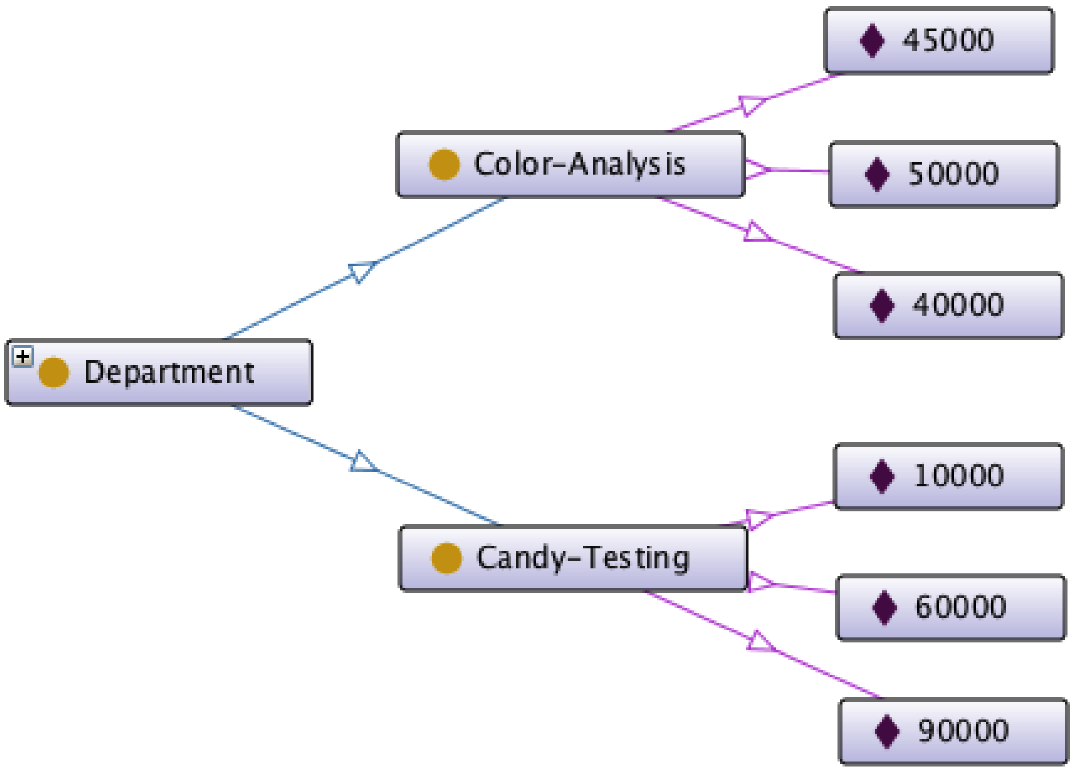
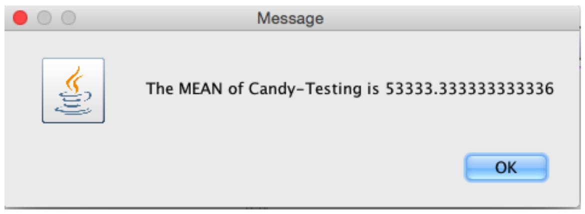
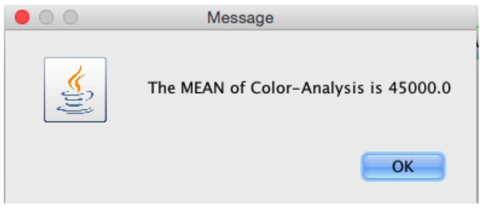


Figure 4.5: The specific portion of the ontology for Example 2.

By interpreting the knowledge, we are able to easily process them using the functions that are inherent to the respective interpretation, and thus leverage the interpreted concept for the information we desire – in this case, the mean or minimum salaries.



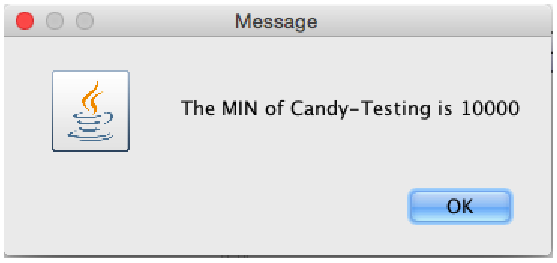
The MEAN of Candy-Testing is 53333.333333333336



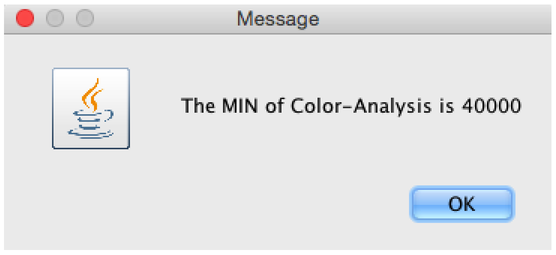
The MEAN of Color-Analysis is 45000.0

Figure 4.6: The means of the two departments.

We can inspect the information we have received from Conto by interpreting the concepts as lists. Figure 4.6 tells us the mean salaries of each department ($53,333.34 and


Message

The MIN of Candy–Testing is 10000


Message

The MIN of Color–Analysis is 40000

Figure 4.7: The minimum values of the two departments.

$45,000), and Figure 4.7 tells us the minimum salaries you can find of each department. The minimums were discovered by taking advantage of the List interpretation that was applied to the concepts – the minimum values are simply the head of the list. The means are discovered by an additional function that Conto offers: a mean function is provided for numerical lists by taking advantage of the numerical data. If we were to compare just the mean salaries between the two departments, we would conjecture that the salaries of each department would be similar. However, with the additional information of the minimum salary, we learn this conjecture is not true. In fact, although the means are similar, the deviation of the means are not. The deviation of the Candy-Testing department is a staggering $43,333.34 compared to the deviation of $5000 for the Color-Analysis department. The final conclusion may imply that the Candy-Testing department has a greater variance among salaries than the Color-Analysis department, or even perhaps the data was incorrectly entered, resulting in the abnormally low minimum. With further inspection and utilization of the interpretations, we could possibly produce even more telling conclusions.

The ontology, as it was originally, only provided a structure for the data – the two departments could not be compared. However, with this introduction of interpretations, we not only could learn information specific to individual concepts, such as the mean and minimum values, but we could compare them to one another. This comparison of the values allowed us to conjecture results that inferred about statistics such as the variance of the data, or the correctness of the input data. Ultimately, these conjectures were created by a domain expert as no reasoner has the ability to utilize these interpretations or their functions, and thus be able to create these conjectures.

4.3 Example 3 - Creating New Concepts via Interpretation

For our final example, we will observe an arbitrary weather ontology that conceptualizes the months and populates them with daily temperature readings. The example ontology is shown in Figure 4.8. We have created concepts for only the first three months of the year, and only populated the first two months (January and February) with data for illustrative purposes. Our goal is to determine the maximum temperature that both January and February experience, as well as the average temperature between the two months. To ensure a deterministic result, two temperatures are considered identical if they round to the same integer using the *round half up* method, which is $q = \lfloor x + 0.5 \rfloor$. If for any two temperatures (x_1 and x_2), they have the same q, then they are the same temperature.

To achieve the first goal of acquiring the maximum temperature both January and February experience, we must first gather the temperatures that are in common between the two months. This can be accomplished by determining the intersection. To intersect the two months, we must first interpret them both as a Set (or Bag), and apply the intersection function. By intersecting the two concepts, we create

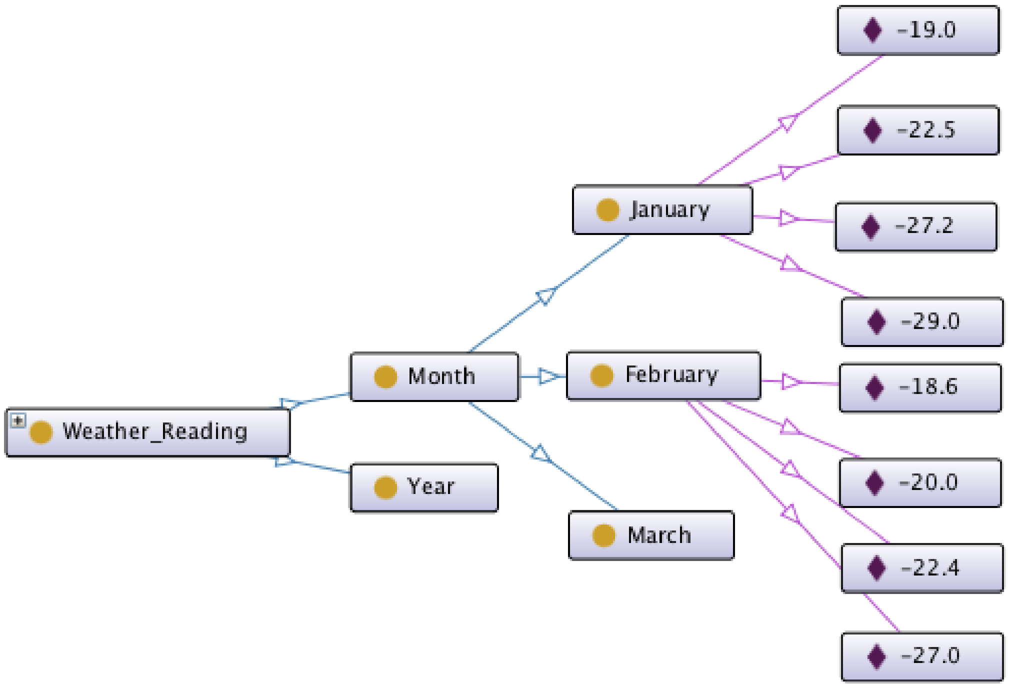


Figure 4.8: The Weather Ontology.

a new concept that is populated solely by the data that is shared among the two concepts being intersected. By observing Figure 4.9, we can see that it is populated by temperatures "-22.5" and "-27.2". These values were determined by noticing that the temperatures -22.4 and -22.5 are identical (they both round up to the value of -22), and keeping the value from January (-22.5). Similarly, the February temperature -27.0 is calculated to be identical to the value of -27.2 from January, and so the -27.2 is kept. We can continue investigating the similarities of the months by interpreting the resulting concept from the intersection as a list to determine the maximum temperature, demonstrated in Figure 4.10.

As shown in Figure 4.10, we notice the maximum to be -22.5. Similar to how the

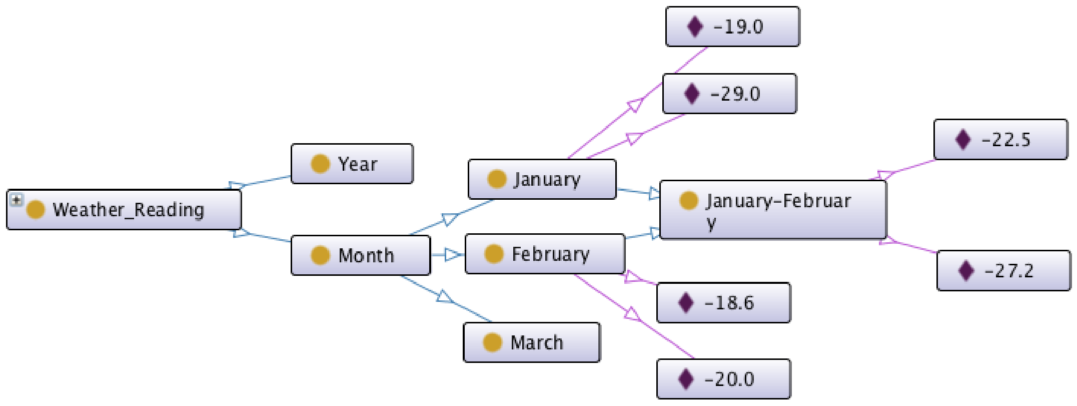
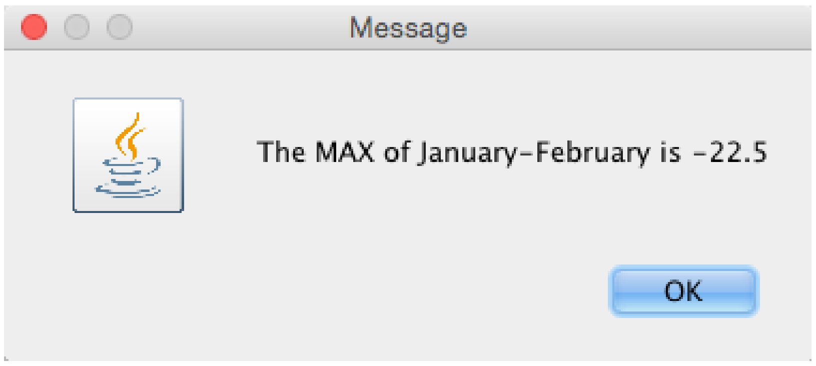


Figure 4.9: The result of intersecting the concepts January and February.

minimum function is taking advantage of the head of the list, the maximum takes advantage of repeatedly taking the tail of the list until the size of the tail is 1. What this tells us is the highest temperature that both January *and* February experiences, is -22.5. Likewise, we could also determine the minimum temperature both months experience together.



The MAX of January-February is -22.5

Figure 4.10: The maximum temperature that is shared between January and February.

We have finished the first part of our task, which was discovering the maximum temperature that both January and February experiences. However, we must now complete the second task: to find the average temperature between both months. We take the original ontology (before the months were intersected), and instead take the union of the two months. By taking the union of both months, we create a new concept that is populated by data that is found in January *or* February. This new concept, shown in Figure 4.11, shows that the concept is populated by every individual that was in either of the two sub-concepts.

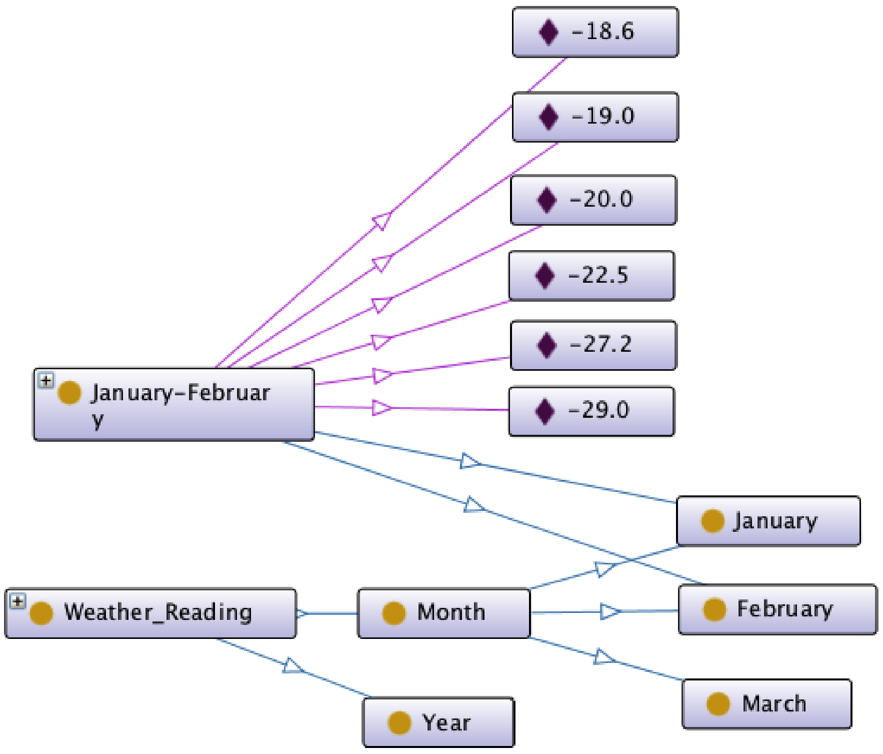
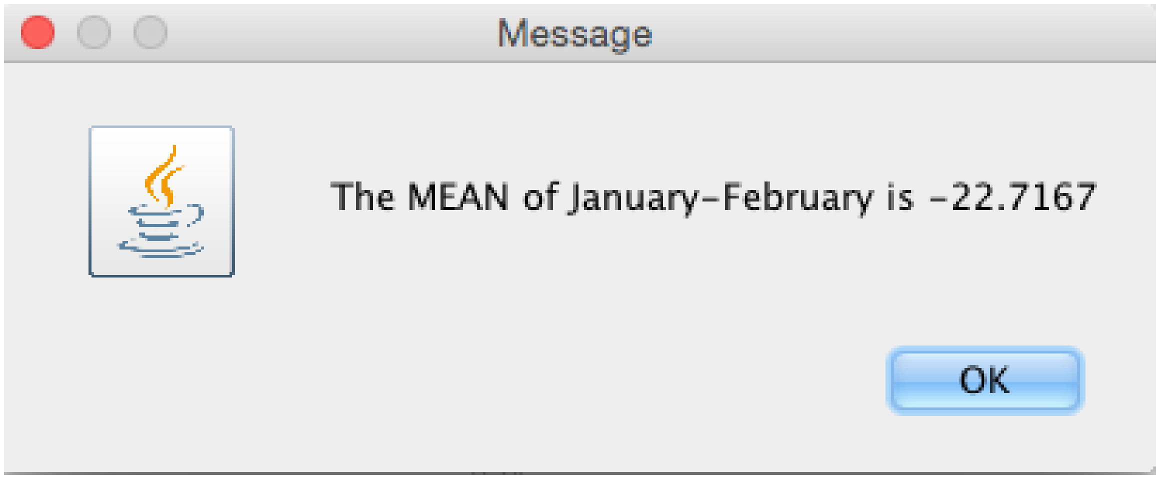


Figure 4.11: The result of the union between January and February.

We wish to find the average temperature that both months experience, so we interpret this new concept as a list, and determine its mean value by taking advantage of the fact it has been interpreted as a numerical list. As shown in Figure 4.12, the mean temperature between January and February is -22.7167.


Message

The MEAN of January-February is -22.7167

Figure 4.12: The average of the union between January and February.

Without the notion of concept interpretation, the information that was discovered in this example would be unreachable. In this example we demonstrated that we can discover new concepts within the ontology that are populated by data that already exists within the ontology. These new concepts represent the union or intersections of concepts. The concept which is created by the union of two other concepts represents the *thing* that contains individuals that are either of the concepts – it is a super-concept. The result of an intersection is its dual, it is a *thing* that contains individuals that are found in both concepts – it is a sub-concept. We used this understanding of union and intersection to determine what we can learn about the data. Specifically, we wished to know what we could learn about information pertaining to January *and* February (the intersection), and about information pertaining to January *or* February (the union). This information – the notion of these new concepts which are populated from already existing data – is able to be discovered and utilized because of the interpretations.

4.4 Conclusion

In this Chapter, we have gone over three different examples. Example 1 was a simple demonstration of what a simple interpretation can do. By interpreting a concept, we discovered duplicate data. We then illustrated an example with Example 2 where we compared concepts through metrics such as minimums or averages. This example provides insight to reasoning potential – the comparing of concepts by methods otherwise impossible. In the example, we discovered that although the averages of the two departments were similar, the variances were much different. In the final example, Example 3, we composed several interpretations and functions to acquire new concepts. We ultimately ended up with knowledge of the maximum temperature that January and February experience, as well as the average temperature between both months. This knowledge was acquired solely because of the interpretations – by interpreting the two months as Bags, we were able to intersect or union them, which resulted in a new concept. This new concept could then be interpreted as a list, which provided us with the maximum or average.

Although we illustrated several points with these examples, there exist other functions that were not shown, such as the median of a list. As the number of interpretation methods grows, the amount of knowledge that can be acquired from a domain increases.

Chapter 5

Discussion

In this chapter, we discuss various aspects of the problem of Conto. In Section 5.1, we discuss some possible application domains for which Conto technique presented in Chapter 3 is suitable, and will be related to the Examples presented in Chapter 4. We also discuss the importance of such techniques and applications. In Section 5.2, we assess the strengths and weaknesses of the main contributions.

5.1 Discussion

As it was mentioned in Chapter 1, several fields of research already contain unimaginable amounts of data, and are acquiring more at an unimaginable rate. Examples of these fields are astronomy [FB12] and businesses [GSLG12]. The information that each respective domain acquires is respective to their own domain – whether it be planetary data, or business transaction – formatted as entries in databases. With the research of translating databases into ontologies, such as the DB2OWL tool [CGY07] which performs said task of translating, we arrive at a situation which Conto is designed for: providing another way to understand an ontology so that we maximize the knowledge that can be acquired.

Even though the domains used for illustrative purposes in the previous chapter were narrow in scope and populated by minimal individuals, we could see valuable knowledge was acquired. It is important to note that this knowledge was always present, it was just unable to be accessed due to the incapability to interpret the concepts. We did not add new data or manipulate the data itself. We changed how we understood the concepts. Conto can thus be applied to the domains that are much larger in scope and with magnitudes more individuals (such as the astronomy or business domain) to provide these new insights, so long as the structure of their ontology is suitable for the tool.

The prototype of Conto demonstrates the usefulness of the interpretation of concepts by allowing us to access this new knowledge that was previously unreachable. If a reasoning process were to be extended to include the interpretation of concepts, then interesting formal structured rules could be found and isolated from the uninteresting rules. Currently, Conto can only output the knowledge that is used for the generation of a rule, and leaves it up to the domain expert to infer a rule from the knowledge. Also, since Conto is utilized by a domain expert, the knowledge being discovered is limited to the configuration the domain expert is focused on. However, Conto is open to concurrent configuration, where it processes all possible configurations of the ontology. By doing this concurrent configuration, all the knowledge that is discovered from the ontology would truly be maximized.

As it was discussed in Section 1.2, one of the primary dilemmas in the field of ontologies is the static representation of concepts. In current literature, concepts are defined in relation to other concepts. With Conto, we believe we have helped conquer this problem by allowing for the interpretation of concepts. We have tackled the problems laid out in Section 1.3, and developed a tool that can configure an ontology and produce novel results.

5.2 Assessment of the Contributions

In this section, we discuss the strengths and weaknesses of the main contributions presented in this thesis. It is important to highlight both the strengths and weaknesses of Conto so that we are able to further refine a solution to the problem of being able to configure ontologies with interpreted concepts.

5.2.1 Strengths of the Contributions

The strengths of Conto lie in three key points: validating the creation of Conto, the versatility of interpretations, and the usefulness of the interpretation functions.

Conto is a prototype to show that the interpretation of concepts is crucial to the reasoning process. Conto excels at this, as the Examples have shown; information is acquired – which can ultimately be understood as rules – that would otherwise be impossible to acquire. The original problem statement stated that the current limitation on concepts inhibits the reasoning process due to reasoners not being able to access some knowledge. Conto effectively removes, or dampens, this limitation.

Conto is able to configure ontologies built using mereological relations between the concepts. Conto already contains three interpretations – Set, Bag, and List – which are extremely useful in providing knowledge, as demonstrated with the Examples. It also boasts a plug-and-play architecture, allowing new interpretations to be incorporated and utilized with ease. This allows a domain expert to specifically mine for information they may seek using an interpretation unique to their domain, such as a domain expert in the business domain interpreting a concept as fiscal, thus having associated tax rates, inflation adjustments for different years, and conversions to different currencies.

Conto not only includes these interpretations, but the functions alongside them, such as Set Union or Intersection. This allows the user to create new concepts from old ones that have been interpreted. Previously inaccessible knowledge can be accessed by functions available from the interpretations. Thanks to the graphical interaction with Ontograf, the user can also visually see how the concepts and individuals may relate via unions or intersections. Much like how the plug-and-play architecture allowed for new interpretations to be made, new functions can also be made (or old ones be extended), such as adding the inverse function for Set.

What this means is that Conto has met the goals set out by this research, and helped reduce this limitation on concepts during the reasoning process. Also, it has been built in a way that allows it to continue to grow. More interpretations can be created to increase the amount of knowledge that can be acquired from an ontology.

5.2.2 Weaknesses of the Contributions

Although Conto provides insight into the interpretation of concepts and the configuration of an ontology, there are limitations and weaknesses to it.

To properly utilize Conto and ensure correct results, the assumptions that were detailed in Section 3.1 in Chapter 3 must be followed. In particular, the ontology that is being investigated must be built off mereological relations. When studying Example 1, it should be noticed we had to modify the Travel Ontology. This is due to the ontology containing relations which resulted in an ontology that was not hierarchical. Conto cannot properly process these non-mereological relations that possibly refer to ancestral concepts. Although this assumption critically hinders the number of ontologies that can be processed using Conto, it should be acknowledged that this can be remedied by a domain expert including the definitions and behaviors of non-mereological relations to Conto. The assumption was originally made due to the impossibility of Conto being able to 'understand' all possible relations that may exist within an ontology, and thus, we have left it to the domain expert to handle how Conto handles these relations on a case-by-case basis.

The other assumptions, such as how identical individuals are defined or how individuals are limited to being instantiations of one concept, may also limit the ontologies that Conto can process. However, these were assumptions that were made during the creation of Conto to avoid unforeseeable problems during the processing of such ontologies. This means that with future updates of Conto, these assumptions will be addressed and lifted.

Another crucial weakness is the lack of reasoning support. There does not exist a reasoner in current literature that utilizes the interpretation of concepts in the reasoning process. However, Conto does provide the knowledge that can ultimately be used to generate rules that a reasoner would normally output. This means that in the current incarnation of Conto, if the user is a sufficient expert within the domain they are studying, they can utilize Conto to acquire valuable knowledge they would ordinarily not be able to. With the eventual introduction of a reasoner, it would add the insurance that no knowledge would be ignored due to negligence or error on the domain experts part, and we also add the ease for any user to acquire knowledge from the domain.

5.3 Conclusion

As discussed in this Chapter, Conto has several immediate uses. Several domains already currently exist with such huge stores of data they can not be properly processed without the use of ontologies. Conto allows for the configuration of ontologies, remedying the current limitation of how concepts within an ontology are defined. With the tool, we are able to interpret concepts so that more knowledge can be extracted that would otherwise be inaccessible. Although the tool achieves all goals that were set out in the beginning of this research, it has its own limitations. Several of the weaknesses come from the assumptions that were made when creating Conto. These assumptions limit the ontologies that can be processed using Conto. However, several of these assumptions will be addressed as we move from a prototype to a commercial tool, and in doing so, lift the limitations of Conto.

Chapter 6

Conclusion and Future Work

In this Chapter we conclude our work. In Section 6.1 we discuss future directions that this research can take. We provide specific work with the purpose for their work, and what they will contribute to the work. Finally, we conclude the work in Section 6.2 with Closing Remarks.

6.1 Future Work

As we have discussed, although Conto helps alleviate the limitation on concepts within an ontology, there is still work to be done. In this section, we evaluate the main future work for the research. The potential gains from completing the listed future work are detailed, as well as how this work has created the potential for the future work.

6.1.1 Reasoning Methods

A critical aspect to knowledge discovery is the reasoning process that is used. The type of knowledge that can be acquired depends on the ontology formalism and the reasoning process. As it currently stands, Conto does not have a reasoner that utilizes the concept interpretations, thus limiting the knowledge that can be acquired. Currently, the domain expert can only acquire knowledge by further processing the output of Conto into rules.

With the creation or modification of a reasoner that utilizes Conto, we eliminate this further processing that the domain expert is required to do, as well as we increase the efficiency of the rule generation. Without a reasoner, the domain expert can only configure an ontology one configuration at a time (then process that configuration). With a reasoner, it could be possible to process multiple configurations concurrently.

6.1.2 Applications

In our work, we presented a few examples to demonstrate the utilization and purpose of Conto. The ontologies that were used were either modified already-existing ontologies, or novel ontologies created for the sake of illustrative purposes. Future work into applications would strengthen the argument for the necessity of the interpretation of concepts. We wish to be able to take an existing ontology and process it using Conto to demonstrate the new knowledge that is acquired.

6.1.3 Automation/Concurrency

In the current incarnation of Conto, the domain expert is required to input the interpretations desired for each concept, and to configure the ontology. However, the ability to upload an ontology and simultaneously interpret a concept indicates that there is a possibility for concurrency in the future. The goal of being able to process every configuration concurrently is a future goal. What this means is that it would be possible to learn everything of significance from a domain, in an efficient and total way.

6.2 Closing Remarks

In this work we have developed Conto, a prototype tool that allows for the interpretation of a concept with an abstract data type. From studying the current literature, we learned that a severe limitation on the reasoning process is the static definition of concepts, resulting in knowledge that is often shallow and non-interesting. By introducing the configuration of an ontology, where a configuration is composed of the interpretations applied to the concepts, we reduce this limitation. We provide a way for a domain expert to understand their domain in multiple ways, and glean knowledge from these understandings that would otherwise be inaccessible and thus forgotten. With Conto, we extended the Ontograf plugin within the Protégé ontology editor so that it is possible to configure an ontology. We detailed the architecture and design of Conto, and highlight the plug-and-play design allows for the easy addition of new interpretations, allowing for Conto to meet the needs of the domain expert. Examples demonstrate the use of Conto over several different ontologies, and describe the type of knowledge that can be acquired that normally is inaccessible. We finally list the strengths and weaknesses of Conto, focusing on the strength of the plug-and-play design and that several weaknesses are results of the assumptions made which will be addressed in future versions.

In conclusion, Conto addresses a critical problem that we believe hinders the reasoning process for ontologies. As it currently stands, the knowledge that is acquired is limited due to how concepts are defined. Conto remedies this by allowing a domain expert to retain the original relations to other concepts, but also imbue the concept with a structure that allows an understanding of its individuals that was not originally there. This structure that has been incorporated into the concept can be used to compare the concept to other concepts, or perhaps to investigate the individuals that belong to the concept. With the creation of Conto, we aim at better understanding the vast amounts of data so that new knowledge can be acquired that is otherwise unobtainable. Although there exists weaknesses within Conto, such as its reliance on the assumptions, we strongly believe Conto is the first steps in a new way of understanding ontologies, that will ultimately be necessary to address the problems with Big Data.

Appendix A

Detailed Class Diagram

This appendix contains the detailed class diagram of Conto in Figure A. This figure goes into detail of the classes that were introduced in the analysis class diagram of Figure 3.3.

The controller classes – Compiler, Interpreter, and Uploader – are detailed as such: classes that have functions to perform their respective responsibilities. The Ontology Holder stores the information and can be seen as such by the get and set functions. The Interpretation and Interpretation Functions represent the repository, and can be seen to store all information related to the interpretation methods and functions.

Of note is that the Interpretation class is an abstract class depicting the player-role. The default interpretations of Set, Bag, and Ordered List are inherited classes of this abstract class (denoted by the hollow pointed arrow).

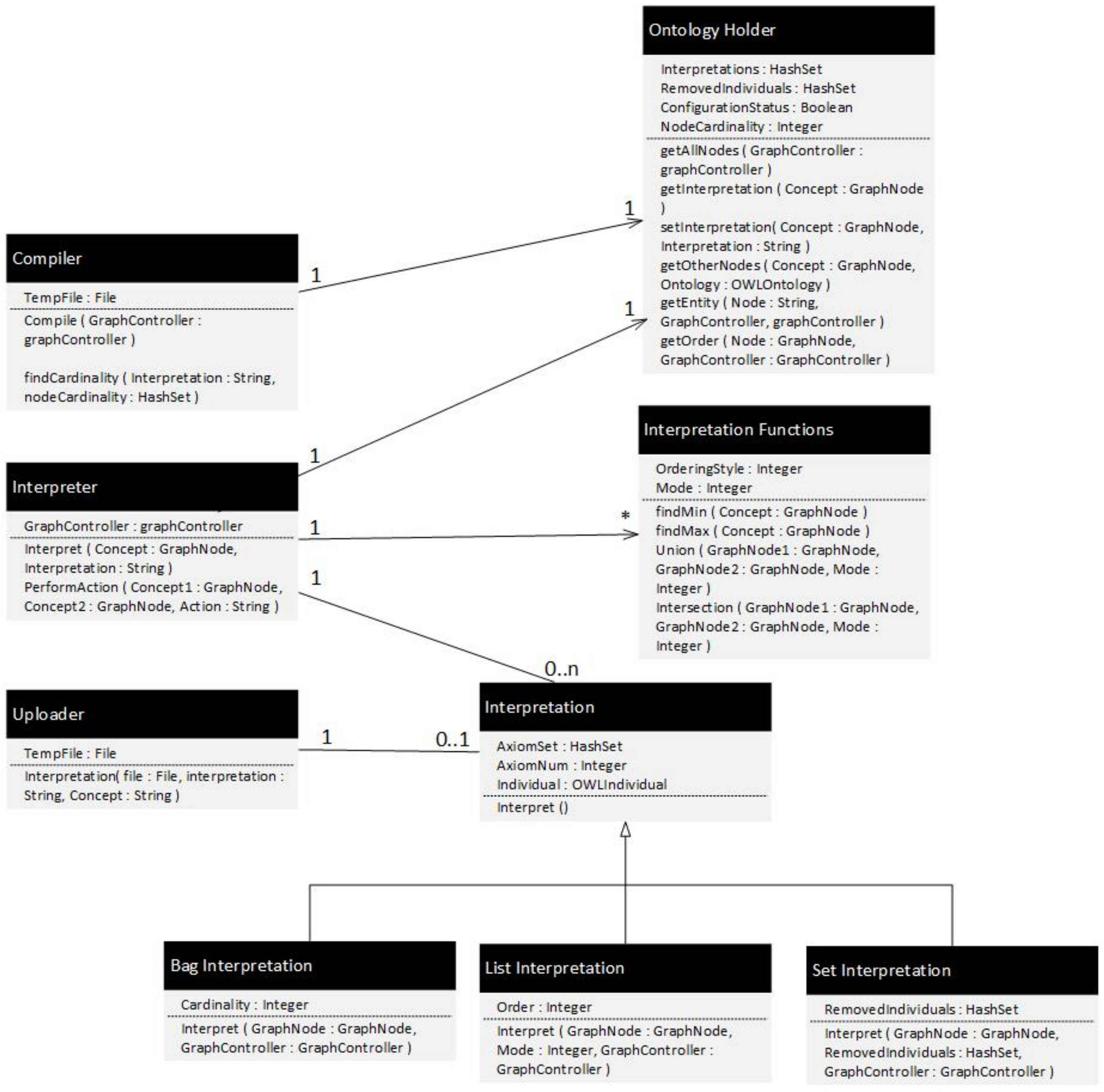

Ontology Holder
Interpretations : HashSet
RemovedIndividuals : HashSet
ConfigurationStatus : Boolean
NodeCardinality : Integer
getAllNodes (GraphController : graphController)
getInterpretation (Concept : GraphNode)
setInterpretation(Concept : GraphNode, Interpretation : String)
getOtherNodes (Concept : GraphNode, Ontology : OWLOntology)
getEntity (Node : String, GraphController, graphController)
getOrder (Node : GraphNode, GraphController : GraphController)
Compiler
TempFile : File
Compile (GraphController : graphController)
findCardinality (Interpretation : String, nodeCardinality : HashSet)
Interpreter
GraphController : graphController
Interpret (Concept : GraphNode, Interpretation : String)
PerformAction (Concept1 : GraphNode, Concept2 : GraphNode, Action : String)
Interpretation Functions
OrderingStyle : Integer
Mode : Integer
findMin (Concept : GraphNode)
findMax (Concept : GraphNode)
Union (GraphNode1 : GraphNode, GraphNode2 : GraphNode, Mode : Integer)
Intersection (GraphNode1 : GraphNode, GraphNode2 : GraphNode, Mode : Integer)
Uploader
TempFile : File
Interpretation(file : File, interpretation : String, Concept : String)
Interpretation
AxiomSet : HashSet
AxiomNum : Integer
Individual : OWLIndividual
Interpret ()
Bag Interpretation
Cardinality : Integer
Interpret (GraphNode : GraphNode, GraphController : GraphController)
List Interpretation
Order : Integer
Interpret (GraphNode : GraphNode, Mode : Integer, GraphController : GraphController)
Set Interpretation
RemovedIndividuals : HashSet
Interpret (GraphNode : GraphNode, RemovedIndividuals : HashSet, GraphController : GraphController)
1
1
1
1
1
*
1
0..n
1
0..1

Figure A.1: Detailed Class Diagram of Conto.

Glossary

Conto Is the tool that has been made for the purposes of this research.

Protégé Is the Ontology editor that Conto plugs into.

Acronyms

ABox	Assertional Box.
DL	Description Logic.
FCA	Formal Concept Analysis.
FLogic	Frame Logic.
FO	Frame Ontology.
GO	The Gene Ontology.
KIF	Knowledge Interchange Format.
LTL	Linear Temporal Logic.
MO	The Music Ontology.
OO	Object Oriented.
OWL	Web Ontology Language.
TBox	Terminological Box.

Bibliography

[ABB+00] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[AIS93] Rakesh Agrawal, Tomas Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD '93 Proceedings of the 1993 ACM SIGMOD Internation Conference on Management of Data*, 1993.

[Ala03] Harith Alani. Tgviztab: an ontology visualisation extension for protégé. 2003.

[Art94] W Brian Arthur. Inductive reasoning and bounded rationality. *The American economic review*, pages 406–411, 1994.

[BAF+06] Djamal Benslimane, Ahmed Arara, Gilles Falquet, Zakaria Maamar, Philippe Thiran, and Faiez Gargouri. *Contextual ontologies.* Springer, 2006.

[BFGR07] Andrea Bellandi, Barbara Furletti, Valerio Grossi, and Andrea Romei.

Ontology-driven association rule extraction: A case study. *Contexts and Ontologies Representation and Reasoning*, page 10, 2007.

[BGL12] Franz Baader, Silvio Ghilardi, and Carsten Lutz. Ltl over description logic axioms. *ACM Transactions on Computational Logic (TOCL)*, 13(3):21, 2012.

[BL04] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Elsevier, 2004.

[BL12] Chitta Baral and Shanshan Liang. From knowledge represented in frame-based languages to declarative representation and reasoning via asp. In *KR*, 2012.

[BS03] Alex Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. In *Journal on Data Semantics I*, pages 153–184. Springer, 2003.

[BSSH08] Elena Beisswanger, Stefan Schulz, Holger Stenzhorn, and Udo Hahn. Biotop: an upper domain ontology for the life sciences. *Applied Ontology*, 3(4):205–212, 2008.

[Bun12] Mario Bunge. *Treatise on Basic Philosophy: Ethics: The Good and The Right*, volume 8. Springer Science & Business Media, 2012.

[CCC$^+$09] Chang Choi, Miyoung Cho, Junho Choi, Myunggwon Hwang, Jongan Park, and Pankoo Kim. Travel ontology for intelligent recommendation system. In *Modelling & Simulation, 2009. AMS'09. Third Asia International Conference on*, pages 637–642. IEEE, 2009.

[CFF+98] Vinay K Chaudhri, Adam Farquhar, Richard Fikes, Peter D Karp, and James P Rice. Okbc: A programmatic foundation for knowledge base interoperability. In *AAAI/IAAI*, pages 600–607, 1998.

[CGP00] Oscar Corcho and Asunción Gómez-Pérez. A roadmap to ontology specification languages. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 80–96. Springer, 2000.

[CGY07] Nadine Cullot, Raji Ghawi, and Kokou Yétongnon. Db2owl: A tool for automatic database-to-ontology mapping. In *SEBD*, pages 491–494, 2007.

[Cha07] Hans Chalupsky. Loom project home page. `http://www.isi.edu/isd/LOOM/`, July 2007.

[CJB99] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 1999.

[CK94] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[CMMV13] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Ivan Varzinczak. Towards practical defeasible reasoning for description logics. 2013.

[CP13] Paolo Ciccarese and Silvio Peroni. The collections ontology: creating and handling collections in owl 2 dl frameworks. *Semantic Web - Interoperability, Usability, Applicability*, 2013.

[dBH08] Jos de Bruijn and Stijn Heymans. On the relationship between description logic-based and f-logic-based ontologies. *Fundamenta Informaticae*, 82(3):213–236, 2008.

[DCTK11] Kathrin Dentler, Ronald Cornet, Annette Ten Teije, and Nicolette De Keizer. Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web*, 2011.

[EH95] Carola Eschenbach and Wolfgang Heydrich. Classical mereology and restricted domains. *International Journal of Human-Computer Studies*, 43(5):723–740, 1995.

[FB12] Eric D. Feigelson and G. Jogesh Babu. Big data in astronomy. *Significance*, 2012.

[fBIR15] Stanford Center for Biomedical Informatics Research. Protégé. `http://protege.stanford.edu/`, 2015.

[fBO15] The National Center for Biomedical Ontology. Ontology for general medical sciences. `https://bioportal.bioontology.org/ontologies/OGMS/?p=summary`, August 2015.

[FEAC02] Frederico T Fonseca, Max J Egenhofer, Peggy Agouris, and Gilberto Câmara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3):231–257, 2002.

[Fen10] Stefan Fenz. Ontology-based generation of it-security metrics. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1833–1839. ACM, 2010.

[FFR97] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: A tool for collaborative ontology construction. *International journal of human-computer studies*, 46(6):707–727, 1997.

[FHVH$^+$00] Dieter Fensel, Ian Horrocks, Frank Van Harmelen, Stefan Decker, Michael Erdmann, and Michel Klein. Oil in a nutshell. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 1–16. Springer, 2000.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3), 1996.

[FT12] Barbara Furletti and Franco Turini. Knowledge discovery in ontologies. *Intelligent Data Analysis*, 2012.

[GBJR$^+$13] Rafael S Gonçalves, Samantha Bail, Ernesto Jimenez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, Birte Glimm, and Yevgeny Kazakov. Owl reasoner evaluation (ore) workshop 2013 results: Short report. In *ORE*, pages 1–18, 2013.

[Gen14] Gene ontology consortium. `http://geneontology.org/`, 2014.

[GGOP13] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. A non-monotonic description logic for reasoning about typicality. *Artificial Intelligence*, 195:165–202, 2013.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Pearson Education, 1994.

[GHM$^+$12] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:84–101, 2012.

[GHMS10] Birte Glimm, Ian Horrocks, Boris Motik, and Giorgos Stoilos. Optimising ontology classification. In *The Semantic Web–ISWC 2010*, pages 225–240. Springer, 2010.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GSLG12] Vivekanand Gopalkrishnan, David Steier, Harvey Lewis, and James Guszcza. Big data, big business: Bridging the gap. *BigMine '12 Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2012.

[Gua95] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies*, 43(5–6), 1995.

[H+02] Ian Horrocks et al. Daml+oil: A description logic for the semantic web. *IEEE Data Eng. Bull.*, 25(1):4–9, 2002.

[HCC+12] Josh Hanna, Cheng Chen, W Alex Crow, Roger Hall, Jie Liu, Tejaswini Pendurthi, Trent Schmidt, Steven F Jennings, Mathias Brochhausen, and William Hogan. Simplifying mireot: a mireot protégé plugin. In *11th International Semantic Web Conference ISWC 2012*, page 25. Citeseer, 2012.

[HHMW12] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The racerpro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

[HLS+08] Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d'Aquin, and Enrico Motta. The neon ontology engineering toolkit. *WWW*, 2008.

[HLY08] T Huang, W Li, and C Yang. Comparison of ontology reasoners: Racer, pellet, fact++. In *AGU Fall Meeting Abstracts*, volume 1, page 1068, 2008.

[Hoe10] Robert Hoehndorf. What is an upper level ontology? `http://ontogenesis.knowledgeblog.org/740`, 2010.

[Hor05] Ian Horrocks. Owl: A description logic based ontology language. In *Logic Programming*, pages 1–4. Springer, 2005.

[HPsaCAW] Ian Horrocks, Peter F. Patel-schneider, and Deborah L. Mcguinness an Christopher A. Welty. Owl: A description logic based ontology language for the semantic web.

[HQX12] Tianyong Hao, Yingying Qu, and Fang Xia. Domain knowledge acquisition by automatic semantic annotating and pattern mining. *Information Retrieval and Knowledge Management (CAMP)*, 2012.

[HS01] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the shoq (d) description logic. In *IJCAI*, volume 1, pages 199–204, 2001.

[HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic\ mathcal {SHIQ}. In *Automated Deduction-CADE-17*, pages 482–496. Springer, 2000.

[JMY99] Igor Jurisica, John Mylopoulos, and Eric Yu. Using ontologies for knowledge management: An information systems perspective. In *Proceedings of the Annual Meeting-American Society For Information Science*, volume 36, pages 482–496. Information Today; 1998, 1999.

[KCS13] Ridha Khedri, Fei Chiang, and Khair Eddin Sabri. An algebraic approach towards data cleaning. *Procedia Computer Science*, 21:50–59, 2013.

[KFNM04] Holger Knublauch, Ray W Fergerson, Natalya F Noy, and Mark A Musen. The protégé owl plugin: An open development environment for semantic web applications. In *The Semantic Web–ISWC 2004*, pages 229–243. Springer, 2004.

[KK13] Yevgeny Kazakov and Pavel Klinov. Incremental reasoning in owl el without bookkeeping. In *The Semantic Web–ISWC 2013*, pages 232–247. Springer, 2013.

[KL89] Michael Kifer and Georg Lausen. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *ACM SIGMOD Record*, volume 18, pages 134–146. ACM, 1989.

[KPS$^+$06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2):144–153, 2006.

[KSH12] Markus Krotzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *Artificial Intelligence*, 2012.

[LCS12] Renars Liepins, Karlis Cerans, and Arturs Sprogis. Visualizing and editing ontology fragments with owlgred. *I-SEMANTICS (Posters & Demos)*, 932:22–25, 2012.

[LLBN09] Marko Luther, Thorsten Liebig, Sebastian Böhm, and Olaf Noppens. Who the heck is the father of bob? In *The Semantic Web: Research and Applications*, pages 66–80. Springer, 2009.

[LNST08] Jean Lieber, Amedeo Napoli, Laszlor Szathmary, and Yannick Toussaint. *Concept Lattices and Their Applications*. Springer Berlin Heidelberg, 2008.

[LSDS06] Severin Lemaignan, Ali Siadat, Jean-Yves Dantan, and Anatoli Semenenko. Mason: A proposal for an ontology of manufacturing domain. In *Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. DIS 2006. IEEE Workshop on*, pages 195–200. IEEE, 2006.

[MBK06] Christopher J Matheus, Ken Baclawski, and Mieczyslaw M Kokar. Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules. In *Rules and Rule Markup Languages for the Semantic Web, Second International Conference on*, pages 67–74. IEEE, 2006.

[MCWD06] Cynthia Matuszek, John Cabral, Michael J Witbrock, and John DeOliveira. An introduction to the syntax and content of cyc. In *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49. Citeseer, 2006.

[MD13] Travis B. Murdoch and Allan S. Detsky. The inevitable application of big data to health care. *The Journal of the American Medical Association*, 2013.

[MM08] Jakub Moskal and Christopher J Matheus. Detection of suspicious activity using different rule engines—comparison of basevisor, jena and jess rule engines. In *Rule Representation, Interchange and Reasoning on the Web*, pages 73–80. Springer, 2008.

[MMS14] Kody Moodley, Thomas Meyer, and Uli Sattler. Practical defeasible reasoning for description logics (stairs). 2014.

[Mot98] Enrico Motta. An overview of the ocml modelling language. In *the 8th Workshop on Methods and Languages*. Citeseer, 1998.

[NH97] Natalya Fridman Noy and Carole D. Hafner. The state of the art in ontology design: A survey and comparative review. *AI Magazine*, 18(3):53–74, 1997.

[NM01] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. `http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html`, 2001.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001.

[Obi07] Marek Obitko. Description logics. `http://www.obitko.com/tutorials/ontologies-semantic-web/description-logics.html`, 2007.

[oP15] Stanford Encyclopedia of Philosophy. Mereology. `http://plato.stanford.edu/entries/mereology/`, January 2015.

[Orl85] Ewa Orlowska. Semantic analysis of inductive reasoning. *Science Direct*, 1985.

[PGS06] Yury Puzis, Yi Gao, and Geoff Sutcliffe. Automated generation of interesting theorems. *FLAIRS Conference*, 2006.

[RASG07] Yves Raimond, Samer A Abdallah, Mark B Sandler, and Frederick Giasson. The music ontology. In *ISMIR*, pages 417–422. Citeseer, 2007.

[RDB$^+$06] Daniel L Rubin, Olivier Dameron, Yasser Bashir, David Grossman, Parvati Dev, and Mark A Musen. Using ontologies linked with geometric models to reason about penetrating injuries. *Artificial intelligence in medicine*, 37(3):167–176, 2006.

[Run13] B. Runciman. Where are we with big data? *IT Now*, 2013.

[SI02] Xiaomeng Su and Lars Ilebrekke. A comparative study of ontology languages and tools. In *Advanced Information Systems Engineering*, pages 761–765. Springer, 2002.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, volume 432, page 91, 2008.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[Stu03] Greg Stumme. Off to new shores: Conceptual knowledge discovery and processing. *International Journal of Human-Computer Studies*, 59(3):287–325, 2003.

[SWA$^+$94] Guus Schreiber, Bob Wielinga, Hans Akkermans, Walter Van de Velde, and Anjo Anjewierden. Cml: The commonkads conceptual modelling language. In *A future for Knowledge Acquisition*, pages 1–25. Springer, 1994.

[SWR+99] WD Solomon, CJ Wroe, AL Rector, JE Rogers, JL Fistein, and P Johnson. A reference terminology for drugs. In *Proceedings of the AMIA Symposium*, page 152. American Medical Informatics Association, 1999.

[SWS03] Barry Smith, Jennifer Williams, and Schulze-Kremer Steffen. The ontology of the gene ontology. In *AMIA Annual Symposium Proceedings*, volume 2003, page 609. American Medical Informatics Association, 2003.

[TH06] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297. Springer, 2006.

[TSDM04] Yue Tang, Jing Sun, Jin Song Dong, and Brendan Mahony. Reasoning about semantic web in isabelle/hol. *Asia-Pacific Software Engineering Conference*, 2004.

[Uni15] Royal Roads University. Types of reasoning (deductive vs. inductive). `http://library.royalroads.ca/writing-centre/writing/argumentation/building-argument/types-reasoning-deductive-vs-inductive`, 2015.

[Wan89] Yair Wand. A proposal for a formal model of objects. In *Object-oriented concepts, databases, and applications*, pages 537–559. ACM, 1989.

[WD10] Daya C. Wimalasuriya and Dejing Dou. Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Sciences*, 36(3):306–323, 2010.

[Wil05] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*, pages 1–33. Springer, 2005.

[WTLH05] Sung-Shun Weng, Hsine-Jen Tsai, Shang-Chia Liu, and Cheng-Hsin Hsu. Ontology construction for information classification. *Expert Systems with Applications*, 31(1):1–12, 2005.

[WVV$^+$01] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-based integration of information – a survey of existing approaches. *Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*, 2001.

[WW90] Yair Wand and Ron Weber. An ontological model of an information system. *Software Engineering, IEEE Transactions on*, 16(11):1282–1292, 1990.

[WZGP04] X.H. Wang, Da Qing Zhang, Tao Gu, and H.K. Pung. Ontology based context modeling and reasoning using owl. *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22, 2004.