Automatic update of vague ontological concepts

Paolo Pareti

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
August 2011

Abstract

Ontologies can be a powerful tool to structure knowledge and they are a technology which is in the focus of extensive research. Updating the contents of an ontology or improving its interoperability with other ontologies are important but difficult processes [6]. One of the reasons of these difficulties comes from vague concepts [28], which can cause inconsistencies or problems of interoperability with other ontologies. This work adopts a novel perspective on vagueness that does not focus on capturing the degrees of uncertainty of vague concepts (like previous approaches [22]) but instead models them as flexible concepts capable of evolving and adapting to changes. These changes are usually induced by ontological inconsistencies. Concerning inconsistencies, very little work can be found in the literature that proposes different solutions to solve them rather than removing axioms. In particular, it was not possible to find any work that considered numerical restrictions in the definition of ontological concepts as a possible source of inconsistencies. The work that I am here presenting makes use of the first framework to provide an automatic solution to detect inconsistencies caused by cardinality restrictions and data range restrictions for OWL 2 ontologies [29]. The novelty of this approach allows to solve those inconsistencies by modifying internal parameters of axioms without removing any axiom or any part of it. Moreover, the internal parameters of the axioms can be adjusted even when no inconsistency arises. This approach could find applications in the Ontology Change [10] and in the Ontology Alignment [7] fields as it can automatically compute ontological changes that are intended to reduce the misalignment with an external set of ontological data (e.g. another ontology).

Acknowledgements

I would like thank my supervisor, Ewan Klein, to whom I am grateful for the support and collaboration on this project. Special thanks go also to my family and friends, and to Camille, for being a constant and helpful presence throughout this year.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Paolo Pareti*)

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1 Benefits of automatic updates in ontologies

In computer science, an ontology is a structured representation of knowledge. Tom Gruber's entry in [21] provides an extended definition of this concept which includes: "an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members)". After defining the aforementioned primitives, the ontology can be populated by concepts.

A considerable amount of research is recently focusing on areas such as Ontology Change [10] and Ontology Alignment [7]. The main issues addressed by those areas are the following. Concerning Ontology Change, the central problem is how to modify ontologies, possibly automatically or semi-automatically, when there is a need for change. The field of Ontology Alignment instead, focuses on the problem of providing interoperability between different ontologies in an open context. In fact, if a close context is not assumed, there is a concrete risk that different ontologies will be encoded in different languages or that they will use a different vocabulary to refer to the same concepts. Being able to update an ontology when there is the need for it to change, and being able to communicate with other different ontologies are important requirements for many systems. The Semantic Web [24] is an example of an open and dynamic technology where advances (and especially a better automation) in Ontology Change and Ontology Alignment can bring significant benefits.

The framework that I am presenting in this dissertation can be seen as an Ontology Change tool. In fact it can update parts of an ontology automatically maintaining

the consistency of the ontology. The changes resulting from this update will aim to achieve ontological alignment with an additional set of ontological information. If this set is another ontology, the updates will improve the ontological alignment between the ontologies.

The update will only affect the information contained in an ontology and not its signature. For this reason, it is assumed that the additional set of ontological information used to compute the update is encoded using the same ontological language of the ontology to update. More precisely I will focus on OWL ontologies and the additional set of ontological information will consist of a set of OWL axioms.

1.2 The problem of vagueness in ontologies

The interpretation of an ontology is dependent on the knowledge representation language used to describe it and on its logical formalism. In order for the interpretation to be meaningful, it should not entail logical inconsistencies. When inconsistencies occur (e.g. if some axiom contradicts another) then it is usually not possible to be sure of the soundness of any information asserted in the ontology or inferred from it.

Vague concepts are potential sources of inconsistencies in an ontology, because they lack a precise definition. If a concept can be precisely defined, then it cannot be regarded as vague. As an example, let us consider the vague concept of *tall*. Ontology O could consider *tall* only persons that are at least 1.70 meters tall. Now, imagine adding to O the assertion that an individual i, which has a height of 1.68 meters, is *tall*. This assertion will cause O to become inconsistent because i is labeled as *tall* even though according to the definition of *tall,* it should not be labeled as such. To solve this inconsistency automatically (without being examined by a human expert), standard Ontology Repair techniques [3] would propose to delete either some information about this new individual (for example his/her height), or the definition of the concept *tall*. However, if parts of an ontology are removed to solve an inconsistency, some information will be lost. The height of i, the assertion that i is *tall*, or the definition of *tall* are examples of such pieces of information that could be discarded to repair the ontology O.

My thesis is that it is possible to extract information about an inconsistency in an ontology O and use it to update the definitions of the vague concepts in O. These updates are aimed to solve the inconsistency, and they can be performed without removing any part of the ontology (therefore without losing any information). For example, if

the information about the *tall* person of 1.68 meters comes from a trustworthy source, we might want to extend our definition of *tall* to all the persons that are at least 1.68 meters tall. This change would solve the inconsistency without removing any information of the ontology (besides the value 1.70 in the definition of the concept *tall* which will be substituted by the value 1.68). Instead, if we do not have complete trust of the source that stated that a 1.68 meters tall person is *tall*, we can still use this information to realise that our definition of *tall* was too restrictive and we could perhaps extend it to everybody who is at least 1.69 meters tall. In this last case the inconsistency is not solved, and to solve it we will still have to remove (and therefore to lose) some information. The advantage, in this last case, is that the inconsistency was used to adjust automatically the definition of the vague concept *tall*, hopefully improving it.

1.3 Structure of the dissertation

Having discussed the motivation that underlies this research, the following chapter (chapter 2) provides a more detailed description of the problems that will be addressed. It will include the relevant background information and the related work that it was possible to find in the literature. Chapter 3 presents the design of a framework to perform automatic updates of ontological concepts. Important considerations about this framework will then be discussed in chapter 4. The description of an implementation of this framework will follow in chapter 5. The evaluation of this implementation is then presented and discussed in chapter 6. In chapter 7 the results of this research will be summarised along with a discussion about possible future research directions.

Chapter 2

Background

2.1 The ontology used and its semantics

Before providing a definition of what a vague concept is, it is necessary to clarify the use that will be made here of the term "concept". The approach described in this dissertation is focusing on OWL 2 ontologies [29]. Therefore the logical formalism that will be used here is Description Logic [2] as its semantics are used to assign meaning to OWL ontologies. The term "concept" will then be used according to its definition in Description Logic.

In OWL ontologies, concepts are represented as classes. The instances of a concept are called individuals, and they can be expressed as OWL objects. In Description Logic, roles define relations between individuals and data types, or other individuals, and they are represented as OWL Properties. More precisely, relationships between individuals and data types can be represented as OWL Data Properties. Relationships between individuals and other individuals can be represented as OWL Object Properties. The knowledge contained in a particular ontology is encoded as a set of ontological axioms. The term "axiom", in this context, refers to a statement that is assumed to be true by the ontology. Class definitions and property assertions are examples of such axioms.

Given an ontology O, being C the set of concepts defined in O, an interpretation $I = \langle \Delta^I, \cdot^I \rangle$ can be defined by a non empty set Δ^I (called *domain*) and a function $\cdot^I$ which maps every concept $c \in C$ to a subset of the domain $c^I \subseteq \Delta^I$. This subset c^I is called the *extension* of c under interpretation I and it represents the set of individuals that are instances of concept c. An individual i is said to be an instance of concept c (or equivalently a member of class c) under interpretation I if and only if $i \in c^I$.

We can call c^I the *positive extension* of c. Given $\neg c$ as the complement of concept c, we can call $(\neg c)^I \subseteq \Delta^I$ the *negative extension* of c. In order for the interpretation to be logically consistent the positive and negative extensions of a concept have to be disjoint ($c^I \cap (\neg c)^I = \emptyset$).

2.2 A definition of vagueness

Being a necessary characteristic of natural language, vagueness also permeates knowledge representations formalisms, such as ontologies [28]. Typical examples of vagueness are concepts such as *many* or *tall*. However, even well-defined concepts such as the units of measure can be, to some extent, vague.

As described in the previous section, the formal meaning of a concept c can be defined by its positive and negative extensions $\left[c^I, (\neg c)^I\right]$ under an interpretation I. What is the meaning of a vague concept? Two different interpretations of this meaning will now be described:

- PARTIAL INTERPRETATION [1]. Under a *partial interpretation*, it is possible for an individual i not belong either to the positive and the negative extension of a vague concept c ($i \notin c^I \cup (\neg c)^I$), even assuming a complete knowledge about the instance i. For example, an individual i, which is 1.70 meters tall, might not be regarded either as *tall* or as $\neg$*tall*, despite the fact that his/her height is known. Under this interpretation, given an assertion a that states that an individual i is an instance of concept c, then a plausible reasoning processes could give to the following formulas $a \vee \neg a$ and $a \wedge \neg a$ the same undefined truth value. Moreover, the formula $a \vee \neg a$ can not be regarded as a tautology.

- SUPERVALUATION [9]. A *partial interpretation* can be made more precise by extending it to a *supervaluation*. In a *supervaluation*, vague concepts are precisely delineated by sharp cut-off points making them crisp concepts. Given sufficient information, every instance i should be considered a member of the positive extension of a vague concept c or of its negative extension: $\forall i \in \Delta^I.\ i \in c^I \cup (\neg c)^I$. If i is found not to belong either to the positive and the negative extension of a vague concept c, then this is due to lack of information in the ontology. For example, if the height of an individual i is not known, it might not be possible to classify i as a *tall* or $\neg$*tall* person. However, if the height h of an individual i is known, then a possible delineation for the vague concept of *tall* could interpret

individual i as an instance of concept *tall* if $h > X$ and as an instance of concept $\neg$*tall* otherwise. Given an assertion a that states that an individual i is an instance of concept c, the formula $a \vee \neg a$ is a tautology in a *supervaluation*.

In this dissertation vague concepts are defined as concepts that will receive a total interpretation that can be considered as a *supervaluation* dependent on a delineation δ. A vague concept differs from a crisp concept in the fact that there could be multiple admissible delineations for it. The meaning of a vague concept is therefore dependent on the delineation used to define it (as a different delineation might result in a different interpretation of its positive and negative extensions) and this delineation can be subject to changes. The delineation δ of a vague concept used in an ontology O can change into a different delineation δ' as a result of an update of the ontology or as an alignment with an another ontology O'.

But how is it possible to compute a delineation for a vague concept? The framework presented in this dissertation uses the approach of *vagueness as ignorance* [5]. This approach can be interpreted in the following way. If we had to state if a number of individuals are instances of a vague concept, or of its complement (excluding any third possibility), we will end up with a total evaluation of those individuals for this vague concept. At this point it is possible to compute a delineation δ that would result in this same total evaluation (or at least an approximation of it) under a *supervaluation* interpretation.

Using a sharp boundary to define the meaning of a vague concept results in a simple and practical model of vagueness. In Informatics, this could often be desirable. In fact, while the meaning of a vague concept can be subject to revision over time, an application might be interested in having available a precise definition of it. For example, the application might be asked to decide if an individual is an instance of a vague concept or not, without considering a third possibility of undecidability.

The first step in defining a vague concept is choosing how to represent the delineation that should model its vagueness. This delineation will be dependent on a number of parameters. To define the vague concept of *tall*, for example, we could use the height of an individual as one of such parameters. The delineation of a vague concept will then consist of a set of restrictions over those parameters. For example, a person could be considered *tall* if his/her height is above 1.70 meters. If the value "1.70" is changed to another value, a different delineation for the concept *tall* is found. In general, I will use the term "adaptor" to refer to one of such values that define the delineation of a vague concept. Different delineations for a vague concept will be computed by chang-

ing the values of such adaptors. For example, the vague concept of *tall* can be defined as the set of persons taller than X meters. In this example the symbol X represents an adaptor. Changing the value associated with X will change the positive and negative extensions of the vague concept *tall*.

Adaptors will be identified in OWL ontologies using special labels. More specifically, if a value v used in axiom A is labeled with an unique identifier associated with adaptor X, then it is possible to say that X is currently holding the value v and that the axiom A is *dependent* on the adaptor X. When the value of adaptor X will be required to change to a new value z, then the value v in axiom A will be substituted with the new value z. If multiple axioms of an ontology are dependent on adaptor X, then all their internal values associated with X will change accordingly. The cardinality of OWL Properties, and the data range of OWL Data Properties are the parameters that can be restricted using a value associated with an adaptor. Therefore, these are the parameters that can be used in the delineation of a vague concept.

2.3 Ontology Change

It is desirable that an ontology could change over time to adapt to changes in the environment. Assuming that an ontology will never change is often unrealistic. In fact we might want to add or to remove some content of the ontology, we might want to correct mistakes, or to update some of its content which is no longer correct. Changing an ontology is also desirable to update vague concepts because vague concepts, by definition, cannot be formulated precisely once and for all. For this reason the ontology might be required to change to adjust their definition when they are found to be wrong or imprecise.

Work in this area generally falls under the name of Ontology Change or Ontology Evolution [10]. Two of the major issues in this area are the following. The first one is how to automate the evolution process [31]. In fact, manual updates might be too impractical if ontologies are large or if the updates need to be frequent. The second one, tightly connected to the first one, is how to make sure that the evolved ontology is still consistent [19]. The changes that an ontology is required to adopt might be incompatible with other axioms already present in the ontology thus making the ontology inconsistent.

A few solutions have been proposed to avoid the problem of inconsistencies for evolving ontologies but all of them present some disadvantages. A possibility is to

develop strategies to reason with inconsistencies. This process however, is hard to automate and it can be more time consuming than traditional reasoning processes. An example of such approach can be found in [23]. Another solution is to restrict the possible updates only to those that will preserve consistency, as proposed in [12]. This kind of solution, nevertheless, will deeply reduce the possibilities of evolution for an ontology as many kinds of changes will not be allowed.

If inconsistencies can occur, then it might be necessary to adopt a system to restore the consistency of an inconsistent ontology, for example using Ontology Repair tools. One possibility to automate an Ontology Repair system is to remove some of the axioms that cause the inconsistency [11]. However the axioms removed might roll back the new changes that were introduced in the ontology or delete part of the ontology that should have been preserved.

A concrete ontological change of an ontology is usually considered the process of adding or removing parts of it (for example axioms). One possibility to automate the process of Ontology Change can be achieved by adding new axioms to an ontology and then, if the extended ontology is inconsistent, the consistency is restored by removing some of its parts which are responsible for the inconsistency. However if these approaches are used to change an ontology automatically, there is no guarantee on how much ontological information will be lost while removing axioms, and which and how many of the axioms of the ontology will survive the change.

Unlike most of the Ontology Changes approaches proposed in the literature, which rely on removing axioms to solve the inconsistency problem, the system that is presented in this dissertation can modify axioms just by changing their internal parameters. An advantage of such a system is that it can be used to modify an ontology automatically, as many times as it is required, maintaining its consistency and making sure that no axiom (or part of axiom) is deleted in the process.

2.4 Ontology Alignment

Ontologies could be required to interact with other ontologies or with other semantic data in an open environment (for example, in the Semantic Web). In this setting the main unsolved problem that arises is how to make two different ontologies *understand* each other. In fact, in order to interact successfully, two ontologies should share a common vocabulary. If they are not explicitly adhering to the same semantic standards then they could differ at various levels. For example they could be written in two

different ontological languages, or they could represent the same concepts in different ways.

At a syntactic level, the same concept could be named in different ways. For example an individual could be classified as a "Person" in one ontology and as a "Human" in another. To improve the interoperability between ontologies, Ontology Mapping [7] tools aim to find a mapping between concepts such that two concepts will be mapped together if they have the same semantic meaning regardless their name, which could differ.

At a semantic level, two different concepts could be named in the same way by two different ontologies. This is particularly likely for vague concepts. The concept of *Adult* for example, though it might have a very precise definition in the legal system of a country, can be considered vague because in different places it might be interpreted in different ways. In China, for example, the concept of *Adult* could be defined as the set of all persons of age 18 or older. In New Zealand however, this concept could be defined as the set of all persons of age 20 or older.

The framework that I present in this dissertation could be used to detect this last kind of misalignment between two ontologies and, if possible, to reduce it. This type of alignment is called *extension based* [30] as the similarity between two classes will be estimated by comparing the sets of the known entities that belong to them. For example, we can define the concept of *Adult* as a vague concept dependent on the age of a person. In Manchester syntax [14] this definition could look like this axiom template: $Adult = Person\ that\ hasAge\ only\ integer\ [\geq X]$. An axiom template is an axiom with zero or more occurrences of an adaptor. In this case, the symbol X identifies an adaptor. Each axiom template can be instantiated to a ground axiom by substituting each adaptor with a value. For example, instantiating X to 24 would generate the following axiom: $Adult = Person\ that\ hasAge\ only\ integer\ [\geq 24]$. Given a list of Chinese adults and their age as a *training set*, the value X should be updated to the value of 18. If the list of adults comes from New Zealand instead, X might get the value of 20.

2.5 Related work

In the literature it is possible to find a number of papers addressing the problem of vagueness in ontologies. A survey of those approaches can be found in [22]. Those approaches share the common objective of representing the uncertainty of vague concepts inside an ontology considering vagueness as a static property of concepts. Fuzzy

logic and probabilistic techniques are some of the solutions proposed. The work I am presenting here, instead, aims to address the problem of vagueness considering its implications in Ontology Evolution and Ontology Alignment. The difference lies in the fact that my framework is not concerned with representing vagueness inside an ontology, but with considering it while evolving or aligning the ontology. As such, I focus on the dynamic properties of vague concepts whose uncertainty is represented by their capacity to change and to adapt in different contexts.

A related work is presented in [20]. This work proposes a tableaux algorithm capable of identifying which sub-part of an axiom is responsible for an inconsistency in ontologies using the ALC Description Logic. However this approach differs from the one I am presenting here on several points. Firstly, it cannot deal with cardinality restrictions and restrictions on the value of data properties. My approach, on the other hand, is designed to work with them. Secondly, the approach proposed in [20] can only identify parts of axioms that could be removed to restore consistency. The approach that I am presenting here, instead, can suggests a rewriting of the axioms (without removing any of their parts) that can be used to solve an inconsistency or that can be used to update some of the axioms even when no inconsistency arises.

Another related work is described in [26] where a "continuous design-time matching" approach is mentioned. In particular, this paper suggests its importance to deal with ontology matching in P2P environments. However, this approach is dissimilar to the one I am presenting because it is triggered only in the case of a failed interaction and the alignment steps are focused only to solve the failed interaction.

Chapter 3

System architecture

3.1 The inputs taken by the system

The first input for this system is the main ontology we want to update or align. This could be the ontology used by an agent that wants to align it to the other agent's ontologies. Or it could be just the ontology among all those that are used by an application, that needs to be updated. This ontology will be called *original ontology* to distinguish it from the *updated ontology* which will be the output of the system. The *updated ontology* could differ from the *original ontology* only by having different values for the adaptors. No axiom of the ontology will be removed and no one will be added.

The second and last input for this system is a set of axioms, here called *training axioms* that will be used to update the *original ontology*. This set of axioms could be the whole or part of another ontology that the *original ontology* wants to align to. Or it could be just some semantic data used to validate the correctness of the *original ontology* and, possibly, to update it. There is no restriction on the type or on the content of the *training axioms*. However they need to be expressed in the same ontological language as the original ontology (e.g. OWL), otherwise it would not be possible to reason over them and over the *original ontology* at the same time (if this is the case, they need to be translated).

The vague definitions contained in the *original ontology* will be modified if they are found inconsistent with the individuals found in the *training axioms*. For example, the *original ontology* might contain a vague definition that states that adults are only persons over the age of 20. If in the *training axioms* it is asserted that an 18 years old person is an adult, then the vague definition of adult contained in the *original ontology* is found to be inconsistent and it could be modified.

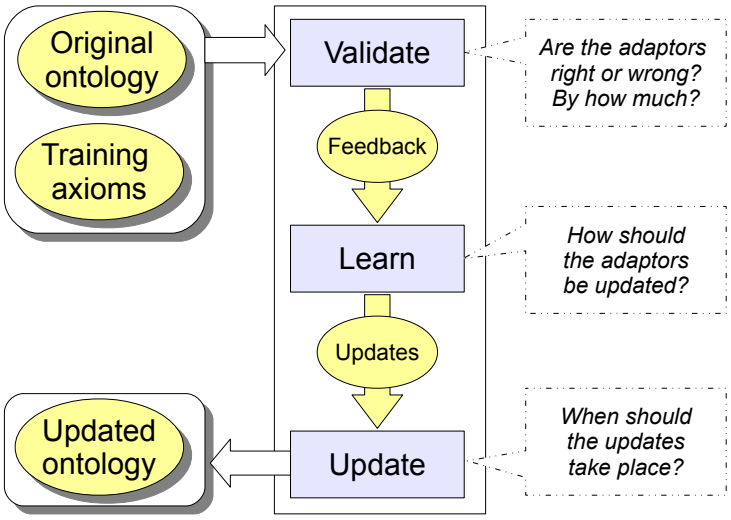

Figure 3.1: Schema of the system architecture

3.2 Overview of the system architecture

A schematic representation of the architecture of the system is illustrated in figure 3.1. Given the original ontology and the training axioms as inputs, the system will output an updated version of the ontology. The whole process of computing this output can be divided into three main phases: the validation phase, the learning phase and the update phase.

The first phase is the validation phase. The purpose of this part of the system is to extract as much feedback as possible regarding the adaptors of the *original ontology* by "validating" them against what is asserted in the *training axioms*. This part of the system has the task of answering the following question:

- Is it possible to state that an adaptor X is *incorrect*? An adaptor X will be found to be *incorrect* if, due to its value, one or more of the *training axioms*, if added to the *original ontology*, would generate an inconsistency. If this is the case, which is the minimal change in its value that, if adopted, would make X no longer generate such inconsistency?

A more detailed description of this phase will follow in section 3.3.

The second phase is the learning phase. Given as input the feedback on the adaptors produced in the validation phase, the aim of the learning phase is to determine how the adaptors should be updated. This could be done in many different ways. For example it is possible to keep some information from the previous iterations of the system (to

use it as a learning history) or to set the level of confidence in the current values of the adaptors. While no learning strategy will perform optimally in all the possible environments some learning strategies might be able to generalise well in different situations. More details about this phase will follow in section 3.4.

The last phase is the update phase. This part of the system uses as input the set of recommended updates to perform in the ontology computed in the previous phase. The purpose of the update phase is to control when those updates will be concretely applied to the *original ontology*, generating the *updated ontology*. For example it might be desirable not to update the ontology too often, or to submit some of these updates to a human expert. A more extensive description of this phase can be found in section 3.5.

3.3 The validation phase

3.3.1 Overview of the validation phase

This is the first phase of the system and the one that will process the "raw" data of the *training axioms* producing a number of *feedback* objects. Those objects will contain a compact representation of all the information that the next phase (the learning phase) will need as input. Let us imagine that in the *original ontology* it is asserted that adults are only persons over the age of X (in Manchester syntax this assertion could be represented by the following axiom: $Adult = Person\ that\ hasAge\ only\ integer\ [\geq X]$) and that X currently holds the value of 18. Now imagine that the *training axioms* consist of the following assertions:

1. Individual John is an Adult and he is 16 years old
2. Individual Jack is an Adult and he is 26 years old

Given this *original ontology* and these *training axioms*, the validation phase should output the *feedback* shown in table 3.1. A more detailed description of how to compute this output will follow in subsections 3.3.2 and 3.3.3.

3.3.2 Generating feedback for adaptors found to be incorrect

In the example previously defined in section 3.3.1, the first assertion of the *training axioms* (individual John is an Adult and he is 16 years old) will clash with the definition

	Feedback 1	Feedback 2
Unique identifier	$adaptorX$	$adaptorX$
Was this adaptor correct?	false	true
Current value of this adaptor	18	18
Required value for this adaptor	16	26

Table 3.1: Examples of feedback for an adaptor X

of Adult (a Person of age 18 or older) in the *original ontology* generating an inconsistency. In fact, John is a 16 year old Adult but the *original ontology* believes that Adults should be at least 18 years old. Is it possible to solve this inconsistency changing the adaptor X? It is easy to see that the answer is yes. In fact, if the value of X would have been different (e.g. if $X = 10$) no inconsistency would have been generated. But 10 is not the only value for X that would have solved the inconsistency. We can notice that no inconsistency would arise for all the values of X inferior or equal to 16. However we are only interested, among all the alternative values of X that would solve the inconsistency, in the one that differs from the current value of X the least, in this case it is 16. In fact there is no reason to change the value of X more than necessary, in order to solve the inconsistency.

But how is it possible to automatically determine, among all the possible values of X, the one that will solve the inconsistency while differing from the current value of X the least? Fortunately, if this value exists, it is already contained in the *training axioms* (or it is its first successor or predecessor, as it will be soon explained). For this reason it is sufficient to consider only a small set of possible values. Given a set of inconsistent axioms and an adaptor X, algorithm 3.1 shows how to extract this set of values.

Given the set of values V computed by algorithm 3.1 for a set of inconsistent axioms S and an adaptor X, if there is a value that, substituted to X, will solve the inconsistency for axioms S, then this value is included in V, or it is the first predecessor or the first successor of a value in V. We could define the value b to be the first successor of a value a if $b > a$ and there is no other value c such that $c > a$ and $c < b$. In the same way we could define b to be the first predecessor of a value a if $b < a$ and there is no other value c such that $c < a$ and $c > b$.

For each value v, it is necessary to consider also its first successor or its first predecessor to deal both with strict and not strict inequalities. In fact, if an inconsistency can be solved by making the following condition true: $v \geq X$ then X could take the

Algorithm 3.1 Algorithm to compute all the possible values to consider for an adaptor X and a set of inconsistent axioms S

```
1  computeAlternativeValues(inconsistent_axioms, adaptor)
2    values = empty set
3    data_relations = the set of relations in inconsistent_axioms
         restricted by the adaptor on the value of their target
4    cardinality_relations = the set of relations in the
         inconsistent_axioms restricted by the adaptor in their
         cardinality
5    all_individuals = the set of individuals in inconsistent_axioms
6    FOR EACH individual IN all_individuals
7      FOR EACH r IN data_relations
8        data_values = set of all the values that individual is related
             to by relation r
9        values ADD ALL data_values
10     ENDFOR
11     FOR EACH r IN cardinality_relations
12       cardinality = number of relations r that the individual has
13       IF cardinality > 0
14         values ADD cardinality
15       ENDIF
16     ENDFOR
17   ENDFOR
18 RETURN values
```

value of v. However, if the inequality is strict, the condition to make true would be: $v > X$ and giving to X the value of v would not make this condition true. In this last case, we are interested in giving to X a value that is smaller than v but, at the same time, it should be as close to v as possible. This value is the first predecessor of v. It is necessary to consider the first successor for the same reason. For example,f X could only take integer values, then the first successor of v is $v+1$ and the first predecessor is $v-1$.

Let us consider again the *original ontology* described in section 3.3.1, containing a definition of the concept of Adult ($Adult = Person\ that\ hasAge\ only\ integer\ [\geq X]$). If we add the *training axioms* to the *original ontology* an inconsistency will arise. It will be shown here an example of how to determine the value to substitute to the variable X, such that it will solve the inconsistency generated from the *training axioms* while differing from the current value of X the least (using algorithm 3.1).

First of all, it might be possible to find a subset of axioms that cause an inconsistency. The Pellet reasoner [27], for example, offers such function to explain inconsistencies in OWL ontologies by extracting the subset of the axioms that is causing the inconsistency. Instead of considering all the axioms of the *original ontology* plus the *training axioms*, it can be desirable (but not necessary) to consider only the subset of axioms that is generating an inconsistency. This subset, in fact, tends to be much smaller than the original set of axioms and therefore reasoning over it can be done more efficiently.

In the example previously proposed, it is possible to notice that the subset of axioms S that generate the inconsistency has those members:

1. the vague definition of Adult ($Adult = Person\ that\ hasAge\ only\ integer\ [\geq X]$) with adaptor $X = 18$,
2. the assertion that John is an Adult ($John\ instanceOf\ Adult$),
3. the assertion that John is 16 years old ($John\ hasAge$ 16)

As X is the only adaptor involved, we could try to run the algorithm 3.1 to determine possible alternative values for the adaptor X. This algorithm will detect that *hasAge* is the only relation restricted by the adaptor X, and thus it will look for instances, in the inconsistent set of axioms S, that are described with such relation. In this case, John is the only instance that is described by the relation *hasAge*, and it's value, for such relation, is 16. For this reason, the algorithm will return only the value 16. This means that, if it is possible to solve the inconsistency in the set of axioms S changing the value of the adaptor X, then the value to change X into either 16 or its first successor or predecessor (if we assume that the age of a person must be an integer, then the first successor would be 17 and the first predecessor would be 15).

At this point, we have only three values to consider in order to solve the inconsistency: $X = 15$, $X = 16$, $X = 17$. We can now just substitute these values to X, starting from the one closer to the current value (in this case 17 is the closest to the current value 18) until we find one that solves the inconsistency or until we have tried all the possibilities and failed. In this example, $X = 16$ will be the first substitution that solves the inconsistency and at this point enough information has been extracted to produce a feedback on the adaptor X. This feedback should state that the adaptor X, currently holding the value of 18, was found to be wrong (as it generated an inconsistency) and it can be corrected by changing its value to 16. This feedback is schematically represented in table 3.1 under column "Feedback 1".

3.3.3 Generating feedback from adaptors found to be correct

During the validation process, it is possible to modify adaptors even when no inconsistencies arise. For example, we could state in the original ontology that adults are only persons over the age of 5. Even if this definition is not causing inconsistencies, we can still notice that all the adult individuals contained in the training axioms are, for example, above 15 years old. In such case, there is the possibility to make the definition of adult more restrictive and to increase the threshold that was found to be too loose.

This process starts in the validation phase by generating feedback about adaptors that are found to be correct. This information is then passed to the learning phase, that could decide to modify an adaptor even if it is not causing inconsistencies. But how is it possible to state that an adaptor X is *correct*? Two possible definitions will here be proposed.

- DEFINITION 1. An adaptor X currently holding the value c will be found to be *correct* if it is not generating an inconsistency, and if it exists a value v, with $v \neq c$ such that changing the value of X to v would generate an inconsistency. If this is the case, which is the maximal change in the value of X that would still maintain the correctness of X?

- DEFINITION 2. An adaptor X will be found to be *correct* if, thanks to its value, it is possible to entail another axiom Y and Y can also be entailed by other axioms which are not dependent on adaptors. If this is the case, which is the maximal change in the value of X that would still maintain the entailment of Y by the axioms dependent on the adaptor X?

Generating feedback from adaptors that are correct is a computationally expensive process. In section 5.3.3 a possible implementation of this process will be discussed. This implementation will make use of DEFINITION 2 because under this definition it is possible to restrict the subset of axioms to consider to only those that contribute to the entailment of Y. Being able to reduce the subset of axioms to work with is an important factor to improve the efficiency of the system.

3.4 The learning phase

3.4.1 Overview of the learning phase

The learning phase is responsible for computing the updates that the *original ontology* should adopt, given the *feedback* objects extracted from the validation of the *training axioms*. It is here called "learning" phase because the feedback objects can be seen as training data and the aim of this phase as to learn the "optimal" value to give to the adaptors, given this training data. However, what counts as "optimal" depends on the preferences of who is using this system. In an Ontology Alignment setting, for example, the updates should try to generate an *updated ontology* which is more likely (compared to the *original ontology*) to interact with the ontology defined by the *training axioms* without generating inconsistencies. In an Ontology Change setting, instead, the *training axioms* could be used just to update the values of the adaptors of the *original ontology*. In this case, resolving the inconsistencies between the *original ontology* and the *training axioms* might not be a priority.

The whole system (and therefore also the learning phase) will execute every time a set of *training axioms* is used to update the *original ontology*. In fact, multiple sets of *training axioms* could be used to train the *original ontology* over time. Each iteration of the learning phase should not be seen as a learning step that adjusts the value of the adaptors getting them closer to an hypothetical "optimal" value. Each iteration of the learning phase should compute the exact value that is believed to be the "optimal" value that each adaptor should adopt, possibly analytically. This guarantees that the *updated ontology* will always contain values for the adaptors that are optimal. At least optimal to the best of its experience based on the previous sets of *training axioms* examined. For this reason, an ontology can be correctly updated using just one set of *training axioms*.

The feedback generated in the validation phase should provide the necessary evidence to justify a change in the adaptors. The following section describes a possible way to use this evidence to compute by how much an adaptor should be updated.

3.4.2 Learning by considering the evidence that supports a change

A possible learning strategy is to adjust the value of an adaptor according to the evidence that is found that suggests its change. If an adaptor was found to be wrong, then it reasonable to assume that there is evidence to support its change. More specifically,

given feedback about an adaptor X, currently holding the value c, that was found to be wrong as it generated an inconsistency solvable by changing its value to v, then there is evidence to support the change of the value of X by an amount equal to $v - c$.

But if an adaptor is found to behave correctly, is there evidence to suggest its change? This depends on how we want to update the adaptors. If our main goal is to prevent inconsistencies, then it is reasonable to claim that an adaptor that behaves correctly does not need any change. If this is the case, then there is also no need, during the validation phase, to compute any feedback about adaptors that are found to be correct (as it would not be used in the learning phase).

However, we might be interested in adjusting an adaptor even when it appears to behave correctly. Let us consider again the example about the *original ontology* containing a vague definition of what an Adult is (as described in section 3.3.1). Assume that the *training axioms* will consist only of a list of individuals, optionally labeled as Adults, along with their age. We can observe that, if an Adult is found to be younger than expected, this would generate an inconsistency (and eventually an evidence suggesting to reduce the threshold between adults and not adults). However, no evidence might be generated to suggest an *increase* in the threshold. In fact, if a person is asserted to be old enough to be considered an Adult, no inconsistency will be generated by this fact. As a result, the adaptor that defines the threshold used to tell if a person is an Adult or not could only decrease. This will create a problem as the concept of Adult will risk, over time, to become too "loose" (meaning that the concept will have more members than it should have), without having the possibility of becoming more restrictive. In this situation, it could be desirable to change the value of an adaptor even when it is found to be correct, as a way to make a vague definition of a concept more restrictive, when it is found to be too loose.

This is also due to the fact that the reasoning process considered so far worked under the Open World Assumption. Under the Closed World Assumption, instead, more inconsistencies could be generated by the *training axioms* and, consequently, more feedback could be generated about adaptors found to be incorrect. For a more detailed discussion about the implications of the Open and Closed World Assumptions see section 4.1.

But what change should be computed when an adaptor is found to behave correctly? Given a feedback about an adaptor X, currently holding the value c, that was found to be correct up to the value of v, then there is evidence to support the change of the value of X by an amount equal to $v - c$. In fact, we might want to make the value

of X the most restrictive as possible, while maintaining its correctness.

After those considerations it can be seen that each feedback object generated in the validation phase can provide evidence supporting a change of an adaptor. But what happens when there are two different pieces of evidence supporting two different changes for the same adaptor? The learning phase should only output up to one update for each adaptor. When different pieces of evidence support a change for the same adaptor than it is necessary to compute an average over them. Suppose that, for the same adaptor X, there are n pieces of evidence supporting changes $[v_1, v_2, ..., v_n]$. The unique update u could then be the mean $\bar{v}$ of those pieces of evidence: $\bar{v} = \frac{1}{n}(\sum_{i=1}^{n} v_i)$.

For example, given three feedback objects about an adaptor X, two of them might provide evidence to increase the value of X by 4, and the third one provides evidence to reduce it by 2. Summing the evidence and dividing it by the number of feedback considered, the update computed would be $((+4)+(+4)+(-2))/3 = +2$. This means that, after considering those three feedback objects, the adaptor X should be increased by 2.

For practical purposes, however, it could be desirable to compute the update in a more sophisticated way. For example, we could give different weights to different types of evidence by introducing a weight function $w(v) = [0,1]$. In fact an evidence generated from an inconsistency (encoded by a feedback object about an adaptor found to be incorrect) could be considered more important than evidence generated from a consistency (encoded by a feedback object about an adaptor found to be correct).

Moreover, if the information contained in the *training axioms* is subject to noise, it could be desirable to reduce the importance of the pieces of evidence that are found to be far from the mean $\bar{v}$. This would reduce the risk that some erroneous information would influence too much the final result. For example, given the following pieces of evidence: $[+6, +9, +2, +4, +5, -984751]$, the piece of evidence -984751 could be reduced in importance as most likely generated by noise. It is possible, for this purpose, to use a sigmoid function ($s : \mathit{Reals} \mapsto [0,1]$) to reduce exponentially the importance of a piece of evidence v the further it is from the mean $\bar{v}$ (reduction factor computed by: $s(|v - \bar{v}|)$). Pieces of evidence very distant from the mean should be scaled by a factor close to 0 ($s(\infty) = 0$) while if they are close to the mean they should be scaled by a factor close to 1 ($s(0) = 1$). This sigmoid function could be automatically scaled by an amount proportional to the standard deviation $s = \sqrt{\frac{\sum_{i=1}^{n}(v_i - \bar{v})^2}{n-1}}$. Including these

additional considerations, the update u can be computed in the following way:

$$u = \frac{1}{n}\left(\sum_{i=1}^{n} v_i\, w(v)\, s(|v - \bar{v}|)\right)$$

3.5 The update phase

This last phase is responsible for producing the *updated ontology* applying the set of updates computed in the previous phase to the *original ontology*. Each update will suggest by how much to modify an adaptor. When the updates take place, the *original ontology* will be searched for all the occurrences of an adaptor and each of them will be updated.

But when to perform those updates? This depends on the preferences of the user of this system. It could be desirable to update the ontology very quickly, every time an update is produced. Or it could be preferable to perform the updates only at specific times, or only after a human expert approved the updates. The updated ontology could then substitute the original ontology or it could become a new version of the *original ontology* which will be kept as a previous version.

This phase is also responsible for making sure that the *updated ontology* remains consistent. A possible way to ensure this is to apply only one update at a time discarding the updates that generate inconsistencies.

Chapter 4

Considerations about the framework

4.1 Open and Closed World Assumptions

4.1.1 Difference between Open and Closed World Assumption

The interpretation of an ontology could be different under different assumptions. For example, two different automatic reasoners might infer different facts from the same ontology, if they make different assumptions about it. Two of those assumptions are the Closed World Assumption (CWA) and the Open World Assumption (OWA) [8].

The OWA is based on the principle that the ontology has only a partial knowledge of its domain of interest. More specifically, given an ontology O and its closure under inference O' (O' containing every assertion that can be inferred from O), if the O' does not contain a fact X or its negation $\neg X$, then this fact X is considered to be unknown. The OWA is desirable not to have to explicitly represent everything that is true or false and to leave ontology potentially open to extension (the addition of new information).

The CWA, on the contrary, is based on the assumption that the ontology has a complete knowledge of its domain of interest and that everything that is true must be entailed by what is asserted in the ontology. This implies that, given O' as the closure under inference of an ontology O, if O' does not contain a fact X, then this fact is considered to be false in O. The CWA could be preferred when it is important to constrain the information contained in the ontology without assuming that it could be extended by additional information.

The work I am presenting in this dissertation follows the OWA. The reason is because OWL ontologies are designed to be deployed in an open environment, such as the Semantic Web, and therefore they are commonly interpreted using the OWA. Pellet

[27] and HermiT [25] are two OWL reasoners that use the OWA and that do not currently support reasoning under CWA. In fact it was not possible to find any reasoner supporting CWA for OWL ontologies.

4.1.2 Consequences of OWA for cardinality restrictions

Ontological instances can be connected to other instances or to literals (data types) by relations. Cardinality restrictions can be used to restrict the maximum or minimum number of such relations that the instances of a class should have. When an adaptor X is used to restrict a cardinality, it might not generate an inconsistency even if it is incorrect. Without generating an inconsistency, it will be hard to detect that X is incorrect, and how to change it.

For example, imagine that all the instances of the vague class *PopularPerson* should have more than X friends, therefore having more than X relations of type *hasFriend*. If an instance I of *PopularPerson* is found to have less than X friends, no inconsistency is generated because, under the OWA, the constraint of having more than X friends is still satisfiable. In fact, in this setting it is not possible to state that I has less than X friends (thus violating the constraint) because under the OWA, it is possible that other *hasFriend* relations for instance I exist, even though they are not explicitly asserted in the ontology. If we want to reason excluding (or ignoring) the possibility that additional relations (not asserted in the ontology) exist, then we might be interested in using the CWA. This could be the case, for example, if we have access to a comprehensive list of persons and their friendship relations, and we are not interested relations not asserted there.

If CWA reasoning is desired, but a suitable reasoner is not found, the axioms of the ontology could be changed to simulate CWA reasoning. A possibility is to assert the cardinalities of the relationships restricted by adaptors as data types, and therefore treated as fixed values (excluding the possibility that more of such relationships exist). For example, the vague concept $PopularPerson = Person\ that\ hasFriend\ min\ X$ (restricted in the cardinality or relationship *hasFriend*) could be expressed as:

$$PopularPerson = Person\ that\ numberOfFriends\ some\ integer\ [\geq X]$$

In this last case, X no longer restricts a cardinality, it now restricts the target value of the (functional) data property *numberOfFriends*. However, in order to perform this conversion, an automatic system should be used to convert the cardinalities found in an ontology into data properties. If the reasoner used can deal with data properties

restrictions more efficiently than with cardinality restrictions, this conversion could also be used to improve the performances of the reasoning process.

4.2 Interconnected vague concepts

An ontological concept can be defined as vague, not only if it interpretation is dependent on an adaptor, but also if it is defined by another concept which is vague. In fact a vague concept C might not contain any adaptors in its definition but, if it is defined by some vague concepts, its meaning will still be (indirectly) dependent on some adaptors. More specifically, the adaptors that *influence* the meaning of a concept C are:

1. the adaptors used in the definition of C
2. for every vague concept C' used to define C, the adaptors that *influence* the meaning of C'

After this consideration, it is possible to provide an alternative definition of a vague concept: a concept can be called vague if there is at least one adaptor that *influence* its meaning.

An implication of this fact is that whenever a concept C is found to be responsible for an inconsistency, all the adaptors that *influence* its meaning could be modified in order to solve the inconsistency.

For example, let's imagine a class A defined by some adaptors. In order to adjust those adaptors, the ontology containing this class is validated against a set of *training axioms*. Let's assume that some individuals in the *training axioms* are classified as members of class A but no individual in the *training axioms* is classified as a non-member of class A. This is a realistic assumption because ontologies usually state when individuals are members of a class but they rarely state explicitly when an individual is not a member. This would imply that all the examples provided in the *training axioms* will be members of class A but no individual will be classified as a non-member of class A. Without negative examples (instances that do not belong to a concept) it becomes more difficult for the adaptors to define a precise threshold between what counts and what does not count as a member of a class. However, if class A is used to define another concept B then it is possible to infer non-members of class A from the members of class B.

For example, according to ontology O a city could be defined as $Suitable$ to open an automotive business if there are less than X $BigCompetitor$ in the same city. A

BigCompetitor could be defined as a business that sells more than Y cars every year. Now imagine to receive the information, from a trustworthy source (e.g. an expert in the field), that city C is *Suitable* for opening the business. However an inconsistency arise because, according to ontology O, city C is not found to be *Suitable* as there are more than X *BigCompetitor* in that city. If X and Y are adaptors, then there are two ways to solve this inconsistency:

1. the maximal number of acceptable *BigCompetitor* in the same city (X) should should increase

2. some of the businesses that were considered *BigCompetitor* should no longer be considered as such (increasing Y)

The second way of solving this inconsistency will generate negative examples for the class *BigCompetitor*. In fact, the adaptor Y should be modified in such way to exclude some of the instances that were previously classified as *BigCompetitor* from this class. As a result, the assertion that an instance is a member of a class might imply the non-membership of other instances in other classes. This is important in order to extract information about both members and outliers of a class.

This example also shows that inconsistencies could be solved in more than one way, when more than one adaptor is involved. If multiple values for an adaptor X are found to solve the inconsistency, only the one that is closer to the current value of X should be considered (as discussed in section 3.3.2). However if multiple adaptors are involved in an inconsistency (as in the example previously described), a feedback can be generated for each of those adaptors that, if modified, would solve the inconsistency.

4.3 Single modification of adaptors

How many adaptors should be changed in order to solve an inconsistency? Without prior assumptions, an inconsistency dependent on n adaptors might require the concurrent modification of all of them, in order to be solved. The possible combinations of changes in the adaptors that should be tried before solving an inconsistency (or before having examined all the possibilities) grow exponentially as the value n grows. In such a situation, using a large number of adaptors in an ontology might not be possible in practice.

In a minimal set of inconsistent axioms, however, if the inconsistency can be solved, it can be solved by modifying just a single adaptor. A set of inconsistent

axioms can be called minimal if no strict subset of it is inconsistent. Algorithms to identify such minimal sets exists, such as the ones proposed in [18].

Each adaptor restricts a relation of the ontology. Whenever this restriction is violated, an inconsistency can be generated. If the minimal set of inconsistent axioms is computed for this inconsistency, it will include only the axiom whose restriction is violated, and a minimal set of of axioms that can prove the violation.

Let's imagine a minimal set of inconsistent axioms S whose inconsistency is generated by the violation of relation r restricted by adaptor a. Since this set is minimal, there is no other relation r' violated by any other adaptor. In fact, if this happened to be the case, the violation of r' could constitute a set of inconsistent axioms by itself (meaning that S was not minimal). Since constraints are expressed as inequalities ($x < a$, $x > a$, $x \leq a$, $x \geq a$) there must exist a value for a that can solve the violation of the constraint.

Chapter 5

Implementation

5.1 The ontology used in the simulation

The simulation is designed to use OWL 2 ontologies serialised in RDF/XML. However OWL ontologies do not support the notion of vagueness and additional information has to be added to the ontology in order to represent adaptors. In order to preserve the functionalities of the OWL ontology used, any additional information required by this system to work can be encoded as axiom annotations, or as attributes in the RDF/XML serialization of the ontology. For this reason this information will be transparent to any reasoner and it will not change the official semantic interpretation of the ontology. This implies that any OWL ontology could be annotated with some adaptors without affecting its normal use.

The annotations that are required by the system to work properly are the following:

- each axiom which contains one or more adaptor should be labeled with an unique identifier
- each adaptor should be identifiable with an unique identifier.

The identifier of an adaptor X should be an attribute of every XML element of the RDF/XML serialization of the vague ontology that directly contains a value that we want to bind to X. When X is updated, then all the values that are bind to X will change accordingly. For example, the ontology described in section 3.3.1 contained a definition of the concept of Adult defined as a person of age 18 or older. This definition can be encoded by the class equivalence axiom: $Adult = Person\ that\ hasAge\ only\ integer\ [\geq 18]$. This definition, however, does not specify that 18 has to be considered as an adaptor.

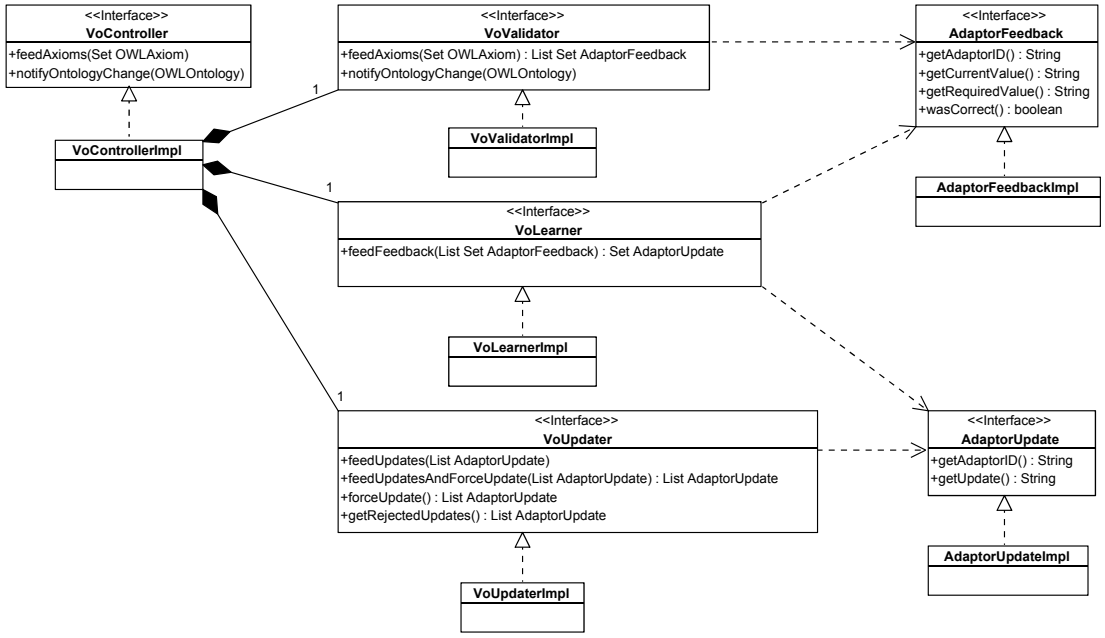

Figure 5.1: UML class diagram of the main classes of the java program

To do so, we have to annotate the value 18, inside the XML/RDF serialization of this axiom, with the identifier of X as an attribute.

In the simulation implemented adaptors can only be numerical values. As such, we can distinguish two different usages for an adaptor X:

1. it can be used to determine the cardinality restriction for object or data properties (e.g. PopularPerson = Person that hasFriend min X Person)

2. it can be used to restrict the data range of numerical data properties (e.g. TallPerson = Person that hasHeight some double [>= X])

5.2 Overview of the program used

The program to run the simulation was implemented using the Java 1.6 programming language. The UML class diagram in figure 5.1 shows the main interfaces used in the program, along with their implementing classes and the main operations that they support. Each of the three main phase of the system (as described in section 3.2) is implemented by the functionalities of the three main classes of the system. More precisely:

- The validation phase is implemented by a VoValidator object

- The learning phase is implemented by a VoLearner object
- The update phase is implemented by a VoUpdater object

The VoController object have the task to coordinate the entire flow of the system. More specifically, it deals automatically with the routine of passing the inputs of each of the three main phases to the following one. Each VoController is meant to operate on a single *original ontology*. An iteration of the validation, learning and update phases is triggered every time a set of *training axioms* is given to it as an input. The VoController will also automatically update its internal fields (including the VoValidator, VoLearner and VoUpdater objects) every time the *original ontology* is changed into an *updated ontology*. After each iteration of the system, the *updated ontology* computed can become the new *original ontology*, ready to be updated again by another iteration of the system.

To work with the OWL 2 ontologies, I used the OWL API [13], version 3.2.3. Adaptors are identified and modified accessing the XML/RDF serialization of the ontology. To manipulate XML data I used the JDOM api [15], version 1.1.1.

5.2.1 The data types used: AdaptorFeedback and AdaptorUpdate

Two different objects are used as data types by the system. The first one is the AdaptorFeedback object and it represents a single feedback given to an adaptor. As such, it is used as the output of the validation phase and as the input of the learning phase. Its internal fields should include at least the following:

- *ID*: the identifier of the adaptor to which this feedback refers to
- *currentValue*: the value that the adaptor was holding during the validation time
- *wasCorrect*: a boolean that is set to false if the value *CurrentValue* was causing an inconsistency, true otherwise
- *requiredValue*: if *wasCorrect* is false, *requiredValue* is the closest value to *currentValue* that is found to be correct. If *wasCorrect* is true, *requiredValue* is the most distant value from *currentValue* that was found to be correct.

The second object used by the system is the AdaptorUpdate object and it represents a suggested update for a single adaptor. It is used as the output of the learning phase and as the input of the update phase. Its internal fields should include at least the following:

- *ID*: the identifier of the adaptor to which this update refers to
- *update*: the change that is suggested to be performed on the adaptor. It is represented as a relative increase or decrease in the value of the adaptor (e.g. +3 or -10)

For simplicity, the simulation I implemented is only dealing with integers values for adaptors. In theory, however, any comparable data type (a data type that supports the "<", "=" and ">" operators) could be used by this system (for example dates). For this reason, in order to generalise to different data types, the values of the adaptors (*currentValue*, *requiredValue* and *update*) are encoded as strings objects. The identifier *ID* is also encoded as a string object.

5.3 The implementation of a VoValidator object

5.3.1 The main functionality: computing the feedback

In the simulation, the interface VoValidator was implemented by the concrete class VoValidatorImpl. This class is initialized with the *original ontology* that needs to be validated. Given a set of *training axioms*, the main functionality that this class provides is to compute a list of sets of parameter feedback. Each of these sets should normally contain just one feedback object. In fact it is reasonable to assume that only one adaptor should be modified in order to solve an inconsistency, as discussed in section 4.3. However this set might contain multiple feedback objects about different adaptors if only the concurrent modification of all of them could solve the inconsistency. Extracting feedback about adaptors that are found to be correct is optional, as in some circumstances it might not be necessary. It is necessary to specify this option while instantiating a VoValidatorImpl object.

Computing this feedback can be done by algorithm 5.1 (this algorithm makes use of algorithms 5.2 and B.1). First of all, the *training axioms* are added to the *original ontology* generating a *combined ontology*. If the *combined ontology* is found to be inconsistent, the explanations about this inconsistency are generated (in algorithm 5.1, this function is called *computeExplanations()*). The Pellet reasoner provides a functionality to compute the explanations for an inconsistent ontology. Each explanation is a minimal set of axioms that generate an inconsistency (the adjective minimal indicates that no strict subset of these axioms can generate an inconsistency). If an in-

Algorithm 5.1 Algorithm to compute feedback from the *training axioms*

```
1  feedAxioms(training_axioms)
2    feedback = empty list
3    combined_ontology = original_ontology + training_axioms
4    IF combined_ontology is inconsistent
5      explanations = computeExplanations(combined_ontology)
6      FOR EACH inconsistent_axioms IN explanations
7        feedback ADD computeFeedback(inconsistent_axioms)
8      ENDFOR
9    ENDIF
10   IF compute_feedback_from_consistency
11     IF combined_ontology is inconsistent
12       removeInconsistentAxioms(combined_ontology)
13     feedback ADD computeFeedbackFromConsistency(
         combined_ontology)
14   ENDIF
15 RETURN feedback
```

consistency has different causes, the explanation will contain multiple sets of axioms (each set representing an explanation). Each of those inconsistent sets of axioms are then analysed to extract feedback from them (in algorithm 5.1, this function is called *computeFeedback()*). A more detailed definition of this function will be discussed in section 5.3.2.

After computing the feedback generated from inconsistencies, the validation phase could be finished and the resulting feedback can be returned. However, if this feature is enabled, there is the possibility to compute feedback about adaptors that were found to be correct. This function will be discussed in section 5.3.3 (in algorithm 5.1, this function is called *computeFeedbackFromConsistency()*). However it is only possible to extract feedback about adaptors that are found to be correct on a consistent ontology. For this reason, if the *combined ontology* is found to be inconsistent, it should be repaired (the inconsistencies should be removed). A simple way to repair an ontology (in algorithm 5.1, this function is called *removeInconsistentAxioms()*) is to progressively remove from the ontology the axioms that are causing an inconsistency until no more inconsistency is found. This solution can work well in practice, but it does not guarantee an optimal repair of the ontology. Ideally the set of axioms removed should be the

minimal set of axioms that, if removed, would restore the consistency of the ontology. However approaches that can guarantee this property (such as the algorithm proposed in [11] to find a "maximal consistent subontology") have an increased computational complexity.

5.3.2 Computing feedback from a set of inconsistent axioms

Once a subset of axioms *S* that generate an inconsistency is identified, it is possible to determine if the inconsistency can be solved modifying some adaptor. This process is described in algorithm 5.2. First of all, the algorithm determines the set of adaptors contained in *S*. If there is none of such adaptors, the algorithm terminates returning an empty set. In fact, only adaptors can be modified by this system in order to solve an inconsistency.

Algorithm 5.2 Algorithm to compute feedback from a set of inconsistent axioms

```
1  computeFeedback(inconsistent_axioms)
2    feedback = empty set
3    adaptor_set = set of adaptors in inconsistent_axioms
4    FOR EACH adaptor IN adaptor_set
5      alternatives = empty set
6      relations_restricted = set of relations restricted by
         adaptor
7      FOR EACH relation IN relations_restricted
8        alternatives ADD values found as targets of the relation
           in inconsistent_axioms
9      ENDFOR
10     FOR EACH value IN alternative
11       substitute(adaptor, value, inconsistent_axioms)
12       IF substitution solves the inconsistency
13         feedback ADD feedback about this substitution
14       ENDIF
15     ENDFOR
16     removeDominatedFeedback(feedback)
17   ENDFOR
18 RETURN feedback
```

After identifying the set of the adaptors involved, it is necessary to determine the

possible alternative values for the adaptors. This set of alternative values is dependent on what types of relations is the adaptor restricting. Only values associated with those relations will be considered as alternatives. For example, if adaptor X is restricting the relation *hasAge*, and among the axioms S it is asserted that an individual *hasAge* 17, then 17 is a possible alternative value for X. In this example, if X is not restricting any other relation, than no other value associated with other relations is considered (e.g. if for some instance the relation $distanceMeasure$ 8 is asserted in S, 8 will not be considered as a possible alternative value for X).

If, instead, an adaptor X is restricting a cardinality of a relation r, then the possible alternative values are computed by calculating the cardinality that instances in S have for relation r. For example, if X restricts the cardinality of the relationship $parentOf$, and an instance in S is asserted to have 3 of such relations, then 3 would be a possible alternative value for X.

When the set of the alternative values for an adaptor is computed, it is possible to try those substitutions (in algorithm 5.2 this function is called $substitute()$). As discussed in section 3.3.2, also the immediate successor and the immediate predecessor of those values might have to be considered. After substituting a new value for an adaptor it is possible to test if this change did solve the inconsistency. If this is the case a new feedback object can be added to the list of the feedback objects to return.

The set of feedback object calculated should not have more than one feedback for each adaptor. If multiple feedback objects are found to refer to the same adaptor, only the one that dominates the other feedback objects will be returned. A feedback object $F1$ dominates another feedback object $F2$ if both refer to same adaptor X, and if $F1$ states that the inconsistency in S can be solved modifying X by an amount $d1$, smaller than the amount $d2$ that $F2$ is suggesting. For example imagine that X currently holds the value of 20, that $F1$ states that the inconsistency can be solved changing X to 22, and that $F2$ states that the inconsistency can be solved changing X to 30. In this case, $F2$ should be discarded as we are interested in solving the inconsistency with the smallest modification of the adaptors as possible.

5.3.3 Computing feedback from a set of consistent axioms

In a set of consistent axioms, no feedback can be generated about incorrect adaptors. However it might be possible to generate feedback about correct adaptors. The definition of a correct adaptor that will here be used is the second definition (DEFINITION

2) provided is section 3.3.3. This definition states that an adaptor X can be defined correct if, thanks to its value, it is possible to entail a fact y which is certain. A fact y is said to be certain if it is entailed in an ontology without considering the vague axioms. This means that y would be considered true in the ontology for every possible values of the adaptors.

The algorithms B.1 and B.2 in appendix B can be used to extract feedback from a consistent ontology. The basic principle behind these algorithms is to identify a set of facts Y that are certain in the ontology. Each fact $y \in Y$, being a certain fact, is entailed by some axioms of the ontology that are not vague. These axioms can be temporary removed to make y no longer certain in the ontology. At these point, if y is still entailed by the ontology, it must be entailed by some vague axioms S. The adaptors that influence the meaning of the vague axioms S can then be changed using alternative values. If one of those adaptors X, once changed, makes the fact y no longer entailed, then it is known that X was responsible for the correct entailment of the certain fact y. At this point there is enough information to create a feedback object about adaptor X.

5.4 The implementation of a VoLearner object

The VoLearnerImpl class is a concrete implementation of the VoLearner interface, responsible for providing the functionalities required in the learning phase. The VoLearner main functionality is to compute a set of AdaptorUpdate objects given a list of AdaptorFeedback objects. The values to update an adaptor with are computed analytically, using the general principles described in section 3.4.2. Each AdaptorFeedback object suggests a change in the value of an adaptor. The update computed will be an average over those changes.

To reduce the risk of computing a wrong update due to noise in the data, changes that are found to be far from the mean will be reduced in importance while computing the average. The standard deviation of those changes will determine how far a change should be from the mean to be significantly reduced in importance. Given x as the distance from the mean and s as the standard deviation of the changes, the reduction is computed by a generalised logistic function in the following form:

$$l(x) = \frac{100}{\left(1 + (s/q)e^{-b(x-2s)}\right)^{q/s}}$$

This logistic function is dependent on parameters q and b by default initialised with values 50 and 0.1 (as those values were found to generalise well between the different

use cases tested). They can be modified if a different value is preferred. This logistic function will output the loss of importance (as a double precision percentage) of a change that has x as the distance from the mean (if $l(x) = 0$ the change will be entirely considered and if $l(x) = 100$ the change will not be considered).

The VoLearnerImpl class also allows to store information about the previous learning iterations, to use it as an additional information while computing the updates. If a memory of size n is permitted, the previous n changes that were examined are stored and they will be considered while computing the updates. A value of n greater than 0 will result in more confidence about the current value of the adaptors and their change will be less flexible. This can be desirable if the *training axioms* come from an unreliable source or if they are subject to noise.

5.5 The implementation of a VoUpdater object

The interface VoUpdater defines the tasks required by the final phase of the system: the update phase. The class used in the simulation for this purpose is the VoUpdaterImpl class. It's main functionalities are two. The first one is to store the updates computed by the previous part of the system, the learning phase, until they are concretely applied to the *original ontology* (how to decide when to concretely apply them is discussed in section 3.5). The second functionality is to perform the updates on the *original ontology* generating the *updated ontology*.

Until the updates are applied to the ontology, it is necessary to store them. It is possible to store all the updates generated from the past iterations of the system, but this is probably not desirable. Imagine that the value of an adaptor X is 10, but the correct value for it is 15. In a first iteration, the update computed for the adaptor X could then be $+5$. In a second iteration, the correct value for X is 16 and this time the update computed might be $+6$. In this situation, it is not desirable to apply both updates, as the resulting value for $X = 10 + 5 + 6 = 21$ is not the correct one. For this reason, one of those updates should be discarded. In general, only the most recent update among all those that refer to the same adaptor should be kept (in this case $+6$). Even though this should be the standard approach, it is possible to disable this option to keep all the updates without discarding any. This can be preferred if, for example, a human expert wants to examine all the different updates computed to for the same adaptor.

The concrete update of the *original ontology* into the *updated ontology* can be

triggered by calling the $forceUpdate()$ method on a VoUpdater object. This update can happen as soon as the updates are received as inputs, if this is desired. Whenever this process of update starts, the required updates are applied one at a time. Each update will modify the value of an adaptor in the *original ontology*. If this modification maintains the consistency of the ontology, then the next update is considered, until there are no more updates to apply. If an update instead generates an inconsistency, it is discarded and the next one is considered.

It is necessary to check the consistency of the ontology after updating one or more adaptors. In fact it is not possible to know if the effect of an update (or the combined effects of multiple updates) will generate an inconsistency. The updates that generate an inconsistency, and that for this reason are discarded, will be kept in case the information they contain could be used in some other way. For example they could be considered again the next time an update is performed on the ontology.

If estimating the consistency of an ontology is an expensive process, and the risk for the updates to generate inconsistencies is low, another approach could be to compute all the updates at once. In this last case, the consistency of the ontology can be checked only once, after all the updates are applied. If the ontology is found to be inconsistent, however, all the updates might be discarded.

Chapter 6

Evaluation

6.1 Evaluation with artificial data

6.1.1 An ontology about persons and their ages

The first evaluation of the system uses an ontology describing persons and simple relations between them. The most important definitions in this ontology are the following:

- *Person*: a general class representing a human being
- $Minor = Person\ and\ (hasAge\ only\ integer\ [< a_1])$: a class representing a young person (defined as a person under the age of a_1)
- $Adult = Person\ and\ (hasAge\ only\ integer\ [\geq a_1])$: a class representing an adult (defined as a person of age a_1 or older)
- $BusyParent = Person\ and\ (parentOf\ min\ a_2\ Minor)$: a class representing the vague concept of a busy parent (defined as a person with at least a_2 young children)
- $RelaxedParent = Person\ and\ (parentOf\ some\ Person)\ and\ not\ BusyParent$: a class representing the vague concept of a relaxed parent (defined as a parent that is not busy)
- *hasAge*: a functional data property with domain *Person* and range integer values
- $parentOf$: an object relation between two *Person*: a parent and a child

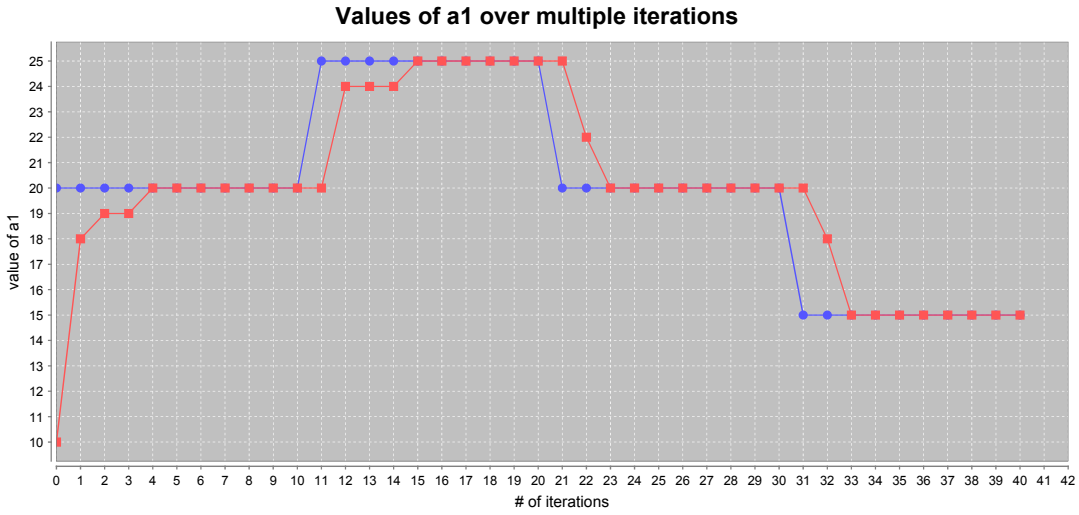

Figure 6.1: Plot of the values of adaptor a_1 across 40 iterations of the system (no noise). The red squares indicate the value computed by the system, the blue circles indicate the correct value for that iteration.

The *training axioms* are produced automatically generating instances of the above mentioned classes, and the relations between them. Since the data is produced automatically, it is possible to know the exact value that the adaptors should have, as it is the value used while producing the data. For example, data can be produced generating n persons and their age (in the simulation the age is an integer between 0 and 80) and then each of those persons is labeled as an *Adult* if his/her age is greater or equal than a_1, and it is labeled as a *Minor* otherwise.

Additionally, some noise can be added to those axioms. For example, after labeling a person as an *Adult* or as a *Minor*, his or her age can be modified by a random value.

6.1.2 Learning the threshold between adults and minors

This first scenario shows the behaviour of the system while it tries to adjust the adaptor a_1 that defines the threshold between being *Minor* and being *Adult* as defined in the previous subsection. The simulation will run for a number of iteration and, at each iteration, a new set of *training axioms* will be randomly generated. Each of those sets of axioms describes 30 *Persons* along with their age (randomly generated). Those individuals are also labeled as *Adult* or as *Minor* according to their age and the correct value of $a_1$in that iteration. The adaptor a_1 is initially set to the value of 10. The validation system used in this simulation is only extracting feedback for adaptors that generate inconsistencies. No feedback is generated for adaptors found to be correct.

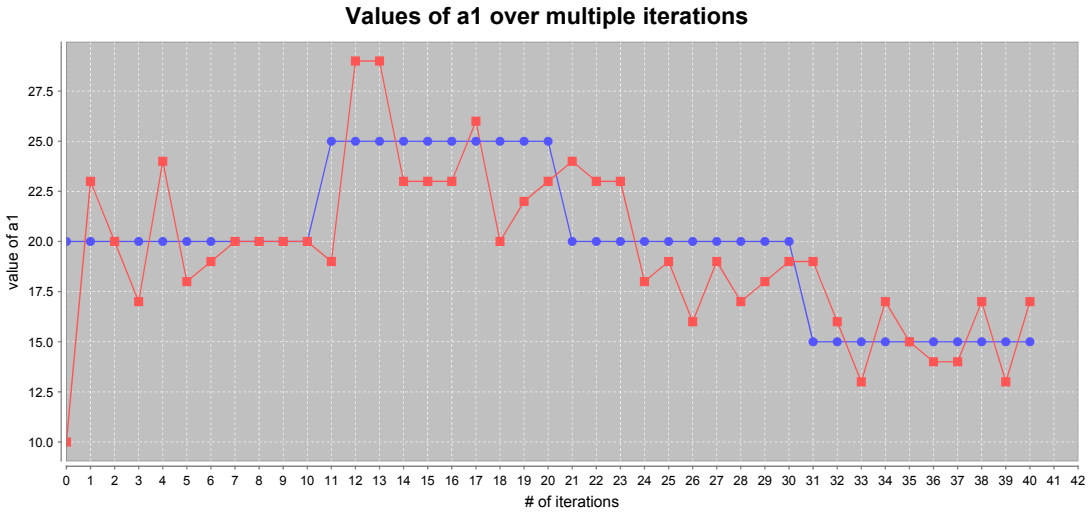

Figure 6.2: Plot of the values of adaptor a_1 across 40 iterations of the system (with noise). The red squares indicate the value computed by the system, the blue circles indicate the correct value for that iteration.

The red squares in figure 6.1 show the value of adaptor a_1 across 40 iterations of the system. The blue circles indicate the correct value that a_1 should have in each iteration, this value changes every 10 iterations to test if the system can adapt to changes in the environment. At each iteration, a set of *training axioms* is generated and used to update the *original ontology* (and therefore the value of a_1). The *updated ontology* will then become the *original ontology* in the following iteration. Figure 6.1 shows that under these conditions the adaptor a_1 can quickly converge to the correct value.

In some circumstances it might be necessary to consider the possibility that the *training axioms* are subject to noise. To simulate noise in the data, the age of each person in the *training axioms* is randomly modified by a value in the range $[-5, +5]$. Figure 6.2 shows the updates of adaptor a_1 under the same conditions as in the previous simulation, but with the addition of noise. It is possible to notice that the value of a_1 is still approaching its correct value but it cannot reach a stable convergence.

To reduce the influence of the noise on the convergence of the adaptors, it is possible to keep some learning history to use in the learning phase. Figure 6.3 shows the value of a_1 across 40 iterations of the system under the same conditions as before, but using memory in the learning phase. More precisely, the VoLearnerImpl object used is now allowed to keep in memory the two latest feedback examined, and to use them while computing the updates. Comparing figure 6.3 with figure 6.2 it is possible to notice that using some learning history makes the system having some confidence on the

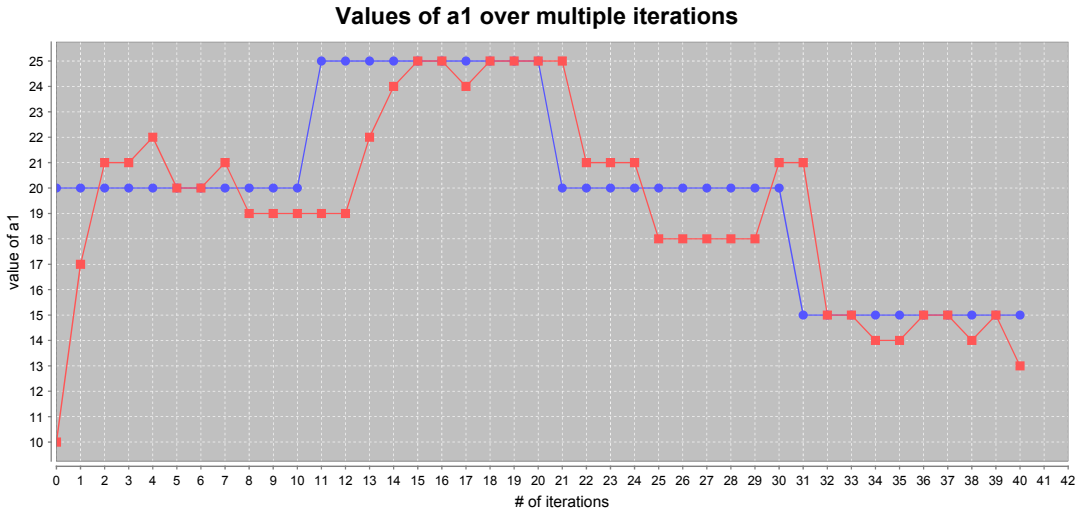

Figure 6.3: Plot of the values of adaptor a_1 across 40 iterations of the system (with noise and using memory in the learning phase). The red squares indicate the value computed by the system, the blue circles indicate the correct value for that iteration.

current value of the adaptors. This confidence makes it harder for the adaptors to adopt small changes and consequently the effect of the noise on the updates of the adaptors is reduced.

6.1.3 Learning the correct value for a cardinality restriction

The simulation described in the previous section was focused on updating an adaptor restricting the data property *hasAge*. The simulation presented in this section will show how an adaptor that restricts cardinalities is updated. Considering again the ontology described in section 6.1.1, the concept of *BusyParent* is found to be dependent on the adaptor a_2 ($BusyParent = Person\ and\ (parentOf\ min\ a_2\ Minor)$). The concept of $RelaxedParent$ is dependent on a_2 too. In fact, even though the definition of $RelaxedParent$ does not make use of a_2, it make use of the concept of *BusyParent* and therefore it is indirectly dependent on the adaptor a_2. In the ontology used, a_2 is initialised with the value 1.

This simulation consists of 40 iterations of the system. For each of them a set of *training axioms* is randomly generated according to the the correct value that a_2 should have in that iteration. Each set of *training axioms* contains the information about 10 instances of the *BusyParent* class, along with their children in number equal or greater than a_2. It also contains 10 instances of the class $RelaxedParent$, along with their children (less than a_2 of them being $Minor$).

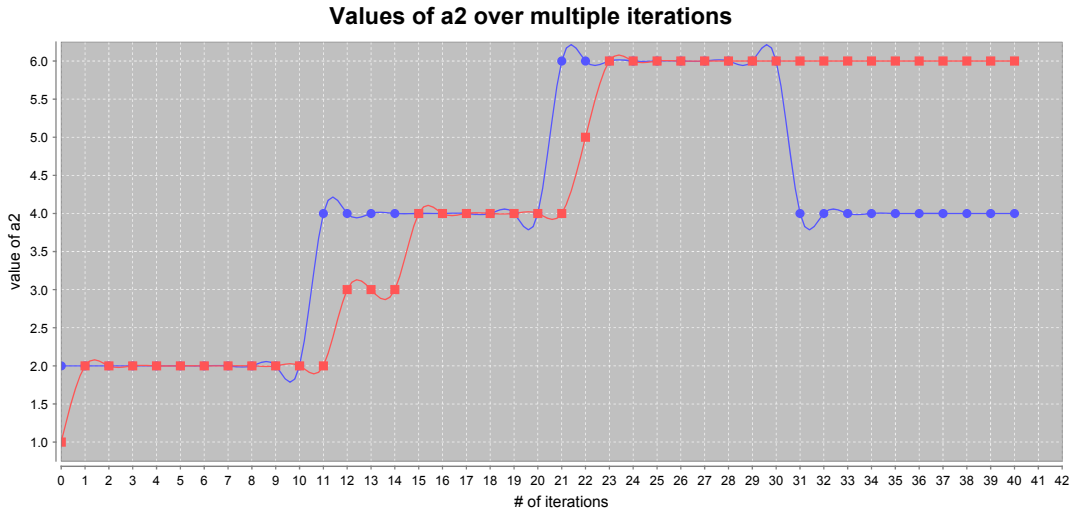

Figure 6.4: Plot of the values of adaptor a_2 across 40 iterations of the system. The red squares indicate the value computed by the system, the blue circles indicate the correct value for that iteration.

Figure 6.4 shows a run of this simulation. The red squares indicate the value of a_2 in each iteration. The blue circles indicate the correct value that a_2 was supposed to have in that iteration. This result shows that an adaptor restricting a cardinality (in this case a_2) can converge on its correct value, as long as the convergence implies an increase in its value. If the convergence instead implies a reduction in the value of the adaptor (as in the last 10 iterations of this simulation), the adaptor will not change. For this reason, a_2 is not able to adapt to the last change in the simulation, namely reducing its value from 6 to 4.

The inability to reduce the value of a_2, which might seem undesirable from a practical point of view, is a logical consequence of the Open World Assumption made by the reasoner used (as discussed in section 4.1). Concerning cardinalities, in fact, the only situation that could generate an inconsistency is when the maximum cardinality of a relationship is violated. For example, if an instance I, belonging to class $RelaxedParent$, is found to have 3 children that are $Minor$ and the concept of $BusyParent$ is defined as all the persons with more than 2 children, then I will be classified also as a $BusyParent$. Since the classes $BusyParent$ and $RelaxedParent$ are disjoint, an inconsistency will arise. This inconsistency could then be solved increasing the value of a_2. However, no inconsistency will arise in this situation if a_2 is found to be greater than its correct value. To decrease its value, in fact, it would be necessary to prove that a certain cardinality does necessary have to be inferior than a_2. However, under the Open World

Assumption, the cardinalities for each kind of relationship are potentially infinite (unless otherwise stated).

A possible way to overcome this problem could be to use adaptors such as a_2 in the definition of more than one concept. If an adaptor is used to define multiple concepts, in fact, it is more likely that it will generate an inconsistency if it is found to be incorrect.

The adaptor a_2 is not the only one that the classes of *BusyParent* and *RelaxedParent* depend on. In fact, the definition of the class *BusyParent* uses the definition of the class *Minor*, which is dependent on the adaptor a_1. For this reason, some of the solutions that the system suggested (to solve the inconsistencies generated in this simulation) involved a_1. More precisely, if the value of a_1 happened to be reduced, some individuals that were at first classified as *Minor* would be then be classified as *Adult*. As a consequence, an instance of class *RelaxedParent* that had too many *Minor* children, after a decrease in the value of a_1, might result having less children that are *Minor* (or even none at all). Even though reductions to the adaptor a_1 were suggested by the system, the value of a_1 remained close to its correct value. In fact, as soon as a_1 diverges from its correct value, instances of the *Minor* or *Adult* classes are no longer correctly classified and inconsistencies are generated. Those inconsistencies are then detected and the value of a_1 is pushed again to converge toward its correct value.

6.2 Evaluation with web data

6.2.1 An ontology about places and distances

This second evaluation deals with an hypothetical scenario where the user of this system is a company that owns libraries in cities around the world. This company is interested in developing an automatic system to discover cities where it can be profitable to open a new library. More specifically, given a city X , the system could look for available semantic information about that city (for example from sources such as DBpedia or Google) and use it to determine if X can be considered a profitable place to open a new library in. This system could be used to select, out of a large number of cities, a small subset of them that could be potentially profitable for business. This subset could then be analysed by human experts to decide if opening or not a new library in one of those cities.

In a real scenario, many factors might be used to make such choice. In this sim-

ulation however, for simplicity, only one factor will be considered. A city will be classified as profitable to open a business if there are only few competitors libraries near the city centre. The concept of near could vary significantly in different contexts. For example, it usually depends on the mean of transport that can be used to cover a distance (on foot, 5 kilometers might seem a long distance but by car they might seem a short one). In the centre of a city, potential customers of books are assumed to reach a library on foot. Therefore in this context the concept of "near" should be interpreted as a walking distance. However, even after this clarification, two vague concepts remain undefined. How many libraries should there be in a city centre to be able to say that they are "too many"? And what counts as near? How many meters away is a "distant" place?

These vague concepts are defined in an OWL ontology using two adaptors. The first one is c, it determines the maximum number of nearby libraries that a place considered profitable can have. The second one is d, it determines the maximum number of meters that two nearby places can have in between. This ontology uses a ternary relation to define distances. In logic, this relation could be expressed as: $distance(x,y,z)$ meaning that there are z meters between place x and place y. OWL ontologies, however, support only binary relations. For this reason the concept of distance cannot be represented as a relation but it must be represented as a class having, as properties, two places and a measure of distance.

The most important definitions contained in the *original ontology* used in this simulation are the following:

- $SpatialThing$: a general class that represents a real place
- $Distance = (distBetween\ exactly\ 2\ SpatialThing)\ and\ (distMeasure\ exactly\ 1\ integer)$: this definition states that an instance of class *Distance* should be related to two $SpatialThing$ (the places between which the distance is computed) and to one integer (the meters between the two places).
- $CloseDistance = Distance\ and\ (distMeasure\ only\ integer\ [\leq d])$: if there is a $CloseDistance$ between two places, then they can be considered near. This definition states that a $CloseDistance$ is a *Distance* that measures no more than d meters.
- The definition of $ProfitablePlace$ can be easily expressed in natural language as follows: a place can be considered profitable if there are no more than c libraries

nearby. Its definition, expressed in Manchester syntax, is displayed below. This definition of $ProfitablePlace$ might appear to be more complex than expected. This is due to the fact that the concept of "near" cannot be expressed as a relation but it has to be represented as a class.

$$ProfitablePlace = SpatialThing\ and$$
$$(hasDistance\ max\ c\ (CloseDistance\ and\ (distBetween\ some\ library)))$$

- $library$: the instances of this class represent real libraries.

Having defined the *original ontology* used, the next section will describe how the *training axioms* are extracted from real web data.

6.2.2 The training axioms generated using web data

In order to learn the proper values to give to the adaptors c and d, a series of *training axioms* is fed to the system. These axioms contain information about places and about the distances between them. Moreover, some places might be classified as $ProfitablePlace$ or as $library$ and some distances might be classified as $CloseDistance$. We could imagine that for some cities, it is already known that they are profitable. For example, a human expert already classified some of them, or perhaps they are cities where a profitable business is already present. The city centre of those cities can then be classified as a $ProfitablePlace$ and then additional information about that city can be extracted using web data. This information is then converted into OWL axioms and treated as a set of *training axioms*. In this simulation, the city centres of the 30 largest cities of the United Kingdom are assumed to be profitable places for the opening a new library. This is an arbitrary assumption adopted just for the purpose of generating plausible *training axioms*.

For each of those cities X a set of *training axioms* is generated. The coordinates used to determine the position of the city centre of city X will be extracted from DBpedia [4]. This city centre will then be classified as a $ProfitablePlace$. The service offered by Google Places API [17] will then be used to determine the locations of libraries within 2 kilometers from the city centre. The distance between each of those libraries and the city centre is calculated using the service offered by the Google Distance Matrix API [16].

For each library l that were found to be near the city centre, the Google Places API provides the name of a location p "near" l. In this simulation I adopted the simplified

assumption that if a place is considered "near" by the Google Places API, then it should be considered near also by the *original ontology* that this simulation is using. For this reason, the distance (in meters) between p and l is computed using the Google Distance Matrix API and then it is classified as as $CloseDistance$.

For each city, an amount of information is extracted from web data as it was just discussed. This information is then converted into OWL axioms to form the set of *training axioms* that the system will use to adjust the values of the adaptors c and d. The next section presents the results of the simulation performed using these *training axioms*.

6.2.3 The results of the simulation

This simulation consists of 30 iterations of the system. In each iteration, a city X is considered and set of *training axioms* is generated as described in the previous section. This set of axioms is then used to update the *original ontology* into an *updated ontology*. The *updated ontology* generated is then used as the *original ontology* in the following iteration of the system. The names of the 30 cities used in this simulation can be found in appendix A.

This simulation used a VoValidator object that computes only feedback about adaptors found to generate inconsistencies. The VoLearner object used is allowed to have memory about previous iterations of the system and it can store in memory up to 15 previous feedback objects. The values that adaptors d and c held in each iteration of the system can be seen in figures 6.6 and 6.5. Other statistics about these adaptors can be found in table 6.1.

Figure 6.5 shows the evolution of adaptor d, across the 30 iterations of the system. To test if an adaptor can quickly adjust its value, d is initialised with the incorrect value of 0. What could be the correct value for d will be discussed later in this section. In the meanwhile it is reasonable to assume that 0 is incorrect. In fact, if the value of d is 0, every distance will be considered a long distance and no place will be considered near.

After one iteration d increased to the value 383 and after two to the value 1838. In this example the adaptor d managed, in one iteration, to reach a more plausible value to determine what counts as far for a person walking on foot (expressed in meters). From the second iteration to the last one, d held values ranging from 1360 to 2517.

The information obtained from the web is subject to noise and missing information.

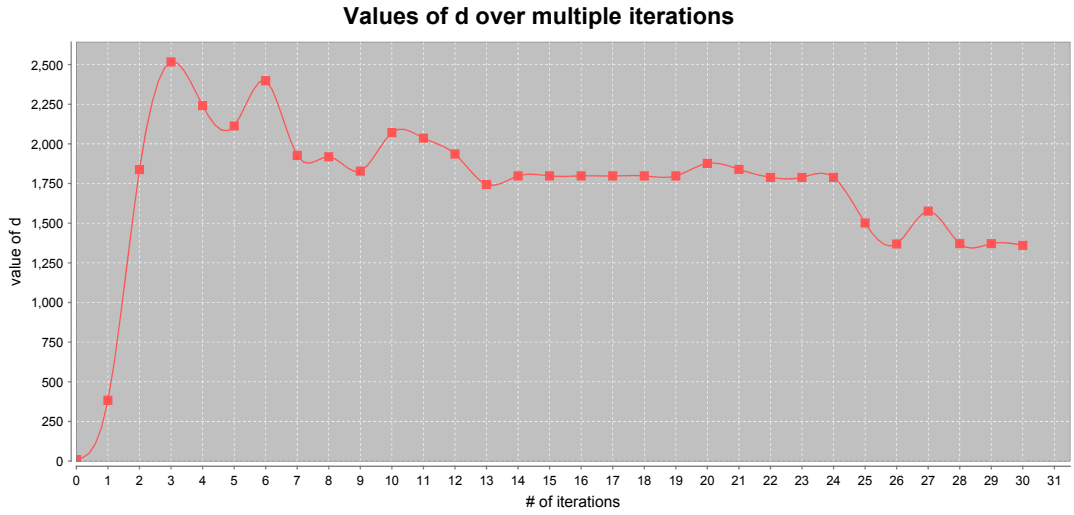

Figure 6.5: Plot of the values of adaptor d across 30 iterations of the system

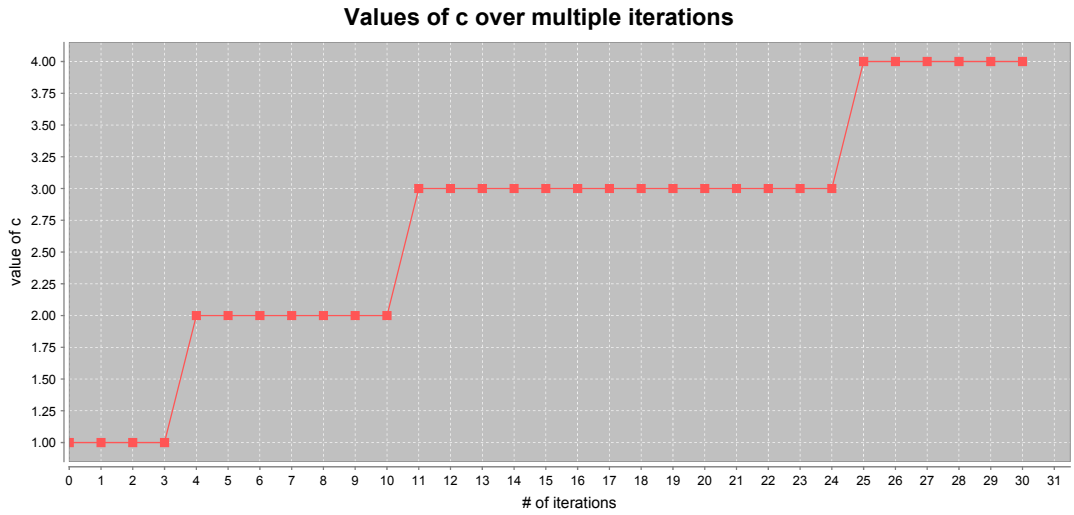

Figure 6.6: Plot of the values of adaptor c across 30 iterations of the system

For example, several distances measuring more than 10 kilometers were classified as *CloseDistance*. The sigmoid function used in the learning phase (as described in section 5.4) reduced the effect that values greatly differing from the mean have on the value of the adaptors. Without this reduction, the value of d would have been less stable and it could have suddenly changed its value of several thousands units in just one iteration.

It is possible to notice that, compared to the first 10 iterations, the adaptor d started to become more stable in the last 20 iterations. This is a resulting effect of the memory accumulated from the previous iterations. This memory increases the confidence that d had on its current value and it reduces its flexibility to change.

Adaptors:	d	c
Starting value	0	1
Ending value	1360	4
Min value computed	383	1
Max value computed	2517	4
Average value	1778.7	2.8
Estimated correct value	1325	2.49 or 3.54
Difference from the correct value (% of the standard deviation)	4%	162% or 39%

Table 6.1: Statistics about the evolution of adaptors d and c

Figure 6.6 shows the evolution of adaptor c (concurrent to the evolution of d shown in figure 6.5) during the 30 iterations of the system. Changes in this parameter occurred when a city centre, classified as $ProfitablePlace$ was found to have too many (more than c) libraries nearby thus violating the definition of class $ProfitablePlace$. Increasing the value of c is a possible way to solve this inconsistency. Another possibility would be to reduce the value of d as this would result in less libraries to be considered "near".

As already observed in section 6.1.3, an adaptor that is only used to restrict a cardinality (in this case d) might not be able to reduce its value. In figure 6.6, in fact, it can be seen that the adaptor d is only increasing. This problem can be partially solved by initializing to a low value an adaptor that is only restricting cardinalities, and letting it increase through the iterations. In this simulation, d was initialised with the value 1.

6.2.4 The correct values for the adaptors

This system ended its series of iterations reaching the conclusion that if a place in the centre of a city has less than 4 libraries within 1.36 kilometers, then it is probably a profitable place where to open a new library. In order to estimate if these values are correct, the average number of libraries near a city centre and the average distance between near places should be computed considering all the *training axioms* involved in the simulation.

The average measure of the $CloseDistance$ instances considered is 504 meters, and the standard deviation is 821 (these values are computed reducing the importance of samples too far away from the mean as described in section 5.4). Adaptor d aims to

capture the concept of how many meters away is a far place. For this reason we could consider as its "correct" value, the average measure of a *CloseDistance* instance plus the standard deviation observed. Distances that are found to exceed this measure could be intuitively considered "far distances". After these consideration, we could state that in this simulation the correct value for adaptor d is $504 + 821 = 1325$. The value actually computed by the system for adaptor d, after completing all the iterations of the system, is 1360. We can consider, as a measure of error, the percentage of the standard deviation that equals the difference between the correct value and the value actually computed. In this simulation, the computed value differs from the correct value by 4% of the standard deviation. The adaptor d then managed to approximate the correct value with a 4% error.

It is possible to identify a correct value for adaptor c in a similar way. We could claim that the correct value for c is the average number of libraries near each city centre, plus its standard deviation. Before computing this value, it is necessary to state what counts as "near". Using the "correct" definition of near discussed before, we could consider a place near if it is less than 1325 meters away. After this clarification, the average number of libraries near the city centre (for the cities examined) is 1.56, with standard deviation 0.93 (as before, values far from the mean are reduced in importance by a sigmoid function). The correct value for c can then be said to be 2.49. The value that was computed for c after all the iterations of the system is 4, which differs from the correct value by 162% of the standard deviation.

One of the main reasons to explain this large error is that it is computed assuming the "correct" definition of near (a place more than 1325 meters away). However, during the iterations of the system, the concept of near was different. In fact the correct value for adaptor d was only approximated in the end. The average value of d across the 30 iterations of the system is instead 1778.

We could now consider an alternative interpretation of the correct value for adaptor c, no longer based on the correct definition of near (with $d = 1325$) but using the definition that in practice (in average) was used during the simulation (with $d = 1778$). In this case the average number of libraries near the centre of a city is 2.37, with standard deviation 1.17. The correct value for d in this case is 3.54. The value computed for d at the end of the system (4) differs from the correct value by 39% of the standard deviation.

6.2.5 Extracting feedback about correct adaptors

The simulations discussed so far computed and used feedback only about incorrect adaptors involved in inconsistencies. In appendix C it is possible to find the results of a simulation that extracted also feedback from adaptors found to be correct. The most significant difference in the results in this variant of the simulation is that the average value of both adaptors across the iterations is significantly reduced. In this variant, in fact, both average values are below the correct ones. In the simulation described before (that did not extract feedback about correct adaptors) instead, the average values for both parameters were above their correct values. The reason for this difference is that feedback about a correct adaptor will be used, during the learning phase, to make the relations it restricts "more restrictive". In this situation, since both adaptors were used to restrict the maximum value of a relation, this maximum value was reduced (making it more restrictive).

The effects of extracting feedback about correct adaptors did not result in a significant improvement in the precision of the system. This precision is estimated by calculating the difference between the correct value and one actually computed for each adaptor (these values can be found in table C.1 for the new variant and in table 6.1 for the old one). For this reason, this additional process of extracting feedback about correct adaptors appears to be unnecessary and it can be avoided in order to save computational resources. This process could be used, however, if it is desirable to update the values of the adaptors even if few or no inconsistencies are generated.

6.3 Note on the computational complexity

This system uses automatic OWL reasoners (such as Pellet and HermiT) and therefore its computational complexity is dependent on the size (the number of axioms) of the *original ontology* and of the *training axioms* used. The specific implementation of those reasoners, and the amount of axioms they have to reason with, are the most important factors that determine this complexity.

Generating the explanations for an inconsistent ontology is the operation that consumes most of the resources. This process identifies sets of axioms, subsets of the ontology used, that are responsible for an inconsistency. Only after an explanation is generated the system will check if it contains vague concepts. If this is the case, the system will attempt to extract feedback from the inconsistency. An explanation

will typically contain only a small subset of axioms and thereby only a small set of adaptors. Therefore, extracting feedback from an explanation is a significantly faster process compared to the process of generating the explanation in the first place. For this reason, the number of adaptors (and consequently the number of vague concepts used in an ontology) are not important factors to determine the computational complexity of the system. If desirable, all the cardinality and data restrictions used in an ontology could make use of adaptors.

To improve the performances of the system, efficient OWL reasoners should be used. Additionally, the size of the *original ontology* and/or the size of the *training axioms* can be reduced (as this will reduce the time spent by the reasoning process). A possibility to reduce the size of a large set of *training axioms* is to divide it into smaller subsets. In the simulation presented in section 6.2, for example, the *training axioms* computed for each city were kept separate from each other and they were used one at a time. To reduce the size of the *original ontology*, it is possible to remove some if its parts that are known to be unrelated with the axioms that are dependent on adaptors.

Reducing the size of the axioms considered by the system, however, can reduce the amount of information that it is possible to infer from them. In this situation less inconsistencies might be generated and consequently less feedback will be produced during the validation phase. For this reason, the size of the *original ontology* and of the *training axioms* should be reduced only if necessary.

Chapter 7

Conclusion

7.1 Concluding remarks

Ontologies are a formal representation of knowledge. But knowledge can often be imprecise and vague. Nevertheless, most of the current ontological languages, such as OWL, do not provide solutions to deal with ontological vagueness. In a static context, it might be possible to ignore this issue. In a dynamic environment, however, it is inevitable to deal with the effects of vagueness. The problem of vague ontological concepts in fact, is identified as central both in the fields of Ontology Evolution and Ontology Alignment.

This work presented a novel approach to deal with vagueness: a vague concept is defined as a concept that receives a total interpretation which, however, can be subject to change. More precisely, the meaning of a vague concept is dependent on a number of values, called adaptors, which can be automatically updated. These adaptors can be used to define cardinality restrictions and data range restrictions for OWL Properties.

The definitions of the vague concepts of an ontology (here called *original ontology*) can be automatically updated by validating the *original ontology* against a set of *training axioms*, generating an *updated ontology*. Inconsistencies generated by the union of the *original ontology* and the *training axioms* are interpreted as a misalignment between those two sets of ontological information. This misalignment can be reduced solving these inconsistencies by modifying the values of the adaptors used in the *original ontology*. It is also possible to update the definitions of the vague concepts (and consequently the values of the adaptors they depend on) even when no inconsistencies arise. If the axioms of another ontology are used as *training axioms* for the *original ontology*, then the update will result in an improved alignment between the

two ontologies.

In order to compute this update, a framework was proposed. This framework defines the three main phases that are required to update the vague concepts of an ontology. The first phase is responsible for computing the feedback about the adaptors. In particular, this phase should detect if inconsistencies between the *original ontology* and the *training axioms* are caused by some of the adaptors, and if it is possible to solve them by changing the values of those adaptors. The second phase is responsible for using the feedback computed in the previous phase to determine suitable updates for the adaptors. The final phase consists in applying those updates to the *original ontology* generating an *updated ontology*, which contains updated definitions of the vague concepts.

A possible implementation of this framework was then described and evaluated using two different simulations. The first simulation used artificially generated *training axioms* while the second one extracted them from web sources. The results of these simulations show that if the *training axioms* contain sufficient information to provide a precise delineation of a vague concept, then the adaptors used by this vague concept can quickly (less than 5 iterations) converge to their correct value (the value required to define the correct delineation of the vague concept). A quick approximation of their correct value can also be achieved if the *training axioms* are subject to noise or if they contain only partial information about the delineation that the vague concept should adopt. These preliminary results suggest that this framework could be effectively used to update the definitions of vague concepts in order to evolve a single ontology or to improve the extension-based alignment between multiple ontologies.

7.2 Future work

The preliminary results presented in this dissertation could be extended by a more accurate evaluation of this framework. A number of use cases could be designed to evaluate the efficiency of this system using more complex and realistic ontologies. In particular, it would be interesting to study the effect that interconnections between vague concepts would have on the evolution of their definition. In fact, if vague concepts are defined by other vague concepts or if they make use of the same adaptors, then a network of interdependence between the adaptors would emerge. This network will influence the evolution of the adaptors and consequently of the definitions of the vague concepts. It is possible that the convergence of the adaptors to their correct

values could be enforced by this network.

The framework presented in this simulation could also be extended in a number of different directions. Minor extensions includes increasing the number of data types supported and designing a more general learning strategy to use during the learning phase. A major extension to this framework would be to integrate parts of its functionalities into an automatic reasoner. In fact the extraction of feedback during the validation phase could be done more efficiently if it is done simultaneously with the reasoning process and not after. It is possible that when a reasoner detects an inconsistency it would have already collected enough information to determine if that inconsistency is due to the value of an adaptor or not.

Another major extension to this framework consists in allowing other parts of the axioms to change as a result of the evolution process. At the moment adaptors can only be numerical values used in restrictions. In theory however, other parts of an axiom defining a vague concept could contribute to its delineation. For example, adaptors could be extended to have class expressions as values.

Appendix A

List of cities used in the simulation

In the simulation described in section 6.2.3, thirty cities of the United Kingdom were used to generate *training axioms*. The names of the cities used are the following (in the order they were used in the simulation):

1 London
2 Westminster
3 Edinburgh
4 Wakefield
5 Glasgow
6 Manchester
7 Plymouth
8 Swansea
9 Portsmouth
10 York
11 Manchester
12 Birmingham
13 Leeds
14 Sheffield
15 Bradford
16 Liverpool
17 Bristol
18 Cardiff
19 Coventry
20 Nottingham
21 Leicester
22 Belfast
23 Brighton
24 Wolverhampton
25 Derby
26 Southampton
27 Aberdeen
28 Oxford
29 Peterborough
30 Dundee

Appendix B

Computing feedback from a consistent ontology

The following algorithms describe how to compute feedback from a consistent ontology about adaptors found to be correct, as described in section 5.3.3.

Algorithm B.1 Algorithm to compute feedback from a consistent ontology

```
1  computeFeedbackFromConsistency(ontology)
2    feedback = empty set
3    vague_classes = classes of the ontology dependent on adaptors
4    not_vague_ontology = ontology without axioms in vague_classes
5    FOR EACH v_class IN vague_classes
6      members = members of v_class found in the ontology
7      FOR EACH individual IN members
8        class_assertion = assertion that individual is a member of class v_class
9        IF not_vague_ontology entails class_assertion
10         reduced_ontology = not_vague_ontology
11         WHILE reduced_ontology entails class_assertion
12           explanations = axioms of reduced_ontology that entail class_assertion
13           reduced_ontology = reduced_ontology without axioms in explanations
14         ENDWHILE
15         reduced_ontology = reduced_ontology plus axioms in vague_classes
16         IF reduced_ontology entails class_assertion
17             feedback ADD computeFeedbackForCorrectEntailment(reduced_ontology,
                   class_assertion)
18         ENDIF
19       ENDIF
20     ENDFOR
21   ENDFOR
22 RETURN feedback
```

Algorithm B.2 Algorithm to compute feedback for a particular class assertion in a consistent ontology

```
1  computeFeedbackForCorrectEntailment(reduced_ontology,
       class_assertion)
2   feedback = empty set
3    adaptor_set = set of adaptors in reduced_ontology
4    FOR EACH adaptor IN adaptor_set
5      alternatives = empty set
6      relations_restricted = set of relations restricted by adaptor
7      FOR EACH relation IN relations_restricted
8        alternatives ADD values found as targets of the relation in
             reduced_ontology
9      ENDFOR
10     values_that_keep_entailment = empty set
11     values_that_remove_entailment = empty set
12     FOR EACH value IN alternative
13       modified_ontology = substitute(adaptor,value,reduced_ontology)
14       IF modified_ontology entails class_assertion
15         values_that_keep_entailment ADD value
16       ELSE
17       values_that_remove_entailment ADD value
18       ENDIFELSE
19     ENDFOR
20     IF values_that_remove_entailment is not empty
21       IF values_that_keep_entailment is not empty
22         best_value = value from values_that_keep_entailment that
               differs the most from the current value of adaptor
23         feedback ADD feedback about adaptor with best_value as a
               possible alternative value
24       ENDIF
25     ENDIF
26     removeDominatedFeedback(feedback)
27   ENDFOR
28 RETURN feedback
```

Appendix C

Results of extracting feedback from correct adaptors

The following table and figures show the evolution of adaptors d and c. These plots refer to the simulation described in section 6.2 with the following variant: the feedback about correct adaptors was extracted in the validation phase and used in the learning phase.

Adaptors:	d	c
Starting value	0	1
Ending value	1153	3
Min value computed	383	1
Max value computed	1542	3
Average value	932.6	2
Estimated correct value	1325	2.49 or 3.54
Difference from the correct value (% of the standard deviation)	20%	54% or 46%

Table C.1: Statistics about the evolution of adaptors d and c

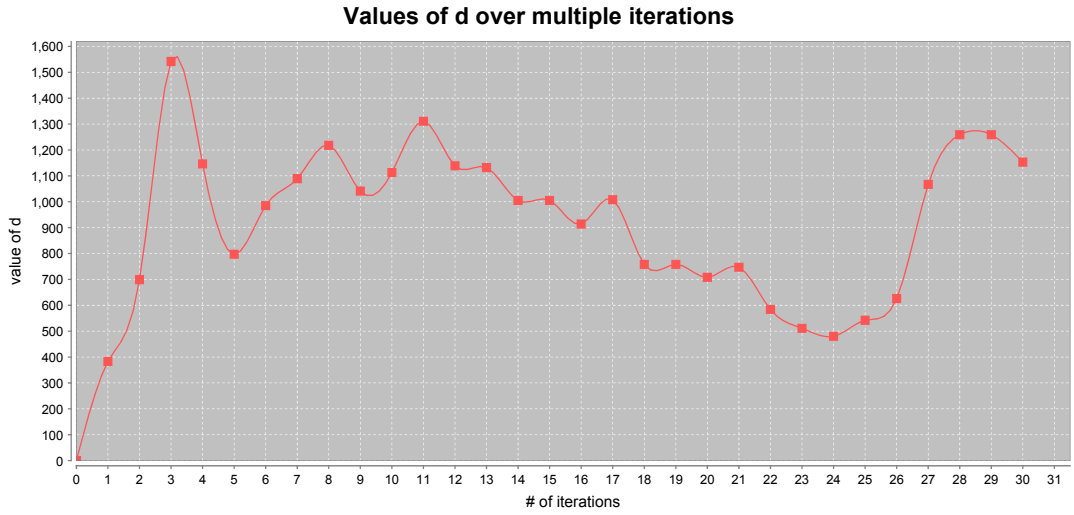

Figure C.1: Plot of the values of adaptor d across 30 iterations of the system (extracting also the feedback about correct adaptors)

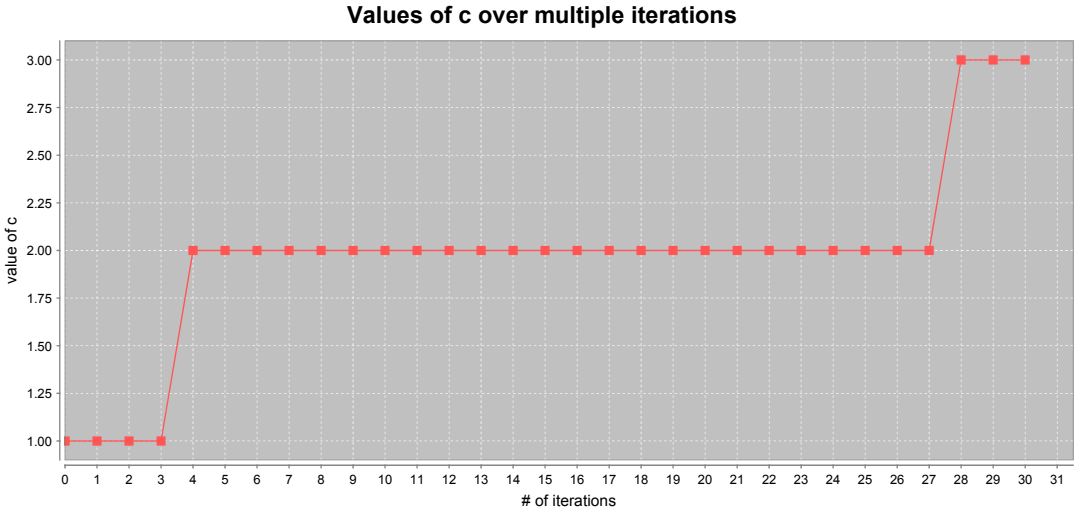

Figure C.2: Plot of the values of adaptor c across 30 iterations of the system (extracting also the feedback about correct adaptors)

Bibliography

[1] Peter Achinstein. Theoretical terms and partial interpretation. *The British Journal for the Philosophy of Science*, 14(54):pp. 89–105, 1963.

[2] Fritz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3. Elsevier, 2007.

[3] David Bell, Guilin Qi, and Weiru Liu. Approaches to inconsistency handling in description-logic based ontologies. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, volume 4806 of *Lecture Notes in Computer Science*, pages 1303–1311. Springer Berlin / Heidelberg, 2007.

[4] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009.

[5] Nicolao Bonini, Daniel Osherson, Timothy Williamson, and Riccardo Viale. On the psychology of vague predicates, 1999.

[6] Alan Bundy and Fiona McNeill. Representation as a fluent: An AI challenge for the next half century. *IEEE Intelligent Systems*, 21(3):85–87, May 2006.

[7] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35:34–41, September 2006.

[8] Nick Drummond and Rob Shearer. The Open World Assumption. In *eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web*, 2006.

[9] Kit Fine. Vagueness, truth, and logic. *Synthese*, 30:265–300, 1975.

[10] Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: Classification and survey. *Knowl. Eng. Rev.*, 23:117–152, 2008.

[11] Peter Haase and Ljiljana Stojanovic. Consistent evolution of owl ontologies. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications*, volume 3532 of *Lecture Notes in Computer Science*, pages 182–197. Springer Berlin / Heidelberg, 2005.

[12] Jeff Heflin and James A. Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 443–449. AAAI Press, 2000.

[13] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *OWLED 2009, 6th OWL Experienced and Directions Workshop*, 2009.

[14] Matthew Horridge and Peter F. Patel-Schneider. Owl 2 web ontology language: Manchester syntax. Technical report, W3C, 2009. URL `http://www.w3.org/TR/owl2-manchester-syntax/`.

[15] Jason Hunter and Brett McLaughlin. JDOM. *Website accessed on: 12/08/2011*, http://jdom.org/.

[16] Google Inc. The Google Distance Matrix API. *Website accessed on: 12/08/2011*, http://code.google.com/apis/maps/documentation/distancematrix/.

[17] Google Inc. The Google Places API. *Website accessed on: 12/08/2011*, http://code.google.com/apis/maps/documentation/places/.

[18] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, pages 267–280, 2007.

[19] A.M. Khattak, Z. Pervez, Sungyoung Lee, and Young-Koo Lee. After effects of ontology evolution. In *Future Information Technology (FutureTech), 2010 5th International Conference on*, pages 1–6, 2010.

[20] Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 428–434. IEEE Computer Society, 2006.

[21] Ling Liu and M. Tamer Özsu. *Encyclopedia of Database Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[22] Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):291 – 308, 2008.

[23] Yinglong Ma, Beihong Jin, and Yulin Feng. Dynamic evolutions based on ontologies. *Knowledge-Based Systems*, 20(1):98–109, 2007.

[24] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96 –101, 2006.

[25] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, 2008.

[26] Pavel Shvaiko, Fausto Giunchiglia, Marco Schorlemmer, Fiona Mcneill, Alan Bundy, Maurizio Marchese, Mikalai Yatskevich, Ilya Zaihrayeu, Vanessa Lopez, Marta Sabou, Ronny Siebes, Spyros Kotoulas, and Coordinator Pavel Shvaiko. OpenKnowledge Deliverable 3.1.: Dynamic ontology matching: a survey, 2006.

[27] 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.

[28] Kees van Deemter. *Not Exactly: in Praise of Vagueness*. Oxford University Press, 2010.

[29] W3C OWL Working Group, editor. *OWL 2 Web Ontology Language Document Overview*. W3C, 2009. URL `http://www.w3.org/TR/owl2-overview/`.

[30] Shenghui Wang, Balthasar Schopman, Lourens van der Meij, Stefan Schlobach, and Frank van Harmelenr. *Automatic subject alignment experiments*. TELplus project, 2010.

[31] Fouad Zablith. Dynamic ontology evolution. In *ISWC Doctoral Consortium*, 2008.