Intuitive Ontology Authoring using Controlled Natural Language

Ronald Denaux

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

University of Leeds

School of Computing

February 2013

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Chapter 2, the background literature, is based on: R. Denaux, C. Dolbear, G. Hart, V. Dimitrova, and A. G. Cohn, Supporting domain experts to construct conceptual ontologies: A holistic approach, Web Semantics: Science, Services and Agents on the World Wide Web, vol. 9, no. 2, pp. 113-127, Jul. 2011. The original background literature for this paper was based on my own work and the Chapter in the thesis updates and goes into more detail about the related work.

The technical description of ROOin Chapter 3 is based on: R. Denaux, V. Dimitrova, A. Cohn, C. Dolbear, and G. Hart, Rabbit to OWL: Ontology Authoring with a CNL-Based Tool, in Controlled Natural Language, vol. 5972, N. Fuchs, Ed. Springer Berlin / Heidelberg, 2010, pp. 246-264. This paper was mostly written by me, and reviewed and improved by the other authors. The design of the Rabbit controlled natural language and the Kanga methodology was done by the Ordnance Survey (C. Dolbear and G. Hart). An initial Rabbit BNF was also done by the Ordnance Survey. I implemented the Rabbit parser on my own. During the implementation of the parser I updated the Rabbit BNF where necessary in collaboration with G. Hart. The initial design of ROO and the guide dog functionality was done in collaboration with Ordnance Survey and V. Dimitrova. The implementation of ROOand the guide dog is my own work.

The evaluation of ROOin Chapter 3 is based on: V. Dimitrova, R. Denaux, G. Hart, C. Dolbear, I. Holt, and A. G. Cohn, Involving Domain Experts in Authoring OWL Ontologies, in Proceedings of the 7th International Semantic Web Conference, ISWC, 2008, pp. 116. The evaluation discussed in this work was done by me and V. Dimitrova with suppervision by the other authors. G. Hart also contributed to the analysis of the results together with V. Dimitrova.

Parts of Chapter 4 were published as: R. Denaux, D. Thakker, V. Dimitrova, and A. G. Cohn, Interactive Semantic Feedback for Intuitive Ontology Authoring,

in 7th International Conference on Formal Ontology in Information Systems, 2012. This research was driven by me with guidance from my supervisors. V. Dimitrova and D. Thakker helped to design and perform the evaluation study, while A. G. Cohn supervised the work.

to... domain experts, I suppose.

Acknowledgements

My immediate thanks to both of my supervisors for their valuable comments and nudges in the right direction. Another thanks to them for their dedication, while at the same time giving me the space to go in my own direction.

More thanks go to Glen Hart for his trust in me and for supporting the work on ROO. I am also grateful to the other members of the Ordnance Survey –Cathy Dolbear, Ian Holt, John Goodwin, Paula Engelbrecht– team with whom I collaborated and helped me to get ROO started in the right direction.

Thanks to the members of the reading group through the years for keeping things lively and interesting and for providing feedback on my work. Special thanks to Dhaval Thacker for helping out with the Entendre evaluation.

Thanks as well to Lora Aroyo, who first brought me in contact with this research area and with Leeds.

Finally, also a big thanks to all those who have provided valuable software that I could reuse and build on for my research. Special thanks to Brian Davis and Kaarel Kaljurand for providing their tools.

Thanks go, of course, to my parents, brother, close friends and good teachers who have all contributed in one way or another to this thesis being created.

A final huge thanks goes to Clare for enduring me during those days before a big deadline and for helping me put research aside once in a while.

Abstract

Ontologies have been proposed and studied in the last couple of decades as a way to capture and share people's knowledge about the world in a way that is processable by computer systems. Ontologies have the potential to serve as a bridge between the human conceptual understanding of the world and the data produced, processed and stored in computer systems. However, ontologies so far have failed to gather widespread adoption, failing to realise the original vision of the semantic web as a next generation of the world wide web: where everyone would be able to contribute and interlink their data and knowledge as easily as they can contribute and interlink their websites.

One of the main reasons for this lack of widespread adoption of ontologies is the steep learning curve for authoring them: most people find it too difficult to learn the syntax and formal semantics of ontology languages. Most research has tried to alleviate this problem by finding ways to help people to collaborate with knowledge engineers when building ontologies; this approach however, requires the wide availability of knowledge engineers, who in practice are scarce. In the context of the semantic web, recent research has started looking at ways to directly capture knowledge from domain experts as ontologies. One such approach advocates the use of Controlled Natural Languages (CNL) as a promising way to alleviate the syntactical impediment to writing ontological constructs. However, not much is yet known about the capabilities and limitations of CNL-based ontology authoring by domain experts. It is also unknown what type of automatic tool support can and should be provided to novice ontology authors, although such intelligent tool support is becoming possible due to advances in reasoning with existing ontologies and other related areas such as natural language processing.

This PhD investigates how CNL-based ontology authoring systems can make ontology authoring more accessible to domain experts by providing intelligent

tool support. In particular, this thesis iteratively investigates the impact of providing various types of intelligent tool support for authoring ontologies using the Web Ontology Language (OWL) and a controlled natural language called Rabbit. After each iteration of added tool support, we evaluate how it impacts the ontology authoring process and what are the main limitations of the resulting ontology authoring system. Based on the found limitations, we decide which further tool support would be most beneficial to novice ontology authors. This methodology resulted in iteratively providing support for (i) understanding the syntactic capabilities and limitations of the chosen controlled natural language; (ii) following appropriate ontology engineering methodologies; (iii) fostering awareness about the logical consequences of adding new knowledge to an ontology and (iv) interacting with the ontology authoring system via dialogues.

The main contributions of this PhD are (i) showing that domain experts benefit from guidance about the ontology authoring process and understandable syntax error messages for finding the correct CNL syntax; (ii) the definition of a framework to integrate the syntactical and semantic analyses of ontology authors' inputs; (iii) showing that intuitive feedback about the integration of ontology authors' inputs into an existing ontology benefits ontology authors as they become aware of potential ontology defects; (iv) the definition of a framework to analyse and describe ontology authoring in terms of dialogue moves and their discourse structure.

Abbreviations

ACE	Attempto Controlled English
AG	Annotation Graph
BNF	Backus Naur Form
cf	communicative function
CL	Controlled Language
CNL	Controlled Natural Language
DE	Domain Expert
DL	Description Logic
dm	dialogue move
fs	functional segment
fundep	functional dependency
GUI	Graphical User Interface
IC	Informational Component
ISU	Information-State-Update
KE	Knowledge Engineer
NL	Natural Language
NLP	Natural Language Processing
$\mathcal{O}$	Ontology
OA	Ontology Authoring or Ontology Author
OAs	Ontology Authors
POI	Point of Interest
POS	Part of Speech
sc	semantic content
SW	Semantic Web

Contents

List of Figures

Chapter 1

Introduction

The process of formally capturing human knowledge has been investigated for many years in the field of computer science. Ontologies [56] have been proposed and studied in the last couple of decades as a way to formally capture and share people's knowledge about the world in a way that is processable by computer systems. Ontologies have the potential to serve as a bridge between the human conceptual understanding of the world and the data produced, processed and stored in computer systems. The Semantic Web [12] envisages ontologies as a key component for enabling the large scale integration and sharing of the rich data sets held by public organisations and businesses. This vision requires the availability, and thus the development, of ontologies ranging from small domain ontologies to large ontologies linked to legacy datasets [3, 39]. However, the time and effort required to create ontological structures is one of the major reasons for the reluctance of large organisations and businesses to utilise SW technologies [3, 90].

Ontology Authoring is the process of developing ontologies. This process is performed by one or more *ontology authors*, people who directly contribute to the formal capture of knowledge in the form of ontological structures. Most ontology authoring tools are designed to be used by *knowledge engineers*, specialists with appropriate knowledge engineering skills but who may lack the necessary domain expertise to create the relevant ontologies [47]. Finding knowledge engineers competent in a specific knowledge domain to be captured in an ontology is a luxury. The most common case is to ask domain experts, people who are subject matter

experts, to provide relevant knowledge sources, or apply knowledge elicitation techniques to discover information directly from the expert, while knowledge engineers encode the ontology using available ontology authoring tools. Apart from creating an extra layer of bureaucracy in the development cycle [90], this approach can hinder the ontology construction process and may have a negative impact on the quality of the resultant ontology (e.g. poor documentation, inconsistency of terminology used, incorrect or incomplete knowledge constructs). There is thus a need for *intuitive ontology authoring*: enabling domain experts to directly contribute their knowledge to ontologies without requiring extensive training in knowledge engineering or ontology authoring tools.

Existing research has shown strong support for using a Controlled Natural Language (CNL) – an engineered language that is the subset of a natural language and is computer processable – as a basis for making ontology authoring tools more intuitive [33, 47, 80, 122, 125]. CNLs enable the syntactic expression of knowledge in a way that can be automatically converted into an appropriate logical formalism [40]. CNLs seem to be a a step in the right direction; however, domain experts still need support about other aspects relevant to ontology authoring [93].

This thesis investigates how CNL-based ontology authoring systems can make ontology authoring more accessible to domain experts by providing intelligent tool support. In particular, this thesis iteratively investigates the impact of providing various types of intelligent tool support for authoring ontologies using the Web Ontology Language (OWL) [115] and a controlled natural language called Rabbit. After each iteration of added tool support, we evaluate how it impacts the ontology authoring process and what are the main limitations of the resulting ontology authoring system. Based on the found limitations, we decide which further tool support would be most beneficial to domain experts who are novice ontology authors. Following this methodology, this thesis presents a novel holistic framework for combining syntactic, semantic and interaction analyses in order to provide intelligent support for (i) understanding the syntactic capabilities and limitations of the chosen controlled natural language; (ii) following appropriate ontology engineering methodologies and (iii) fostering awareness about the logical consequences of adding new knowledge to an ontology.

The hypotheses that drive this PhD are that: *Holistic intelligent tool support based on CNL can enable the active involvement of domain experts in ontology authoring. In addition, integration of syntactic support (in the form of a CNL) with semantic and interactive tool support can improve the effective involvement of domain experts in the ontology authoring.* From these hypotheses we derive the following research questions which we address in this thesis:

- How can CNL-based tool support be integrated with support for following an ontology authoring methodology and how does such combined tool support affect ontology authoring by domain experts?
- How can the syntactic analysis required for understanding textual inputs (such as CNL) be formalised and integrated with semantic analysis of the inputs in order to provide understandable feedback to domain experts?
- How can dialogue systems be used to formalise and improve ontology authoring interactions for better support of domain experts?

We will present a novel ontology authoring tool that enables domain experts to build ontologies using a CNL and that guides them through an ontology authoring methodology. We formalise the (i) syntactic analysis process required to understand textual inputs and generate syntactic feedback (ii) semantic analysis process required to understand the logical consequences of adding new facts to an ontology and (iii) discourse analysis required to provide intelligent interactions and fine-grained support to ontology authors. At the start of this research we have made the following assumptions:

- OWL and Rabbit are representative examples of ontology languages and controlled natural languages for ontology authoring,
- domain experts:
  - have good knowledge of the domain to be modelled,
  - can provide descriptions and knowledge sources for concepts and relationships of the domain;

– usually have no previous experience building ontologies[1], but may have experience building other types of models in their specific domain.

This thesis resulted in a number of original research contribution:

- a new tool for directly involving domain experts in ontology authoring.
- a novel framework for providing syntactic and semantic feedback for ontology assertions.
- a novel dialogue framework for formalising ontology authoring interactions.

This thesis is organised in six chapters. We first will provide an overview of the research context for ontology authoring in Chapter 2, where we will justify the need for making ontology authoring more intuitive. We then will present various approaches for involving domain experts and motivate our choice for using CNLs as a suitable basis to achieve this goal. We will also review the main issues and research challenges on the topic and relate them to this research.

The first iteration of tool support for intuitive ontology authoring will be presented in Chapter 3. We will present ROO, a CNL-based ontology authoring tool designed to support the involvement of domain experts. We will present an existing design for a CNL and an ontology authoring methodology, which we use as the blueprint for designing and implementing the tool. The chapter will then present original work by giving a detailed description of the ontology authoring tool. This description will include various design decisions about how to implement a CNL parser for an expressive language using standard natural language processing tools. We will also describe how to provide guidance to domain experts based on an ontology authoring methodology. This chapter concludes by describing an evaluation study of the tool compared to a similar CNL-based tool.

Based on the experiences with ROO, we performed a second iteration of our research, which is described in Chapter 4. In this second iteration, we noticed that extending our CNL parser to provide feedback about inputs which are not proper CNL was cumbersome and could be improved. We also noticed that

[1]In this thesis, when we use the term *novice ontology author*, we assume that this person is also a *domain expert*.

use of ROO could result in ontology defects being introduced. The chapter will introduce a formal description of the process of syntactically and semantically analysing the inputs of ontology authors (OAs). This chapter will then present a novel integration analysis strategy that is applied to generate feedback for ontology authors. We will present an evaluation to control whether ontology authors benefit from such semantic feedback and whether they understand it without having a background in formal logics.

The evaluations of ROO and the integration analysis from Chapters 3 and 4 indicate that appropriate feedback during ontology authoring could benefit from complex interactions between the ontology author and the system; this led to a third iteration of our research, presented in Chapter 5. This chapter will address interaction problems in ontology authoring systems by presenting a framework for describing ontology authoring interactions in terms of dialogue plans. We will present existing work related to dialogues for ontology authoring and will introduce the general area of dialogue systems. The dialogue framework for ontology authoring is then defined in detail and validated by (i) formalising an existing ontology authoring interaction as a dialogue (ii) adapting and formalising a dialogue to make it more intuitive for novice ontology authors and (iii) presenting the implementation of the dialogue framework and the formalised dialogue interactions.

Finally, Chapter 6 will summarise the main aspects of the research, discussing the main contributions of each iteration and future directions of research.

Chapter 2

Intuitive Ontology Authoring

Ontology languages appear to offer a unique opportunity for people to formally describe their knowledge. The vision of the Semantic Web sees such formal descriptions being used to give context to data that is published online, which in turn can be exploited by computers to foster sharing and discoverability of knowledge on the web. However, capturing formal descriptions in the form of ontologies is not a trivial task. This has resulted in a broad literature on ontologies, ontology languages, ontology engineering and ontology authoring. This chapter aims to define the area of ontology authoring; in doing so, this chapter also narrows the scope of this PhD.

The next sections will set the context and the theoretical foundations that underpin this research. The sections show the need for making the ontology authoring process more intuitive and accessible to people, in particular to domain experts. Furthermore, we motivate our decision to use Controlled Natural Languages as our main approach for achieving more intuitive ontology authoring.

2.1 Ontology Authoring

In this PhD, we use the term *Ontology Authoring* for the process of creating or editing a formal ontology. Vital steps in this process are the *understanding of existing knowledge* and the *formulation of new knowledge* in an ontology language. This section aims to provide an overview of ontology authoring. We do this by defining what are ontologies and introducing the general research area of ontology

engineering. We also describe the classical tools which are used for ontology authoring, which are meant to be used by knowledge engineers.

2.1.1 Preliminaries

Ontologies

The output of the ontology authoring process is an ontology. A common definition of *Ontology* is given by Gruber as a "formal, explicit specification of a shared conceptualisation" [55]. In this PhD we assume that the goal of ontology authoring is to produce such an ontology. However, achieving the conditions of "explicitness" and "shared conceptualisation" can take a long time, thus we will also call any intermediate formal representation of knowledge an ontology[1] .

In order to produce a formal representation of knowledge we assume the use of an *ontology language*: a formal language that can be used to represent knowledge about the world in a manner that minimises ambiguity. Various such languages have been developed which vary in terms of expressivity, underpinning logic, level of standardisation, computational complexity, tool support, etc. Recent examples of ontology languages are the Web Ontology Language (OWL) [2], Common Logic [34], F-Logic [88], the Resource Description Framework (RDF)[3] and the RDF Schema language (RDFS)[4].

It is worth noting that current ontology languages are based on decades of research into knowledge representation and expert systems. Early expert systems, like MYCIN [20] used rules to capture the domain knowledge and identified the need to capture knowledge in formal languages. Another early language for capturing knowledge is Prolog[129], a declarative (and general programming) language which is still in use today and enables the specification of knowledge in terms of facts and rules. Besides rule-based knowledge representation, there

[1] That is: in this thesis, we will not require ontologies to be shared between a community or have some level of coverage of the domain. This is because in this thesis, we focus instead on the knowledge formulation aspects of ontology authoring.

[2] `http://www.w3.org/TR/owl2-overview/`

[3] `http://www.w3.org/TR/rdf-primer/`

[4] `http://www.w3.org/TR/rdf-schema/`

are also various frame-based languages such as the Frame Representation Language (FRL) [116], KL-ONE [18], CycL [100]. Description logics have led to new languages such as DAML [71] and OIL [70], the direct precursors to OWL.

In this PhD we will target ontologies based on OWL. The main reasons for choosing this language are as follows:

Expressivity OWL is based on the $\mathcal{SROIQ}$ description logic. This means it is more expressive than lightweight ontology languages such as RDF and SKOS, but less expressive than F-Logic or Common Logic. OWL is a fairly expressive language: it enables the definition of complex terminological knowledge (concepts and relations) as well as factual knowledge (instances). OWL is a good choice for this PhD because we assume that the more expressive an ontology language is, the more tool support is required to use the language correctly. Hence, we expect OWL to be a good choice for investigating tool support for enabling more intuitive ontology authoring since it provides the right balance of expressivity.

Note that in this thesis, we distinguish between *lightweight and heavyweight ontologies*, which are terms that are related to the ontology language expressivity. Lightweight ontologies are those that only provide a glossary of concepts and relations as well as some simple hierarchical organisation of those concepts. Ontologies languages with limited expressivity like RDF and SKOS can be used to create such ontologies. Heavyweight ontologies include more complex relations between concepts and relations like, for example disjunction of concepts, anonymous concepts and property chains. More expressive languages can be used to create such ontologies, but can also be used to create lightweight ontologies, simply by using only a subset of the provided language constructs.

Standardisation OWL is a W3C standard. This is important to minimise compatibility issues in the future and to ensure a fixed language definition. Several other languages (RDF, Common Logic) have also been standardised.

Popularity and Tool Support OWL and RDF have gained great popularity in the last decade. This means that example ontologies are readily available.

Furthermore, the popularity of these languages means that there is a pool of research and production tools and algorithms that we can reuse in our research.

Designed for the Web OWL has been designed with the Web in mind. This makes it more likely that existing ontologies in this language are published in the web. This availability of existing ontologies can be useful for our research.

There is a push to creating ontologies, because their formally encoded knowledge can be used in various ways [87]: enabling and improving search; retrieval of information; improving the presentation of information; facilitating the integration of heterogeneous information; enabling reusability and interoperability of information systems; and enabling personalisation services.

Ontology Engineering

The push for creating useful ontologies, paired with the realisation that ontologies are software artifacts has resulted in a new discipline called *Ontology Engineering*. This discipline is very broad and studies:

methodologies for ontology construction which prescribe steps that need to be taken when constructing ontologies. Example methodologies are METHONTOLOGY [44], Uschold & King [138], On-to-Knowledge [131] and the NeON methodology [130].

approaches that can be taken for building ontologies. Example approaches include automatic learning of ontologies from existing data sources and construction of ontologies based on existing thesauri [105, 142].

tools for supporting ontology construction and validation. This includes ontology authoring systems [91, 94, 134], tools for matching ontologies [42], tools for checking the consistency of ontologies [106], tools for ontology debugging [82] etc.

ontology evaluation defines methods for evaluating created ontologies [43, 57].

The first methodologies, approaches and tools often assumed that only **Knowledge Engineers**, individuals trained in formal logics and ontology modelling, would build ontologies. This is reflected in the mainstream ontology authoring systems such as Protégé and TopBraid, which require extensive training before being usable. This resulted in a bottleneck for ontology creation, since:

- only a limited number of people had the skills to build ontologies, but lacked knowledge about the domain to be captured in an ontology.
- conversely, the domain experts, individuals who have the knowledge about the domain to be modelled, lacked the skills to formulate this knowledge as ontology constructs.

Recently, this resulted in a series of new methodologies, approaches and tools that try to make ontology authoring more intuitive. The goal of these new approaches is to facilitate the active involvement of domain experts in ontology authoring. We discuss these developments in the next section.

2.1.2 Involving Domain Experts

Promoting the *active* involvement of domain experts in the ontology authoring process has been an important trend in recent research efforts in Ontology Engineering.[1] In the area of the semantic web, two recent ontology engineering methodologies have suggested possible ways to achieve the involvement of domain experts in ontology engineering.

DynamOnt The DynamOnt project [54] pointed out that existing methodologies did not support domain experts because they lacked appropriate support for communities and collaboration. The project aimed at producing a methodology where domain experts could create lightweight ontological models that could be used as part of an "evolving conceptual model". To

[1] Note that the wider problem of acquiring knowledge from domain experts is a longstanding problem in expert systems and various techniques have been researched to facilitate knowledge acquisition, for example in [17]. The novelty now is that, in the context of the web, the presence of a knowledge engineer is more unlikely, while the ease to publish ontologies on the web makes it more important for domain experts to be able to create ontologies.

achieve this, DynamOnt proposed to reuse existing methodologies (Uschold & King [138] and CommonKADS [121]) while (i) adding guidance for domain experts so they can act as knowledge engineers; (ii) encouraging collaboration; and (iii) grounding the resultant ontologies using foundational ontologies such as DOLCE [50].

HCOME Similar to DynamOnt, the HCOME methodology [92] argued that traditional ontology construction methodologies such as METHONTOLOGY [44], Uschold & King [138], On-to-Knowledge [131], rely too much on the knowledge engineer for development, maintenance and evolution of ontologies and minimize the role of the domain experts. The HCOME methodology proposes to support individual domain experts by enabling them to collaborate in the construction of ontologies with a community of knowledge workers.

DynamOnt and HCOME have both pointed out limitations of traditional ontology construction methodologies and suggested to involve domain experts by (i) considering ontology construction as a *joint process* involving both domain experts and knowledge engineers and (ii) providing domain experts with *suitable guidance* to ensure their active involvement in ontology authoring. Below, we will discuss various approaches and tools that have been suggested to foment the involvement of domain experts.

More recently, the **Kanga** methodology [93] for ontology construction has been developed. Kanga has been informed by experiences at Ordnance Survey when building several ontologies in the topographical domain. Kanga adds to the existing ontology methodologies focusing on domain experts' involvement by clearly identifying the assumptions about domain experts and distinguishing the phases where domain experts or knowledge engineers should be involved. Most other methodologies also explicitly include the domain expert. Where Kanga differs is in the emphasis it places on the domain expert and the central role that the expert plays. Kanga requires the domain expert to take the lead role, guided by the knowledge engineer but nevertheless in control. So where Kanga differs is not in that it involves the domain expert where others do not, rather the degree to which it involves them. Additionally, Kanga does not sacrifice the

expressivity of the resultant ontologies and describes how domain experts can be involved in the construction of highly expressive and interconnected ontologies by using a Controlled Natural Language interface. Thus Kanga has largely adapted best practice to place greater emphasis on the domain expert and therefore the novelty in Kanga results purely from this shift of emphasis. Since we will base our work in Chapter 3 on the Kanga methodology, we provide an overview of Kanga in Section 3.1.

Approaches and Tools

Several approaches and tools to involve domain experts in the ontology construction process have been proposed in recent years. We discuss these approaches here and highlight target contributions for this PhD. Note that the following approaches are not exclusive of each other; that is, these approaches are often compatible and can be (and have been) combined to minimise drawbacks of the separate approaches.

Ontology Engineering tools that improve collaboration focus on supporting a community of people (including domain experts) to build ontologies, e.g. HCONE[1] [92] and Web Protégé [136]. These tools provide communication and Web 2.0 techniques – such as discussion forums – to aid users to propose, document and implement changes to the ontology. The main advantage of this approach is that it encourages the formation of a community of both domain experts and knowledge engineers to collaborate in building the ontology. These tools improve the communication between domain experts and knowledge engineers, which may motivate domain experts to provide more input into the ontology construction process. However, the means to edit the ontology are similar to traditional tools, e.g. Protégé, which makes domain experts heavily dependent on knowledge engineers to formalise the ontologies. Recent studies explore customised interfaces that domain experts can be comfortable with (predefined forms or excel sheets) and can be converted into OWL [120, 137]. This clearly has potential for facilitating domain experts' involvement in ontology engineering,

[1]Note that HCONE is the ontology authoring tool that supports the HCOME methodology.

however their participation is currently restricted to discussions and the population of ontologies with specific instances and subclasses without being directly involved in the adding of new formal definitions. Hence, the ontology constructs are actually composed by a group of knowledge engineers (who may or may not be domain experts), while the domain experts without knowledge engineering experience mainly provide the knowledge sources and are involved in the verification of the ontology. The reported experimental studies indicate that this approach enables the involvement of domain experts; however this involvement is limited, since the presence of knowledge engineers is still required in order to produce ontologies.

Semantic Wikis [51] are extensions allowing the wiki manager to define a broad ontology structure that corresponds to wiki pages. Users then refine the ontology by editing and semantically tagging wiki pages. The wiki interface hides the ontology formalisms from the users, in this case domain experts, who can add information to the ontology model by editing wiki pages. Note that to make the interaction intuitive, an initial ontology needs to be created with input from both domain experts and knowledge engineers (e.g. to create semantic forms in Semantic Media Wiki). Semantic wikis offer a flexible approach for lightweight ontology engineering. However, they are inappropriate for heavyweight ontology engineering which requires more expressive logical formalisms, such as description logic and OWL.

Ontology Maturing[19, 119] aims to reuse semi-structured data produced by knowledge workers such as emails, tags and existing schemas and classifications to produce lightweight ontologies and eventually heavyweight ontologies. This approach looks at ways for users to add formal semantics to existing data one layer at a time. Proposed tools, e.g. SOBOLEO [19], for extracting a lightweight ontology based on a set of tags or existing schema provide intuitive ways for domain experts to encode their knowledge of the existing data. However, this work does not seem to have follow-up studies that show the suitability of this approach for producing heavyweight ontologies. The available literature indicates that this approach only achieves minor contributions from people without knowledge engineering skills to the ontology maturing process.

Ontology Understanding aims to make it easier to understand what type of knowledge is represented by a particular ontology. This is achieved through a variety of techniques such as (i) ontology summarisation where the main concepts in an ontology are extracted [113]; (ii) showing relevant metadata [62]; (iii) showing *Ontology Visualisations* to gain insights into the structure of ontologies and linked data [108]. Domain experts can benefit from these approaches by getting a high-level understanding of existing ontologies that they can reuse or extend. However, these techniques do not always translate to enabling domain experts to add new knowledge to the ontologies. From the three techniques mentioned above, only ontology visualisations can be used to provide visual interfaces for editing ontologies [94], but still require learning the available interface and learning about how the interface relates to the ontology language.

Limitations of Knowledge Formulation

A common limitation of all of the presented approaches and tools is that they *only enable knowledge formulation of lightweight ontologies by domain experts*. A possible cause for this limitation is that they all rely on Graphical User Interfaces (GUIs) to enable knowledge formulation. The problem with this approach is that the GUI must *provide a different GUI component for each construct in the target ontology language*. In this case, domain experts need to learn the different types of components and they also need to have a rough idea of the meaning of each component. Designing such GUIs in an intuitive way can be very challenging. For example, a GUI that show subsumption hierarchies can be misinterpreted by users, resulting in the use of mereological relations instead [60]. Another example of a component that is difficult to make intuitive is shown in figure 2.1; this is a component for entering object property restrictions in OWL where the user needs to know the meaning of OWL terminology such as *Restriction type* (existential, universal or cardinality), a *Restricted property*, *filler* and *cardinality*.

The discussion of the various approaches that use GUI components indicate that it is difficult to produce a GUI that is sufficiently flexible and intuitive in order to facilitate knowledge formulation for expressive ontology languages [87, 137]. As a way to get past these limitations, a novel approach has been proposed based on Controlled Natural Languages, which we discuss next.

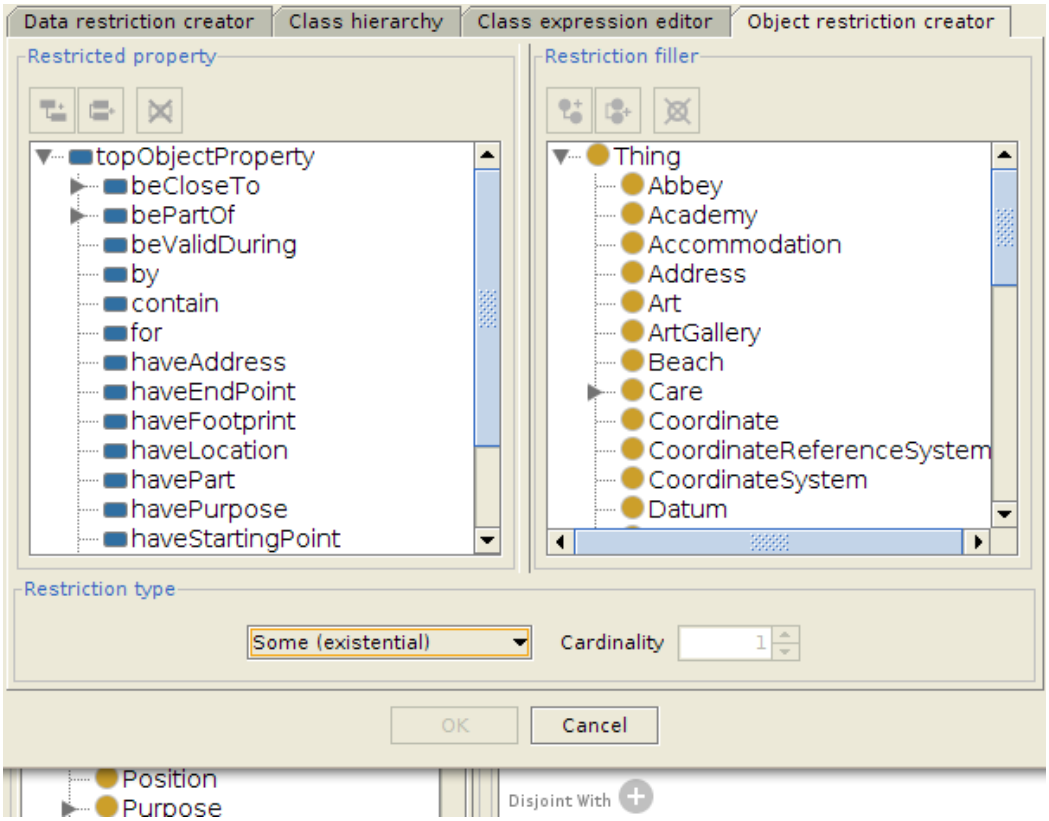

Figure 2.1: Screenshot of the *Object Restriction Creator* component in Protégé 4.

Controlled Natural Language as a Way Forward

Recently, a new approach that relies on the use of *Controlled Natural Language (CNL) interfaces* [46] to perform ontology engineering has been explored. A CNL, in this case, is a formal language that closely resembles natural language. The main advantage of such an approach is that it drastically lowers the barrier for domain experts to understand existing ontologies and to contribute statements to the ontology. Compared to the other approaches discussed above, this is the only approach that does not require collaboration with a knowledge engineer while still allowing the full use of the ontology language.

Because this is such a promising approach, this PhD will use a CNL as the way to achieve knowledge formulation by domain experts. We discuss existing research on CNLs for ontology authoring in more detail in the following section.

To sum up this section, the presented approaches show that in practice, it is

very difficult to achieve effective domain experts' involvement without appropriate tool support. This PhD aims to contribute to this area of research by investigating a number of novel ways to provide tool support to domain experts.

2.2 CNL-Based Ontology Authoring

A **Controlled Natural Language** (CNL) is a language that is based on a natural language (e.g. English), but that is restricted (i.e. controlled) in some aspect [141] (e.g. its vocabulary, morphology, syntax or interpretation). The idea of formally defining a sublanguage of a natural language for computer processing was pioneered in the 1970s. For example, in [111] an early form of a controlled natural language is used to construct a "conversational domain", where the translation between a natural language sentence into a formalised structure is performed by a human intermediary. As another early example of a CNL, [107]proposes the definition of a CNL as an interface for writing SQL queries. The idea of defining a sublanguage for English and French was used to automatically translate restricted domains such as meteorology and aviation manuals [73].

The pioneering work of the 1970s and 1980s has evolved and currently, in the context of ontology authoring for domain experts, CNLs can be used to help them:

Understand existing Ontologies. CNLs can be used to generate CNL sentences from an existing ontology [2, 16, 139]. In this case, the focus when defining the CNL is on making the CNL sentences easy to understand and making sure that they reflect the semantics of the ontology language. Such CNLs can improve domain expert participation, however since such a CNL is not necessarily easy to write, the participation of domain experts is limited. That is, domain experts cannot directly contribute their own knowledge to the ontology, but have to rely on somebody who can formulate knowledge in the formal ontology language. An example of a CNL that is designed for readability but not writability is presented in [128].

Query existing Ontologies. CNLs can be designed to help domain experts formulate questions to be presented to the ontology. This is helpful for

ontologies that are quite large, since browsing through the ontology may be difficult in such cases. This is also helpful for heavyweight ontologies, since such ontologies can contain implicit knowledge that follows from the explicitly stated knowledge and the ontology language semantics. Having a CNL to formulate questions helps domain experts understand what knowledge is captured in the ontology. CNLs that have been proposed for query answering include [4, 13, 23, 132]. These languages make it easier for domain experts to evaluate an ontology, as they can pose questions that need to be answered based on the ontology and a set of instances. To the best of our knowledge these languages can only be used to formulate queries; they cannot be used to construct ontologies because they do not support the input of new knowledge into the ontology. They may however, be closely related to languages that do allow authoring.

Formulate Knowledge. CNLs can be defined which focus on both understandability and writability. In particular, these types of CNLs can be automatically converted into some target ontology language. Such CNLs are very promising for involving domain experts, since they allow for the direct involvement of the domain authors. The promise of such CNLs is that domain experts will no longer be dependent on people who can formulate knowledge in an obscure logical syntax. Below, we look at existing CNL-based approaches for knowledge formulation in more detail.

2.2.1 CNLs for Knowledge Formulation

We now discuss various CNLs that have been proposed in the literature to facilitate knowledge formulation by domain experts. For each CNL we:

- discuss any relevant design decisions and history. In particular we describe how the grammar is specified, whether there is an intermediate representation that is used and what the target logical language is for the CNL.
- state whether CNL is linked to a tool that provides support for entering valid inputs;

- discuss any findings and limitations that have been reported in the literature and
- provide a small number of example sentences in the CNL to provide a feeling for the language.

ACE Attempto Controlled English [45, 80] is the most mature CNL having originally been created to translate into First Order Logic. The grammar of ACE is specified using a Definite Clause Grammar [112], which is then parsed into an intermediate representation in the form of a Discourse Representation Structure [83]. This structure can then be converted into first order logic.

A subset of ACE is now used to drive an ontology authoring applications called ACE View [77] where the resulting sentences are translated into OWL and SWRL. ACE View provides a CNL interface to enter and view ontology knowledge that is automatically translated into OWL.

A major limitation for achieving active involvement by domain experts is that ACE View requires knowledge engineering expertise in order to be used effectively. This is due to the lack of guidance through the ontology construction process, which is missing in ACE View.

ACE View provides bi-directional conversion between OWL and ACE. This allows ACE View to import existing OWL ontologies and generate corresponding ACE sentences. The bi-directionality also allows ACE View to adapt and expose advanced functionality from Protégé 4. For example, inferred axioms can be shown as ACE sentences and explanations of some inferred axioms can also be rendered as lists of ACE sentences.

ACE has also been used in various promising directions for making ontology authoring more intuitive. ACE has been integrated in a multilingual semantic wiki [78]. It has also served as the basis of a speech recognition system [79], demonstrating the versatility of the CNL approach, which is not bound to purely textual inputs. Finally, ACE has been used to provide a paraphrasing service to help domain experts get an alternative rendering

for ontological statements [81]. It is not clear how useful these features are since no systematic evaluation of these features has been reported.

Examples of ACE sentences taken from [123] are:

- `If something X is larger than something Y then Y is not larger than X.`
- `Every river-stretch has-part at most 2 confluences.`
- `For every factory its part is a building whose purpose is a manufacturing.`

CPL The Computer-Processable Language is another mature CNL, developed at Boeing, and used in the HALO project [4, 24, 26]. HALO improved over other CNL approaches by providing a more holistic approach: the CNL is provided in conjunction with support of the ontology construction process and not just as a standalone tool for entering knowledge into the system. Within the HALO project, CPL is used to formulate queries, rather than to capture domain knowledge [1]. A missed opportunity in this case is that no systematic evaluation was performed to study the impact of the CNL aspect combined with tool support to aid domain experts in ontology authoring tasks.

CPL is parsed using a broad coverage chart parser [58], which generates an intermediate representation called a "logical form" which resembles a abstract syntax tree, but includes some logic-type symbols. This logical form is finally translated into a frame-based language called Knowledge Machine [27], which is based on first-order logic. Example sentences from [24] are:

- `An object is thrown from the top of a cliff.`
- `The object falls from the top of the cliff to the ground.`

SOS The Sydney Ontology Syntax [29] is a CNL that targets OWL. We are not aware of any tool support for the Sydney OWL Syntax, although SOS is

[1] Knowledge capture is performed using a graphical representation.

a language that is based on PENG (Processable English), which provided tool support in the form of an editor with auto-completion features [124]. The PENG editor also proposed the use of reasoning services to provide warnings and error messages based on results of theorem provers; however, no evaluation was reported to show how useful such services were in practice [124].

PENG uses a similar approach as ACE by starting with a Definite Clause Grammar to define the language constructs. It differs from ACE in that it then uses a top-down chart parser before constructing the Discourse Representation Structure, which is finally translated into a first-order notation. In the case of the Sydney OWL Syntax, the final step is to convert from that first-order notation to OWL. Example SOS sentences from [123] are:

- `If X is larger than Y then Y is larger than X.`
- `Every river stretch has at most 2 confluences as a part.`
- `Every factory has a building as part that has a manufacturing as a purpose.`

CLOnE Controlled Language for Ontology Editing is a CNL developed using standard Natural Language Processing techniques [47]. The reported CLOnE language is a proof of concept in the sense that it only covers a small subset of OWL. This small subset is enough for building lightweight ontologies. One contribution reported in [47] was performing a comparative study between CLOnE and Protégé. This study showed increased efficiency in building ontologies using a controlled natural language as opposed to the classic Protégé user interface.

CLOnE is defined in terms of a JAPE [30, Chapter 7] transducer that matches regular expressions over basic NLP annotations as its input: sentence splitting, part-of-speech and morphology analysis. Matched patterns are directly translated into OWL constructs without an intermediate representation. Example sentences from [47] are:

- `Persons are authors of documents.`

- `Carl Pollard and Ivan Sag are authors of 'Head-Driven Phrase Structure Grammar'.`

Rabbit is a CNL developed by the Ordnance Survey as part of a methodology for involving domain experts (in the case of the Ordnance Survey: geographers, cartographers, etc.) in ontology authoring. When this PhD started, the language consisted only of a rough specification of the types of sentences that should be allowed in Rabbit. No tool support was available for parsing the language, converting it to OWL or helping domain experts to write this CNL. However, due to the combination of the ontology engineering methodology, our collaboration with Ordnance Survey and the following design decisions of the CNL, we have chosen this CNL as the starting point for our investigations:

- the use of one CNL pattern to represent one OWL axiom type
- lightweight intermediate representation (no requirement for anaphora resolution, no need for Discourse Representation Structure);
- direct control of the language and tool development (none of the other available CNLs at the time were open source)[1];
- full support of OWL constructs.

We provide a more detailed look at the Rabbit language in Section 3.1.1; a few example sentences taken from [123] are:

- `The relationship ‘‘is larger than’’ is asymmetric.`
- `Every River Stretch has part at most two confluences.`
- `Every Factory has a part Building that has Purpose Manufacturing.`

The CNLs discussed above differ in some of their design choices regarding how strongly they aim to recreate natural language sentences and how easy it is to define computable linguistic restrictions on the language. In the end, however, languages targeting the same ontology language tend to be similar to each other [123].

[1]The ACE parser has now been open sourced.

Conclusion

Evaluation of existing CNLs for ontology authoring have shown positive results regarding the intuitive understandability of CNL sentences [41, 96], at least in the case of small sets of CNL sentences. Results are less positive regarding the writability of CNLs, and point at the need for tool support for producing syntactically correct sentences. However, one of the main benefits of CNLs is that they enable the provision of such tool support due to the formal syntactic specification of CNLs. In the next section we summarise the types of syntactic support that they enable.

2.2.2 Syntactic Support

CNLs require a formal definition of their syntax (a language grammar), which can be used to provide a variety of ways to help ontology authors enter syntactically correct inputs:

1. A first use of this syntax information is to *provide templates* that the user can select and then fill in. This works best when the expected input sentences have keywords that are fixed. For example in Rabbit `Every <concept> is a kind of <concept>` is a template for declaring a new subsumption relation between two OWL classes. However, since OWL allows the expression of complex restrictions (anonymous classes) it is difficult to predict (or provide) all templates that authors may require.

2. Another use of the language grammar is to provide *predictive input support* [124], where the system can analyse the current (partial) input and predict what type of input is necessary at the position of the cursor in order to get a grammatically correct sentence. This analysis is typically used to show a list of completions: e.g. a list of OWL entities or keywords that can be used next in order to have a grammatically correct sentence. This approach tends to work well to remind users of the available vocabulary and to avoid typing. The approach is less useful when the grammar allows for a great number of variations or when the list of entities or keywords is very large; in such cases users may require a long time to find a suitable

suggestion in the list of completions. This approach can also be seen as intrusive.

3. The language grammar can be used to *display syntax highlighting* of the input. This generally means highlighting which parts of the input are keywords and which parts of the input refer to the different types of OWL entities. This helps the user learn the different templates of the language as well as the main types of entities supported by the language.

4. Another use of syntactic information is the *generation of paraphrases* [81]. For this, some extra syntactic information is required besides the grammar of the language: the system needs to know that different syntax trees can have an equivalent mapping onto OWL. When this is the case, the system can generate a paraphrase: a different syntactic representation of the input that has the same OWL translation. This can help the user to decide whether the input is being interpreted correctly.

The CNL literature has focused on providing technical solutions to achieve these forms of tool support. However, evaluation of these tool support techniques has not been systematic and it is not clear how useful the tool support is in practice for domain experts. There is thus a need for conducting controlled studies of domain experts performing realistic ontology authoring tasks.

2.3 Discussion and Open Issues

This chapter provided an overview of the research context for intuitive ontology authoring. We summarise the main issues and research challenges that we found:

- Traditional ontology authoring tools such as Protégé are useful for building expressive ontologies. They provide several reasoning tools which are crucial for analysing and debugging ontologies. However, such tools are intended to be used by knowledge engineers with a background in logic and ontology modelling. A main research challenge is to **facilitate the building expressive of ontologies by domain experts**. This challenge can

be subdivided in the facilitation of (i) knowledge formulation; (ii) following ontology engineering methodologies and (iii) ontology evaluation and debugging through the use of reasoning services.

- In order to involve domain experts, various new approaches have been proposed based on Web 2.0 and social aspects. Most of these approaches only partially facilitate knowledge formulation as they only enable the building of lightweight ontologies by domain experts (see Table 2.1 for a summary of the approaches). A promising approach is the use of CNLs for ontology authoring as it seems to facilitate knowledge formulation using the full expressivity of ontology languages. However, syntactic tool support is necessary, which has been proposed but has not been evaluated in realistic settings. The main research challenge here is to **evaluate the practical use of CNLs to enable domain experts to build expressive ontologies**. One of the challenges in performing such an evaluation is that CNLs only provide support for knowledge formulation, while ontology authoring also requires support for the process of ontology engineering. Another challenge is thus to **provide holistic support for knowledge formulation and following ontology engineering methodologies to domain experts**.

A recent review of Semantic Content Authoring (SCA), summarises the problems with current authoring ontologies:[1]

> "Many current SCA systems bear a bewildering amount of functions and algorithms which confuses both the novice and expert users. This problem causes a significant impediment for a broader use of SCA systems." [87]

The work presented in this PhD tackles the research challenges identified above by contributing to research on making ontology authoring more intuitive for

[1]The difference between SCA and Ontology Authoring is one of focus on the type of knowledge being captured. While SCA focuses on capturing factual knowledge, OA focuses on gathering conceptual knowledge (both approaches support the capture of both types of knowledge, though).

Ontology Engineering Tools	**Traditional Tools**	**Collaborative Tools**	**Semantic Wikis**	**Ontology Maturing Tools**	**CNL-based Tools**	**Research Target**
Example	Protégé, TopBraid Composer	Web Protege, HCONE	Semantic MediaWiki	SOBOLEO	ACE View	–
Ontology Engineering experience required	Yes	Yes (at least one team member)	No (but initial ontology required)	No	Yes	No
OWL knowledge required	Yes	Yes	No	No	No	No
Stages requiring Knowledge Engineer	All	All	Creation of initial ontology	Formalisation (only for heavyweight ontologies) and Validation	Validation All stages (for guidance)	Validation
Supported Ontology Expressivity	All	All	Lightweight	All, but currently more suitable for lightweight ontologies	All	All
Difficulty for domain experts	Very difficult	Difficult	Easy	Easy	Easy	Easy

Table 2.1: Comparison of existing ontology construction tools and target contribution for this PhD.

domain experts. In the next chapter we start by building and evaluating a CNL-based tool that enables domain experts, with no knowledge engineering experience and without the direct help of knowledge engineers, to build both lightweight and heavyweight ontologies. Our research intentionally avoids the collaborative aspects of building ontologies (present in the collaborative and semantic wiki approaches in Table 2.1) in order to focus on how domain experts can create ontologies that represent their knowledge of their domain. Contributions of our work are compatible with – and could improve – collaborative approaches as it allows communities to be less dependent on knowledge engineers and to produce more complex ontologies if necessary.

Chapter 3

ROO: CNL-Based Ontology Authoring

The main objective of this PhD is to investigate how intelligent tool support can help domain experts to become actively involved in ontology authoring. The previous chapter made the case that ontology authoring systems based on Controlled Natural Languages are a promising approach to achieve this goal. However, existing CNL-based tools for ontology authoring do not provide enough support to domain experts. For example, they focus solely on providing a CNL interface, while ignoring the *whole* ontology construction process, and thus assume that the user of the system already has good knowledge engineering skills. Our first step is thus to investigate how to improve on existing CNL-based ontology authoring tools through the use of intelligent techniques.

This chapter presents a CNL-based ontology authoring tool designed to support the involvement of domain experts who are novice ontology authors. We base this tool on an existing design for a CNL and an ontology authoring methodology, which we present in Section 3.1. The chapter then presents our original work by introducing the ontology authoring tool (in Section 3.2) and proposing an approach for providing intelligent tool support for (i) formulating valid CNL statements (Sections 3.3 and 3.3.3) and (ii) following an ontology engineering methodology to build an ontology (Section 3.3.4). Finally, we describe an evaluation of our approach (Sect. 3.4), discuss other practical experiences with the tool and provide a summary and conclusions in Sect. 3.6.

3.1 An Existing Methodology for Involving Domain Experts in Ontology Authoring

As we mentioned in the previous chapter, the Ordnance Survey (the mapping agency of Great Britain) has designed Rabbit, a CNL for authoring OWL ontologies. The design of this CNL was part of larger effort by the Ordnance Survey for developing a modular topographic domain ontology to facilitate the description and reuse of its topographic data by third parties [93]. At the heart of ontology development is the *active involvement of domain experts* (e.g. geographers and ecologists), which is reflected in *Kanga*, the Ordnance Survey's methodology for ontology construction [93]. Kanga includes several steps:

- Identifying the scope, purpose and other requirements of the ontology;
- Gathering source knowledge and documents and identifying ontologies for reuse;
- Capturing the ontology content in a knowledge glossary;
- Formally defining core concepts and relationships between concepts by using controlled English sentences;
- Converting the controlled English sentences into OWL[1];
- Ontology verification and validation.

Following this methodology, the domain expert is engaged in the construction of a *conceptual ontology* which involves the first four steps. The knowledge engineer then performs the last two steps, which focus on the logical level of the ontology.

A crucial component of the Ordnance Survey methodology is the use of a controlled language for authoring the conceptual ontology — a CNL, called Rabbit, has been developed for this purpose [40].

[1]OWL (Web Ontology Language) is a W3C standard for authoring ontologies intended to be used in the Semantic Web. See `http://www.w3.org/TR/owl-features/`

3.1.1 The Rabbit Controlled Natural Language

The Ordnance Survey has designed Rabbit [59] to be easy for domain experts to digest and produce, allowing them to express what they need to in order to describe their domain[1].

The fundamental principles underlying the design of Rabbit are:

- To allow the domain expert to express their knowledge as easily and simply as possible and in as much detail as necessary. It is written to appear like a natural English statement;
- To have a well defined grammar and be sufficiently formal to enable those aspects that can be expressed as OWL to be systematically translatable;
- To recognise that the domain expert alone cannot produce an ontology and that a knowledge engineer is also necessary;
- To be used in conjunction with tools which help to enforce an authoring method but not to the point where Rabbit is only readable through tools;
- To be domain independent.

Tables 3.1, 3.2 and 3.3 show commonly used sentence structures for describing concepts, relationships and individuals. The tables also show the translation of the Rabbit sentence in the OWL Manchester Syntax [67].[2] The language covers most of the constructs in OWL 2, see [59].

Ordnance Survey has performed a number of user evaluations that helped the design of Rabbit. First, Rabbit sentences were presented to users (geography students) along with a multiple choice of possible different interpretations for the sentence [60]; this showed that most sentences were correctly understood by most users. This study showed that some constructs of the language had to be changed. For example, the construct to denote a class assertion `is an instance of` was

[1]The Ordnance Survey named the Rabbit language after Rabbit in Winnie the Pooh, who was actually cleverer than Owl.

[2]The tables show most of the Rabbit constructs, but not all. For a complete overview of the language see `http://sourceforge.net/apps/mediawiki/confluence/index.php?title=Rabbit_Language_Overview`

Table 3.1: Example Rabbit sentences for describing concepts. Source: the Ordnance Survey hydrology ontology [110].

Description	Rabbit	Manchester Syntax

Concept Declaration	`Stream is a concept.`	`Class: Stream`
		`SubClassOf:`
		`owl:Thing`

Subjunction	`Every Cataract is a kind of Waterfall.`	`Class: Cataract`
		`SubClassOf:`
		`Waterfall`

Defined Class	`A Source is anything that:`
	`is a kind of Spring or Wetland;`
	`feeds a River or a Stream.`
		`Class Source:`
		`EquivalentTo:`
		`Spring or Wetland`
		`and feeds some (River or Stream)`

Existential Quantifier	`Every River flows into a Sea.`	`Class: River`
		`SubClassOf:`
		`flowsInto Some Sea`

Negation	`No Backwater has a Current.`	`Class Backwater:`
		`DisjointWith:`
		`have some Current`

Union	`Every Bifurcation is part of`
	`one or more of River or Stream.`
		`Class: Bifurcation`
		`SubClassOf:`
		`bePartOf some (River or Stream)`

Minimal Cardinality Restriction	`Every Confluence flows from at least`
	`two River Stretches or Streams.`
		`Class: Confluence`
		`SubClassOf:`
		`(flowFrom min 2 (RiverStretch`
		`or Stream))`

Qualified Cardinality Restriction	`Every Channel has exactly 2 Banks.`	`Class Channel:`
		`SubClassOf:`
		`hasBank exactly 2 Bank`

Universal and Existential Quantifiers	`Every River only flows into a Sea.`	`Class River:`
		`SubClassOf:`
		`flowsInto some Sea`
		`and flowsInto only Sea`

Built-in Properties for Anonymous Classes	`Every Reservoir has purpose`
	`Storage of Water.`
		`Class: Reservoir`
		`SubClassOf:`
		`havePurpose some (Storage`
		`and (Rabbit:of some Water))`

Complex Objects	`Every Spout gushes Water that`
	`flows from the Ground.`
		`Class: Spout`
		`SubClassOf:`
		`gush some (Water)`
		`and (flowFrom some Ground))`

Disjointness	`Canal and Disused Canal are`
	`mutually exclusive.`
		`Class: Canal`
		`DisjointWith:`
		`DisusedCanal`

Modality	`A Broad usually is located in East Anglia.`	no translation
Homonym Disambiguation	`Dam (Water) is a concept.` `Dam (Structure) is a concept.`	`Class: Dam_Water` `Class: Dam_Structure`

Table 3.2: Example Rabbit sentences for describing instances. Source: the Ordnance Survey hydrology ontology [110].

Description	Rabbit	Manchester Syntax
Instance Declaration	`Somerset is a County.`	`Individual somerset: Types: County`
Same Instance	`UK and United Kingdom are the same thing.`	`Individual: uk SameAs: unitedKingdom`
Different Instance	`River Wharfe and River Aire are different.`	`Individual: riverWharfe DifferentFrom: riverAire`

Table 3.3: Example Rabbit sentences for describing relationships. Source: the Ordnance Survey hydrology ontology [110].

Description	Rabbit	Manchester Syntax
Relationship Declaration	`enables is a relationship.`	`ObjectProperty: enable`
SubProperty	`The relationship flows in is a special type of the relationship is connected to.`	`ObjectProperty: flowIn SubPropertyOf: beConnectTo`
Inverse Relationship	`The relationship is fed by is the inverse of feeds.`	`ObjectProperty: beFeedBy InverseOf: feed`
Range Restriction	`The relationship is capital city of must have the object Country.`	`ObjectProperty: beCapitalCityOf Range: Country`
Functional Property	`The relationship is capital city of can only have one object.`	`ObjectProperty: beCapitalCityOf Characteristics: Functional`

replaced by `is a`. A second evaluation compared the user comprehension of Rabbit sentences versus Manchester Syntax statements [60]. This study *confirmed that Rabbit is easier to understand than Manchester Syntax.*

Although users seem to correctly understand the meaning individual Rabbit sentences, further studies by Ordnance Survey have shown that this understanding does not automatically translate into ability to author ontologies[41, 60]. In these studies, participants were given an introduction to the language, a crib sheet with common Rabbit sentences and a natural language text that had to be translated into Rabbit. The studies found that, on average, users only captured 60% of the information into Rabbit. Furthermore, 50% of the written Rabbit sen-

tences, contained syntactic errors. These results clearly suggest that tool support is required for helping users to formulate correct Rabbit sentences.

Finally, Rabbit has been compared to other controlled languages (ACE [77] and Sydney OWL Syntax [29]) that target OWL in [123]. This comparison showed seemingly minor differences between the languages at the lexical and syntactic level. One of the conclusions is that, for all 3 languages, support is required to formulate correct sentences and to help users understand the logical meaning of some constructs.

Relation to this PhD The Ordnance Survey's methodology for ontology authoring has been of great influence for this PhD. Note that the methodology and the design of the Rabbit language are **not** contributions made by this PhD. However, the specification of a methodology (and its CNL) are not directly useful to domain experts without training or direct support by an existing Rabbit expert. For this reason, the Ordnance Survey wanted to provide tool support for both the methodology and the Rabbit language. This corresponded to our view that there was a need for intelligent tool support for domain experts and led to our choice of Rabbit as the CNL to base our work on.

The following sections describe the first original contributions of this PhD: (i) a description of how we used the Rabbit language specification to provide tool support for domain experts and (ii) an evaluation of the proposed tool support. In the following sections, we describe the implementation of a parser based on the Rabbit specification and the role of that parser in providing a user interface that is easy to use for domain experts.

3.2 ROO: Rabbit to OWL Ontology Authoring Tool

ROO is an ontology creation tool based on Protégé 4 that assists domain experts to build conceptual ontologies. ROO uses Rabbit to enable domain experts to automatically formalise their knowledge in OWL. ROO provides easy to understand suggestions and task-specific messages to help the user enter correct CNL

constructs. Appropriate feedback is given to help users recognise concepts, relationships and individuals when writing CNL sentences. Syntax highlighting based on the parsed structure helps the user recognise CNL patterns.

Although ROO is based on technologies such as OWL and natural language processing, ROO avoids exposing technical terminology to ontology authors. ROO prefers to use conceptual terminology that may not be well-defined in a technical sense, but which is easier to understand for novice users. In the case of OWL, ROO will avoid introducing terminology such as *Object Property*, opting to use *relationship* instead. In the case of the natural language processing, ROO avoids using linguistic terminology such as *determiner* or *adjective* as much as possible. When technical terminology is introduced, ROO tries to give specific examples, preferably coming from the domain (i.e from the ontology itself).

ROO helps users to avoid introducing ambiguity in the resulting ontology. The CNL tool avoids making assumptions by requiring the declaration of concepts, relationships and individuals. ROO is aware of cases when parsed sentences could be ambiguous and warns the user accordingly, preferably suggesting ways to avoid ambiguity.

ROO provides guidance about how to build ontologies that are easy to reuse. ROO incorporates a model of Kanga [93], the Ordnance Survey's ontology engineering methodology, and can make suggestions to the user regarding tasks that need to be performed to improve the reusability and general quality of the ontology.

ROO provides easy access to documentation to explain the user interface, the controlled natural language and the ontology creation process.

3.2.1 Architectural Overview

Fig. 3.1 shows the main architectural elements of ROO, and how these elements interact with each other. Here, we introduce each architectural element.

The user of ROO interacts with the *ROO User Interface*, which is a collection of components that enable domain experts to create and edit ontologies by writing Rabbit sentences. The user interface uses services provided by the Rabbit Language Processor (to parse sentences), the ROO Model Manager (to construct the

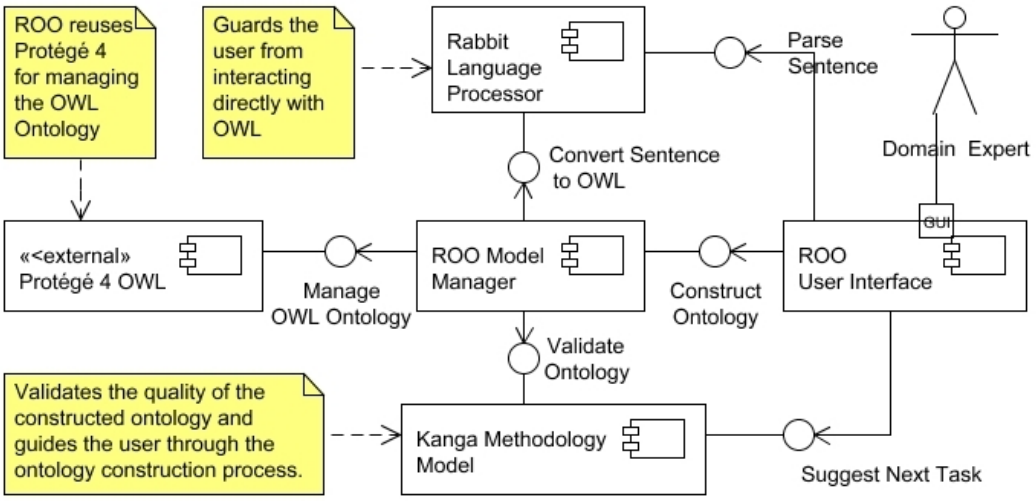

Figure 3.1: UML 2.0 functional architectural view of ROO shows the architectural elements, interfaces and inter-element connections.

ontology) and the Kanga Methodology Model (to suggest ontology construction tasks to the user).

The *Rabbit Language Processor* consists of a Rabbit parser – that parses text into an unambiguous intermediate representation– and a Rabbit to OWL converter – that converts the unambiguous intermediate representation into OWL. The ROO Model Manager and the ROO User Interface use the services provided by the Rabbit Language Processor.

The *ROO Model Manager* acts as a central point between the underlying OWL ontology and the other architectural elements. It provides methods to construct the ontology by accepting Rabbit sentences and uses the Kanga Methodology Model to validate that the ontology complies with the rules set by Kanga Methodology. The ROO Model Manager is implemented as a thin layer on top of Protégé 4 OWL, which already provides services to manage OWL ontologies.

The *Kanga Methodology Model* provides services to check whether a given OWL ontology contains annotations and entities that would be expected if the ontology was built using the Kanga Methodology for ontology construction. If the ontology is missing annotations or entities, the Kanga Methodology Model can suggest ontology construction tasks from the Kanga Methodology.

In the next sections we describe the components of ROO in more detail, in particular, we describe how the various components can be used to provide intelligent tool support for domain experts.

3.3 Providing Domain Expert-Specific Tool Support for Rabbit

This section gives an overview of how the Rabbit parser and OWL generator are implemented (section 3.3.1), gives a detailed description of how ambiguity is used by the Rabbit parser to provide extra support for domain experts (section 3.3.2) and explains how the parser implementation is used to provide a user interface that domain experts can learn quickly (section 3.3.3).

3.3.1 RabbitParser

The Rabbit parser consumes a document containing Rabbit sentences and produces a parse tree (an intermediate representation of the document). If the parsed document contains sentences which are not valid Rabbit sentences, the parser marks these sentences and attaches error messages that should help the users to correct the sentence.

The Rabbit parser consists of a pipeline of linguistic Processing Resources as pictured in Fig. 3.2. The implementation of the parser included in ROO follows the CLOnE [47] approach and is based on the GATE[1] text processing environment. Fig. 3.2 provides an overview of the pipe-line (the current pipeline consists or around 30 Processing Resources) and shows the three main phases of the parsing process:

First, there is a pre-processing phase where we use GATE — more specifically ANNIE[30, Chapter 8] — to perform the basic natural language pre-processing such as tokenization, sentence splitting, Part of Speech (POS) and morphological tagging.

[1] http://gate.ac.uk

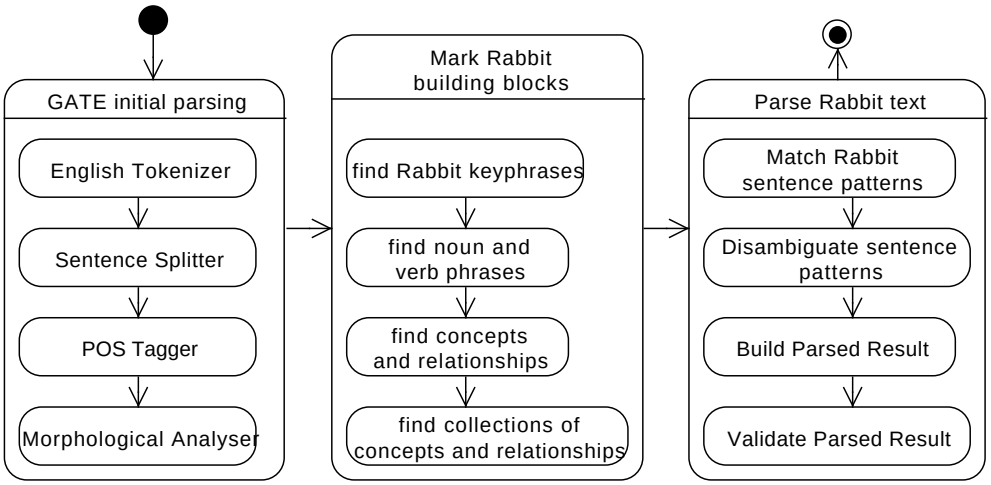

Figure 3.2: Pipeline of Processing Resources for parsing Rabbit constructs. The input is a file containing Rabbit sentences and the output is a parse tree that gives access to the information found by the Processing Resources (i.e. the POS of each word, whether a phrase is a Rabbit keyphrase, or a Rabbit concept, etc.)

The second major phase in parsing Rabbit includes using a gazetteer to find Rabbit key phrases and using JAPE[1] transducers to find and process Rabbit constructs based on the annotations gathered during the pre-processing phase. Roughly speaking, there is one JAPE transducer per Backus Naur Form (BNF) rule in the Rabbit grammar as shown in table 3.4, although in some cases, our implementation deviates from the Rabbit BNF in order to achieve a better parsing efficiency or in order to control the amount of ambiguity the user can introduce (see details in Sect. 3.3.2).[2]

The final phase during the parsing is the construction of an intermediate representation of the parsed text. The Rabbit parser is implemented in Java and defines a hierarchy of Java interfaces for representing Rabbit constructs. The base Java interface for Rabbit constructs is an `IParsedPart`, with example in-

[1] Java Annotation Pattern Engine [30, Chapter 7] provides a language for finding annotation patterns during the parsing process. It also provides hooks for invoking Java methods during the parsing of a text.

[2] See Appendix A for links to the full Rabbit BNF and the JAPE implementation.

heriting interfaces `IParsedConcept` and `IParsedKeyphrase`. This intermediate Java representation of the Rabbit input text can be used to implement validations which would be difficult to capture using only the pattern matching capabilities of JAPE. For example, we can detect whether concepts in a sentence have been defined in the ontology and warn the user if this is not the case.

One of the advantages of using the pipeline shown in Figure 3.2 is that it enables processing of the input text with an *open lexicon*. The ANNIE components contain generic rules to determine the part-of-speech and morphology of words but there is no need to have a pre-defined set of words. In comparison, other CNLs, such as ACE, work better if a *closed lexicon* – a list of terms relevant to the domain – has been provided in advance. Use of a closed lexicon is not appropriate for ontology authoring, where users will be introducing new vocabulary. With the open lexicon approach, new terms are added to the ontology and the ontology is used as an extension to the initial NLP rules provided by ANNIE to verify that a term has been introduced in the ontology.

Table 3.4: Example of using JAPE to implement the Rabbit BNF

BNF	JAPE
`<universal subject> ::= Every <concept>`	`Rule: UniversalSubjectPattern ( ({Lookup.minorType == "RABBIT-Every"}): kp1 ({RabbitConcept}):concept ):RabbitSubject`
`<object> ::= [<object prefix>] <concept> [<preposition modifier>]`	`Rule: ObjectPattern( ({ObjectPrefix})? : objectPrefix ({RabbitConcept}): concept ({PrepositionModifier})?: prepositionModifier`

Generating OWL

ROO automatically translates Rabbit sentences into OWL. This task is performed by a program that uses the OWL API[1] (also used by Protégé) to convert the output of the Rabbit parser into OWL. This translation is straightforward because the Rabbit language follows the OWL language closely as shown in tables 3.1, 3.3 and 3.2. These tables only show the minimum of axioms generated to express the OWL equivalent, in practice the generator also includes annotations such as

[1] `http://sourceforge.net/projects/owlapi/`

`rdf:label`s and `rabbit:sentence` which make the resulting ontology easier to read (using ROO or Protégé).

Although the current implementation of the OWL generator uses the OWL API, the mapping from the parsed tree to the corresponding statements in OWL has been isolated by the API. Hence, creating an OWL generator using an alternative API, such as Jena, should be a straightforward task.

Mapping Rabbit Entities to OWL Entities

A crucial step during the parsing and validation of Rabbit sentences is to link Rabbit Entities (`IParsedConcept`, `IParsedRelation` and `IParsedInstance`) to OWL Entities (OWL Classes, OWL Object Properties and OWL Individuals). Similarly to CLOnE [47], we have implemented a canonicalisation procedure to create a key identifier for the entities. This canonicalisation uses the root morpheme of the words comprising a Rabbit entity to create an OWL id. The third sentence in Table 3.3 illustrates the results of this canonicalisation process as the Rabbit relationship `is fed by` has been mapped to the canonical name `beFeedBy`. Although `beFeedBy` is not easy to read, the OWL generator also adds the `rdfs:label "is fed by"` annotation (not shown in table 3.3, which is used by most applications instead of the OWL id.

A possible disadvantage of the canonicalisation process is that it allows Rabbit sentences which are not correct English. For example, the sentence `Every Confluences flows from at least two Rivers Stretch or Stream`, is a valid Rabbit sentence due to canonicalisation even though (i) `Confluences` is plural, which is incongruent with `Every` and the verb `flows from`) and (ii) `Rivers Stretch` and `Stream` are singular, but should be plural to match `at least two`. We could add more rules to avoid such cases, but we leave that as future work. For this PhD, this will not be a problem for authors, since we assume they will tend to write correct English sentences. The described issue can be a problem for generating Rabbit sentences, as Rabbit sentences may be generated which are not correct English.

The advantage of generating class identifiers based on the root morpheme is that the natural language processing engine (GATE, in this case) takes care of

morphology variations, making it easier for domain experts to write sentences in the way they would when writing natural language. For example, users write `Every Confluence flows from at least two River Stretches or Streams`, but only need to declare that `River Stretch is a concept` and `Stream is a concept`, without needing to specify the plural forms. By using the morphological annotations added by ANNIE during the pre-processing stage of Rabbit parsing (see Sect. 3.3.1), the parser maps `River Stretches` and `River Stretch` to the same OWL class.

3.3.2 Handling Ambiguity in Rabbit sentences

Most constructs in Rabbit contain key phrases which makes the constructs unambiguous. For example: `River Wharfe and River Aire are different`, where we have underlined the key phrases. The parser can search for text that matches the pattern `<individual> and <individual> are different`[1].

In order to permit users to give natural names to concepts, relationships and instances, Rabbit allows these constructs to consist of multiple words. Examples of concepts that should be valid are: `Natural Body of Water`, `Water for Irrigation`, `Fire and Rescue Services` and `Formula 1`. At the same time, the following relationships are also valid: `has pet`, `flows into`, `contains`, `contains water for`, `Services` (as a relationship meaning "to provide services to") and `Rescues`. This freedom to name Rabbit entities allows users to construct more natural sentences and to use concepts and relations at an appropriate granularity level for their domain. At the same time, this freedom has the potential to introduce ambiguities. The Rabbit parser and ROO provide support to avoid introducing ambiguity in the context of the ontology being built.

Table 3.5 shows an example of how a Rabbit sentence can be interpreted in different ways depending on its context. The context of a Rabbit sentence is defined by the set of Rabbit sentences that have been entered for the ontology. Context 1 in table 3.5 is the empty context. In this case the parser chooses an interpretation where it expects `Water for Irrigation` to be a concept and `contains` a relationship. However, the parser also has recognised that `Water for`

[1] An `<individual>` can be any number of tokens (excluding keyphrases).

Table 3.5: Handling translational ambiguity in Rabbit. The table shows how the sentence `Every Irrigation Canal contains Water for Irrigation.` is interpreted depending on the context of Rabbit sentences. Note that input sentence is not ambiguous in the traditional sense; but that there are multiple ways to translate the sentence into OWL. This is important since these different OWL representations are not guaranteed to be logically equivalent.

ID	Context Sentences	Manchester Syntax Translation or Error Message when translating sentence `Every Irrigation Canal contains Water for Irrigation.`
1	no context	2 errors: missing concepts `Irrigation Canal` and `Water for Irrigation` 1 error: missing relationship `contains` 1 error: alternative interpretation for `Water for Irrigation'` 1 error: alternative interpretation for `'contains' 'Water for Irrigation'`

2	`Irrigation Canal is a concept.`
	`contains is a relationship.`
	`Water for Irrigation is a concept.`
		`Class: IrrigationCanal`
		`SubClassOf:`
		`contain some WaterForIrrigation`

3	`Irrigation Canal is a concept.`
	`contains is a relationship.`
	`Water is a concept.`
	`Irrigation is a concept.`
		`Class: IrrigationCanal`
		`SubClassOf:`
		`contain some (Water`
		`and (Rabbit:for some Irrigation)`

4	`Irrigation Canal is a concept.`
	`contains water for is a relationship.`
	`Irrigation is a concept.`
		`Class: IrrigationCanal`
		`SubClassOf:`
		`containWaterFor some Irrigation`

5	`Irrigation Canal is a concept.`
	`contains is a relationship.`
	`Water is a concept.`
	`Irrigation is a concept.`
	`Water for Irrigation is a concept.`
		1 error: `'Water for Irrigation'` has an equally possible alternative interpretation `'Water' for 'Irrigation'`

`Irrigation` could interpreted differently as being composed of concepts `Water` and `Irrigation` and linked by the keyphrase `for`. This shows that the parser keeps alternative interpretations in the parse tree and uses these to warn and prevent the user from introducing ambiguity.

The different contexts in table 3.5 illustrate how the parser chooses the correct interpretation based on the context. For example, in order to decide whether `Water for Irrigation` is a single concept or a Rabbit *compound object* (two concepts linked by the built-in relationship `for`) the parse tree validation uses a disambiguation heuristic that ranks each parsing option based on the enti-

ties declared in the context. So, if the user has declared concepts `Water` and `Irrigation`, but not `Water for Irrigation` (as in context 3 in table 3.5), then the disambiguation heuristic will conclude that `Water for Irrigation` is more likely to be a compound object and not a single concept. Alternatively, if the user has declared concept `Water for Irrigation`, but not the concepts `Water` or `Irrigation`, then the disambiguation algorithm will rank the interpretation of `Water for Irrigation` as a single concept higher than as a compound object (see context 2 in table 3.5).

The disambiguation heuristic illustrated above is able to identify potentially ambiguous constructs and benefits the domain experts by giving them more freedom to use their own terminology when describing their domain. However, if the number of alternative interpretations is very large, the heuristic will find too many potentially ambiguous constructs which can have an adverse result: the users will not be able to introduce new terminology due to potential ambiguities. In order to restrict the number of potential ambiguous Rabbit constructs, we:

1. require users to explicitly introduce entities by using the three Rabbit entity declaration sentences;
2. only allow specific constructs in the Rabbit language to be ambiguous and
3. impose linguistic restrictions on what are valid Rabbit concepts and relationships.

We explain these three aspects next.

Rabbit Entity Declaration Sentences

The Rabbit parser is able to recognise potential entities (which we call concept candidates, relationship candidates and instance candidates) when parsing sentences. For example, given sentence `Every River flows into a Sea.`, the parser will recognise that `River` and `Sea` are concept candidates and `flows into` is a relationship candidate. However, this sentence on its own is not a valid Rabbit sentence because the user has not explicitly introduced the required concepts and relationships. The parser will currently show error messages stating that `concept`

`'River' has not been introduced in the ontology yet.` The Rabbit language provides entity declaration sentences to introduce concepts (`<concept candidate> is a concept.`), relationships (`<relationship candidate> is a relationship.`) and instances (`<instance candidate> is a <concept>.`). The disadvantage of using entity declaration sentences is that users needs to write more sentences at the beginning of the ontology construction process. The advantage is that the parser can then use the set of introduced entities to disambiguate sentence constructs. Furthermore, having an explicit declaration sentence forces users to consider whether an entity that is introduced is essential for describing the domain.

Ambiguous Rabbit Constructs

The Rabbit parser only allows two grammatical constructs [1] to be ambiguous: `Object`s and `Relationship Phrase`s. Rabbit `Object`s are translated as (anonymous) classes in OWL. Example `Object`s are: `River`, `Body of Water`, `Irrigation of Water` and `Building that has purpose Education`. `Object`s can be ambiguous because concept names allow prepositions (see Table 3.6 in Section 3.3.2), but Rabbit also provides built-in prepositions to combine two concepts into an `Object`. For example: `Irrigation of Water`, can be an `Object` containing a single concept (introduced by sentence `Irrigation of Water is a concept.`), or it can be an `Object` that relates concepts `Irrigation` and `Water` using the built-in Rabbit relationship `of`.

The second potentially ambiguous construct in Rabbit is the `Relationship Phrase`, which translates as an anonymous class in OWL by combining a relationship and an `Object`. Example `Relationship Phrase`s are: `has purpose Education` or `contains water for Irrigation`. Ambiguity is possible in this case due to the freedom Rabbit gives users to define concepts and relationships. For example, the user has the freedom to define relationships `has purpose` or `has` as well as concepts `Education` or `Purpose Education`. The set of defined entities affects how the `has purpose Education` is interpreted.

[1] Note that these are constructs used only in the intermediate representation and should not be confused with other types of objects from logic or object-oriented approaches.

Linguistic Restrictions on Rabbit Entities

The main intuition in linguistically restricting Rabbit concepts and relationships is that concepts tend to be nouns (`River`) or noun phrases (`Body of Water`); and relationships tend to be verbs (`contains`) or verb phrases (`flows into`). The parser implementation defines JAPE rules that find noun and verb phrases which are then interpreted as possible concepts or relationships in the Rabbit sentence. Table 3.6 shows the main rules with examples of valid concept and relationships. The rules use high-level linguistic definitions, which are themselves defined as rules that use the Part of Speech tag. For example, `Adverb` is any token that has POS `RB` or `VBN` (past participle is interpreted as an adverb to handle concepts such as `Written Article`). See the GATE User Manual [30, Appendix E] for the codes used by the POS tagger. [1]

Table 3.6: Main JAPE rules for detecting noun and verb phrases in the Rabbit parser implementation.

JAPE rule	Example

`Rule: NounPhrase(`
`  (Adverb)* (Adjective)*`
`  (`
`  ((Noun) | (SpecialToken))*`
`  (Noun) (SpecialToken)*`
`  )`
`): RabbitNounPhrase`
	`Extremely Large River`
	`Peer 2 Peer Network`
	`Boeing 737-300`
	`Fire & Rescue Services`
	`Written Article`

`Rule: CompoundNounPhrase (`
`  (Noun) (Prep) (Noun)`
`):RabbitNounPhrase`
	`Body Of Water`
	`School Of Computing`
	`North By Northwest`

`Rule: VerbPhrase (`
`  (Verb)+ (Prep)?`
`  ({RabbitNounPhrase})?`
`  (Prep)?`
`):RabbitVerbPhrase`
	`has`
	`has purpose`
	`produces sound by`
	`runs into sea at`
	`is close to`

`Rule: ComparativeRelation (`
`  (CopulaVerb)`
`  (ComparativeAdjective)`
`  ({Token.root == "than"})`
`):RabbitVerbPhrase`
	`is larger than`
	`is older than`

[1] Appendix A has a link to the full JAPE implementation. The jape files `findNounPhrase` and `findVerbPhrase` contain the relevant rules for finding potential entities.

Rabbit Syntactic Disambiguation Mechanisms

Besides the three mechanisms to restrict the potential for ambiguity in Rabbit sentences, the language itself also provides two built-in syntactic mechanisms to explicitly indicate ambiguity. First, concept names can contain a disambiguation component. For example, the Ordnance Survey Hydrology Ontology defines two different concepts that have the same name `Dam`. A `Dam` refer to both the physical construction that holds water, or it can refer to the water itself. In order to separate the two concepts, while still using the `Dam` name, Rabbit allows users to introduce the concepts as `Dam (Water)` and `Dam (Structure)`. The user has to use these names for all the sentences describing the concepts (e.g. `Every Dam (Water) is a kind of Pool (in River).` and `Every Dam (Water) is located behind a Dam (Structure).`).

The second syntactic disambiguation mechanism relies on the capability of importing and reusing external ontologies. Rabbit allows users to import an ontology by giving that ontology a `label`. For example `Use ontology: [Hydrology] from http://example.com/Hydrology.rbt`, where `[Hydrology]` is the label. In that case, the user can refer to concepts defined in the imported ontology as `Bank [Hydrology]`. which can be used to disambiguate a concept with the same name but from a different semantic definition (e.g. `Bank [Finance]`).

Summary This section discussed a feature of the Rabbit parser, which allows potential ambiguity of entity names in order to make it easier for domain experts to write correct Rabbit sentences. In particular, we illustrated a heuristic to perform unambiguous entity-mapping based on the context of the ontology being built. We also discussed a number of aspects of the Rabbit parser and the Rabbit language which restrict potential ambiguity or allow users to explicitly avoid ambiguous entity names. Finally, we stress that these choices for the Rabbit parser allow only ambiguity regarding entity mapping: at the level of Rabbit sentences, the choice of OWL axiom type for a Rabbit sentence is always unambiguous.

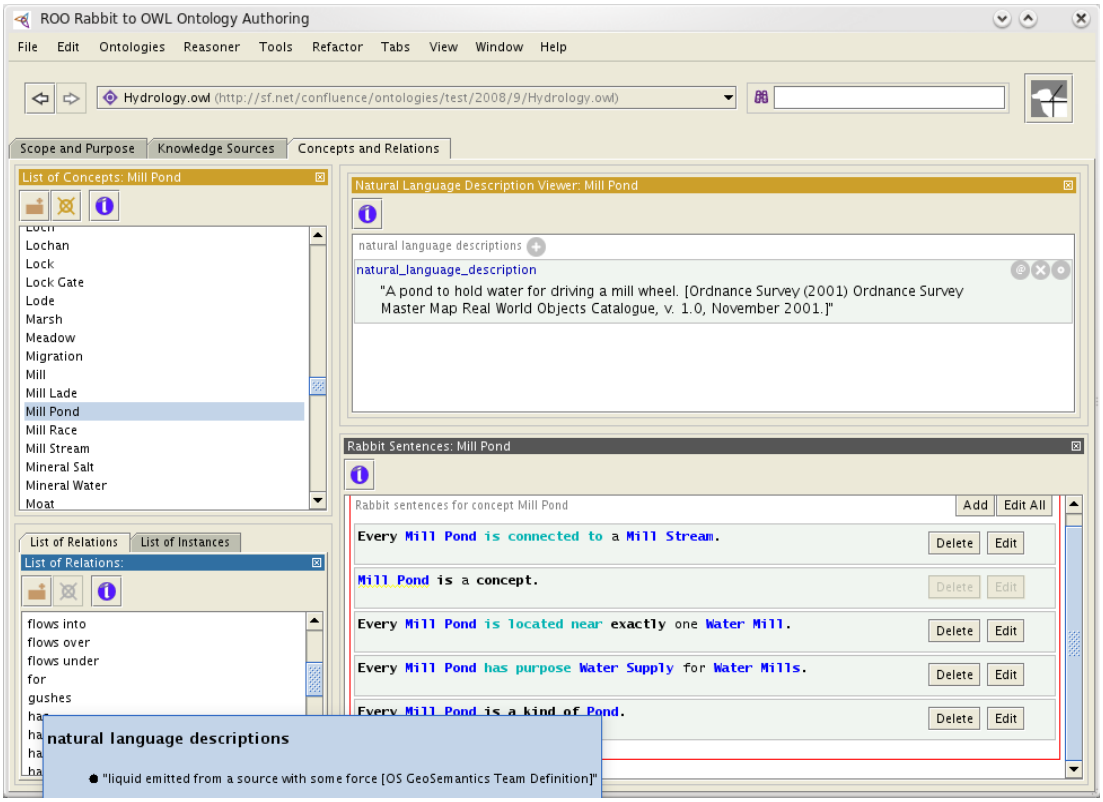

Figure 3.3: Graphical user interface of ROO. ROO provides custom tabs and components to build ontologies that use terminology that is easy to learn by domain experts.

3.3.3 Support for Editing Rabbit

All features described in the previous sections to make it easier for domain experts to build ontologies are made available through a customised user interface. The user interface is built on top of Protégé 4, but provides different tabs and components that use the Rabbit language instead of Manchester Syntax and class hierarchies. Fig. 3.3 shows the user interface of ROO. In this section we present some of the features that ROO provides to make it easier for domain experts to create ontologies, focusing on those features that are related to the editing of Rabbit sentences.

Every component in the user interface provides a button that opens a browser

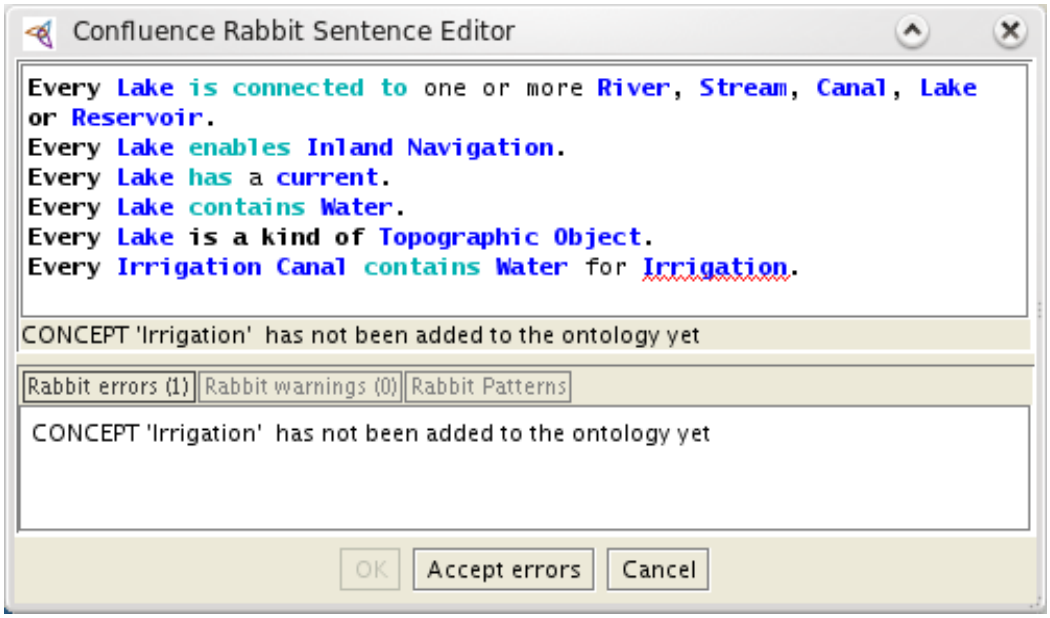

Figure 3.4: Rabbit Editor component showing syntax highlighting and error feedback.

and shows specific help about that user interface component and how it can be used to create ontologies. The documentation also includes an introduction to the Kanga methodology and the Rabbit language. The Rabbit language is presented in different ways to users who are at different stages of the Rabbit learning curve. As an introduction to the language, the keywords in Rabbit are explained, which includes descriptions of how to use keywords in sentences and example sentences. The documentation also provides a reference documentation for the Rabbit language, which presents sentences that can be used to accomplish ontology construction tasks such as *introduce a concept*, or *describe a relationship specialisation*. We also provide an introduction to Rabbit for people who already know Manchester Syntax and a set of example ontologies to help users get an idea of how to use Rabbit and ROO.

Once users have received an introduction to Rabbit, they can add knowledge to the ontology by writing Rabbit sentences. The user interface provides a Rabbit editor which provides syntax highlighting that uses the intermediate representation provided by the Rabbit parser. The syntax highlighter uses different colours that

help the user see which parts of the sentence are recognised as concepts, relationships, instances, labels of imported ontologies and Rabbit keywords.

The Rabbit editor also gives feedback about syntax errors and warnings. This is done in three different ways: (i) as a list under the editor; (ii) by showing squiggly lines under the word (or words) causing the error (in red) or warning (in yellow) and (iii) by showing the error message as a tooltip when the user hovers over the sentence with the mouse. Fig. 3.4 show an screenshot of the Rabbit editor that shows syntax highlighting and error feedback.

Finally, the editor also has a list of sentence structures to remind the user of the correct Rabbit syntax. This list is used in combination with the help documentation described above.

3.3.4 Support for Following Ontology Engineering Methodology

ROO *guides the domain expert by following an appropriate ontology construction methodology.* This is achieved by providing a user interface that reflects the phases of the Kanga methodology where each tab corresponds to a phase in the methodology. For example, when a user creates a new ontology or opens an existing ontology, the first tab is for `Purpose and Scope`. The interface components encourage entering an annotation for the ontology purpose and a different annotation for the ontology scope. The annotation URIs have already been defined for the user as they are not expected to learn the OWL annotation system. This has the advantage that the encoding of the purpose is standardised by the tool, so it is easy to check whether an ontology has defined its purpose and scope. This is used by ROO's *Guide Dog*, a component which contains an internal model of the Kanga methodology to check the progress of the user in building the ontology. The user can ask the guide dog for advice regarding building the ontology. When this happens, the guide dog inspects the state of the ontology to determine the current phase in the Kanga methodology and suggests a task to the user that is appropriate for the current phase. The following types of tasks are suggested: scope and purpose definition, knowledge source definition, declaration of

concepts and relationships (OWL entities), free text definition of OWL entities, CNL definition of OWL entities.

The Guide Dog functionality in ROO is driven by a rule-based task planner implemented in JBoss Drools.[1] The task planner defines 11 rules for determining when a set of ontology construction tasks should be carried out according to the Kanga methodology. Each rule has the form `when LHS then RHS`. Where LHS is a condition expressed in terms of the ontology being constructed. Typical conditions are: whether the ontology contains a specific annotation(e.g. `scope` annotation); whether the ontology defines more than a specific number of OWL classes or whether an OWL entity in the ontology contains a specific annotation(e.g. `related_rabbit_sentence` annotation). These LHS conditions are checked using the OWLAPI to inspect the Ontology, its axioms and its annotations. The RHS is only triggered when the LHS condition is met. The RHS adds a task to a list of tasks that the Guide Dog will suggest to the user. Appendix A contains a link to the full list of rules. As an example, the prescription made by Kanga that: "Users should enter a natural language description for each concept in the glossary" is encoded in the following rule:

```
rule ‘‘Enter free-text definition for
       Entity X’’
when
  IOntologyWrapper(
    hasScope == true,
    hasPurpose == true,
    numberOfKnowledgeSources > 0)
  ew : IOWLEntityWrapper(
    hasFreeTextDef == false,
    numOfSent:numberOfRabbitSentences)
then
  ntc.add(NextTaskSuggestionType.
     EnterFreeTextDefinitionForAEntity,
     ew);
end
```

[1] http://jboss.org/drools/

When a user invokes the Guide Dog, all the rules are triggered populating the Guide Dog with a list of tasks that can be presented to the user. The first suggestions for an empty ontology related to entering the scope, purpose, knowledge sources and initial entities. Once these have been followed, the main suggestions relate to entering the free-text definition or Rabbit sentences for particular entities.

When the ontology becomes larger, multiple entities may be missing free-text or Rabbit definitions resulting in a large number of suggestions by the Guide Dog. In order to keep the focus of the interaction, we reuse the *selection model* defined by Protégé 4, which keeps track of which entity is currently selected in the user interface. The Guide Dog sorts the tasks to give priority to tasks that are related to the currently selected OWL Entity. Thus, if the user is viewing the concept `Lake`, the Guide Dog will first suggest to add a Rabbit sentence for this concept, instead of suggesting the same for some other, potentially unrelated, concept.

Summary of Tool Support in ROO This section showed the main features in ROO that provide support to domain experts:

- The Rabbit Parser
  - analyses users' inputs to validate the correct use of the Rabbit language. The parser aims to provide easy-to-understand messages to help users produce correct sentences.
  - automatically converts correct Rabbit sentences into OWL.
  - uses NLP and disambiguation techniques to enable, in a controlled manner, the use of natural terminology for domain experts. The used techniques allow the introduction and detection of potentially ambiguous entity names and Rabbit constructs.
- The user interface reflects the steps of the Kanga methodology, guiding the domain experts through the ontology engineering process.
- The Rabbit editor provides syntax highlighting, error feedback and access to documentation of the Rabbit language.

- The Guide Dog provides domain experts with help for deciding what to do next, based on the Kanga methodology.

3.4 Evaluation

This section presents a *comparative user study* we performed to study how well our approach works. The main goal of the evaluation was to assess the effectiveness of ROO, while following criteria for evaluating ontology tools [61]. The study addressed three groups of questions:

1. What is the interaction with the tool like? How usable is the tool? Can domain experts without knowledge engineering skills create OWL ontologies with ROO?

2. How well does ROO facilitate the ontology construction process? Do users develop ontology modelling skills as a result of the assistance the tool provides?

3. What is the quality of the resultant ontologies produced with ROO? Is the quality influenced by assistance provided by the tool?

In this section we discuss some preliminary usability studies we conducted, the experimental design and the experimental results. Based on the results, we will draw conclusions about the benefits of the tool support provided by ROO to assist domain experts' involvement in ontology authoring. We will also outline open issues and point at further development.

3.4.1 Preliminary User Studies

Before conducting the study, we first conducted three preliminary *usability studies* with 3-4 users. These studies were conducted during the development of ROO while new features were still being added. Music was chosen as the domain due to the availability of users with subject knowledge. Users were asked to build an ontology of musical instruments, corresponding to the material in the

UK A-level[1]. All participants had studied this specialised level and had played different musical instruments. The users did not have experience and knowledge in ontology engineering, their computer background varied from programming to general computing literacy skills. The sessions were recorded using `CamStudio`[2], and a member of the ROO team would always be present at the first session for a user to observe how newcomers would start using the tool. This enabled us to tune the interface and, most importantly, to polish the user guidance and the support provided with the CNL interface. It also allowed us to elicit interaction patterns, which were further examined in follow up studies.

3.4.2 Experimental Design

The study followed a *task-based*, *between-subjects* experimental methodology to compare ROO with a baseline system.

Baseline System. The study compares ROO with a similar tool that allows the user to author in a CNL. From the available CNL tools for ontology authoring, ACEView [77] for Protégé was chosen because the user interaction with it is the closest to the user interaction with ROO: both tools extend Protégé as plug-ins, support text input in a CNL compatible with OWL-DL, provide error messages for sentence composition, and produce an ontology in OWL[3]. The main difference between ROO and ACEView is that ROO offers assistance with the whole ontology authoring process.[4]

Participants. The study involved 16 volunteers from the departments of Geography (8 students) and Earth and Environment (8 students) at the University of Leeds. The participants were chosen to closely resemble domain experts who may perform ontology modelling tasks at Ordnance Survey (Hydrology) or the

[1] A specialisation which can be chosen in UK upper secondary schools.

[2] See `http://camstudio.org/`

[3] The other available CL ontology authoring tools are CLONE and PENG. They were used during a pilot but discarded for the actual study. CLONE is more suitable for users with some knowledge engineering skills, while the users in our study did not have such skills. The interaction with PENG is pattern-based and is notably different from the ROO interface.

[4] ACE, the CNL used in ACE View was introduced in Section 2.2.

UK Environment Agency (Flooding and Water Pollution). The main requirement for attending the study was to have knowledge and experience (confirmed with the modules attended and practical work done) in Hydrology, for Geography students, and Flooding and Water Pollution, for Environmental Studies students. In each domain, 4 participants used ACEView and 4 used ROO; this was assigned on a random basis. None of the participants was familiar with ontologies or ontology construction tools. They had not heard of RDF or OWL. None had previous background in encoding knowledge and for most participants structuring knowledge meant writing reports/essays in a structured way.

Scenarios. The study involved two ontology authoring scenarios.

Scenario 1 (Geography participants) This scenario resembles ontology modelling tasks performed by domain experts at Ordnance Survey to describe geographical features whose spatial representations are included in Ordnance Surveys OS MasterMap®[1]. The participants were asked to describe several hydrology concepts: `River`, `River Stretch`, `River Bank`, `Ditch`, `Catch Drain`, `Balancing Pond`, `Canal` and `Reservoir`. These concepts are included in a large Hydrology ontology[2] defined by Ordnance Survey. The Geography participants were familiar with OS MasterMap®, which is used at the School of Geography at Leeds University.

Scenario 2 (Environmental Studies participants) This scenario resembles ontology modelling tasks performed by domain experts at one of Ordnance Surveys customers –the Environment Agency of England and Wales– who can use OS MasterMap® for flooding and water pollution analysis. The participants were asked to describe: `River`, `Catchment`, `Flood Plain`, `Ditch`, `Water Pollution`, `Sediments`, `Colloids`, `Land Use` and `Diffuse Pollution`. These concepts were selected from a list derived by

[1] OS MasterMap® `www.ordnancesurvey.co.uk/oswebsite/products/osmastermap/` is a nationally contiguous vector map containing more than 450 million individual features down to street, address and individual building level, spatial data to approximately 10cm accuracy.

[2] `www.ordnancesurvey.co.uk/ontology`

an Ordnance Survey researcher interviewing an expert from the Environment Agency as part of a project to scope a semantic data integration scenario. Many of these concepts required references to hydrology features from OS MasterMap® but the participants were unaware of this. None of the Environment subjects had knowledge of OS MasterMap®. Ontologies for geography and environment were also produced by Ordnance Survey and were used as comparators with the ontologies produced by the participants.

Procedure and Materials. [1] Depending on their background, the participants were sent the corresponding list of concepts, and were asked to prepare brief textual descriptions for these concepts by using specialised dictionaries or other sources. Each session was conducted individually and lasted 2 hours. It included several steps.

Pre-study questionnaire (20 min) included a brief introduction to the study and several questions to test the participants ontology modelling background.

Introduction and training with the ontology authoring tool (10 min) was given to each participant by an experimenter, describing the main parts of the interface and entering of several definitions from a Building and Places[2] ontology. The examples used for the ACEView and ROO sessions were similar (the differences came from the CL and the errors given by each tool). The training with ROO also required entering the ontology's scope and purpose and knowledge sources.

Interaction with the tool (60 min) The participants had to use the tool allocated to them to describe the concepts following the descriptions they had prepared. Each session was monitored by an experimenter who provided some general help when the participants got stuck with the language. Help materials with printed examples of the corresponding CL were provided. The interactions were logged and video recorded. The experimenters kept notes of the user interaction.

[1] All materials are available from `www.comp.leeds.ac.uk/confluence/study.html`

[2] `www.ordnancesurvey.co.uk/ontology`

Post-study questionnaire (20 min) included checking the participants ontology modelling background (repeating questions from the pre-study questionnaire); a usability questionnaire using seven-point Likert scale; and open questions about benefits, drawbacks, and future improvement of the tool used.

General impression and clarification (10 min) included a brief interview with each participant about their general impression of the CL used, interaction with the tool, and any additional aspects the participants wished to mention.

Data Collected. The following data was collected during the study:

- *Questionnaires.* Used for examining the usability of each tool and examining possible changes in the participants understanding of ontology modelling;
- *Log data, video records of the sessions, and experimenters notes.* Used for clarifying aspects of the interaction with each tool;
- *Resultant OWL ontologies.* The quality of these ontologies was analysed following the O2 framework [49]. The data was analysed quantitatively and qualitatively. The quantitative analysis used Mann-Whitney U test[1] for discrete measurements and t-test for interval data.

3.4.3 Results

Comparing the Interaction with ROO and ACE-View

Interaction Patterns. Both tools have fairly simple interfaces and were easy to use. The first quarter of the interaction was usually slower as the participants had to learn to formulate sentences in the corresponding CNL. During this time,

[1] Mann-Whitney U test is a nonparametric equivalent of the between-subjects t-test [126]. It determines whether the median of a variable for one group is significantly different (for 2-tail) or higher (for 1-tail) from the median of that variable for the second group.

the definition of the first concept `river` (common for both scenarios) was completed. Both tools offer a tab to show the *CNL errors*, this was used extensively. Initially, most users did not realise that the error messages refer to incorrect CL grammar that the computer could not parse or translate into a logical form, rather than incorrect domain facts. From the second quarter, the users established a routine to describe a concept, including:

1. *Check the NL description for the currently entered concept and identify a statement with knowledge to be encoded.* The ACEView users had a printout of the descriptions they had prepared, while the ROO users followed the NL descriptions the tool prompted them to enter.

2. *Look for a CL pattern that matches the NL statement.* The ACEView users used only the printed list of CL examples provided, ROO users could, in addition, see the available patterns within the tool, and they gradually moved to using this.

3. *(Re)Formulate the NL statement in a CL pattern.* This usually involved simplifying the constructs or taking away unnecessary detail, e.g. simple patterns were easily created, more complex patterns were normally not written correctly in the first instance and required several iterations and checking the system feedback.

4. *Check for error messages*; if there are no error messages, continue with another NL statement (i.e. go to step 1). When there are error messages, the users would usually repeat steps 2-4. Some participants would be persist, reformulating the CNL statement until there were no errors (and it was translated to OWL), while others would continue and leave the CNL statement with errors (i.e. not encoded in OWL).

For both tools, the users were occupied mostly with steps 3 and 4 and would often refer to step 2 for a quick check. Two of the eight ACE-View users entered sentences to describe all concepts from the given list (see scenarios), while none of the ROO users managed to complete the descriptions; in most cases the last two concepts were not defined. The main interaction problems were:

Error messages lack detail. When the CL pattern entered was not recognised, the users would not always get informative error messages. In such cases, the users had to guess what may be misleading, e.g. in ACE-View: `The sentence is not correct ACE syntax.` In ROO: `Sentence is not recognised as correct Rabbit sentence.`

Error messages confusing. When the user entered sentences which could not be recognised, they sometimes received error messages that were misleading. ACEView messages included `???` to indicate unrecognised parts in the sentence or referred to grammatical constructs which some users found hard to follow. ROO gave at times misleading suggestions when the sentence was unrecognised.

Dealing with adjectives and compound noun phrases. Recognising a concept which includes a compound noun phrase (e.g. adjective-noun) can be a challenging problem. ACEView users often received the message `adjectives are not supported`, in which case they had to use hyphenation (see above problem). ROO parses for compound noun phrases and in most cases could make helpful suggestions about what the concept might be, e.g. `natural waterway, man-made feature`. However, when the compound nouns were not recognised and this led to confusing error messages, e.g. `natural body of water` was not recognised as a possible concept.

Dealing with a specialised vocabulary. The parsers in both tools could not recognise some specialised vocabulary which did not allow entering certain concepts, such as: `ACEView: sediment, irritation`; ROO: `watershed`. ACEView deals with this by pre-entering classes. However, it would be hard to predict in advance what phrases a user may enter. A more flexible way would be to allow the user to enter a phrase which should be added to the vocabulary used by the NL parser.

Next task suggestion not always useful. This problem only applies to ROO since ACE-View does not have this feature. On several occasions, users ignored the task suggestions and commented that not all of them were useful. E.g.

Question (1-Strongly disagree; 4-Neutral; 7-Strongly agree)	ROO median	ACEView median	U (Mann-Whitney, 1-tail)	p	Significance
The error messages helped me write CL sentences	5	4.5	16.5	p≤0.1	LOW
The error messages were confusing	2	4.5	11.5	p≤0.025	YES
The guide dog was helpful	5	—	—	—	—
The guide dog suggestions were not easy to understand	2	—	—	—	—
I did not follow the suggestions from guide dog	4	—	—	—	—
The interaction was demanding	3	4	39	p>0.1	NO
I had no idea what I was doing	2	1.5	16	p>0.1	NO
It took me too long to compose what I wanted	4	3	21	p>0.1	NO
The interaction was intuitive	5	3.5	11.5	p≤0.025	YES
The feedback was prompt and timely	5	4.5	24	p>0.1	NO
It was clear to me what to do in this tool	5	4.5	24	p>0.1	NO
The tool was frustrating	3	5	5.5	p≤0.01	YES (HIGH)
The tool was unnecessary complex	2.5	3.5	18	p≤0.1	LOW
I'd like to use the tool again	5	4	18.5	p≤0.1	LOW

Table 3.7: Summary of the comparison of the usability of both tools (post-study questionnaire). From the 14 questions, 11 did not yield any statistically significant differences between ROO and ACE-View. However, the results indicate that the error messages in ROO were not experienced as confusing, while ACE-View users had more trouble understanding error messages. Also, users of ROO found the interaction intuitive, while this was not the case for ACE-View. Finally, users did not found ROO frustrating, while they tended to find ACE-View frustrating.

> ROO suggested that the participant enter definitions for concepts that were not directly relevant to the ontology, such as `man` or `bacteria`.

Usability. Table 3.7 summarises the findings from the usability questionnaire. For both tools, the users were positive. ROO was found to be significantly less frustrating than ACEView, which may be due to the much more intuitive interface, much less confusing error messages, and the help offered from the guide dog. The messages in ROO were more helpful, the tool was less complex than ACEView, and users would be more willing to use ROO again (although note the low significance).

Ontology Modelling Skills. The answers to six ontology modelling questions (covering the main steps and building blocks in conceptual models, definition of ontology, concepts, and relations) in the pre- and post-study questionnaires were compared to examine whether the users ontology modelling skills had changed as a result of the interaction with the tool. Two evaluators with a sound ontology background worked independently and marked the users answers. The following scheme was applied to each question: -1 (the understanding has worsened, e.g. because the user was confused); 0 (no change to the users understanding on the questions), +1 (correct aspects are added but gaps exist), +2 (the understanding is improved, and now is correct and complete). The marker compared their results and the discrepancies were clarified in a discussion. The maximum score, if a user had not had any ontology modelling knowledge and has become an expert, would have been 12, while the worst score meaning a user was an expert and became totally confused would have been -6.

The ROO users scored significantly higher than the ACE-View users — ACE-View score mean 0.38, STDEV 2.97; ROO score mean 5, STDEV 2.78; U (Mann-Whitney)=8.5, $p \leq 0.01$. This shows that the users understanding in ontology modelling improves significantly more when using ROO than when using ACE-View.

Quality of the Resultant Ontologies

The resultant ontologies were analysed following the ontology evaluation framework in [49], considering structural, functional, and usability ontology measures.

Ontology Structural Measures. Since the size of the ontologies is limited, we have used fairly simple structural metrics based on [133], calculated by hand based on metrics provided by Protégé 4. We found no significant differences in the structural characteristics of the ontologies created, with exception to annotations per entity, as shown in Table 3.8.

The results show that ontologies built with ROO have a *significantly better readability* than ontologies built with ACEView. Both systems store the entered sentences as annotations in the ontology. Since both Rabbit and ACE are quite readable for humans, these annotations can be used to understand the meaning

	Average Class Count	Average Object Property Count	Average Properties Relative to number of Classes	Average Annotations per Entity	Average Subclass Axioms per Class (Inheritance Richness)
ROO	21.875	8.250	0.367	2.625	0.634
ACE	28.125	11.875	0.420	0.582	0.877
p (t-test)			0.263	0.000	0.095
U (Mann-Whitney)	19.5	21.5			
p (Mann-Whitney)	0.104	0.147			

Table 3.8: Summary of ontology structural measures.

of the OWL entities. The main reason why ROO ontologies are more readable is that ROO encourages users to provide additionally natural language descriptions for both concepts and relationships. When Rabbit sentences are translated and new classes and properties are added to the ontology, an appropriate `rdf:Label` is added. In contrast, ACEView does not add annotations when classes or properties are added.

We measured inheritance richness based on OntoQA [133]. ACEView ontologies had higher inheritance richness (Table 3.8), i.e. the classes built with ACEView had more connections to other classes. However, the functional measures (see Table 4 below) indicate that ACEView ontologies were more tangled than ROO ontologies. Domain experts seemed slightly more productive using ACEView than using ROO but the Mann-Whitney U-test does not provide conclusive significance.

Ontology Functional Measures. A domain expert who is also a knowledge engineer[1] at Ordnance Survey produced two benchmark ontologies to quantify the fitness-for-purpose of the participants ontologies. A scoring system was devised:

+1 point per matching axiom for each axiom produced by the participant ontology that exactly matched[2] an axiom from the benchmark ontology;

[1]We were lucky that such an expert existed, making it possible to examine in depth the functional dimensions of the ontology.

[2] Some interpretation was required owing to variances in terminology.

Scenario	ROO (mean)	ACEView (mean)	U (p)
Geography	1.25	-3.5	3.5(p>0.1)
Environment	3.75	-5	0(p≤0.025)
Combined	2.5	-4.25	9 (p≤0.1)

Table 3.9: Summary of the scores from the functional analysis of the resultant ontologies.

+1 point for additional valid axiom , i.e. axioms that were considered to be valid even though an equivalent did not exist in the benchmark;

-1 point per absent axiom , i.e. a point was deducted for each axiom in the benchmark but absent the users ontology;

-1 point per modelling error , i.e. we deducted a point for any axiom containing a modelling error.

The participants did not define axioms for all the concepts they were given. Where this was the case, we did not count any metrics for that concept for that participant. We only scored against axioms belonging to the concepts in the concept list given to the participants. The total score for each ontology was therefore the sum of the points added or deducted.

Subjectively, the ACEView ontologies appeared to be more complete, whereas the ROO ontologies appeared to be better structured and with fewer modelling errors.

The data for each set of ontologies was analysed statistically using the Mann Whitney U test (Table 3.9). At a 95% confidence level this indicates that there is no significant difference between the sets of data collected for the geography ontologies but that ROO out-performs ACEView with respect to the environmental ontologies and overall (geography and environment combined). The weakest participant by far was a ROO geographer who despite only recording axioms for three concepts achieved a negative overall score, but this alone would not have accounted for the overall differences even given the small sample sizes.

ACEView users tended to describe more concepts and add more axioms (Table 4). This applied to both the in scope concepts and also those out of scope. Some of the latter group were secondary concepts necessary to define the core concepts – for example `water body` used to super class `river` and `reservoir`. But others were irrelevant clutter, such as `Scotland`, and it was not clear why they were added.

ACEView users did better than ROO in getting exact axiom matches with the benchmark ontologies (with a mean that was 1.5 matches higher per person). They also had a higher mean for providing additional axioms, with an average of three more per person. However, ACEView users did very much worse when it came to the number of errors they made, that is the number of axioms that were deemed to be incorrect, averaging 8 errors per person more that ROO users. Even taking into account that ACEView users enter more axioms proportionately they enter 0.4 errors per axiom, compared to 0.13 errors for ROO users. Erroneous axioms were not included in the other axiom counts. If included, it would show that ACEView users are even more prolific - it seems to be a case of quantity over quality. The following is a summary of the modelling problems that occurred:

Multiple tangled inheritance. Both tools showed this problem, although it occurred much less frequently in ROO. This was a very common error in ACE ontologies. In the worst case `Drainage` had five separate immediate simple super classes: `Artificial Object`, `Depression`, `Drainage`, `Long Trench` and `Narrow Trench`. An error was scored for each extra entanglement so in the case above a score of 4 would have been recorded. The axioms would have been included in the overall total of axioms. Although also occurring in ROO ontologies, the rate and degree of multiple inheritance was much lower.

Definition of an instance instead of a class. Both tools showed this problem. There were a number of occasions where a class was recorded as an instance. ACEView example: in one ontology `Flood-Plain` is declared to be an individual of class `sediment-deposition`. In examining the ACE log file the first mention of `flood-plain` is the sentence: `Flood-plain borders a river.` There is no use of `every` in the sentence so ACE assumes

`Flood-Plain` is an individual, and so records the assertion `Flood-plain` is an individual of the anonymous class "`borders some River`". The next correct sentence: `Flood-plain is a sediment-deposition` has the effect of adding `Flood-plain` as an individual of the class `sediment-deposition`. ROO example: user entered `Flood Plain is a Land Area` rather than the correct `Every Flood Plain` ***`is a kind of`*** `Land Area`.

Generation of 'random' individuals Only ACE-View suffered from this problem. ACEView also appears to generate random individuals. For example the sentence: `Scotland contains a farm and contains a forest and contains a reservoir.` Generates three individuals. It is probable that what the user meant was that `Scotland` (also an individual) contains *some* farms, forests and reservoirs. What is even less clear is why the user felt it necessary to add this out of scope information at all.

Repeated Knowledge Both tools showed this problem, although much less frequently in ROO. In a number of cases ACEView users tended to enter axioms that were similar to axioms already entered. An example is:

- `Every flood-plain experiences flooding` and
- `Every flood-plain experiences periodic-flooding.`

Such repetitiveness also occurred in the ROO ontologies, but much less frequently.

Ontology Usability. None of the ontologies as produced would have been usable without modification. This is unsurprising given the fact that the users were essentially untrained in the language and knowledge modelling techniques. No user produced an ontology that provided a complete description of the concepts, but again this is unsurprising given the experience levels and time available. In simple terms the ROO ontologies were less complete, containing fewer concepts and fewer axioms. However, the greater number of modelling errors in the ACE-View ontologies, combined with the amount of unnecessary clutter in terms of out-of-scope concepts and axioms would indicate that it would take longer to get them to a usable state. ROO ontologies were certainly better annotated and this

helped significantly in terms of evaluating the usability of ontologies for a certain purpose.

3.4.4 Benefits and Limitations of CNL-based Interaction for Ontology Authoring

In this section we summarise the main benefits and limitations of CNL-based interaction for Ontology Authoring that follow from our comparative evaluation study.

Efficiency

The results from the evaluation study give positive evidence that **CNL-based interaction makes it possible to quickly involve domain experts in ontology construction**. The time users spent familiarising with CNL interfaces (both in ROO and in ACE View) was relatively short. This result confirms existing findings that CNL sentences are intuitive to understand [41, 96]. Furthermore, CNL provides a unified way for defining all knowledge constructs, such as entering concepts/relationships, specifying hierarchical links, and formulating axioms.

Our evaluation does not compare CNL versus non-CNL ontology authoring (e.g. direct editing of OWL statements, using forms, or visual interfaces). Hence, we have no direct evidence that CNL-based ontology engineering is faster than non-CNL. Some work in this direction has been done elsewhere, for example [47] reports that a CNL interface is more efficient for performing simple ontology construction tasks than traditional tools such as Protégé.

Abstraction

Defining ontological constructs in a CNL requires some level of abstraction, albeit the CNL reduces the cognitive complexity of this process. Typical sentences for defining a concept in a natural language tend to be information rich. For instance, a text definition of `Flood Plain` given by a participant in our comparative study is *Flood plain is the area of land surrounding a river, which is usually flat, and is prone to flooding.* The domain expert has to learn that this text is not suitable

for the ontology and has to be broken down into several sentences. In our study, this was explained at the start of the session when introducing the tool and CNL. However, participants used help material such as example CNL sentences and the tool feedback (error messages) to explore the CNL and learn the limitations of the language. For example, a participant broke the above natural language definition of `Flood Plain` into the following Rabbit constructs:

- `Every flood plain is a kind of area of land.`
- `Every flood plain is around a river.`
- `Every flood plain is prone to flooding.`

Our studies showed that the process of *breaking down* natural language definitions into CNL sentences is a crucial step when using a CNL for building ontologies. In most of the cases, the participants performed this breaking correctly after an initial phase when they learn the restrictions imposed by the CNL and staying within those limits while describing concepts from the ontology. All ROO users, as well as the ACE View users in the comparative study, were able to quickly decide how to rephrase most natural language definitions.

Although the reduced complexity of abstraction from natural language to ontological constructs is a key advantage of CNL-based ontology authoring, it also brings a **crucial limitation**. **Making the formulation of ontological statements fairly easy can be misleading**. In our studies, the participants without knowledge engineering background would focus mainly on the formulation of the CNL constructs. None of them questioned the logical implication of what they had entered. In contrast, users with previous ontology engineering experience not only managed to quickly formulate Rabbit sentences but were more dubious about the exact meaning in OWL terms. These users would often open the Protégé Class Description View to check the OWL translation of the entered sentences. This points at the **need for offering intuitive ways for presenting feedback about the logical consequences** of added sentences, as discussed below.

Ambiguity

In Section 3.3.2 we presented how the Rabbit language and parser allow for restricted ambiguity of entity names. During the user studies with ROO and ACE-View, we observed both expected and unexpected cases of ambiguity. We discuss some of the unexpected cases here. Firstly, some users found built-in relationships in the CNLs ambiguous. For example, commonly confused were *is a kind of* and *is a* in Rabbit. – the former is used to enter subclasses, while the latter is used for defining instances. Users without knowledge engineering background did not realise the difference, and often mixed instances and classes, see Section 3.4.3. Such ambiguity problems are not handled by the mechanisms discussed in Section 3.3.2 because they stem from built-in vocabulary as opposed to user-provided entities. However, they can be anticipated and corresponding prompts added. In the most recent version of ROO, appropriate error messages are added when a possibility for mixing *is a kind of* and *is a* is detected (e.g. when a partial pattern of class definition is recognised but an *is a* relationship is used, the user is reminded that *is a* is used for defining instances).

The second ambiguity type observed was caused by inability of the CNL parser to determine the part-of-speech for some words, most commonly when a word could be tagged as either a verb (hence, corresponding to a relationship) or a noun (hence, corresponding to a concept). For example in a Hydrology domain, a user stated that `Flow is a concept.` when trying to describe the Water Flow of `River`s and other bodies of water in terms of their flow of water. However, the Rabbit parser (at the time of the study) tagged `Flow` as a verb (e.g. to flow) instead of a noun, and the sentence was not accepted. ACE View had similar problems that were solved by enabling a user to extend the glossary of terms. A solution for this ambiguity type is to make the authoring tool aware of such cases and to allow the user to override the part-of-speech tagger of the parser at runtime. The tool should help the user realise that there is a danger of introducing ambiguity, e.g. the user could always state that `flow is a relationship`, which results in an ontology having an object property and a class with potentially the same name. The latest version of ROO includes corresponding warnings when this type of ambiguity is recognised.

While we were able to work around some of these ambiguity problems, these fixes are performed in an ad-hoc manner by adding extra steps to the Rabbit parsing pipeline. This indicates that **there is a need for a principled classification of the types of lexical, syntactic and semantic ambiguities that can occur when analysing ontology authors' inputs**.

Coverage

Learning to use a CNL resembles learning a new language – starting from basic constructs and gradually adding more complex statements. This was confirmed in the experimental studies with ROO - most domain experts utilised only a subset of the full set of Rabbit (and ACE) sentences. The resultant OWL ontologies varied from $\mathcal{ALE}$ to $\mathcal{ALCOI}$ and $\mathcal{ALCOQ}$. That is, the resultant OWL statements included definition of subclasses, anonymous classes (concept union and intersections) universal and existential restrictions and qualified cardinality restrictions. Domain experts rarely used (or did not use at all) CNL sentences that translated into disjoint, equivalence and negation axioms as well as role hierarchies and role inclusions. The reason for this might be that users tried to follow natural language descriptions of concepts – one rarely describes a concept in terms of what it is not (disjoint classes or complex concept negation).

These axioms are crucial for the quality of the resultant ontology as they are vital when using OWL reasoners for automatic classification. This indicates that **there is a need for helping domain experts use more expressive sentence types**. One way to do this would be to extend the *Guide Dog* feature in ROO. However, this functionality has to be invoked by the domain expert and this can give rise to jumps in the scope of the suggestions. Ideally, **the system should pro-actively elicit specific knowledge from the ontology authors based on recently added sentences**. For instance, by scanning the created taxonomy for suitable candidates it can generate *connecting statements* that will enable the reasoning, and can ask the user to confirm or reject these statements. Similarly, domain experts can be directed to use a more systematic approach to combine axioms following ontology design patterns [48], which can be a crucial feature in ontology authoring tools geared towards domain experts.

Quality

The evaluation study showed mixed results regarding the quality of ontologies resulting from domain experts using CNL-based tools. A positive impact that we found on the quality of ontologies is that, since both Rabbit and ACE are easy to understand, including the CNL sentence as annotations *improves the readability of the resultant OWL ontologies*. Ontologies created with ROO were slightly more readable because ROO encourages users to provide natural language descriptions for both concepts and relationships, which are also included as annotations.

A negative finding was that *none of the ontologies produced during the comparative user study would have been usable without modification*. This finding is not surprising because the users in this study had only a limited amount of time to build the ontologies, had no knowledge engineering background, and may have not realised the importance of choosing appropriate knowledge sources. Our experimental design followed closely real situations when domain experts enter knowledge constructs. This, however, led to reliance on domain experts' judgement in choosing the natural language definitions and deciding what statements should be entered. A more controlled experimental design, e.g. providing the correct natural language definitions and asking users to formulate corresponding ontological statements, could be used to further examine the effect of the tools on the ontology quality.

Even taking into account the evaluation study limitations, it seems clear however, that in order to improve the quality of the resulting ontologies **domain experts should be made aware of the logical implications of their assertions**. In the next chapter we propose a way to do this.

3.5 Practical Experience with ROO

After the comparative evaluation study between ROO and ACE-View, we released ROO and used it to develop ontologies in practice. This section presents the experience of developing ontologies in the context of two European projects, ImREAL[1]

[1] `http://www.imreal-project.eu`

and Dicode[1]. In both projects, the development of ontologies was performed in an iterative and modular fashion, involving domain experts and knowledge engineers working together using ROO. These recent practical experience of using ROO confirmed that the semantic aspect of ontology authoring – overseeing the logical consequences of adding OWL axioms to an ontology – is still a major burden to domain experts and knowledge engineers. A minor problem in comparison is that, in some cases, authors also required assistance entering sentences as they were not valid Rabbit sentences and ROO failed to produce a useful error message that could be used to produce a valid Rabbit sentence.

Before using ROO, domain experts and knowledge engineers sat together to discuss the scope and purpose of the ontology. As part of this, domain experts explained the domain to be modelled to the knowledge engineers, revealing in the process that certain documentation (e.g. tables and diagrams) is required to describe the domain. Domain experts then produced the required documentation before the first joint session using ROO.

In the first sessions using ROO, the domain expert and knowledge engineer used ROO to add concepts, relations and axioms to an ontology. The focus on these sessions was to formalise the knowledge encoded in the gathered documents. The knowledge engineer generally provided a supporting role in these sessions, *guiding the domain expert through the authoring process* in the following ways:

- Since the ontology was started from scratch, the knowledge engineer *demonstrated the basic ROO interface* to show how to enter concepts, relations and sentences (when no KE is available, a screencast can be used instead).
- The KE also *explained limitations and capabilities of both the CNL language and OWL when relevant*: for example that Rabbit typically accepts both singular and plural forms of a concept or that a certain inference would be made based on existing sentences.
- *When the KE noticed potential logical issues, the KE intervened* to explain how the current sentence could interact with existing sentences to produce unintended inferences; this usually triggered a process of refinement of the

[1] `http://www.dicode-project.eu`

input sentence. The result was either a sentence that satisfied both the DE (it described the domain satisfactorily) and the KE (it did not introduce any logical problem).

- The KE noticed tacit assumptions and suggested ways to capture them. For example, DEs tend to use generic relationships between concepts such as "Every A has a B", which hides the specific relation between the concepts. In such cases, the KE prompted the DE to suggest a more specific relation.

- The KE also provided *support regarding the general architecture of the ontology* by advocating the creation of modules and abstractions from the constructs proposed by the DE. The DE usually accepted such suggestions, although this was done reluctantly when it involved introducing terminology extraneous to the domain.

Besides these joint sessions, DEs also used ROO on their own, without the support of a KE[1]. During the development of the ontology, the knowledge engineers spent considerable time rectifying the ontology produced by domain experts, mainly working on logical implications such as concept satisfiability, consistency, redundancies and novel concepts with un-intended consequences. Such problems had either been overlooked by the KE during the joint session, or were introduced by DEs during a ROO session without support from the KE. The rectification of such cases is often time consuming, due to the logical dependencies between ontological constructs and because rectification often needs further interaction between KE and DE.

We stress that the problem of authors not understanding the logical implications of their inputs seems to stem from the *combination of several CNL sentences* into an ontology, as there is evidence that even novice ontology authors can understand the logical semantics of most individual CNL-sentences (and thus also of individual OWL axioms) [41, 95]. Indeed, there is evidence that understanding the logical consequences of a set of statements – not necessarily in CNL – is a hard task even for experienced authors [68]. This problem manifests itself when

[1]The DEs during these experiences seemed to prefer working together with a KE, probably to enjoy the immediate support that the KE can provide.

authors using a CNL-based tool produce ontologies that contain logical defects. Such ontologies require significant refinement – typically performed by knowledge engineers – before they become useful.

3.6 Conclusion

In this chapter, we presented ROO, a tool that has been designed to cater for the needs of domain experts with little or no ontology engineering experience. We presented a description of how NLP, parsing and GUI techniques can be used to provide an intuitive interface for a controlled natural language and described an implementation for the Rabbit CNL. In doing so, we gave a detailed description of the various novel aspects of ROO:

- custom parser enabling the use of a consistent and easy to understand terminology for error feedback;
- disambiguation techniques to enable, control and detect the use of natural — but potentially ambiguous — terminology by domain experts;
- a user interface mirroring and providing guidance through an ontology engineering methodology.

This chapter also presented an evaluation study on ROO that provide empirical evidence in support of using intelligent techniques to assist ontology authors. Our evaluation showed that CNL-based interaction enabled domain experts to build ontologies from scratch in a short period of time. This suggests that CNL-based interaction can reduce the cognitive complexity associated with the move from natural language sentences to formal ontological statements. CNL shortens the abstraction path by providing an *intermediate level of abstraction* which helps people without formal logical background to formulate knowledge constructs. The task of converting CNL constructs to OWL, which requires formal knowledge engineering skills, is performed automatically by the CNL parsers.

The evaluation study and further practical experience with ROO presented in this chapter also showed several open issues that need to be addressed in order to enable intuitive ontology authoring:

- the presented disambiguation techniques provide ad-hoc solutions to some forms of ambiguity that can occur in CNLs. However, a more **principled approach is required for enabling the syntactic analysis** of the next generation of CNL tools. The next chapter proposes such a principled approach to the syntactic analysis of ontology authoring inputs.

- domain experts without knowledge engineering experience require support that goes beyond ontology engineering steps and syntactic feedback: they also **require explanations and guidance about the logical implications of entered facts**. The lack of such tool support results in ontologies with poor coverage and with a variety of modelling and logical defects. The next chapter proposes a framework that combines the syntactic and semantic analysis of ontology authoring inputs. In particular, the semantic analysis consists of an integration analysis of new axioms to the ontology and can be used to provide relevant feedback.

- domain experts tend to only use a subset of the full spectrum of CNL sentences which has a negative effect on the coverage and quality of the resulting ontology. This indicates that **the system should pro-actively elicit specific knowledge from ontology authors**. In Chapter 5 we look at a generic framework for enabling more pro-active ontology authoring systems based on dialogue systems.

Chapter 4

Entendre: Understanding Ontology Authors' Inputs

The previous chapter showed that a CNL interface could enable domain experts to enter ontology constructs. However, the user studies pointed out that there is a need for:

- appropriate feedback when ontology authors enter sentences which are not recognised by the CNL-parser. In particular, producing error messages and handling ambiguity cases requires adding rules on top of the natural language and CNL grammar processing;
- guidance about the logical implications of new facts. This is especially the case for novice ontology authors, who typically were content when they managed to write a syntactically correct sentence. As a result, ontology defects were introduced which had to be fixed by a knowledge engineer.

In this chapter we claim that both of these problems can be mitigated by providing a formal description of the process of analysing the inputs of ontology authors (OAs). We propose a framework for performing lexical, syntactic and semantic analysis of ontology authoring inputs.[1] A resultant hypothesis from the

[1]Note that since this chapter proposes a generic framework, it focuses on "ontology authors" instead of "domain experts" as in the previous chapter. This is because the framework is useful to any person who enters knowledge into an ontology authoring system, whether that person

above is that: in order to be able to provide relevant tool support to ontology authors during ontology construction, *tools need to be able to*:

1. provide a robust syntactic understanding of ontology authors' inputs and
2. interpret authors' inputs at semantic levels; that is, tools need to be able to analyse the logical effects of new inputs.

In this chapter, we investigate this hypothesis by introducing a framework, called Entendre, that can be used to provide robust syntax analysis and to incorporate semantic analysis of authors' inputs. For the syntactic analysis, Entendre provides an abstraction of the analysis performed by CNL tools as presented in Chapter 3; this abstraction makes it easier to extend existing CNL-parsers in order to successfully parse inputs that could otherwise not be recognised. For the semantic analysis, Entendre proposes an algorithm to classify axioms in relation to a reference ontology. The result of these analyses can be used to generate feedback for inputs that would otherwise not be recognised and to include relevant logical implications in this feedback.

In this chapter we thus investigate:

- whether we can augment CNL-parsers to provide robust syntactic analysis of OAs' inputs (while keeping the CNL language unambiguous and the CNL-parsing efficient)
- whether we can extend CNL-based ontology authoring to take into account the semantics (i.e. the logical aspects) of the ontology that is being built and
- whether such an extension can be used to aid ontology authors to understand the logical aspects of ontology languages (without requiring prior knowledge about formal logics).

This chapter is organised as follows:

is a knowledge engineer or a domain expert. Having said that, the framework is most useful to *novice* ontology authors and thus also to most domain experts.

- Section 4.1 formulates the problem of interpreting the inputs of ontology authors and argues that a combination of syntactic and semantic analysis is desirable in order to provide tool support for ontology authors.
- Section 4.2 discusses existing approaches for understanding the inputs of ontology authors and providing feedback based on this understanding.
- Section 4.3 provides a formal description of Entendre, a framework for analysing ontology authors' inputs; this chapter provides definitions of various syntactic and semantic analyses and how they can be combined to provide an understanding of ontology authors' inputs.
- Section 4.4 describes an implementation of Entendre based on several existing semantic web tools.
- Section 4.5 describes how the Entendre implementation was used to extend ROO. In particular, we show how Entendre enables the generation of *interactive semantic feedback* during ontology authoring.
- Section 4.6 describes how the feedback provided by Entendre in ROO was evaluated.

4.1 Analysing Ontology Authors' Inputs

The user studies and practical experiences with ROO described in the previous chapter indicate that the *support described in Chapter 3 is still insufficient* to allow ontology authors to produce useful ontologies – at least, when ontology authors are not sufficiently aware of the formal semantics and existing services that can be used to check for inferences and defects. Although ROO supports authors through the ontology construction process and also helps them to recognise and resolve some syntactic errors, authors also require support for understanding the logical implications of their sentences.

The described experiences also suggest that there is a need for enabling the *detection of situations where such syntactic and semantic support is required.* In particular, it would be beneficial to ontology authors if a system could identify

when authors require feedback that is relevant to the knowledge being entered. This approach mimics the support provided by the KE.

Although a semantic analysis can be defined separately from the syntactical analysis, a holistic analysis of the input requires a tighter integration between these analyses. The reason for this is that a semantic analysis requires the input to be syntactically unambiguous. Thus syntactic ambiguity should influence our interpretation of the semantic analysis. Alternatively, logical defects discovered by a semantic analysis might be an indication that an alternative syntactic analysis may apply for the same input. Thus a robust analysis of an input that captures the intent of the ontology author may require a combination of the syntactic and semantic analyses for that input.

It is also important to note that some ontology authors, in particular domain experts, may not be aware of – or may not care about – the distinction between syntactic and semantic issues of their inputs. From their perspective, they just want to add knowledge to the ontology. A step towards intuitive ontology authoring tools is thus to help ontology authors with both syntactic and semantic issues in a uniform way.

Understanding Ontology Authors' Inputs In the remainder of this chapter we use the term "understanding ontology authors' inputs" as a shorthand for: *performing syntactic and/or semantic analyses of the inputs in order to facilitate the communication of information/knowledge from Ontology Authors into the system.* We clarify that with our use of the term "understanding", we are not implying that the system achieves an understanding in the colloquial sense of the word. However, we think this term is suitable in the sense that such syntactic and semantic analyses often result in feedback that helps the ontology author to verify whether the input is being interpreted by the system in the way the author intended.

To sum up: there is a great opportunity to increase the efficiency and effectiveness of the ontology authoring process by providing interactive, semantic feedback that warns ontology authors to consider logical consequences of the entered facts. This requirement becomes even more important now as there is a growing interest in linked data [64] and a push for iterative, collaborative ontology

development that favours reusability [8, 53, 65, 103]. In an iterative, collaborative development style, it is important to be aware of the logical implications while contributing or expanding existing facts.

4.2 Approaches to Understanding Ontology Authors

To the best of our knowledge, there is no existing work that attempts to combine an understanding of both syntactic and semantic issues in the area of ontology authoring, as discussed in the previous section. There are, however, several approaches that aim to improve the system's understanding of some syntactic or semantic aspects of authors' inputs. In practice, these approaches are often combined in ontology authoring tools in an ad-hoc manner. In this section we review existing approaches and discuss how they contribute to the system's understanding of ontology authors' inputs.

4.2.1 Understanding the input's syntax

As we discussed in Chapter 2, in this thesis we focus on analysing textual input as a more flexible interface than *GUI interfaces* (see Section 2.1.2). In this section we review the existing approaches for analysing ontology author's textual inputs.

Using Lexical Information

A common approach to enhance the syntactic understanding of ontology authors' inputs is to perform lexical analysis on the input. This means that the input is analysed against a source of lexical information: `rdfs:label` annotations in the ontology being built, a set of morphological rules, a word distance metric or an external lexical ontology such as WordNet. The aim is always to find lexical variants to OWL entity names that match some input segment. This allows for greater flexibility in the input language, as authors may, for example:

- use the singular or plural forms of the same word interchangeably,

- spell an entity name incorrectly,
- use a different tense of the same verb interchangeably or
- use a synonym for the same entity.

The use of lexical analysis greatly increases the flexibility of the system to map an input onto OWL entities. However, it has the following disadvantages:

- the lexical analysis may be incorrect: e.g. the wrong plural form or synonym may be assumed,
- it may introduce ambiguity: an input word may match the lexical form of two or more entities in an ontology: e.g. because two entities have the same `rdfs:label` or because the word is the synonym for more than one entity in the ontology; a input word may be used as both a verb or a noun making it unclear whether it should map onto a concept, instance or a property.
- the lexical rule may not correspond to the semantic meaning. E.g. if the lexical analysis allows different tenses for a verb, but the ontology author wants to capture in the ontology the difference between when a relationship occurs.

Traditional ontology authoring tools generally perform very limited lexical analysis: they require authors to enter full or abbreviated URIs. In doing this, they put the onus on the ontology author, who has to refer to entities by their full or abbreviated URIs. An exception is the Manchester Syntax parser in Protégé 4, which uses lexical analysis based on the `rdfs:label` information attached to entities. However, the current implementation is faulty when two or more different entities share the same `rdfs:label` value. In this situation, Protégé randomly chooses one of the entities which can result in undesired constructs. Some ontology authoring tools that perform such analysis for entering axioms into an ontology are the CNL editors ROO(Chapter 3), ACE View[77] and PENG[122]. Both ROO and ACE keep track of potential ambiguities that can be introduced due to lexical analysis and they issue error messages when an ambiguous sentence is entered.

To sum up, existing ontology authoring tools use a variety of lexical analysis strategies[1] in order to map input segments to entities in the ontology language. The used strategies vary from requiring full URIs to allowing morphological variances and using external lexicons to allow synonyms. Current tools use one or more different strategies for finding these mappings to existing entities. However these tools rarely communicate the lexical assumptions they make to the user, even though they often result in an incorrect understanding of the input. Resolving such an incorrect understanding is very hard to do without further syntactic and/or semantic analysis.

Using Syntactic Information

The most common source of syntactic information that can be used to analyse inputs is the grammar of the input language. All ontology authoring languages that we are aware of (e.g. RDF/XML, Manchester Syntax, Rabbit) specify a formal grammar[2] of the language. It is good to note that the bulk of the grammar is usually only of interest to tool developers; most ontology authors remain unaware of the specific production rules and only need to be made aware of high-level structures such as "sentences", "concepts" and "keywords".

The use of formal grammars to define languages imposes a trade-off between computability (which impacts ease of developing the language itself and the tools to support that language) and ease-of-use for the person writing sentences in that language. Formal grammars are useful for tool and language developers for specifying an *ideal or standard language*: unambiguously defining which sentences belong to the language and how those sentences are composed. For people writing sentences in the language, the grammar can be an obstacle when they are not aware of all the rules in the grammar.

Research into context free grammars is relatively mature and tool support based on these grammars, originally developed to aid software developers, is well understood. This maturity is reflected by the availability of tools that generate

[1] Even when not directly tied to ontology authoring, such as automatic annotation of texts.

[2] By formal we mean a Context Free Grammar or similar formalisms such as Definite Clause Grammars, Categorial Grammars or Regular Languages.

a parser and editor with auto-complete and syntax highlighting[1] based on an initial grammar. Most ontology authoring tools provide some tool support based on syntactic analysis as we saw in Section 2.2.2, this information can be used in a variety of ways to help the ontology author enter syntactically correct inputs:

1. sentence templates make it easy to learn the different sentence types supported by the language;
2. predictive input support helps users to enter syntactically correct sentences;
3. syntax highlighting helps users to understand the structure of the sentences;
4. paraphrasing helps users to understand the meaning of sentences by providing an alternative syntax for sentences that are semantically equivalent.

This type of tool support improves the usability of ontology authoring tools, but they are not specific to ontology authoring and do not take into account that ontology authors can be very different from software developers, especially when using a CNL interface. While software developers can be expected to go through the process of learning a programming language, some ontology authors – in particular domain experts –, are not used to such a process. Also, ontology languages are more accessible than programming languages as they allow authors to write knowledge statements instead of algorithmic statements. These differences mean that syntactic tool support for CNL grammars should be even more robust than typical software development tools.

For example, although context-free grammars can generate syntactically ambiguous sentences (sentences with more than one derivations), language designers strive to avoid such ambiguities [10] because such languages are computationally harder to parse. Tool support is thus focused on detecting ambiguities when designing the language, rather than providing tool support when authoring sentences.

The use of a formal grammar during ontology authoring *imposes limitations to the expressivity of authors*: only sentences that adhere to the grammar are deemed correct by the system. A recent usability study (in the area of query

[1] E.g. `http://www.eclipse.org/Xtext/`

formulation using natural language interfaces, but the conclusions may be applicable to ontology authoring) suggest that the majority of users prefer languages that are flexible [85] and do not impose overly strict grammatical restrictions[1]. From the tool developer perspective, these limitations are required to (i) keep the grammar simple (ii) provide an unambiguous mapping to the formal semantics of the ontology language and (iii) develop efficient parsers.

To counteract the limitations imposed by the grammar, the system should ideally *explain why limitation exists*; it should also *suggest a similar sentence that is valid in the grammar*. We are not aware of any system for authoring ontologies that does this. Most current tools simply issue a syntax error and rely on the user to fix the problem (in the next section we discuss the strategies for providing syntax errors). Some tools use *heuristic rules* to complement the grammar of the language. These heuristic rules are used to fix common syntactic errors. For example, ACE and ROO automatically insert a period at the end of inputs because authors tend to forget to close their sentences. A problem with such heuristic rules is that they are ad-hoc solutions, their execution is rarely explained to the ontology authors and they can only be applied when there is no danger of introducing ambiguity.

Although we have only considered syntactic analysis approaches that rely on a formal grammar, it is in theory possible to provide an interface that treats the input as a bag of words and tries to find a way to *link the words onto OWL assertions without relying on a grammar*. A similar approach has been used for providing a flexible language for querying ontologies [84], but in such cases, the underlying ontology is used to capture commonly asked questions. This limits the range of interpretations that the system produces. In the case of ontology authoring, where the vocabulary and the relations between entities is not known in advance, this approach would probably not be practical.

Other relevant work includes the work on the Computer-Processable Language (CPL), a controlled natural language from Boeing [26]. The CPL language is an example of a, so called, "natural" approach to CNLs: it provides robust parsing at the cost of introducing ambiguity in the language. It does this by relying

[1] Although a minority of users in the study seemed to prefer tools to impose grammatical limitations

heavily on NLP techniques and using a set of hand-crafted heuristic rules to select the most likely interpretation for an input. Because this approach can result in wrong selections, and because the heuristics often depend on the domain, they later added a more restrictive version of CPL called CPL-lite that rules out several cases of ambiguity at the cost of making the language less robust and cumbersome. Since they started from the more robust language, it was relatively easy to specify CPL-Lite and combine the more robust parsing from CPL when a sentence falls outside CPL-Lite. In their experience authors tend to stay within the confinements of CPL-Lite, but it is good to have the more robust CPL to fall back on. The work on CPL and CPL-Lite has not been formalised, Entendre is an attempt to formalise the relation between the more traditional, less robust syntaxes and extensions that are more robust at the price of predictability (See Section 2.2 for more details of CPL and its relation to Rabbit).

In summary, syntactic analysis based on formal grammars of inputs is well understood and provides several ways to support the ontology author. However, the use of grammars to perform syntactic analysis can result in overly restrictive syntaxes. While the use of heuristic rules can make the syntactic analysis more flexible, their use is limited to avoid introducing ambiguities.

Syntax Error Messages

When the approaches for lexical or syntactic analysis described above fail – i.e. the syntax of the input is not understood by the system, which means that no OWL axioms can be mapped to the input – the commonly used approach is to issue a syntax error message to prompt the ontology authors to revise their inputs. The content and understandability of the error messages varies widely depending on the targeted users (type of ontology authors) and the way that the syntax analysis is implemented.

The most common approach for providing syntax errors is to **reuse the standard syntax errors provided by parser generators** such as Antlr[1] or Yacc[2]. This approach results in very cryptic error messages that assume the ontology author is acquainted with the language grammar and how this grammar is parsed.

[1] `http://www.antlr.org/`
[2] `http://dinosaur.compilertools.net/`

Typical error messages expose vocabulary that is only relevant for parser developers: the errors include, besides the line and column number, which production rule(s) failed, which token types were expected and/or which unexpected token types were found. Such error messages are not very intuitive as most ontology authors will not understand these messages due to their terminology[1] and will be unable to rectify any errors(unless they are aware of all the production rules in the grammar). An example of an ontology authoring tool that uses this approach is OPPL[2].

To avoid exposing parsing terminology on syntax errors, **parsers are often custom built or an extra analysis is performed on the auto-generated error messages**. In both cases, the lexical and syntactic analysis of the parsed input needs to record information that can be used to generate error messages. Also, parsing terminology needs to be adapted to the situation at hand: the error message generator needs to know which parts of the grammar can be exposed to the ontology author. This choice often depends on the intended audience. For example, a CNL parsing system may include linguistic terminology in its error messages (e.g. a noun phrase was expected, but a verb was found), while a different parsing system may avoid linguistic terminology and use OWL terminology instead (e.g. an OWL class was expected, but an object property was found). ROO, ACE and the Manchester Syntax parser in Protégé 4 all use this approach.

Error messages may also **include information about the attempted strategies to understand the input's syntax**. An error message can, for example, state that a word in the input was matched to an existing entry in WordNet, but that none of the words in the synset matched a class in the ontology. Such messages help the ontology author understand how inputs are interpreted by the system.

Finally, error messages can also contain **suggestions about how the grammatically incorrect input can be changed to arrive at a valid input**. As stated before, tools based on context free grammars usually do not provide these types of errors as any sentence that is not produced by the grammar is assumed

[1] Assuming the authors have no experience building parsers and no knowledge of the grammar of the syntax being used to author the ontology

[2] `http://oppl2.sourceforge.net/`

to be incorrect. In ROO for example, we perform an extra analysis to find nouns and noun phrases which could be potential concepts that have not been added to the ontology yet. We are not aware of other tools which perform such extra analysis to try to repair the syntax of the input.[1].

Summary of Approaches for Syntactic Analysis

This section presented an overview of existing approaches that are used to analyse the syntax of ontology authors' inputs. We saw that:

At the lexical level , ontology authoring tools use a variety of strategies to map segments of inputs to entities in the ontology. For the ontology author, it is more convenient when the system uses more advanced lexical strategies, because then the ontology author can use more natural language. However, we saw that implementing such advanced strategies is not trivial and can result in incorrect mappings. Furthermore, resolving such incorrect mappings can only be done when taking into account other syntactic and semantic aspects of the input.

At the syntactic level , most ontology authoring tools rely on formal grammars as a way to provide standard tool support for ontology authors such as syntax highlighting and auto-completion. However, further tool support is required for novice ontology authors; which can be achieved through the use of heuristic extensions to formal grammar parsers. A principled way to describe and apply such heuristic extensions is necessary.

Syntactic feedback of varying quality is generated by existing ontology authoring tools. Custom-built parsers are often required in order to produce understandable feedback at the right level of abstraction. Furthermore, the use of more complex lexical strategies and heuristics for syntax analysis requires better feedback in order to inform ontology authors about assumptions made by the system. The lack of a principled way to accommodate

[1] At the lexical level, a well known strategy for building robust parsers is the use of the Levensthein distance. However, we are not aware of any similar approach for the syntactic level of parsers (for either CNLs or other types of languages).

such advanced syntax analysis strategies in a unified manner stands in the way of providing such necessary feedback.

4.2.2 Understanding the input's logical implications

We will now discuss existing approaches for understanding the logical consequences of ontology authors' inputs. These approaches generally assume that the ontology author's input does not contain syntactical errors and is unambiguous: thus, that there is a single interpretation of the input as a set of OWL axioms.

Most of the approaches in the state-of-the-art assume that ontology authors have some knowledge of the semantics of ontology languages. As a consequence, the approaches are only applicable when the ontology authors know how to use existing reasoning services to confirm or explore the logical consequences of the authored ontologies. In other words, these tools are not intuitive and are hard to use by ontology authors who lack knowledge engineering skills.

While existing approaches for analysing the logical consequences of ontology authors' inputs are not intuitive, they are still very valuable to ontology authors as they make it easier to perform the following tasks: (i) verify whether expected inferences have been made, (ii) become aware of unintended inferences and (iii) explore and understand how inferences are made by the reasoners.

The most common approach to understand the semantics of an author's input is to *provide access to generic reasoning services*. This access allows for semi-automatic analysis of the input. In the following sections we will discuss several of these reasoning services. At the end we will also discuss a less common approach to understand the semantic consequences: an automatic approach[1].

Basic DL-Reasoner Services

The most basic reasoning services are those provided by DL reasoners: consistency checking and classification [7]. During ontology authoring, after entering one of more inputs, the ontology author can use the reasoner to classify the ontology. This process will fail if the ontology is inconsistent, which is usually shown by

[1]Because we are focusing on intuitive **tool** support, we do not consider collaborative authoring as an approach for understanding the logical consequences.

authoring tools as an error message. When the classification succeeds, authoring tools present the inferred class graph as a visualisation or as a hierarchy. The classification will also show any unsatisfiable classes as being equivalent to (or subclasses of) `owl:Nothing`.

Another reasoning service provided by DL reasoners is *query answering* or entailment checking. An ontology author can use this service by asking the DL reasoner whether a given axiom is entailed by the ontology or not. Ontology authoring tools tend to provide access to this service by providing a syntax for writing queries, such as Sparql-DL [127] and SQWRL [109] or a subset of Manchester Syntax as is done in the query tab in Protégé.

The information provided by a DL reasoner is useful to authors as it warns them about inconsistency problems, unsatisfiable classes and unexpected subsumption relations. When the author is experienced and the new issues were introduced recently, the author may be able to figure out which inputs contain errors. When the author is inexperienced or does not have enough understanding of the OWL semantics the author may need further support to understand the unexpected inference. Also, when the author discovers the issue after several ontology changes it may be difficult to pinpoint a few axioms that cause the problem. This approach thus suffers from two main drawbacks:

- *lack of transparency* as any of the issues may be the result of complex interactions of different OWL constructs. This is also true for answers to queries: the reasoner determines whether an axiom is entailed by the ontology, but does not provide any justification as to why this is the case.

- *low discoverability of issues*: the classification graph can be very large and unexpected inferences may not be spotted immediately by ontology authors. This problem is exacerbated when authors do not run the reasoner after each ontology change, but rather wait until a large number of changes have been made.

Recently, higher-level reasoning services [1] are becoming available that address the problems mentioned above. In particular, *justification generators* help

[1] We call them *higher-level* because they generally are built on top of the basic services provided by DL reasoners.

to make inferences more understandable while *entailment set extractors* help to discover new entailments that affect a particular entity in the ontology. We discuss how these services are made available by current ontology authoring tools.

Entailment Set Extractors

Reasoning services that extract entailment sets from an ontology can aid authors to become aware of unintended inferences that their ontologies may have. They also help novice authors to get a feeling for the types of inferences that DL reasoners can make based on asserted axioms. The main drawback of these services is that even small ontologies can result in an infinite number of inferences; therefore, a *choice has to be made as to which set of entailments are deemed "interesting" and should be extracted.* The potentially large number of entailments also means that *a strategy has to be defined for showing these axioms to the ontology authors*; this strategy should prevent the author becoming overwhelmed with too many entailments and further narrowing the selection of entailments to a subset that is relevant to the task at hand.[1]

Different ontology authoring tools have made different choices when defining the set of entailments and when presenting these entailments. For example, Protégé 4 provides three different sets of entailments. The smallest set is based on lightweight reasoning support(it does not require a full classification of the ontology by a DL reasoner) to show inherited restrictions of classes; this type of entailments is shown in Figure 4.1. Another set of axioms is shown when an ontology is classified in Protégé 4 (by manually invoking a DL-reasoner); in this case, some components are updated to show extra restrictions on the selected entities (these restrictions correspond to an entailment that has been extracted, e.g. a new subsumption relation determined by the DL reasoner) as shown in

[1]Note that in the above we are dealing with an issue of trying to avoid cognitive overload of the ontology authors; a separate issue is the generation of entailments for very large knowledge bases, which is an issue of computational scalability. In such cases, usually only a subset of the ontology language is used to guarantee the materialisation of entailments in a reasonable time. In the context of ontology authoring, we assume that ontology development avoids the use of very large knowledge bases by using modularisation and alignment techniques, which are out of the scope of this PhD.

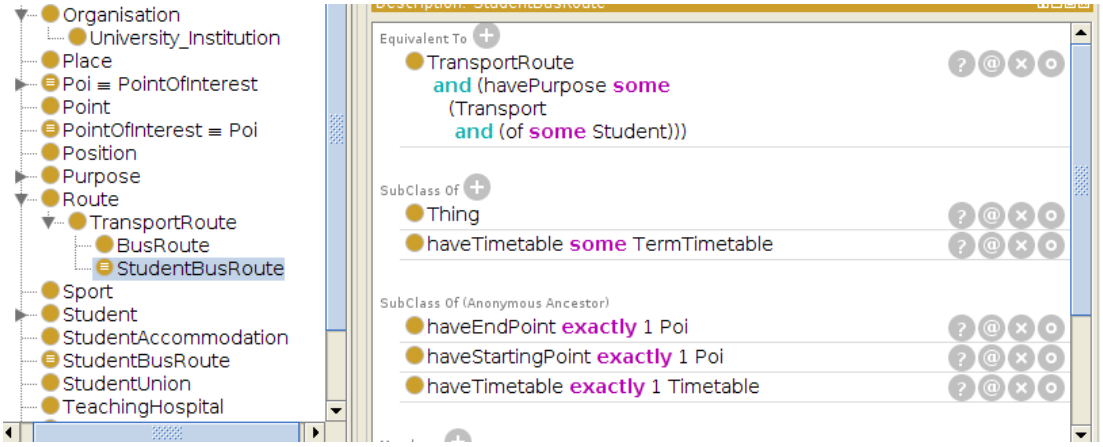

Figure 4.1: Screenshot of Protégé 4 showing both asserted and inherited axioms for class `StudentBusRoute`. The inherited axioms are shown under the heading of "SubClass of (Anonymous Ancestor)". In this case, the inherited axioms have been defined for class `TransportRoute`. No reasoner is required for Protégé 4 to show these inferences.

Figure 4.2. The final set of entailments is shown in a special component called *Selected Entailments*, see Figure 4.3. This component also requires running a reasoner and shows basic inferred relations between named classes such as implicit subsumptions.

Recent work has proposed a way to unambiguously specify some selections of entailments [9]. However, the proposed selections remain fairly simple: for example, they exclude entailments that involve concept expressions. Thus, there is still no standard way to determine what constitutes an interesting set of entailments. Furthermore, there are no empirical studies to compare the impact of providing entailments sets on ontology authoring.

Entailment Justifiers

Justification of inferences as described in [69] greatly helps authors to gain insights into how inferences are derived from the ontology and the OWL semantics. When ontology authors find an unexpected entailment in their ontology (e.g. in a set

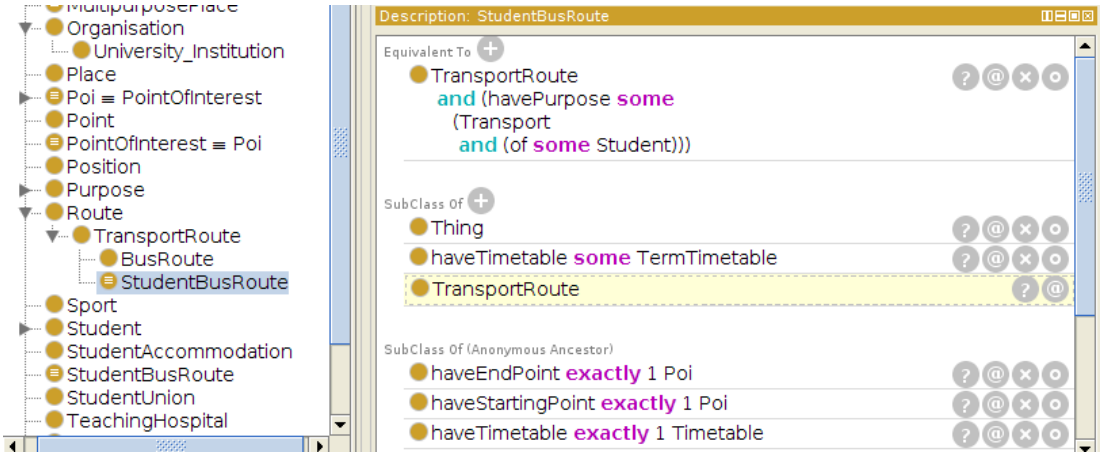

Figure 4.2: Screenshot of Protégé 4 showing asserted, inherited axioms and inferred axioms for class `StudentBusRoute`. The inferred axioms are shown in yellow and require the manual running of a reasoner.

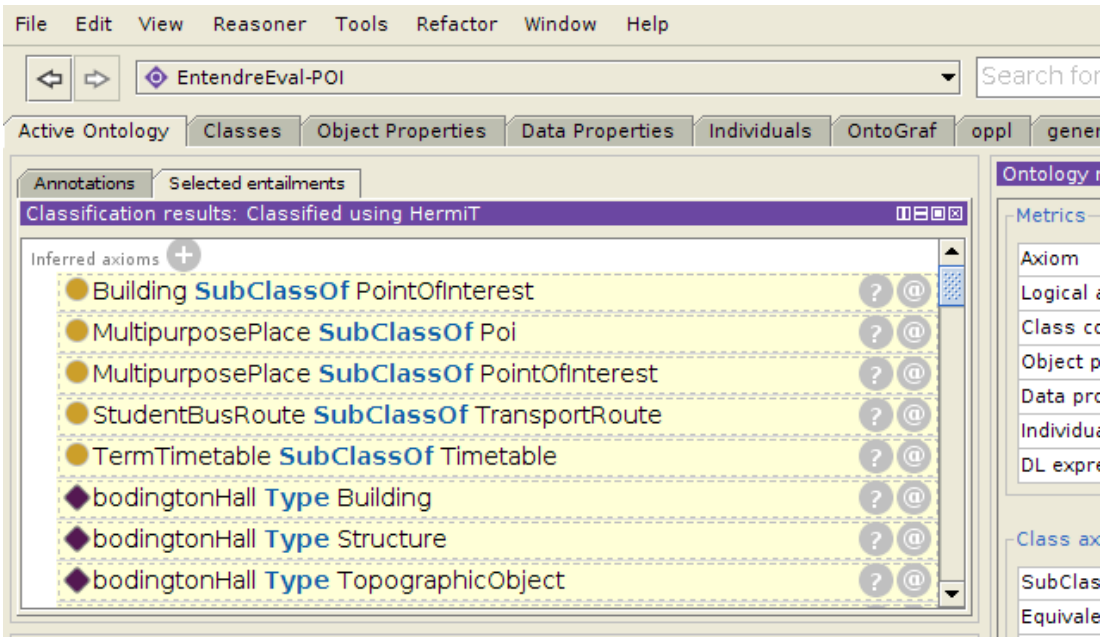

Figure 4.3: Screenshot of Protégé 4 showing the Selected Entailments component. This component only shows entailments after manually running a reasoner.

extracted by the tool, or after a query), they can use an entailment justifier to generate one or more justifications: a minimal subset of the ontology that is sufficient to entail the axiom. The idea is that, by showing only the axioms that influence the entailment, the authors will be able to understand how the entailment was made (and possibly discover whether an existing axiom needs to be altered).

Ontology authoring tools such as Protégé recently added support for justifications by adding a button to generate a justification for entailed axioms that have been extracted [1].

Because the algorithms for finding justifications generally depend on DL reasoners, they only work for consistent ontologies. This means they can be used to debug some issues in ontologies such as unsatisfiable classes and redundancies, but are not directly useful for debugging inconsistent ontologies.

Inconsistency Debuggers

Most DL-Reasoners are not able to classify inconsistent ontologies, nor can they give meaningful answers to queries because all axioms are entailed by an inconsistent ontology. Because of this, other services such as justification generators and entailment set extractors cannot be used with inconsistent ontologies. In order to provide reasoning services for such ontologies, a consistent subset of the ontology must be found first. As there are many such subsets, it is generally very difficult to provide intuitive services that can accurately pinpoint the source of the inconsistency [69].

Due to the difficulty in providing reasoning services for inconsistent ontologies, it is generally better to prevent the ontology from becoming inconsistent in the first place. When this cannot be prevented, it is better to detect inconsistencies as soon as possible and to keep a log of ontology changes; this makes it easier to identify the potential cause(s) of the inconsistency.

[1] Although, in principle, a justification can be found for all entailed axioms [72], the current implementation in Protégé, only provides justifications for a certain axiom types.

Model Generation and Exploration

Another reasoning service that helps to understand the logical implications of ontologies is the *generation and exploration of models for a given ontology*. Such a service is implemented in SuperModel [11], a tool that prompts the user to fill an initial ABox and then uses an OWL reasoner to generate a model[1] that satisfies both the initial ABox and the ontology. This model is presented as a graph that can be explored and refined by the ontology author. The idea is that generated graphs will help the author to discover and understand the implications and restrictions of classes. Providing the seed ABox to generate the model and the model exploration itself are activities that have to be initiated by the ontology author. Furthermore, although the evaluation study presented in [11] indicates the approach can be beneficial for some specific tasks such as improving the understanding of property restrictions on classes, it is not clear whether other, simpler approaches can have the same effect.

Automatic Semantic Feedback

All of the tools described above require the ontology author to be aware of the OWL semantics in order to be able to understand what the tools do, how to use them and how to interpret their results. They are semi-automatic in the sense that the ontology author needs to decide when the tool should be used. Fully automatic semantic feedback for inputs was initially explored by Liebig and Noppens using the OntoTrack system [102].

OntoTrack is an ontology authoring system that employs a *graphical user interface*. The novelty of OntoTrack is that it provides *instant reasoning feedback*. When a new axiom is added to (or removed from) the ontology, OntoTrack automatically updates the presented graph to show a selection of logical consequences provided by an OWL reasoner. This means that the graph presented by OntoTrack is always an extension of the graph that was stated by the ontology author: new nodes and links in the graph show inferred classes and relations.

Although the creators of OntoTrack argue that its input analysis, coupled with instant "feedback obviously will help users to identify faulty or non-intended

[1] This is a model in the Tarski model-theoretic sense.

modelling" [102, Section 3.4]. However, there is a lack of evaluation that validates this claim. Certainly, anybody who has authored an ontology eventually realises that this feature is very helpful. However, we are not aware of any evaluation to determine whether providing this functionality automatically is desirable in practice. For example, the information gathered by this analysis may be viewed as redundant, distracting or confusing by some ontology authors. In Section 4.6 of this thesis, we improve on this situation by reporting on an evaluation study we performed to investigate the effect that such semantic feedback has on ontology authors.

OntroTrack's main contribution was coupling the interactive semantic feedback to the ontology authoring. However, the authors do not provide a formalisation of the system: they merely say that some inferences can be extracted using a reasoner and shown to the user. The authors do not attempt to formalise which inferences should be extracted in order to aid ontology authoring scenarios. In Section 4.3.2 of this thesis, we provide a formal description of an algorithm for performing semantic analysis during ontology authoring; it formally describes several categories of axioms that are relevant to ontology authoring tasks.

Besides the work on OntoTrack there are no other attempts to automatically provide semantic feedback when adding new statements to the ontology. Recent ontology authoring tools, such as the commercial tool FluentEditor[1], automatically run a reasoner on user input, but the analysis and the feedback of the tool are minimal: it only shows unsatisfiable classes.

There is however related work on *Ontology Integration*. A system called *ContentMap* treats a set of ontology mappings as a set of new axioms to be added to the ontology [75]. ContentMap then performs semantic analysis of the axioms in order to aid the ontology authors to select a subset of mappings to avoid introducing ontological defects. ContentMap is designed for advanced ontology authors who already have knowledge engineering skills and focuses on the computability of their integration analysis. In this PhD, we propose a similar semantic analysis, but focus on the feedback that we can generate and present to novice ontology authors. Another difference is that in this PhD, we are only considering the integration of new input facts into an existing ontology instead of the more

[1] `http://www.cognitum.eu/products/FluentEditor/`

general problem of integration of ontologies. Furthermore, because of this narrower scope, we will define categories of axioms that have specific effects on the authored ontology.

4.2.3 Summary

Several reasoning services are available for helping ontology authors to understand the logical consequences of their ontologies. Unfortunately, most of these tools require ontology authors to understand the logical underpinnings of the ontology language, rendering them unsuitable for most domain experts. There has not been much research on making such reasoning services more intuitive. This is unfortunate as the reasoning services complement each other very well and could be combined to provide more intuitive interfaces for ontology authors.

We think it is also important, in order to provide intuitive interfaces for ontology authoring, to take into account results from the syntactic analysis when performing semantic analysis. Although the two types of analysis can be performed independently from each other, it is important for tools to be aware of how syntactic results may affect semantic analysis results: namely, such tool awareness should results in improved feedback for ontology authors.

4.3 Formal Description

In the previous sections we saw that in order to provide appropriate tool support to ontology authors, there is a need for a principled way to (i) integrate various syntactic and semantic analyses strategies and (ii) produce understandable feedback based on the analysis results. In this section we present a generic framework, called Entendre, which provides a formal description of how an ontology authoring system can gain an understanding of an ontology author's input. The overall analysis performed by such an OA system can be seen as a gradual semantic enrichment of the input by analysing that input based on some reference ontology and a set of analysis strategies. Figure 4.4 shows that the overall

analysis comprises lexical, syntactic and semantic analysis steps.[1] Each step in the analysis refines the understanding of the system about the input:

- The entity annotation stage or lexical analysis[2] results in a *lexical reading* of the input. A lexical reading specifies whether the input mentions entities used in the reference ontology, or whether it uses terms that do not occur in the reference ontology.

- A syntactic analysis of the input results in a *syntactic reading* of that input. A syntactic reading maps the input to zero, one or more assertions expressed in the ontology language (these assertions may refer to entities defined in the reference ontology).

- Finally, a semantic analysis of the input results in a *semantic reading* of that input. A semantic reading relates the input to logical consequences that hold for that input based on the reference ontology. For example, a semantic reading may state that an input contradicts assertions in the reference ontology.

In this chapter, we assume that the ontology author's input is a textual input, which may be analysed by using a variety of syntactic strategies such as those defined by various CNLs – such as Rabbit, ACE or CPL –, or more traditional syntaxes such as the Manchester Syntax, or the RDF/XML serialisation syntax for OWL. We also assume that the reference ontology is represented in OWL, although most of the presented approach could be used with other ontology languages.

[1] Note that this diagram only shows a successful analysis; in practice, each phase of analysis may be inconclusive: no enrichment could be found or more than one annotation could apply to the same input. Entendre defines ways to handle these cases as well.

[2] Note that in this thesis, lexical analysis means mapping segments of the input to names in the reference ontology. This should not be confused with the processing of lower-level symbols such as characters in a string, which are not in the scope of this thesis.

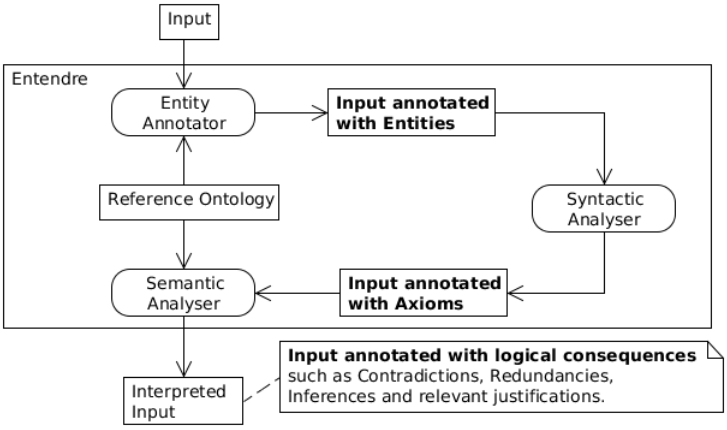

Figure 4.4: Overview of the analyses executed in Entendre and *how they enrich the original textual input* by an ontology author.

4.3.1 Syntactic Analysis

Preliminaries

In this thesis we reuse a formal framework for linguistic annotation introduced by Bird and Liberman in [14]. This formalisation has been shown to be general enough to capture annotation of a wide variety of linguistic corpora. Besides providing a solid grounding to represent annotations, this framework ensures that our formalisation of Entendre – which in this thesis focuses on analysing textual inputs – can be extended for analysing other types of inputs such as speech or video.

We use ι to denote an input by an ontology author. Since we focus on CNL-based interfaces we only consider inputs which are textual inputs; more specifically we assume that our input is a finite sequence of characters (i.e. a string).

Bird and Liberman define an *annotation graph* $\mathcal{G}$ as a triple $\langle N, A, \tau \rangle$ over a label set L and collection of timelines $(T_i, \leq_i)$, where each timeline is a totally ordered set. N is the set of *nodes* (i.e. graph vertices), A is a set of *labelled arcs* (i.e. graph edges) and τ is a *time function* $\tau : N \to \bigcup T_i$. The annotation graph must satisfy two conditions:

1. $\langle N, A\rangle$ is a labelled acyclic digraph (also called directed acyclic graph) that contains no nodes of degree zero

2. for any path from node n_1 to n_2 in $\mathcal{G}$, if $\tau(n_1)$ and $\tau(n_2)$ are defined, then there is a timeline i such that $\tau(n_1) \leq_i \tau(n_2)$.

The conditions mean that an annotation graph always "moves forward" in terms of some timeline.

In this thesis, we will use a narrower version of annotation graphs:

- Because we are concerned with an ontology authoring task, we assume that annotation graphs are defined in the context of some ontology $\mathcal{O}$ – we call this the *reference ontology* – that is written in an ontology language $\mathcal{L}$. Thus in this thesis, an annotation graph will be a tuple $\langle N, A, \tau, \mathcal{O}\rangle$, where $\langle N, A, \tau\rangle$ is defined as above and $\mathcal{O}$ is an ontology. A formal definition of an ontology is given in Section 4.3.2.

- Because we only consider textual inputs, annotation graphs will only be defined over a single timeline. This timeline is given by the indices of the input string. This restriction means that we can assign a time value to any node in the annotation graph.[1]

- For any arc from node n_1 to n_2, n_1 and n_2 must be different nodes. That is: indices in the input string will never be annotated, only segments of the input can be annotated.

The definition of annotation graph means that one cannot have isolated nodes: all nodes must have at least one incoming or outgoing arc. It is possible however, to have a graph that consists of two or more subgraphs which are not interconnected through an arc (see for example Figure 4.5). Also, one can have empty annotation graphs that have no nodes or arcs; we use $\emptyset$ to denote the empty annotation graph.

Annotation graphs can be presented in a graphical form as shown in Figure 4.5. Because we are only considering inputs that are sequences in this thesis, we will

[1]This means we only consider *totally-anchored* annotation graphs as defined in [14].

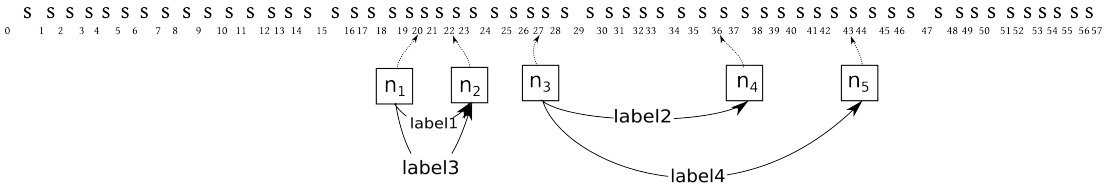

Figure 4.5: Example of an Annotation Graph for an input. From top to bottom this example shows: the input sentence consisting of a sequence of symbols "s", the timeline, the mapping τ, the nodes in AG and the labelled arcs in AG.

use a simpler notation for annotation graphs based on *intervals*. Figure 4.6 shows the same graph as Figure 4.5, but uses the interval notation. In the interval notation, we omit the names for the nodes. Instead, we show only the arcs and their labels as intervals. Each interval corresponds to one or more arcs in the annotation graph that have the same start and end nodes. The labels of those arcs are shown below the interval, as shown in Figure 4.6. Whenever we show intervals in figures, we will also show the original input in order to "align the intervals with the input": the horizontal spatial locations of intervals corresponds to their time values.

Representing Ambiguity with Annotation Graphs

In the next sections we will often encounter ambiguous and unambiguous variants of annotation graphs. We introduce a formal definition of ambiguity based on the structure of the annotation graph.

Definition 1. *An **Unambiguous Annotation Graph** is an annotation graph $\mathcal{G}$ such that:*

- *every node in $\mathcal{G}$ has indegree at most 1 and has outdegree at most 1 (thus every node has degree at most 2) and*
- *for any arc from node n_i to n_j in the graph, the set of nodes $\{n \mid n$ in $\mathcal{G}$ and $\tau(n_i) < \tau(n) < \tau(n_j)\}$ is empty.*

Figure 4.6: The same Annotation Graph as in Figure 4.5 using the *interval* notation. From top to bottom this example shows: the input sentence consisting of a sequence of symbols "s", the timeline, the annotation graph using the interval notation. One interval starts at time value 20 and end at time value 22; this interval has two labels: labels 1 and 3. A second interval, that contains only label2, starts at time value 27 and ends at time value 36. In this case we have given the annotation graph a name: g_n.

Annotation graphs that are not unambiguous are ambiguous.

Intuitively, this means that for an unambiguous AG, there can be at most one arc (and thus also at most one label) that applies to any segment of the input. Using the interval notation, ambiguity is easy to spot as intervals that overlap or encompass each other. For example in Figure 4.6, the intervals for labels 1 and 3 overlap each other; in the same figure, label4 encompasses label2.

It can be shown that any ambiguous AG can be decomposed into 2 or more unambiguous AGs. For example, the ambiguous AG shown in Figure 4.6 decomposes into 4 unambiguous subgraphs, shown in Figure 4.7. In this thesis, we will use $\delta(\mathcal{G})$ to denote the set of unambiguous sub annotation graphs for $\mathcal{G}$.

Lexical Readings

The first step during the syntactic analysis of an input is to determine the vocabulary that the input uses; in other words, find the entities that the input refers to. This process results in a *lexical reading* of the input: this is an annotation graph that maps segments of the input string to entity names in the ontology language.

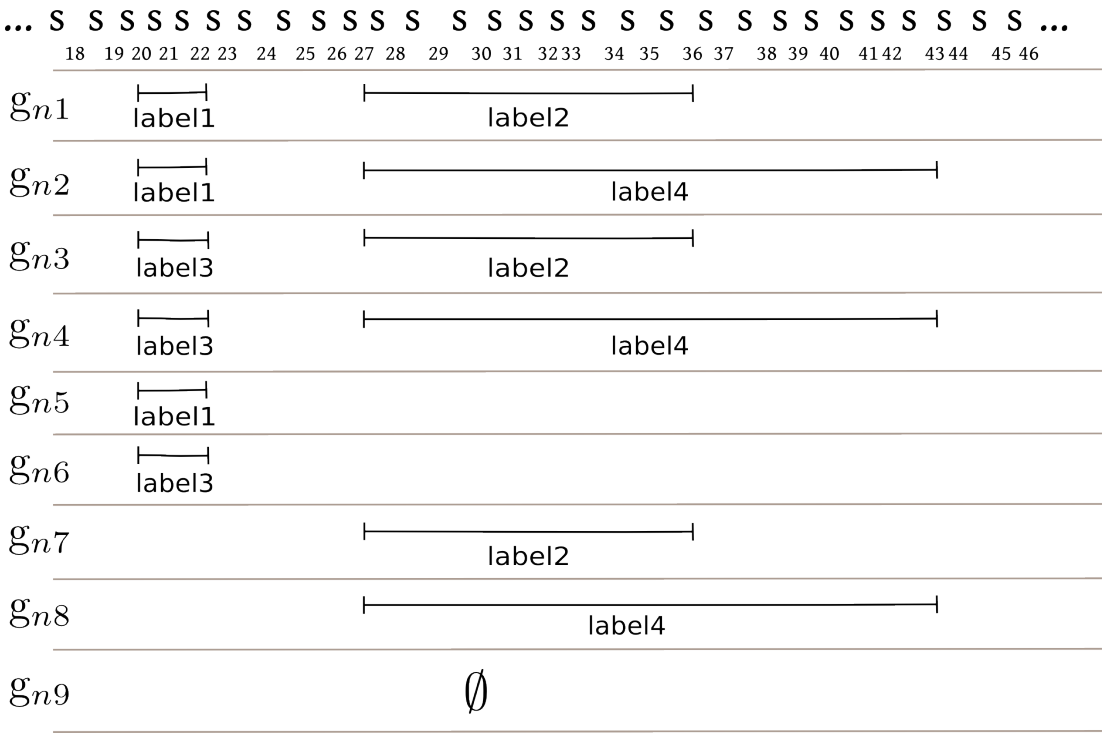

Figure 4.7: The nine unambiguous subgraphs of the ambiguous Annotation Graph shown in Figure 4.6. Note that the empty annotation graph (represented as $\emptyset$) is always an unambiguous subgraph of any other non-empty annotation graph. Note further that the ambiguities that can be represented by Annotation Graphs depend on the contents of the labels. Thus if label1 and label3 are entity names, then the annotated string has an ambiguous mapping to entity names. But if the labels would refer to different axioms in OWL, the ambiguity would be structural or logical.

Definition 2. *Let ι be an input string, Σ the set of entity names in an ontology language and Γ_Σ the set of strategies for mapping input segments onto entity names; then a* ***Lexical Reading*** *of ι is an annotation graph over the set of labels L_{entity}. A label in L_{entity} is a triple $\langle \sigma, n, \kappa \rangle$, where $\sigma \in \Sigma$, n is a natural number and $\kappa \in \Gamma_\Sigma$.*

In the case of OWL, the entity names are IRIs[1]. Intuitively, an arc in a lexical reading means that a segment of the input can be mapped onto an IRI by using one or more mapping strategies. The numbers in the labels act as identifiers, this makes it possible to determine whether two (or more) segment refer to part of the same OWL entity or whether they refer to different entities. See for example Figure 4.8, where input segment "is not contained within" may refer to an object property `:beContainedWithin`[2]. We will generally omit the identifier in our figures unless an entity is split across multiple intervals (e.g. Figure 4.8) or unless an entity is named more than once in the same input.

In this chapter we assume that *lexical mapping strategies* are black boxes that are performed during lexical analysis of the input. Thus, applying a mapping strategy κ_i to an input creates a lexical reading that only contains labels of the form $\langle \sigma, n, \kappa \rangle$ where $\kappa = \kappa_i$. However, because we envisage provision of robust syntactic analysis we allow the lexical analysis to use more than a single mapping strategy, hence the definition of lexical readings over a set Γ_Σ.

The labels in lexical readings refer to *mapping strategies* as a provenance mechanism. This allows Entendre to keep track of heuristics (and their assumptions) that were made during the lexical analysis. The provenance information is necessary for feedback generation and can also be used for disambiguation purposes. For example, in Figure 4.8 we see that the segment of the input corresponding to string "Student" was mapped to two different IRIs by two different mapping strategies. Both strategies first used Part-of-Speech tagging to determine that the string "Student" is a *Noun*. One strategy however, mapped the corresponding

[1] `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/#IRIs`

[2] Here, we are using abbreviated IRIs where the string before the colon is a IRI prefix. For the examples in this chapter we will use prefix `n` to stand for `http://example.com/ontology/newEntities`. When the prefix is empty in an abbreviated IRI, the IRI uses the default namespace of the current ontology.

input segment to a new, unused IRI `n:Student`. The second mapping strategy mapped the noun to an existing entity name from the ontology $\mathcal{O}$ that matched with the `rdfs:label` (`:Student`).

The mapping strategies enable robust syntactic analysis of the input. Consider a case where the ontology author wants to enter the input sentence shown in Figure 4.8 but due to a typo the user wrote "contrained" instead of "contained". In such a case, the default strategy of matching `rdfs:label`s will not succeed. However, another mapping strategy using the Levenshtein distance [101] will be able to correctly map the input to `:isContainedWithin`.

As Figure 4.8 shows, ambiguity is very common in lexical readings.[1] The use of multiple mapping strategies, of course, causes such ambiguity. But ambiguity occurs even when using a single mapping strategy: for example, different entities in the reference ontology can have the same `rdfs:label` value causing multiple mappings for the same input segment. In general, allowing such lexical ambiguity allows ontology authors to be less precise about their inputs, but puts the onus on the system analysing the input to perform disambiguation and to provide variants of the input that are unambiguous.

Algorithms for syntactic analysis generally require unambiguous lexical annotations over the input; Entendre thus requires a disambiguation process as part of the entity annotation stage as shown in figure 4.9. For example, r_0^{Σ}, the lexical reading in Figure 4.8, can be decomposed into 320 unambiguous lexical readings[2].

[1]Note that in this thesis, we use the term "reading" as the result of some analysis of an input string. Readings in this sense represent an intermediate state before attaching a specific meaning to an input. Using this sense of reading, it is then possible to have an analysis result, a reading, that is ambiguous in the sense that no single meaning can be attached to the input. We contemplated using a different term like "interpretation", but this seems to indicate even more that a particular interpretation cannot be ambiguous.

[2]To calculate this number, we first look at intervals that do not encompass other intervals. For example "Student" has two labels and does not encompass any other interval, thus the number of unambiguous graphs for this interval is its number of labels plus one (for the empty graph): $|\delta(\text{Student})| = 3$. Similarly $|\delta(\text{Union})| = 2$, $|\delta(\text{be})| = 2$, $|\delta(\text{contained})| = 3$ and $|\delta(\text{University})| = 4$. For intervals containing other intervals, such as "Student Union", the number of unambiguous graphs is given by their number of labels in the interval plus the combination of the unambiguous graphs of the encompassed intervals (that do not intersect each other). Thus, $|\delta(\text{Student Union})| = 2 + (|\delta(\text{Student})| \times |\delta(\text{Union})|) = 8$. Similarly, $|\delta(\text{contained within})| = 1 + |\delta(\text{contained})| = 4$. Which we can use to calculate that

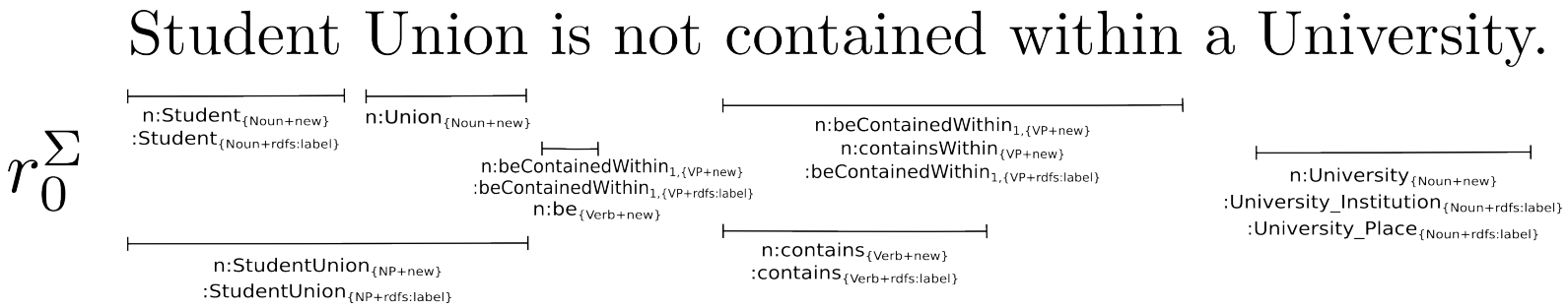

Figure 4.8: Lexical reading for an input sentence. Note that the labels use abbreviated IRIs. The lexical reading shows various ambiguities caused by the use of various entity mapping strategies. The graph also shows that different segments can be mapped to the same IRI: "is" and "contained within" are mapped to IRI `:beContainedWithin`. Note that the `n:` prefix is used to denote a newly coined IRI that was not present in some reference ontology during lexical analysis.

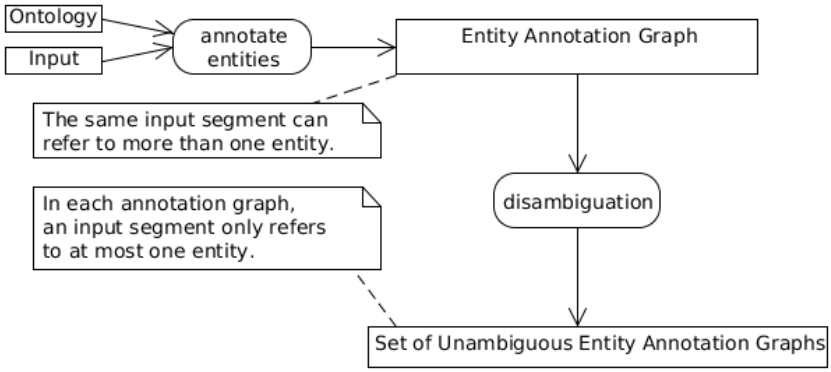

Figure 4.9: Input, processing steps and output of the entity annotation stage in Entendre. Note that this processing is independent of any particular ontology authoring syntax.

Syntactic Readings

Based on one or more unambiguous lexical readings of an input, the input can be syntactically analysed. The result of this analysis is a Syntactic Reading of the input (based on an unambiguous lexical reading). A syntactic reading is similar to a lexical reading: it is also an annotation graph, but the labels in this case refer to assertions in the ontology language (axioms, in OWL terminology) instead of entity names. Accordingly, the labels in syntactic readings usually cover larger segments of the input than the labels in lexical readings.

Definition 3. *Let ι be an input string, r^{Σ} be an unambiguous lexical reading of ι, $\mathcal{L}$ be an ontology language and $\Gamma_{\mathcal{L}}$ be the set of strategies for mapping input segments onto assertions in $\mathcal{L}$; then a **Syntactic Reading** of ι based on r^{Σ} is an annotation graph over ι and a set of labels $L_{\mathcal{L}}$. A label in $L_{\mathcal{L}}$ is a triple $\langle \lambda, n, \gamma \rangle$, where λ is a set of assertions in $\mathcal{L}$, n is a natural number and $\gamma \in \Gamma_{\mathcal{L}}$.*

It is important to note that this definition of syntactic readings is relative

$|\delta(\text{is not contained within})| = 2 + (|\delta(\text{be})| \times |\delta(\text{contained within})|) = 10$. Thus, the total number of unambiguous graphs for the whole graph is given by $|\delta(\text{eg}_0)| = |\delta(\text{Student Union})| \times |\delta(\text{is not contained within})| \times |\delta(\text{University})| = 10 \times 8 \times 4 = 320$.

to an unambiguous lexical reading. As we saw in Section 4.3.1, after the entity annotation stage, we may have a (possibly large) set of unambiguous lexical readings of the input; thus performing syntactic analysis on this set of lexical readings will produce a set of syntactic readings – one syntactic reading for each unambiguous lexical reading.

Note also that the labels in syntactic readings contain *sets of* assertions in the ontology language. Thus, a single input sentence can be mapped to more than one assertion. For example, an instance declarations in Rabbit(e.g. `Leeds is a Place`), is mapped onto two logical axioms in OWL[1]: a class assertion `ClassAssertion(:Place :Leeds)` and an instance declaration `Declaration( NamedIndividual( :Leeds))`.[2][3]

In order to illustrate the concept of syntactic readings, Figure 4.10 shows a syntactic reading of the input shown previously in Figure 4.8. Recall that r_0^{Σ}, the lexical reading for this input was ambiguous and could be decomposed into 320 unambiguous lexical readings. In Figure 4.10 we show a syntactic reading based on one of those unambiguous lexical readings (a subgraph of r_0^{Σ}), which we call r_{42}^{Σ}. The resulting syntactic reading, $r_{42}^{\mathcal{L}}$, associates six labels to the whole input (based on 6 strategies: Rbt+`NSP`, `PME`+Rbt+`NSP`, Rbt, `PME`+Rbt, ManchesterSyntax and ACE) and one label to part of the input (based on strategy BOE+DR). The six strategies are based on three different OWL syntaxes: Rabbit, Manchester Syntax and ACE. None of these syntaxes can parse the input sentence as it does not conform to the expected syntax. We can extend Rabbit to allow `Negative-Split-Passive` verb phrases (Rbt+`NSP`) in order to successfully parse the input sentence. We can even include a heuristic rule `PME` during syntax analysis to prepend the missing keyword "Every" to the input sentence. In Rabbit,

[1]We use the OWL functional syntax

[2]Note that this is partly dependent on the expressiveness and syntax of the ontology language. For example, in the abstract DL syntax there is no notion of a declaration axiom, thus this example would only result in the class assertion `:Place(:Leeds)`.

[3]Besides the 2 logical axioms, the Rabbit parser will also produce 2 more annotation axioms. The first one is attaches a lexical form to the new entity using an `rdfs:Label` property. The second one records the original input using a custom annotation property `rabbit:RelatedSentence`. We omit these two annotation axioms since they are not relevant for the logical interpretation of the input.

the sentence without the "Every"-keyword is interpreted as a class assertion; while the sentence with the keyword is interpreted as a class subsumption. Besides the three OWL syntaxes used, another strategy was applied that does not extend an existing OWL syntax and does not rely on a formal grammar: a bag-of-entities. Such an approach can be useful when the input does not conform to a pre-defined grammar[1]. In this case, the bag-of-entities is used in combination with a strategy to generate domain and range restrictions.

Preferred Input and Rendering Syntax in Ontology Authoring

Lexical and Syntactic readings occur in CNL-based ontology authoring (and also in authoring using e.g. the Manchester Syntax) when authors enter new assertions and the authoring tools analyse this input. As stated in Section 4.2.1, current ontology authoring languages define a language grammar to avoid (or minimise) ambiguity at both the syntactic and lexical levels; in minimising ambiguity, parsers for these languages become unable to recognise some sentences that authors want to enter. Our goal is thus to enable the extension of current parsers to take into account alternative lexical and syntactical analysis strategies in cases where the standard parser fails to recognise an input.

Thus, we assume that during ontology authoring, there will be a *preferred lexical mapping strategy* κ_{pref} and a *preferred syntactic strategy* γ_{pref}. In principle, we allow both of these strategies to produce ambiguous readings although, in practice, current ontology languages do not allow ambiguous syntactic strategies.

Allowing authors to enter ambiguous inputs means Entendre will need to be able to disambiguate the input; often Entendre will then need to confirm the disambiguated sense by consulting with the ontology author. This means we will need to present entity names, assertions (and sets of assertions) in the ontology language to the author; in such cases we require a *rendering syntax*. Thus for an assertion α, we denote the rendering of that assertion with rend(α). We impose the following restriction on the rendering syntax: it must produce unambiguous

[1] Of course, in this example, this approach does not add much value since the `NSP` extension to Rabbit provides a better coverage for the input. However, for some inputs we may not have a suitable extension for Rabbit in which case the BOE strategy can be useful. We include this approach in this example to show that syntactic readings can be used to capture such strategies.

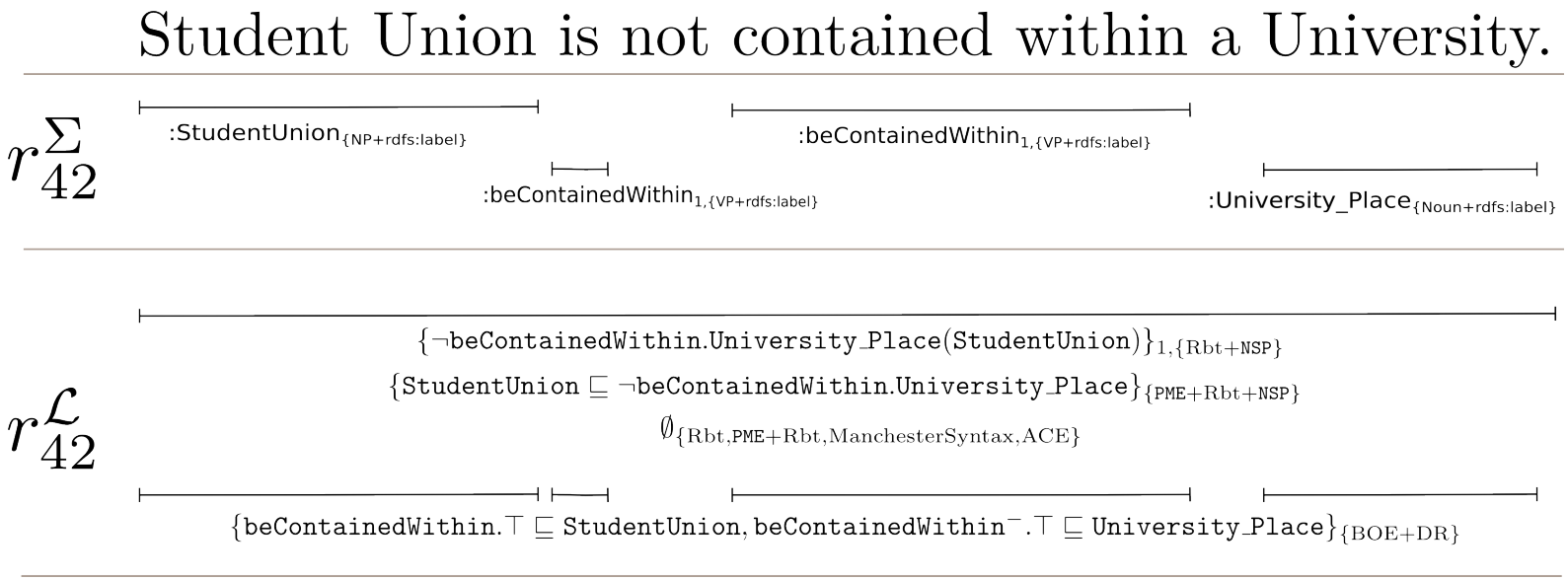

Figure 4.10: Syntactic reading for an input based on an unambiguous lexical reading. From top to bottom: Input, the unambiguous lexical reading r^{Σ}_{42} and the (ambiguous) syntactic reading $r^{\mathcal{L}}_{42}$ over the input. Several syntactic analysis strategies have been used to create $r^{\mathcal{L}}_{42}$: Standard Rabbit(Rbt) does not allow negative verb phrases (the input would have to be rewritten to "Student Union does not be contained within a University"!). In this case, we can introduce an alternative syntactic strategy by extending Rabbit to allow `Negative-Split-Passive`(`NSP`). This new strategy returns a class assertion. By combining the `Prepend-Missing-Every` (PME) heuristic rule, Rabbit and the `NSP` extension a subsumption relation is found. ACE and Manchester Syntax cannot find any matching OWL axioms for the input either. Finally, using a bag-of-entities (BOE) strategy while trying to find domain and range (DR) restrictions, maps several intervals of the input to a set of two assertions (but does not cover the whole input).

renderings in the context of a reference ontology. Thus, for rend(α), there must be a lexical strategy κ_{rend} and syntactic strategy γ_{rend} that can be used to analyse rend(α) and the result of that analysis is an unambiguous syntactic reading with a single label: $\langle\{\alpha\}, 1, \gamma_{\text{rend}}\rangle$. Similarly, we will write rend(σ) for the unambiguous rendering of an entity name σ and rend(λ) for the unambiguous rendering of a set of axioms λ.

Note that the rendering syntax can be very different from the preferred input language. In this thesis, however, we choose for the rendering syntax to be the same as the preferred input language (e.g. Rabbit). We conjecture that keeping the rendering and input syntaxes close to each other will make it easier for ontology authors to understand the Entendre feedback.

Lexical and Syntactic Readings in Ontology Authoring

As stated above, lexical and syntactic readings are created during syntactic analysis of ontology authors' inputs. Since most ontology languages are unambiguous, most lexical and syntactic readings based on these languages will also be unambiguous. Allowing for different lexical and syntactic strategies will result in ambiguous readings and increase the analysis that is required to understand the input.

A naive solution is thus to analyse all inputs using all the available lexical and syntactic strategies as this will generate highly ambiguous readings for the input. The approach that we advocate with Entendre is rather to start from the preferred input language, given by κ_{pref} and γ_{pref}, and to allow tool developers to specify alternative strategies that should be tried. Entendre then makes it easier to determine when alternative strategies should be applied and allows tool developers to create heuristics to apply extra strategies depending on the current readings. To make this possible Entendre proposes a number of categories of readings which we introduce below.

Categories of Readings and Syntactic Feedback

In this section we introduce categories to help us differentiate between syntactic (and lexical) readings. These categories help us to:

- define how different readings for the same (or a similar) input relate to each other;
- understand how we can extend (or narrow) the syntactic analysis to find other possible readings (or to make the current reading more precise);
- build a bridge between the syntax and the semantics of the inputs by identifying which readings can be further analysed using semantics;[1]
- identify feedback that can help domain experts to rephrase the input.

The categories of readings will be defined based on features of the annotation graphs such as ambiguity, input coverage, number of nodes and the number of the assertions in a label. For each reading category, we will describe which problems must be solved (if any) in order to use the input during ontology authoring. Also, for each category we describe how feedback for ontology authors can be generated for readings in that category.

The first distinction we make between readings is whether they are lexically ambiguous or not. Let r^{Σ} be a lexical reading of a non-empty ontology author's input ι. Then we can categorise r^{Σ} as follows:

[1] In the next section we will see how we can define categories based on semantics of the ontology language.

Ambiguous Lexical Reading

Definition: r^{Σ} is an ambiguous annotation graph (see Definition 1).
Feedback: [Provide feedback based on a set of "viable" lexical readings $\mathcal{G}^{\Sigma}_{\text{viable}}$ for r^{Σ}. We initialise $\mathcal{G}^{\Sigma}_{\text{viable}}$ as the set of lexical readings $r^{\Sigma}_{\text{sub}} \in \delta(r^{\Sigma})$ such that there is a $r^{\mathcal{L}}$ that is based on r^{Σ}_{sub} and that is Understood (as defined below). We then have the following options:

- If $\mathcal{G}^{\Sigma}_{\text{viable}} = \emptyset$:
  - try to extend the set of viable lexical readings. First by also allowing lexical readings that lead to Partially Understood syntactic readings. If that is also empty, we can try to extend one or more of the syntactic readings based on r^{Σ} by using an additional syntactic strategy that has not been used yet. If the set of variable lexical readings is still empty, we can extend r^{Σ} itself by using an additional lexical strategy that has not been used yet. After every extension we need to evaluate $\mathcal{G}^{\Sigma}_{\text{viable}}$ again.
  - When $\mathcal{G}^{\Sigma}_{\text{viable}}$ cannot be extended because all available strategies have been used: notify the user that the input cannot be understood as an assertion using $\Gamma_{\mathcal{L}}$. State that the input seems to mention one of the following entities in the reference ontology: rend(σ) for all σ that appear in labels of r^{Σ}.
- If $|\mathcal{G}^{\Sigma}_{\text{viable}}| = 1$, present the feedback for the lexical reading in $\mathcal{G}^{\Sigma}_{\text{viable}}$.
- If $|\mathcal{G}^{\Sigma}_{\text{viable}}| > 1$, state that the input is ambiguous and allow user to choose between the options by showing the feedback for each r^{Σ}_{sub} in $\mathcal{G}^{\Sigma}_{\text{viable}}$. This works best when $|\mathcal{G}^{\Sigma}_{\text{viable}}|$ is small (2 or 3 options). When the set is larger, presenting the full feedback for each r^{Σ}_{sub} will result in too much feedback for the author. In that case we can reduce the amount of information by:
  - only showing the unambiguous renderings for the different r^{Σ}_{sub}.
  - using a partial ordering on $\mathcal{G}^{\Sigma}_{\text{viable}}$ to select a subset of $\mathcal{G}^{\Sigma}_{\text{viable}}$ for which to show feedback. The partial ordering should maximise the use of κ_{pref} and γ_{pref} while minimising semantic defects (see Section 4.3.2).

Defect warning: Rewrite input to make clear which terms you are using (note that the provided feedback will include unambiguous renderings of the input that are unambiguous).

Unambiguous Lexical Reading

Definition: r^{Σ} is an unambiguous annotation graph.
Feedback: [Present the feedback for the syntactic reading $r^{\mathcal{L}}$ based on r^{Σ}. If $r^{\mathcal{L}}$ does not lead to any Understood readings, $r^{\mathcal{L}}$ can be extended by applying an unused syntactic strategy, or r^{Σ} can be extended by applying an unused lexical strategy (this will result in a new ambiguous lexical reading $r^{\Sigma}_{\text{extended}}$ such that $r^{\Sigma} \in \delta(r^{\Sigma}_{\text{extended}})$). Optionally, present a summary of facts known about the entities referenced in r^{Σ} to presented feedback into context.]
Defect warning: [For each label $\langle \sigma, n, \kappa \rangle$ in r^{Σ} such that $\kappa \neq \kappa_{\text{pref}}$, issue the following warnings:

- if the corresponding input segment is not equal to rend(σ), warn that the preferred way to write entity σ is rend(σ) as this is unambiguous.
- if σ is not referenced by the reference ontology $\mathcal{O}$, state that rend(σ) has not been declared in the ontology yet.

Note that an ambiguous lexical reading can be decomposed into a set of 2 or more unambiguous lexical readings. As explained in the previous section, each unambiguous lexical reading can produce a syntactic reading through syntactic analysis. This means that lexically unambiguous readings can be further categorised based on the resulting syntactic reading. Let ι be an input, r^{Σ} be an **unambiguous** lexical reading and $r^{\mathcal{L}}$ be a syntactic reading of ι based on r^{Σ}, then we define the following syntactic reading categories:

Ambiguous Syntactic Reading

Definition: $r^{\mathcal{L}}$ is ambiguous (see Definition 1).
Feedback: [Provide feedback based on a set of "viable" syntactic readings $\mathcal{G}^{\mathcal{L}}_{\text{viable}}$ for $r^{\mathcal{L}}$. We initialise $\mathcal{G}^{\mathcal{L}}_{\text{viable}}$ as the set of syntactic readings $r^{\mathcal{L}}_{\text{sub}} \in \delta(r^{\mathcal{L}})$ such that $r^{\mathcal{L}}_{\text{sub}}$ is Understood (as defined below). Our options for presenting $\mathcal{G}^{\mathcal{L}}_{\text{viable}}$ are anaologous to those of Ambiguous Lexical Readings.]
Defect warning: Rewrite input to make clear what you mean (Note that feedback will provide an unambiguous ways to rephrase the input using the rendering syntax).

Syntactically Unambiguous

Definition: $r^{\mathcal{L}}$ is unambiguous.
Feedback: [Present coverage-based feedback for $r^{\mathcal{L}}$]
Defect warning: same as feedback.

Syntactically ambiguous inputs can thus be seen as a set of two or more syntactically unambiguous inputs. We now further categorise syntactically un-

ambiguous inputs by looking at the *coverage* of the assertion annotation graph over the input. Let ι be an ontology author's input, r^{Σ} an unambiguous lexical reading of ι, $r^{\mathcal{L}}$ an **unambiguous** syntactic reading of ι based on r^{Σ}, then we define the following coverage-based categories for $r^{\mathcal{L}}$:

Not Understood
Definition: $r^{\mathcal{L}}$ is the empty graph (the set of nodes or arcs are empty) or, if $r^{\mathcal{L}}$ is not the empty graph, all labels in $r^{\mathcal{L}}$ are of the form $\langle \emptyset, n, \gamma \rangle$, where n is any number and $\gamma \in \Gamma_{\mathcal{L}}$ (i.e. ι could not be mapped onto an assertion in the ontology language). **Feedback:** This input cannot be understood in terms of $\mathcal{O}$ and the following ontology syntaxes [$\Gamma_{\mathcal{L}}$]. [Optionally, provide auto-completion feedback if available in a supported syntax]. **Defect warning:** same as feedback.

Understood
Definition: $r^{\mathcal{L}}$ completely covers ι. This is the case when $r^{\mathcal{L}}$ satisfies the following conditions: • $r^{\mathcal{L}}$ has at least two nodes • there is a node n_0 in $r^{\mathcal{L}}$, such that $\tau(n_0) = 0$ (because $r^{\mathcal{L}}$ is unambiguous, it follows that n_0 has indegree 0 and outdegree 1) • there is a node n_m in $r^{\mathcal{L}}$ such that $\tau(n_m) = j$, where j is the size of ι (because $r^{\mathcal{L}}$ is Unambiguous it follows that n_m has indegree 1 and outdegree 0) • all nodes in $r^{\mathcal{L}}$ except n_0 and n_m have indegree 1 and outdegree 1 • all labels in $r^{\mathcal{L}}$ are of the form $\langle \lambda, n, \gamma \rangle$ such that $\lambda \neq \emptyset$, n is a number and $\gamma \in \Gamma_{\mathcal{L}}$ **Feedback:** The input can be understood in terms of $\mathcal{O}$. [Arc-atomicity-based feedback for $r^{\mathcal{L}}$]. [ι can only be understood because the following assumptions $\{\gamma \mid \gamma \in \Gamma_{\mathcal{L}}$ and γ is used in $r^{\mathcal{L}}\}$ were made.]

Partially Understood
Definition: All other cases besides **Not Understood** or **Understood** (i.e. some segment of ι could not be mapped to an assertion in the ontology language; either because it is not covered by an arc in $r^{\mathcal{L}}$ or because $r^{\mathcal{L}}$ contains a label mapping a segment of ι to the empty set). **Feedback:** Your input cannot be fully understood in terms of $\mathcal{O}$. The following parts were not understood: [unannotated segments of ι]. If we ignore those segments, your input becomes $\mathsf{trimmed}(\iota)$ which can be interpreted as: [Present feedback for $\mathsf{rewrite}(r^{\mathcal{L}})$, this is a fully Understood rewriting of $r^{\mathcal{L}}$ over $\mathsf{trimmed}(\iota)$.] **Defect warning:** Should we remove the segments that were not understood? (Note that the feedback for the syntactic reading based on $r^{\mathcal{L}}$ without the unannotated segments will include a rendering of the input without the segments that were not understood)

Note that syntactic readings that are Not Understood cannot be further analysed. Partially Understood syntactic readings can be further analysed by "removing" the segments that were not understood from the input; the result is a unambiguous syntactic reading that is Understood and can be further analysed. Partially Understood readings can occur for example when the input sentence contains extra words (e.g. keywords) that are not necessary according to the grammar of the input language. In this case omitting the extra words results in a valid sentence. Partially Understood readings can also be the result of inputs containing multiple sentences, where one of the sentences is not in the input language. In such a case we can ignore the "faulty" sentence and focus on the correct sentences instead (the author can always re-enter the "faulty" sentence after the other sentences have been added to the ontology).

Once we have a reading that is Understood (or Partially Understood), we also have determined a set of lexical and syntactic strategies under which an input can be unambiguously understood as a set of ontological assertions. In principle we can now perform semantic analysis of this set of assertions using ontology integration [75] techniques. However, since the usability of ontology integration has not been explored, we choose in this thesis to focus on the analysis of integrating single assertions into the ontology. The motivation for this choice is that, if authors cannot understand the logical issues of integrating a single assertion to an ontology, then understanding the issues of adding a set of assertions will be even harder.

The above means we need to decompose the Understood readings in such a way that we can provide feedback for each individual assertion in that reading. We do this by further categorising the Understood syntactic readings based on the number of nodes it has (as an annotation graph). The feedback produced by this categorisation is the arc-atomicity-based feedback. Let ι be an ontology author's input, $r^{\mathcal{L}}$ a syntactic reading of ι that is **Understood**; then we define the following categories of syntactic readings:

Syntactically Composite
Definition: $r^{\mathcal{L}}$ has m nodes, where $m > 2$. **Feedback:** ι consists of $m-1$ sentences: [Present feedback for each of the readings in the *sequential decomposition* of $r^{\mathcal{L}}$, $\mathsf{decompose}_{\text{seq}}(r^{\mathcal{L}})$, which is given the algorithm below.].
Syntactically Atomic
Definition: $r^{\mathcal{L}}$ has exactly 2 nodes. **Feedback:** [$\mathcal{L}$-atomicity feedback for $r^{\mathcal{L}}$]

Algorithm 1 Sequential decomposition of a syntactically composite reading

Input: $r^{\mathcal{L}}$, a syntactically composite reading of the form $\langle N, A, \tau, \mathcal{O}\rangle$, with $N = \{n_1, n_2, \ldots, n_m\}$ and $A = a_1, \ldots, a_{m-1}$. For $1 \leq i \leq m-1$, a_i is the arc between n_i and n_{i+1} and has a label of the form $\langle \lambda_i, n, \kappa\rangle$. Also, τ_i is a function based on τ so that, for all j, $\tau_i(n_j) = \tau(n_j) - \tau(n_i)$.

Output: $\mathsf{decompose}_{\text{seq}}(r^{\mathcal{L}})$

$\mathcal{O}_c \leftarrow \mathcal{O}$
$R \leftarrow \emptyset$
for $i = 1 \rightarrow m-1$ **do**
  $r_i^{\mathcal{L}} \leftarrow \langle \{n_i, n_{i+1}\}, \{a_i\}, \tau_i, \mathcal{O}_c\rangle$
  /** $r_i^{\mathcal{L}}$ is defined over ι_i: the segment of ι between $\tau(n_i)$ and $\tau(n_{i+1})$. */
  $R \leftarrow R \cup \{r_i^{\mathcal{L}}\}$
  $\mathcal{O}_c \leftarrow \mathcal{O}_c \cup \lambda_i$
end for
return R

Note that the sequential decomposition of a syntactically composite reading consists of 2 or more syntactically atomic readings. The atomic readings in the decomposition are not simply subgraphs as their semantics need to be evaluated

based on more than just the reference ontology $\mathcal{O}$: the assertions of the previous sentences in the input also have to be taken into account.

Finally, we further categorise syntactically atomic readings based on the number of assertions that are in the label of the syntactic reading. Let ι be an ontology author's input and $r^{\mathcal{L}}$ an syntactic reading of ι that is syntactically atomic; then we define:

$\mathcal{L}$-Composite
Definition: the arc in $r^{\mathcal{L}}$ has a label $\langle \lambda, n, \gamma \rangle$ such that $\lvert\lambda\rvert > 1$.
Feedback: Sentence ι means more than one thing: [Present feedback for each of the readings of the $\mathcal{L}$-decomposition of $r^{\mathcal{L}}$, $\mathsf{decompose}_{\mathcal{L}}(r^{\mathcal{L}})$, which is given by the algorithm below.]

$\mathcal{L}$-Atomic
Definition: $r^{\mathcal{L}}$ has label $\langle \lambda, n, \gamma \rangle$ such that $\lvert\lambda\rvert = 1$.
Feedback: [Semantic feedback for $\alpha \in \lambda$]

Algorithm 2 $\mathcal{L}$-decomposition of a $\mathcal{L}$-composite reading

Input: $r^{\mathcal{L}}$, a $\mathcal{L}$-composite reading of the form $\langle \{n_1, n_2\}, \{a\}, \tau, \mathcal{O} \rangle$, where a is an arc between n_1 and n_2 with a label of the form $\langle \lambda, 1, \kappa \rangle$ where $\lambda = \{\alpha_1, \ldots, \alpha_m\}$, where $m > 1$.

Output: $\mathsf{decompose}_{\mathcal{L}}(r^{\mathcal{L}})$

$\mathcal{O}_c \leftarrow \mathcal{O}$
$R \leftarrow \emptyset$
for $i = 1 \rightarrow m$ **do**
  $r_i^{\mathcal{L}} \leftarrow \langle \{n_1, n_2\}, \{a_i\}, \tau, \mathcal{O}_c \rangle$
  /** a_i is an arc between n_1 and n_2 with a label of the form $\langle \{\alpha_i\}, 1, \kappa \rangle$ defined over ι. */
  $R \leftarrow R \cup \{r_i^{\mathcal{L}}\}$
  $\mathcal{O}_c \leftarrow \mathcal{O}_c \cup \{\alpha_i\}$
end for
return R

Note that an $\mathcal{L}$-composite reading $r^{\mathcal{L}}$ can be decomposed into a collection of $\mathcal{L}$-atomic readings through $\mathsf{decompose}_{\mathcal{L}}(r^{\mathcal{L}})$. The readings in the $\mathcal{L}$-decomposition are similar to subgraphs, but they take into account all of the assertions in the $\mathcal{L}$-composite reading.

Overview of Reading Categories

The syntactic categories defined above form a network of readings that can be traversed to provide feedback about the input. By applying syntactic analysis, one can move from a lexical reading to a syntactic reading. Both lexical and syntactic readings can be expanded by applying a new lexical or syntactic strategy. By following relations between readings such as annotation graph disambiguation δ, sequential decomposition and $\mathcal{L}$-decomposition generally one reaches readings that use fewer strategies and are more specific. The two categories that cannot be further simplified are **Not Understood** and **$\mathcal{L}$-atomic**.

To illustrate such a syntactic reading network, Figure 4.11 shows part of the resulting network for our running example. The topmost reading in the figure is the lexical reading r_0^{Σ} which we showed in Figure 4.8. Via lexical disambiguation we get 320 unambiguous lexical readings, which we can use to produce 320 syntactic readings. Most of these syntactic readings, such as $r_1^{\mathcal{L}}$, are Not Understood. Other syntactic readings, such as $r_{42}^{\mathcal{L}}$ (this graph was shown in Figure 4.10), are syntactically ambiguous and can be further disambiguated, into $r_{42a}^{\mathcal{L}} \ldots r_{42h}^{\mathcal{L}}$.

The figure shows that Unambiguous Syntactic Readings can be Not Understood, such as $r_{42a}^{\mathcal{L}}$ which could be the reading based on the Manchester Syntax. The reading based on Rabbit+NSP ($r_{42h}^{\mathcal{L}}$) is fully Understood and also $\mathcal{L}$-Atomic. The Partially Understood reading ($r_{42f}^{\mathcal{L}}$), is the result of applying the bag-of-entities syntactic strategy. This reading can be rewritten as a fully understood syntactic reading, rewrite($r_{42f}^{\mathcal{L}}$), that is defined over the trimmed input "Student Union is contained within University". This last reading is an $\mathcal{L}$-composite reading (that is further decomposed into two $\mathcal{L}$-atomic readings, one for the domain and one for the range assertion).

We will now use the running example to illustrate how Entendre can be used to provide feedback. As we have noted before, the final network of readings produced by Entendre will depend on the configuration of Entendre – i.e. which lexical and syntactic strategies are available and how these strategies are triggered. For the exact network as shown in Figure 4.11, the feedback would be generated based on r_0^{Σ} (recall that the input is "Student Union is not contained within a University"):

Figure 4.11: Syntactic Reading network for the input sentence `Student Union is not contained within a University`.

Feedback based on reading: r_0^{Σ} (Ambiguous Lexical Reading)
Syntactic Feedback: This input is ambiguous in the context of the "Leeds Point of Interest" ontology. Which of the following four options do you mean: 1. No Student Union is contained within a University(Place). 2. No Student Union is contained within a University(Institution). 3. Student Union is not contained within a University(Place). 4. Student Union is not contained within a University(Institution). Rewrite your input as one of the given options (or as a different sentence) to conform to the Rabbit syntax and to make clear which terms from the ontology you are referring to.
Feedback explanation: r_0^{Σ} is highly ambiguous and the set of viable sub readings is also large. The set of viable readings can be narrowed down by looking at those readings that maximise the use of the preferred lexical (those readings that match the `rdfs:label` of entities in the ontology) and syntactic strategies (those readings based on Rabbit). All readings based on novel entities (e.g. `n:StudentUnion`) will be hidden as well as the readings based on Manchester Syntax, ACE and Bag-of-Entities. This remaining four readings are shown using the default Rabbit rendering. Note that this rendering is quite different from the original input. Note that we could further improve on this result by including the semantic analysis to rank the alternative readings, in which case the mapping to concept `University(Institution)` would lead to an inconsistency (in option 4) or to an unsatisfiable concept(in option 2). Also, since `Student Union` is a concept in the ontology and not an individual, option 3 would introduce a pun. Hence, option 1 would be the top ranked reading and would be chosen as the one to further explain. Since we have not introduced the semantic analysis yet, we have not used this analysis for generating the feedback shown here.).

To illustrate other types of feedback based on the running example, we will add some assumptions in order to get a different network of readings.

Assume for example that the preferred input syntax is the Manchester Syntax and that no other syntactic strategy is available. Assume also that all the lexical strategies are available, but that the ontology does not have concept `University(Institution)`(in order to avoid ambiguity). The feedback would look as follows:

Feedback based on reading: $r_{42a}^{\mathcal{L}}$ (Not Understood)
Syntactic Feedback: Your input cannot be understood in terms of the "Leeds Points of Interest" ontology and the Manchester Syntax. It looks like you are trying to say something about the following concepts and relations in the ontology: concept `Student Union`, relation `is contained within` and concept `University(Place)`. Revise the documentation for the Manchester Syntax to learn how to formulate valid sentences

As another example, assume that we do not have the `Negative-Split-Passive` extension to Rabbit. Also assume that the reference ontology does not contain concept `University(Institution)`. In such a case, only the Bag-of-Entities syntactic strategy will succeed in finding a set of assertions. The lexical feedback would look as follows:

Feedback based on reading: $r^{\mathcal{L}}_{42f}$ (Partially Understood)
Syntactic Feedback: Your input cannot be fully understood in terms of the "Leeds Points of Interest" ontology and the supported syntaxes Rabbit, Manchester Syntax or ACE. By looking at the concepts and relations in your input we make a guess at what you mean. The following parts were not understood: "not", "a". If we ignore these parts your input becomes "Student Union is contained within University", which possibly means the following in Rabbit: • The relationship "is contained within" must have a Student Union as its subject. • The relationship "is contained within" must have a University(Place) as its object. Is this interpretation of your input correct?

Summary

This section presented a generic framework for describing the syntactic analysis of an ontology author's input. The framework describes the results of lexical and syntactic analysis in terms of annotation graphs, which keep track of the analysis strategies that have been used as well as the possible syntactic ambiguities that can occur. To facilitate the generation of feedback messages regarding syntactic issues, we defined a categorisation of syntactic analysis results and described the relations between the categories.

The presented framework provides a principled way to describe the syntactic analysis of ontology authors' inputs, a key requirement for providing appropriate tool support to ontology authors, and in particular to domain experts as discussed in the previous chapter. As this section showed, the framework provides a systematic way of describing the types of syntactic ambiguity that have to be taken into account when analysing inputs, improving on the ad-hoc manner described in Section 3.3.2. The framework also facilitates the systematic addition of new lexical and syntactical analysis strategies into existing tools, opening the way to more robust parsers that provide better feedback about syntactic issues.

4.3.2 Semantic Axiom Integration Analysis

$\mathcal{L}$-atomic syntactic readings can be semantically analysed. The result of such a semantic analysis is a semantic reading of the input. In this case, it is not directly apparent what should be the set of semantic analysis strategies to be used. Also, if we choose to represent a semantic reading as an annotation graph (like the

lexical and syntactic readings), then it is not clear what the labels (or semantic consequences) should be.

Since we want to add assertions to the reference ontology, the type of semantic analysis we want to perform is similar to that performed during ontology integration tasks. We therefore introduce a semantic analysis strategy based on the $\mathcal{SROIQ}$ description logics language, a common and powerful ontology language that underpins OWL. Although this strategy is specific to $\mathcal{SROIQ}$, similar strategies could be defined for other ontology languages. This analysis strategy will result in a categorisation of logical consequences that we can use to provide semantic feedback.

Note that this categorisation is based on our assumption that the input being analysed is meant to be added to an ontology under construction[1]. Furthermore, we focus on analysing logical consequences that can uncover potential logical errors that can be introduced into the ontology; we shortly discuss these ontology defects. After introducing the categorisation, we provide a formal definition of a semantic reading.

Preliminaries

We assume that an ontology $\mathcal{O}$ is built in a language $\mathcal{L}$ that is the $\mathcal{SROIQ}$ description logic(DL)[2]. An axiom α is an assertion in $\mathcal{L}$ which is a well-formed formula.

$\mathcal{O}$ is a finite set of axioms in $\mathcal{L}$. C denotes a concept; R denotes a role, a denotes an individual, and Θ denotes a set of concepts. $\top$ denotes the top concept, i.e. every individual is a member of $\top$, $\bot$ denotes the bottom concept, i.e. no individual is a member of $\bot$, and U denotes the top role. $TBox$ represents all axioms in $\mathcal{O}$ which relate concepts to each other. $RBox$ represents all axioms

[1] A mirror analysis could be performed when removing assertions from an ontology. However we do not consider this case explicitly in this thesis, as the user input in such cases would generally not be a CNL input; rather the user would typically select an existing assertion from an ontology and decide whether to edit or remove it.

[2] Although our approach is suitable for other monotonic DL languages, such as $\mathcal{ALC}$ or $\mathcal{SHOIN}$.

in $\mathcal{O}$ which define role hierarchies and characteristics. $ABox$ represents all axioms which make assertions about individuals.

Λ will be used to denote any set of axioms in $\mathcal{L}$, and a corresponding superscript can be used to assign an additional label to the set. We will use $\models$ to denote that an axiom can be entailed from an ontology (i.e. $\mathcal{O} \models \alpha$), using some appropriate derivation mechanism (e.g. a DL reasoner). $\mathcal{O}^*$ is the deductive closure of $\mathcal{O}$, i.e. the set of axioms in $\mathcal{L}$ that can be entailed by the axioms in $\mathcal{O}$. When an axiom α is entailed from $\mathcal{O}$, we denote with $\mathcal{J}(\alpha, \mathcal{O})$ the **justification** comprising the minimum set of axioms in $\mathcal{O}$ that is sufficient for α to hold. Further definitions and examples of justification are given in [69].

Ontology Defects

Adding α to $\mathcal{O}$, certain defects can be introduced. Entendre considers that:

- $\mathcal{O}$ is **inconsistent** when it includes a set of axioms from $ABox$ which contradict with axioms in $TBox$ or $RBox$.
- $\mathcal{O}$ includes a concept C that is **unsatisfiable**, i.e. $\mathcal{O} \models C \equiv \bot$.
- $\mathcal{O}$ includes an axiom α that is **redundant**, i.e. $\mathcal{O} \setminus \{\alpha\} \models \alpha$.
- $\mathcal{O}$ includes **isolated entities**, i.e. $\mathcal{O}$ can have an isolated concept C that only occurs in the axiom $C \sqsubseteq \top$, or an isolated role R that only occurs in the axiom $R \sqsubseteq U$, or an isolated individual occurring only in the axiom $\top(a)$.

The above defects can be detected *explicitly* using DL reasoners. Inconsistency is considered a defect because it makes further reasoning about the ontology impossible (an inconsistent ontology entails everything). It is possible to reason with ontologies that contain unsatisfiable concepts. However, we consider this a defect, as concepts are usually intended to be satisfiable. Redundant axioms and isolated entities are not necessarily wrong, but they make the ontology cluttered and less concise. Hence, axiom redundancy and entity isolation are considered as bad practice, which should be avoided.

There can also be subjective defects caused by the ontology author's limited understanding of the semantics of the ontology language. Such defects are *not possible to detect* using DL reasoners. Only the authors themselves can discover these defects, e.g. by noticing **unintended inferences** (i.e. inferences that the author considers to be false) or **missing inferences** (i.e. inferences that the author expects to be inferred). Therefore, such defects may be pointed *implicitly* to the ontology author by listing the implications of new axioms.

Axiom Categories and Semantic Feedback

We now define a semantic analysis strategy called $\mathcal{DL}$-axiom-integration that, given a consistent ontology $\mathcal{O}$ and an axiom α in the ontology language $\mathcal{L}$, diagnoses the impact of adding α to $\mathcal{O}$ taking into account the various ontology defects presented above. This will allow *interpreting* the ontology author's input and generating corresponding semantic feedback.[1]

Axioms added to an ontology can be either known or novel: α is **known** by $\mathcal{O}$ when $\alpha \in \mathcal{O}^*$, otherwise α is **novel**. Known axioms can be split into two categories – asserted and inferred.

(A) Asserted Axiom
Definition: $\alpha \in \mathcal{O}$ **Detection:** same as definition. **Feedback:** α is already in $\mathcal{O}$. **Defect warning:** Adding α to the ontology $\mathcal{O}$ is not needed.

(R) Inferred Axiom
Definition: $\alpha \in \mathcal{O}^* \setminus \mathcal{O}$ **Detection:** same as definition. **Feedback:** α is redundant as it can be inferred from $\mathcal{O}$. A set of axioms in $\mathcal{O}$ that implies α is the justification $\Lambda^{\text{justification}} = \mathcal{J}(\alpha, \mathcal{O})$. **Defect warning:** Adding α to $\mathcal{O}$ causes redundancy. Check the axioms in $\Lambda^{\text{justification}}$.

Adding a novel axiom to an ontology will always lead to an infinite number of further implications, Δ_α. We define a subset of these new inferences $\Delta_\alpha^{\mathcal{E}}$ to

[1] The feedback is intended to *inform* authors about potential issues (it may not provide all the information necessary to *resolve* the issues). Because of this, we only show one justification even if there are multiple justifications for an entailment.

represent the finite **set of new relevant implications** brought by adding α to $\mathcal{O}$, i.e.

$\Delta_\alpha^\mathcal{E} = \mathcal{E}_{\mathcal{O}+\alpha} - \mathcal{E}_\mathcal{O}$, where $\mathcal{E}_\mathcal{P}$ is the set of "relevant axioms" entailed by an ontology $\mathcal{P}$. We define relevant axioms as those of the form $A \sqsubseteq B$, $\top \sqsubseteq B$, $A \sqsubseteq \bot$ and $A(a)$ such that A and B are concept expressions[1] that appear in some axiom in $\mathcal{O} \cup \{\alpha\}$ and a is a named individual in $\mathcal{O} \cup \{\alpha\}$. Note that $\Delta_\alpha^\mathcal{E}$ is finite, because the set of concept expressions and individuals appearing in $\mathcal{O} \cup \{\alpha\}$ is finite.

A subset of $\Delta_\alpha^\mathcal{E}$ that is of particular interest is the set of axioms of the form $C \sqsubseteq \bot$, where C is a named concept, as this set helps us to identify the set of new unsatisfiable concepts: $\Theta^{\text{newUnsatisfiable}} = \{C \mid C \sqsubseteq \bot \in \Delta_\alpha^\mathcal{E}\}$.

Adding a new axiom can thus also make the ontology inconsistent or create an unsatisfiable concept. These categories, and the corresponding semantic feedback are defined below.

[1]We use concept expressions here instead of only named concepts. Using only named concepts is more common in the literature and is easier to compute as there are less axioms to be generated. However, we argue that ontologies often make heavy use of concept expressions to describe other concepts without introducing names for those concept expressions. This means that new entailed axioms involving those concept expressions are relevant and should be reported to the ontology author. This is why choose here for the more expensive option, in order to maximise our chances of detecting relevant implications.

(I) Axiom Leading to Inconsistency
Definition: $\alpha \notin \mathcal{O}^*$ and $\mathcal{O} \cup \{\alpha\}$ is inconsistent. **Detection:** $\mathcal{O} \cup \{\alpha\}$ is inconsistent. **Feedback:** α is novel to $\mathcal{O}$. Adding α to $\mathcal{O}$ leads to an inconsistent ontology. The set of axioms in $\mathcal{O}$ that implies $\neg\alpha$ is the justification $\Lambda^{\text{justification}} = \mathcal{J}(\neg\alpha, \mathcal{O})$. **Defect warning:** Check the axioms in $\Lambda^{\text{justification}}$.
(N) Novel Axiom without new Relevant Implications
Definition: $\alpha \notin \mathcal{O}^*$, $\mathcal{O} \cup \{\alpha\}$ is consistent and $\Delta_\alpha^{\mathcal{E}} = \emptyset$ **Detection:** $\Delta_\alpha^{\mathcal{E}} = \emptyset$ **Feedback:** α is novel to $\mathcal{O}$. Adding α to $\mathcal{O}$ does not bring new relevant implications. **Defect warning:** If any entailments were expected, α should be reviewed or $\mathcal{O}$ may have to be extended.
(N+) Novel Axiom with new Relevant Implications
Definition: $\alpha \notin \mathcal{O}^*$, $\mathcal{O} \cup \{\alpha\}$ is consistent, $\Delta_\alpha^{\mathcal{E}} \neq \emptyset$ and $\Theta^{\text{newUnsatisfiable}} = \emptyset$ **Detection:** $\Delta_\alpha^{\mathcal{E}} \neq \emptyset$ and $\Theta^{\text{newUnsatisfiable}} = \emptyset$ **Feedback:** α is novel to $\mathcal{O}$. Adding α to $\mathcal{O}$ brings the set of new relevant implications Λ_α. **Defect warning:** Check that there are no missing or unexpected implications in Λ_α.
(U) Axiom Introducing Unsatisfiable Concept
Definition: $\alpha \notin \mathcal{O}^*$, $\mathcal{O} \cup \{\alpha\}$ is consistent and $\Theta^{\text{newUnsatisfiable}} \neq \emptyset$ **Detection:** $\Theta^{\text{newUnsatisfiable}} \neq \emptyset$ **Feedback:** α is novel to $\mathcal{O}$. Adding α to $\mathcal{O}$ makes the concepts $\Theta^{\text{newUnsatisfiable}}$ unsatisfiable. For each concept $C \in \Theta^{\text{newUnsatisfiable}}$, the set of axioms that makes C unsatisfiable is the justification $\Lambda^{\text{justification}} = \mathcal{J}(C \sqsubseteq \bot, \mathcal{O} \cup \{\alpha\})$. **Defect warning:** Check the axioms in $\Lambda^{\text{justification}}$.

Note that the notation $\neg\alpha$ used in the feedback for axiom category **I** is not standard in description logics. We use it to denote:

- the *opposite of an assertion*: if α is of the form $C(a)$, then $\neg\alpha$ is $\neg(C(a))$ or
- *a counter example of a T-Box (or R-Box) axiom*: if α is of the form $A \sqsubseteq B$, then $\neg\alpha$ is an axiom of the form $(\neg B \sqcap A)(a)$, where a is an individual in the signature of the reference ontology. Note that a given ontology may contain more than one counter example for a given T-Box axiom.

This concludes the definition of axiom categories. Figure 4.12 shows an

overview of the semantic analysis, including the axiom categories defined above and how these are used to generate semantic feedback for the ontology author.

Semantic Readings

Now that we have defined the $\mathcal{DL}$-axiom-integration semantic analysis strategy we can define semantic readings in terms of similar strategies.

Definition 4. *Let ι be an ontology author's input and $r^{\mathcal{L}}$ be an $\mathcal{L}$-atomic syntactic reading of ι over a reference ontology $\mathcal{O}$; then a* ***Semantic Reading*** *of ι is an annotation graph over a set of labels $L_{\mathcal{C}}$. A label in $L_{\mathcal{C}}$ is a triple $\langle \mathcal{C}, n, \varphi \rangle$, where:*

- *n is a natural number;*
- *$\varphi \in \Gamma_{\mathcal{C}}$ is a semantic analysis strategy: it classifies an input axiom based on its logical relation (as defined by the semantics of the ontology language) to the reference ontology;*
- *$\mathcal{C}$ is a (meta) assertion of the form $\alpha \in X$, where α is the ontological assertion in $r^{\mathcal{L}}$ and X is an axiom category defined by φ.*

A semantic reading based on the $\mathcal{DL}$-axiom-integration strategy could, for example, assert that $\alpha \in \mathrm{N}$ or that $\alpha \in \mathrm{U}$. Alternative semantic analysis strategies can be defined, for example we could define $\mathcal{EL}$-axiom-integration that restricts the ontology language to $\mathcal{EL}$ instead of $\mathcal{SROIQ}$. Such a strategy would share many of the same axiom categories, but would also introduce new categories to state that the reference ontology or the input axiom are more expressive than $\mathcal{EL}$.

Another analysis strategy is NullReasoner-axiom-integration. This strategy still accepts $\mathcal{SROIQ}$ as the ontology language, but may use only lightweight reasoning instead of depending on a $\mathcal{DL}$-reasoner. Because of this lightweight reasoning, it may not be able to correctly detect the axiom category for an input axiom (the advantage of such a strategy is that it can perform the analysis quicker than the $\mathcal{SROIQ}$-axiom-integration).

Yet another strategy may be $\mathcal{SROIQ}$-axiom-removal, which analyses the effect of removing an axiom from an ontology. Although much of the analysis

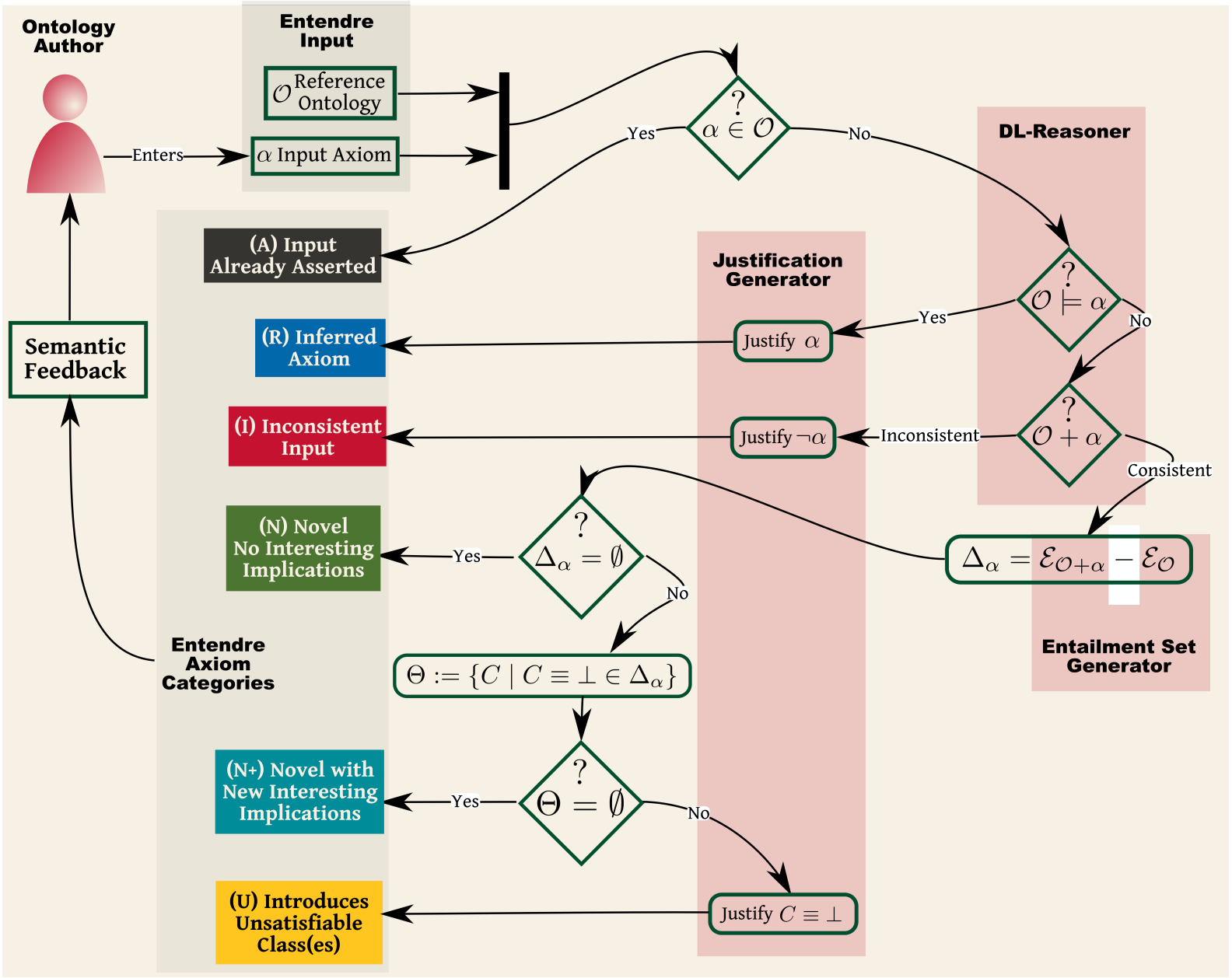

Figure 4.12: Workflow depicting the $\mathcal{DL}$-axiom-integration semantic analysis strategy in Entendre. The analysis is used to categorise the input axiom and generate semantic feedback.

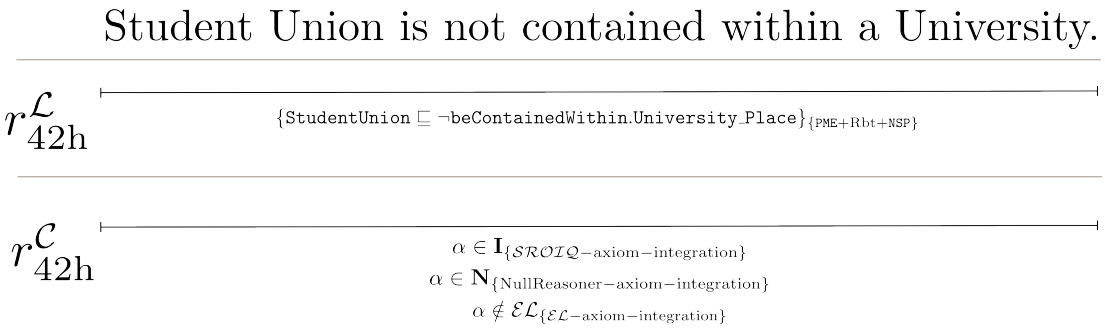

Figure 4.13: Example semantic reading based on various semantic analysis strategies. From top to bottom: the input, a $\mathcal{L}$-atomic syntactic reading and $r^{\mathcal{C}}_{42h}$, a semantic reading of the input. The semantic reading is ambiguous due to the use of different reasoning strategies. The axiom is found to be novel when the NullReasoner-axiom-integration strategy is used. By using a $\mathcal{DL}$ reasoner, the $\mathcal{SROIQ}$-axiom-integration strategy determines that the axiom leads to inconsistency. The $\mathcal{EL}$-axiom-integration strategy on the other hand states that the axiom is not a valid $\mathcal{EL}$ axiom (because it contains a negated concept).

will resemble the axiom-integration strategy, the feedback will be very different. For example, the analogous of the **R** category for axiom-removal would tell the user that it is safe to remove the axiom, as the axiom can be inferred from the rest of the reference ontology. Formally defining all of these alternative semantic analysis strategies is outside the scope of this PhD.

Coming back to our running example, Figure 4.13 shows an example semantic reading based on some of the semantic strategies discussed.

4.4 Implementation

As part of this thesis we have implemented Entendre in the following ways:

- We defined an API consisting of a set of interfaces and classes in Scala[1] that allow us to:

[1]Scala is a strongly typed language that combines the Object Oriented and Functional programming paradigms. The syntax of Scala is similar to Java and it also compiles onto

  - represent the various syntactic and semantic readings discussed in the previous section;
  - represent syntactic and semantic analysis strategies;
  - execute analysis strategies to generate syntactic and semantic readings and
  - generate user feedback based on templates for the different reading categories.

- We implemented two syntax analysis strategies by adapting two existing parsers: the Rabbit parser and the Manchester Syntax parser. These strategies are used to generate syntactic readings (more details about this is given in Section 4.4.2).

- We implemented the $\mathcal{DL}$-axiom-integration semantic analysis strategy in order to generate readings for inputs. And we have initial implementations for other similar strategies: NullReasoner-axiom-integration and and $\mathcal{EL}$-axiom-integration.

4.4.1 Entendre API

The first step for implementing Entendre was to define an API that can be used to describe the different types of readings. The current implementation defines the `EntendreReading` abstract class which is used to represent the different types of readings supported by Entendre. This main class has an attribute `interpretedInput`, which stores the original input value (in our case always a `String`).

An `EntendreReading` is always bound to an `InterpretationContext` (given by the attribute `context` of an `EntendreReading`). The interpretation context keeps track of how this reading was generated. It indicates which reference ontology and which `EntendreAnalysisStrategy`(or strategies) was (were) used to generate the reading. It also keeps track of which are the preferred analysis strategies that should be used as well as what is the rendering syntax.

JVM classes, which means it can be easily combined with existing Java programs. For more information on Scala, see `http://www.scala-lang.org/`

The `EntendreAnalysisStrategy` is a Scala trait[1] that is further specialised into `LexStrategy`, `SynStrategy` and `SemStrategy` for the three types of strategies supported by Entendre. These provide methods to generate readings. For example, `LexStrategy` defines method
`analyse(input: String, context: InterpretationContext)`
that generates a `LexReading`.

Because existing parsers often have a tight integration between the lexical and the syntactic analysis, implementing the `LexStrategy` and `SynStrategy` is often difficult or impossible if one wants to reuse these existing parsers. For this Entendre also includes the trait `EntendreInterpreter` that defines two methods `interpret(input: String, context: InterpretationContext)` and `interpret(input: String, model: LogicalModel)`[2].
Both methods return an `EntendreReading` which can be a lexical, syntactic or semantic reading. Implementations of `EntendreInterpreter` are meant to coordinate existing strategies in order to produce better readings than a single strategy would provide. This interface allows implementations to encapsulate the different analysis that are performed. The following sections contain examples of `EntendreInterpreters`.

Figure 4.14 shows how the main classes and interfaces relate to each other and Figure 4.15 shows an overview of the types of `EntendreReading`s currently supported by the Entendre API.

4.4.2 Syntactic Analysis

Although the Entendre API as described above supports the representation of the various lexical and syntactic reading types, we have only implemented a subset of the API by reusing two existing parsers. We have focused more on the semantic analysis rather than on the syntactic analysis. This is because our experiences show that the need for feedback regarding the logical implications of authors' inputs is greater than the need for more robust syntactic analysis. That authors

[1] A trait in Scala is similar to an interface in Java.

[2] The `LogicalModel` here is a Scala trait that we define to represent a model in some formal logic language. In our case this can be an OWL ontology, a set of axioms or a single axiom.

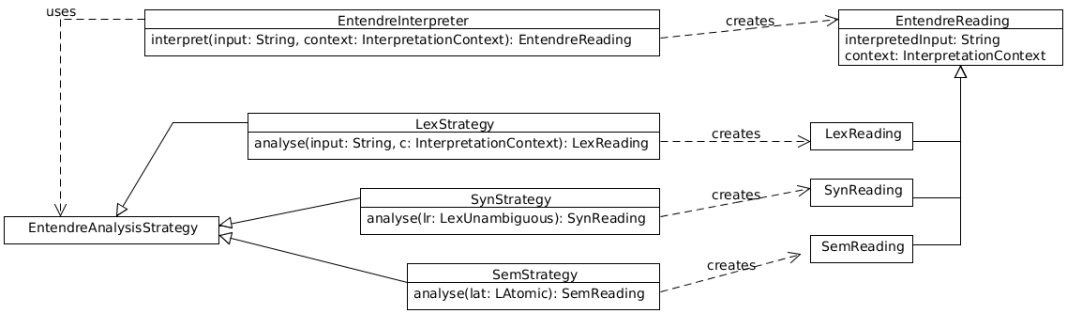

Figure 4.14: Class diagram showing the main classes and interfaces for the Entendre API.

need less support for syntactic issues is due to (i) the ease of learning the CNL by authors and (ii) an existing ad-hoc implementation for supporting some forms of ambiguity and heuristics rules in the existing CNL parsers (as described in Chapter3).

Entendre allowed us to wrap existing parsers. The benefit from wrapping existing parsers is that authors can then benefit from the semantic analysis and feedback that is included in Entendre. In the future, Entendre can also be used to extend existing parsers to, for example, allow and resolve ambiguities during ontology authoring. The two existing parsers that we have adapted to Entendre are the Manchester Syntax parser and the Rabbit parser. We discuss both below.

Manchester Syntax Strategy

The Manchester Syntax parser that we adapted is the implementation that is included in the OWLAPI. Wrapping the Manchester Syntax parser in Entendre was straightforward due to a good decoupling between lexical and syntactic analysis strategies in the OWLAPI. The resulting wrapper is `ManSynOWLAPIInterpreter`.

The OWLAPI provides the `OWLEntityChecker` interface, which provides methods such as `getOWLClass(String name)` that map an input string to an OWL class (similar methods exist for other entity types). An `OWLEntityChecker` is thus similar to an entity mapping strategy. For rendering entities the OWLAPI pro-

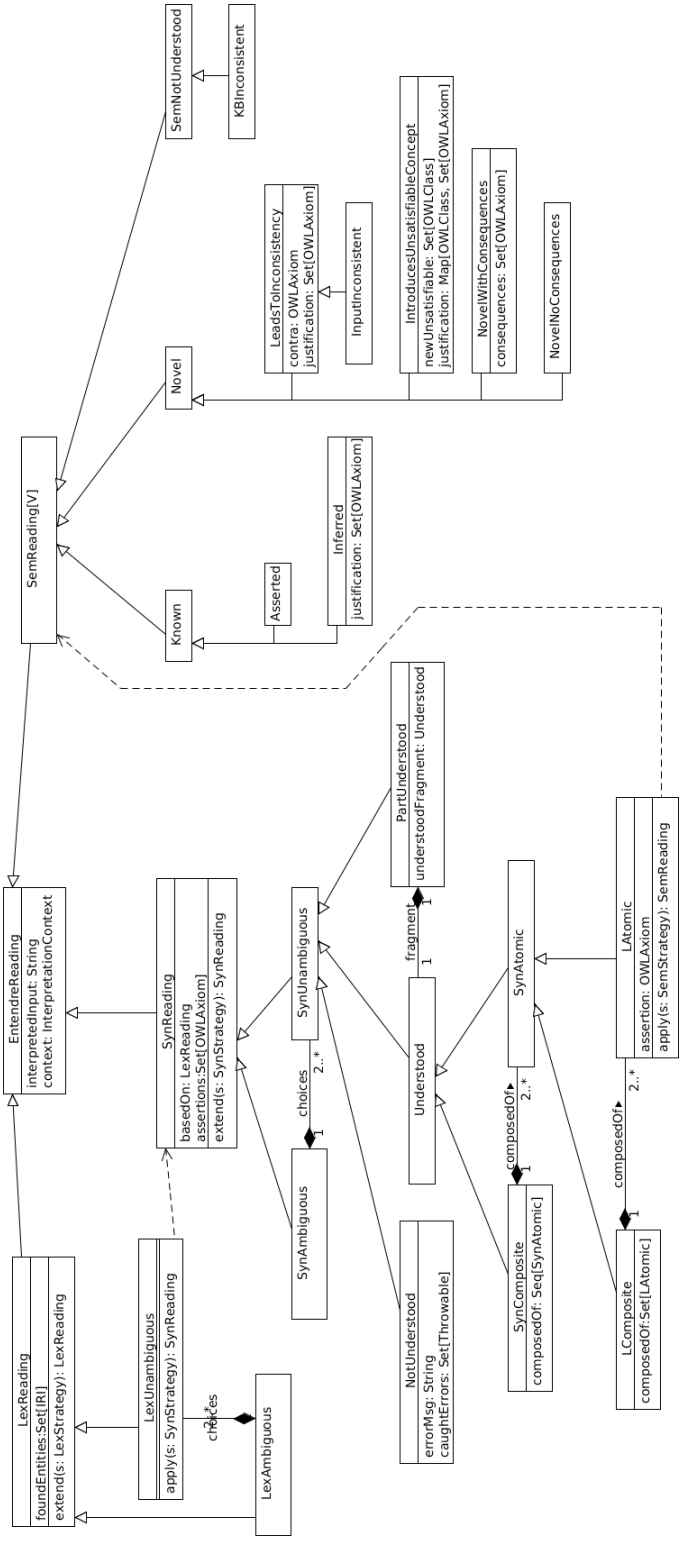

Figure 4.15: Class diagram showing the hierarchy of `EntendreReadings` supported by the Entendre API. The dotted arrows show transitions between different types of readings (e.g. lexical to syntactic reading). The UML aggregation arrows show that some readings can be decomposed into sets or sequences of other reading types.

vides the `ShortFormProvider` interface. Some implementations for this interface are also included in the OWLAPI: e.g. the `AnnotationValueShortFormProvider` can be used to render entities based on an `rdfs:label` value.

The `ManchesterOWLSyntaxEditorParser` class from the OWLAPI can then be used to perform the syntactic analysis of inputs. The parser delegates the lexical analysis to an `OWLEntityChecker` (thus decoupling the lexical and syntactical analyses).

Since full IRIs[1] (and to a lesser extent abbreviated IRIs) are cumbersome and unintuitive for novice authors, we have chosen to use the `AnnotationValueShortFormProvider` as the preferred lexical strategy. As stated in Section 4.2.1 this lexical strategy can introduce ambiguity. While the OWLAPI (and thus also Protégé) does not take this ambiguity into account and simply chooses one interpretation, our implementation takes ambiguity into account by producing an ambiguous reading.

Another improvement in our implementation of the Manchester Syntax is that the default parser fails when no entity can be found (e.g. because there are no entities with a specific annotation value). By using Entendre's features, we were able to extend this default behaviour to include ambiguous readings that introduce missing entities.

Rabbit Strategy

We have developed a wrapper around the existing Rabbit parser which generates either a `NotUnderstood` or an `LAtomic` syntactic reading that can be used for further semantic analysis. Since the existing Rabbit parser already provides detailed feedback when a sentence fails to be understood, we can simply forward the existing error messages.

Adapting the existing parser to produce more detailed readings is part of future work and should be easy to do. For example, we could analyse the `RbtParsedResult` produced by the Rabbit parser to also produce ambiguous readings.

[1] Internationalized Resource Identifier, defined in `https://tools.ietf.org/html/rfc3987` and used as the names for entities in OWL and other semantic web languages.

4.4.3 Semantic Axiom Integration Analysis

In order to generate semantic readings we have implemented the $\mathcal{DL}$-axiom-integration strategy that we introduced in Section 4.3.2. The implementation is fairly straightforward and follows the overall workflow depicted in Figure 4.12. Here, we will give an overview of how the different components were implemented and we will discuss some implementation issues that were required.

The main class in our implementation is called `AxiomUnderstander` which analyses an input axiom based on a reference ontology. Because we are interested in analysing the input based on the full OWL 2 semantics we have a special class `ReasoningAxiomUnderstander` that has access to a description logics reasoner. This class implements the workflow shown in Figure 4.12 by using the methods provided by the other components.

In practice, there are a couple of extra readings that are possible that we did not include in our discussion in Section 4.3.2 because they are corner cases that do not tend to occur (or that should be avoided). The extra readings are [1]:

- `SemNotUnderstood`: readings that are neither Known nor Novel as defined in Section 4.3.2. It contains the following cases:
  - `KBInconsistent`: this occurs when the reference ontology is inconsistent because then we cannot use the $\mathcal{DL}$ semantics to analyse the input axiom.
  - `CannotReasonAboutInput`: this occurs when the input axiom is not an axiom in some subset of description logics. This is not directly usable in the case of $\mathcal{DL}$-axiom-integration as the ontology language is OWL 2. But we could have a case where the target language is, for example, the $\mathcal{EL}$ sub-language. In such cases, if the input axiom contained, for example, a negation, we could not analyse it further.
- `InputInconsistent`: this is a special case of leading to inconsistency where the input itself is inconsistent (regardless of the reference ontology).

[1] We also showed these in Figure 4.15.

Now that we have discussed the extra reading types, we can look at how the rest of the components in Figure 4.12 have been implemented.

We use the **OWLAPI** as the main API to manipulate axioms and ontologies. This API is very close to the OWL2 specifications and already provides several utility methods and classes that we can reuse. For example, with the OWLAPI it is straightforward to determine whether the input axiom is already asserted in the reference ontology. We have however, built a set of utility classes with functionality we feel is missing from the OWLAPI. These are gathered in the Leeds `owlapiutils` library that can be found at `http://sf.net/projects/leedsutils`. For example, one limitation of the OWLAPI is that all the ontologies are mutable; however because a `SemReading` depends on the reference ontology, this reference ontology should not be mutable. The Leeds `owlapiutils` library provides a class for creating such immutable ontologies based on existing ontologies and axioms.

We use Hermit as the **OWL reasoner**. Note however, that we do not use Hermit directly, but rather use the `OWLReasoner` interface defined by the OWLAPI. In theory, this allows us to use any other reasoner that provides that interface (such as Pellet and Fact++).

The **Justification Generator** was implemented by reusing the `BlackBoxExplanation` generator that has been contributed to the OWLAPI by Clark&Parsia. This class is the implementation of the algorithm for generating justifications described in [82]. This algorithm (and thus also the `BlackBoxExplanation` only explains unsatisfiable concepts or concept expressions), but this is enough in practice because it has been shown that any axiom in OWL can be rewritten as a concept unsatisfiability [72]. Unfortunately, the two implementations provided by the OWLAPI(`DebuggerClassExpressionGenerator` and `SatisfiabilityConverter`) to convert an axiom into its equivalent concept unsatisfiability are only partial: not all of the axioms types can be converted. As part of this thesis, we created `UnsatisfiableCEConverter` which improves on the coverage of axioms that can be converted. By combining the `BlackBoxExplanation` and the `UnsatisfiableCEConverter` we were able to generate a justification for relevant inferences.

The **Inferred Axiom Generator** was again implemented by reusing some functionality already provided in the OWLAPI. The OWLAPI contains a set of classes (e.g. `InferredSubClassAxiomGenerator`) which generate axioms of a given axiom type (e.g. `subClassOf`) based on combinations of named entities. For this thesis, we implemented class `InferredAxiomExtractor` that combines the different axiom generators to generate different types of axioms. Also, because we found that some expected inferences where missing (inferences involving class expressions that had already been used in the reference ontology), we extended some of the axiom generators to include such inferences.

To compare the sets of generated inferences we originally used the axiom extractor with the reference ontology and with the merged ontology to produce two sets of inferred axioms; these sets could then be compared to each other. However, we found this to be inefficient because generating these sets is computationally expensive. The current implementation is more efficient as it only generates the set for the merged ontology. We then use this set to check whether there are any axioms that are not entailed by the reasoner for the reference ontology.

Another limitation of the OWLAPI is that it does not provide any method for generating a counter-axiom. This is necessary when an axiom introduces an inconsistency: the counter-axiom is used for generating the justification for the inconsistency. We already indicated how the notion of counter-axiom is defined in Section 4.3.2. We implemented class `CounterAxiomFinder` which we only implemented for some of the axiom types in OWL. It is able to generate counter-axioms for A-box axioms. It can also find counter examples for T-Box axioms. At the moment we have not implemented finding counter examples for axioms involving object properties or data properties.

Besides the $\mathcal{DL}$-axiom-integration strategy which uses full DL reasoning ($\mathcal{SROIQ}$) and has been thoroughly tested. We also have implemented a `NoReasoning`-axiom-integration strategy that simply uses the asserted ontology (hence can only detect a few of the axiom categories). We also have an initial implementation of an $\mathcal{EL}$-axiom-integration based on the JCel reasoner[1].

All of the implementation is free software and is available on-line at `http://sf.net/projects/entendre`.

[1] `http://jcel.sourceforge.net/`

4.5 Application: Feedback in ROO

In order to evaluate whether Entendre has an impact making ontology authoring more intuitive, we integrated Entendre and ROO (see Chapter 3). Before the Entendre integration, when authors added knowledge in ROO, they wrote Rabbit sentences using the *Rabbit editor*, which only provided feedback based on syntactic analysis of their inputs. Once the sentence was parsed correctly, the author could accept the sentence, which was converted to OWL axioms and added to the ontology.

Since ROO is based on the Rabbit syntax. We implemented an Entendre interpreter that sets Rabbit as both its preferred input syntax and its rendering syntax. We considered using other standardised OWL syntaxes (e.g. OWL functional syntax or Manchester syntax with full or abbreviated IRIs) as the rendering syntax, because (i) they entirely exclude ambiguity, while the current Rabbit parser allows some degree of ambiguity through the use of heuristic rules and (ii) we did not have an OWL to Rabbit renderer. However, after an initial session with an ontology author we found that the use of different syntaxes for input and rendering added too much cognitive complexity.

We thus implemented a simple Rabbit renderer. The renderer uses the basic Rabbit sentence templates to render OWL axioms as Rabbit sentences. This means that the produced sentences produce correct Rabbitand that the Rabbit parser should be able to parse these renderings without using any heuristics extensions (i.e. the renderings are syntactically unambiguous syntax). In order to exclude lexical ambiguity we require all entities in the ontologies in scope to have a unique syntactic representation using the disambiguation features provided by the Rabbit language 3.3.2.

The integration with Entendre enables the inclusion of *semantic feedback* to the Rabbit editor. Before the ontology author accepts an input, the input is converted to OWL and semantically analysed. Based on the logical consequence categorisation, ROO now generates appropriate semantic feedback.

The semantic feedback provided by ROO is illustrated below [1].

[1] Asserted axioms are not included as their feedback is trivial.

Rabbit Input: *Every Student Accommodation is a kind of Accommodation.*

Input category: Atomic
Axiom category: N [Novel Axiom without new Relevant Implications]

Syntactic Feedback: Your input can be understood in terms of the "Leeds Points of Interest" ontology and the Rabbit language.

Semantic Feedback: This assertion is **novel**: it has not been added to the ontology yet. However, **no further implications** could be made based on this input. This generally means that the terms you are using have been loosely defined in the ontology. It is also possible that there are some implications that could not be found.

Advice Add the sentence and maybe add more new definitions if you want the ontology to be able to make more inferences.

Rabbit Input: *Teaching Hospital is a Hospital.*

Input category: Ambiguous
Axiom categories: MissingEntity, N+

Semantic Feedback [N+]: This assertion is **novel**: it has not been added to the ontology yet. This input **implies** 6 new things. Have a look at the list of implications to make you agree with the implications. If you do not agree, it may be that you are using the wrong terminology.

Check the new implications:

- *Every Teaching Hospital has footprint a Footprint.*
- *Organisation and Teaching Hospital are mutually exclusive.*
- *Training Centre and Teaching Hospital are mutually exclusive.*
- *Every Teaching Hospital is a kind of Topographic Object.*
- *Every Teaching Hospital is a kind of Place.*
- *Teaching Hospital and University (Institution) are mutually exclusive.*

Rabbit Input: *Every University has part at least one Point of Interest.*

Axiom category: R [Inferred Axiom]

ROO Feedback: This statement is **redundant**. Although the sentence itself is not included in the ontology, this sentence can be inferred from the following sentences that have been said to be true.

Check the justification: Existing Rabbit statements implying that *Every University has part at least one Point of Interest*:

- *Every University has part a Building that has purpose Education of University Students.*
- *Every Building is a kind of POI.*
- *POI and Point of Interest are equivalent.*

Rabbit Input: *Every Student Union is contained within a University(Institution).*

Axiom category: U [Axiom Introducing Unsatisfiable Concepts]

ROO Feedback: This sentence makes concept `Student Union` **unsatisfiable!** This means that nothing can be a `Student Union` anymore. **Advice** You should not add an unsatisfiable concept to an ontology because this concept becomes practically unusable. This is especially true if you make a concept unsatisfiable and that concept was defined by somebody else, as you are probably not using the concept in the way it was intended.

Check the list of contradicting sentences:

- *Organisation and POI are mutually exclusive.*
- *Every Student Union is contained within a University (Institution).*
- *Every University (Institution) is a kind of Organisation.*
- *The relationship contains must have subject POI*
- *The relationship is contained within is the inverse of contains.*

Rabbit Input: *Edge contains a Swimming Pool.*

Axiom category: I [Axiom Leading to Inconsistency]

ROO Feedback: This sentence makes the ontology *inconsistent!* This means that this sentence contradicts what has been said in other sentences in the ontology.

Advice: You should never enter a sentence that makes the ontology inconsistent because:

- it is very hard for computers to reason about inconsistent ontologies
- You are likely using a term (concept, relation or instance) in a way that was not intended by the people who defined the ontology. You should probably find an alternative term that you can use instead. See also the provided list of sentences that contradict this sentence.

Check the list of contradicting sentences:

- *Edge is contained within UoL Campus.*
- *UoL Campus does not contain a Swimming Pool.*
- *The relationship contains is transitive.*

Hereafter, when we mention ROO, we refer to the new version of ROO that is extended with Entendre.

4.6 Evaluation

In order to evaluate the impact of Entendre on ontology authors, we performed an *experimental study*. Because the semantic analysis performed in Entendre is novel and we did not have any indication about how ontology authors would react to feedback about logical implications, we decided to evaluate only this part. Also, much of the syntactic feedback that Entendre could provide was already evaluated in our previous work (see Chapter 3). An advantage of focusing on the semantic feedback only is that we were able to design a more focused evaluation study.

4.6.1 Experimental Design

An experimental study with ROO was conducted to examine users' reactions to semantic feedback and whether this feedback affected users' ontology authoring behaviour. The following *research questions* were addressed:

- ***Q1:*** *How did users characterise the semantic feedback provided by ROO?*
- ***Q2:*** *Did users find the semantic feedback helpful and for what?*
- ***Q3:*** *Did users understand the logical aspects indicated in the semantic feedback provided by ROO?*

Domain and Ontology ($\mathcal{O}$). The study followed a task-based approach which involved using ROO to add new axioms to an ontology $\mathcal{O}$. Points of interest (POI) was chosen as the domain because of its increasing importance, broad application, and familiarity to people. $\mathcal{O}$[1] was created by reusing the W3C POI data model[2] and Ordnance Survey's Buildings and Places ontology[3] using ROO by entering Rabbit sentences (see [36]). $\mathcal{O}$ described main points of interest relevant to Leeds University, including buildings and places related to accommodation, eating and drinking, health services, and transport.

Participants. The study involved 10 participants recruited on a volunteering basis; 6 were from the School of Computing, Leeds University, and 4 were

[1] Available at `http://www.comp.leeds.ac.uk/confluence/Entendre-Study`
[2] http://www.w3.org/2010/POI/
[3] http://www.ordnancesurvey.co.uk/oswebsite/ontology/

from outside. All participants had general IT background and used computers as part of their everyday practice. They were grouped into: **Group 1 KE-novices** (5 users who had no logical background and had never been involved in ontology construction tasks) and **Group 2 KE-experts** (5 users who had logical background or had built ontologies as part of their research).

Procedure and Materials. All evaluation materials and data from the experimental study are available online.[1] Each participant had an individual session observed by an experimenter (from the first three authors). Each session comprised of two steps. In **Step 1** [*5 to 10 minutes*] the participants were given a list of classes, instances, and relationships from $\mathcal{O}$ to examine in ROO. In **Step 2** [*60–90 minutes*] the participants were asked to enter new facts, formulated as Rabbit sentences([36]). There were 15 sentences in 3 batches; a batch included examples from each of the axiom categories : N, N+, R, I, and U (defined in Section 4.3.2). After entering a sentence in ROO, the participants were asked to press the *Semantic Feedback* tab, read the provided feedback, answer a series of questions about their opinion on the feedback, and indicate whether they would add or discard the sentence.

Data. The collected data included the participants' answers related to each sentence and the observers' notes. The analysis is presented below.

4.6.2 Participants' opinions about semantic feedback.

Participants stated their opinion about feedback by selecting characteristics from a given list: informative, relevant, trustworthy, reassuring, confusing, overwhelming and misleading. Table 4.1 presents a summary of all sentences, the five axiom categories are compared in figure 4.16, and the two user groups – in figure 4.17.

	Info	**Relev**	**Trust**	**Reass**	**Conf**	**Over**	**Misl**	**Help**	**NotSure**	**NotHelp**
Overall	78%	56%	38%	16%	10%	10%	1%	91%	8%	1%

Table 4.1: Summary of the participants' opinion on semantic feedback.

Overall, the feedback was found **informative** and **relevant** by all participants. KE-experts found the feedback **reassuring** mainly because their *assumption*

[1] http://www.comp.leeds.ac.uk/confluence/Entendre-Study/

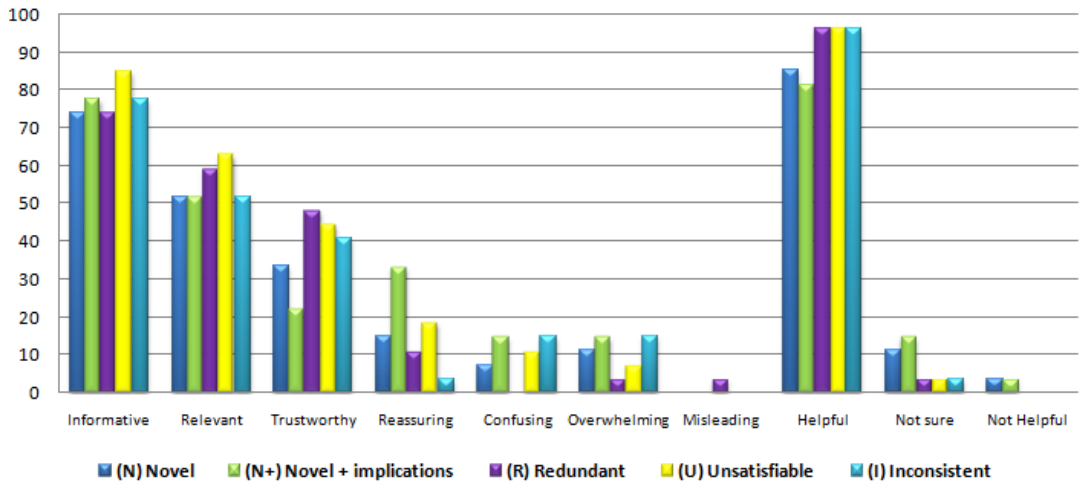

Figure 4.16: Participants' opinions on feedback distributed over axiom categories, together with participants' ranking of feedback as 'Helpful', 'Not sure' or 'Not helpful'. The values are percentages based on all messages from each axiom category.

about what impact an axiom would have on the ontology was confirmed. KE-novices found feedback reassuring in few cases (9%), mainly for *novel with new relevant implications* axioms, as the feedback helped them decide to add the axioms. In more than one third of the cases users found the semantic feedback **trustworthy**. KE-novices trusted the feedback more often, and saw it as a *crutch* to give support when they were unsure. KE-experts in many cases preferred to double-check everything themselves, although they did find the feedback very informative.

There was only one case of **misleading** feedback indicated by a KE-expert. The KE-expert pointed out that although it was possible to infer one of the axioms representing cardinality constraints from the (R) category of the existing ontology, feedback should not encourage the user to discard it, as it would still be valuable to state cardinality constraints explicitly. We also analysed the cases when participants found feedback **confusing** or **overwhelming** and notice that confusion seems to decrease with time: the first batch of 5 sentences includes most misleading and confusing cases, while the last batch has only one occurrence. The

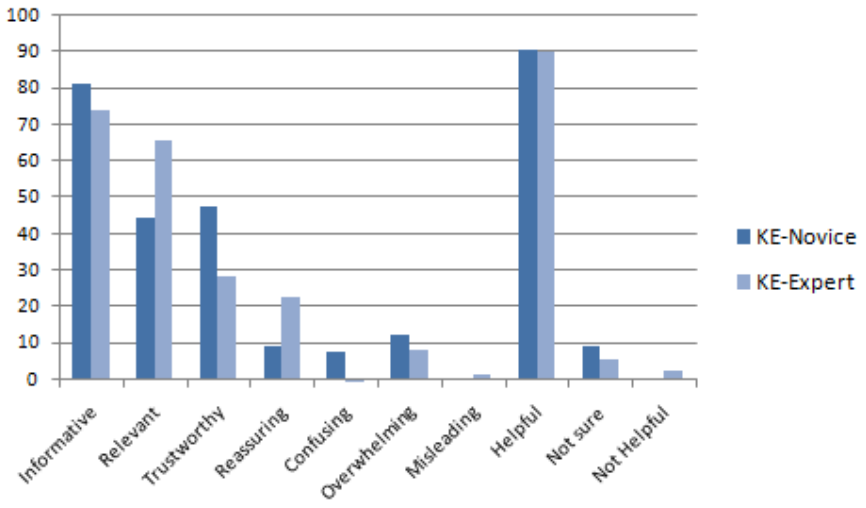

Figure 4.17: Summary of the opinions on feedback for the two groups. The values are percentages based on all messages from each axiom category for the corresponding group.

analysis indicated that the confusing and misleading feedback was associated with certain sentences. These sentences could be grouped in: **Confusing terminology:** KE-novices found feedback about axiom category unsatisfiable (U) hard to follow. They understood that the sentence should not be entered but were confused about what else to do. We plan to work on the usability aspects of the semantic feedback and will consider improving the feedback terminology. **Too abstract:** This concerned mainly feedback containing justifications (R, N+, I and U); users from both groups felt that certain abstract concepts (e.g. Footprint) made it more difficult to understand the feedback message and suggested that appropriate filtering should be done. **Oversimplified:** This was pointed by KE-experts who felt that advice to not enter sentences which made the ontology inconsistent was inappropriate (the sentences were considered valuable, and should have been entered; instead existing sentences should be edited). **Insufficient:** Two KE-experts pointed out that in the cases when the novel(N) axioms could make existing axioms redundant (e.g. entering *Every student bus route is a kind of bus route* made the existing axiom *Every student bus route is*

a kind of transport route redundant but this was not detected) the feedback was insufficient. ROO can be improved to consider this form of redundancy.

4.6.3 Helpfulness of Semantic Feedback

For every feedback message, participants were asked to indicate whether the feedback was helpful and to clarify why. The results, summarised in table 4.1, are very encouraging, as 91% of feedback messages were acknowledged to be helpful. Further analysing (figure 4.16) results by considering the axiom categories, there are notable differences between both groups.

Group 1 - KE-novices - considered feedback as: (a) **Providing new information**, which they did not know (e.g. *'Tells me that a new fact may have impact on the ontology(category N+)'*, *'Informed me about the ontology and the links between the concepts (category R)'*; (b) **Preventing ontology defects**, e.g. *'Told me about inconsistency, I would have not checked otherwise(category I)'*, *'The feedback explained why the sentence should not be included (category U)'*); and (c) **Providing hints on what to do next**, which was mostly for novel facts without relevant implication, as feedback pointed out that further connections should be entered (e.g. *'I have the hint that something may be missing (category N)'*.

Group 2 -KE-experts - found the feedback helpful for: (a) **Developing ontology awareness**, the users found additional information about the ontology provided with the contradicting sentences (category I), implications (category N+), or sentences which make an axiom redundant (category R) useful to gain awareness of the ontology *'(getting the right information at the right time)'*; (b) **Providing warnings**, when something may be overlooked, as one participant commented *'Helps keeping the ontology foolproof (category I).'*; (c) **Providing assurance**, when the KE-experts knows what may happen, (e.g. *'Gives me assurance that I was right in the first place (category R)'*); and (d) **Facilitating decision making**, when further action is needed, e.g. information about contradicting sentences was considered helpful (e.g. *'Directed me what to change from the ontology (category I)'*).

Participants were not sure of the helpfulness of feedback when: (a) it was confusing or misleading (see previous section); (b) did not provide much new information (which was pointed by KE-experts); (c) did not provide enough information what to do next. The two occurrences of *Not helpful* were on novel axioms category and came from the same KE-expert who commented that, apart from telling that a sentences was novel, the feedback was not much useful.

4.6.4 Understanding of Logical Aspects and Impact on User behaviour

For every sentence to be entered, participants were asked three questions to test their understanding of the logical implications relating to the sentence. To avoid asking questions that followed trivially from the feedback given, the questions used rephrasings and slightly different terminology as that used in the feedback. For example, for category N+, we asked whether "the ontology *already knew* the fact that 'X"', where X was one of the presented new entailments. Some questions also inverted the information given in the feedback: for category I, we asked whether the opposite of the input sentence was already known.

We classified the score for each participant's answers to indicate the level of awareness about logical implications: *confusion*, *neutral* or *understanding*. Overall, both groups showed a high level of understanding: 69% for **Group1(KE-novices)** and 86% for **Group 2(KE-experts)**. There are notable differences between both groups (see Table 4.2). Particularly surprising was that **Group 2(KE-experts)** showed signs of confusion when answering some of the questions. We note however that, in the case of axioms leading to inconsistency (I) this apparent confusion matches with this group's opinion that the advice was oversimplified.

We also reviewed participant responses for measuring the impact of semantic feedback on their behaviour. The experiment study included a question on what actions the participants would take in response to the semantic feedback. The possible answers were: they will (a) add the sentence to the ontology, (b) discard the sentence or try to find an alternative sentence, (c) seek further clarification and (d) do not know. We analysed the answers (user actions) and compared

	Group1	KE-novices		**Group2**	KE-experts	
	Confusion	**Neutral**	**Understanding**	**Confusion**	**Neutral**	**Understanding**
N	0%	15%	85%	0%	7%	93%
N+	0%	15%	85%	7%	0%	93%
R	8%	38%	54%	0%	0%	100%
I	8%	31%	62%	21%	14%	65%
U	15%	8%	77%	7%	14%	79%
Overall	6%	25%	69%	7%	7%	86%

Table 4.2: Participants' understanding of logical aspects in feedback.

them against the advice from the feedback. The results are very encouraging as **Group 1(KE-novice users)** accepted 96% of advice compared to the 92% for the **Group 2(KE-experts)**, i.e., participants agreed with the advice and followed the action suggested by the semantic feedback message. In the few cases that the advice would not be followed, KE-novices often indicated that they would seek further advice, which indicates that they had been made aware of the problem at hand, but did not have enough information to resolve the situation.

4.7 Discussion

This chapter presented Entendre , a framework for facilitating (i) the systematic analysis of ontology author's inputs and (ii) the generation of understandable feedback to support ontology authors to formulate new facts to add to the ontology. In particular, we formally defined:

1. a framework that combines both syntactic and semantic analyses of an ontology authoring input;

2. the main syntactic analyses of an input that can be performed and the main results of such analyses: lexical and syntactic readings.

3. a categorisation of lexical and syntactic readings, including relationships between these reading categories (e.g. how an $\mathcal{L}$**-Composite** reading relates to a **Syntactically Atomic** reading). This categorisation (i) provides an overview of the types of syntactic ambiguity that can occur and (ii) can

be used to provide targeted feedback about syntactical issues to ontology authors.

4. an axiom-integration analysis strategy that categorises input axioms based on the logical effects it has on the ontology being built. This categorisation provides an overview of ontology defects that can be introduced and can be used to provide targeted feedback about semantic issues to ontology authors.

This chapter discussed an implementation of the Entendre framework that showed the applicability of the framework. The syntactic analysis part of the framework was used to build wrappers around the Rabbit CNL and the Manchester Syntax. The semantic analysis part of the framework was used to implement the axiom analysis strategy; this showed how Entendre facilitates the integration of various existing ontology reasoning services.

This chapter finally showed an evaluation that focused on the semantic analysis part of Entendre. The evaluation results presented in this chapter show that the Entendre framework facilitates the next step to intuitive ontology authoring – embedding *intelligence* in the ontology authoring tools to make them *active listeners* that understand the user's actions and respond accordingly. In particular, the evaluation results show that:

Responsive ontology authoring tools benefit ontology authors. The study strongly indicated support for the philosophy that *authoring tools can act as active listeners* that offer immediate, interactive, and intuitive feedback at the time a new axiom is to be added. It showed: (a) it is possible to develop such tools (ROO is just an example; by following Entendre, feedback features can be embedded in any ontology authoring tool); and (b) users are enthusiastic about such tools. All KE-experts in the study reacted extremely positively to the *embedded* feedback and commented that it would potentially save them substantial time and effort to maintain the ontology (they followed the advice in 92% of the cases). All KE-novices were also pleased to see immediate response to their actions and followed advice in 96% of the cases. Some users commented that feedback helped

them to consider what was required when authoring an ontology, and even suggested that ROO would be useful to assist people learn about ontology authoring. This can be addressed in further studies with ROO (e.g. using it as a learning tool in BSc or MSc courses on knowledge engineering).

Semantic feedback benefits ontology authors. Both KE-novices and KE-experts found semantic feedback helpful (91%), informative (78%), and relevant (56%). In 38% of the cases, feedback was seen also as trustworthy. KE-experts considered feedback as providing *reminders* of actions they might forget or *reassurance* that their actions were appropriate. It helped them develop *awareness* of the ontology and *take decisions* what actions to perform (including what additional changes to the ontology would be needed). On the other hand, KE-novices saw feedback as a *watch-dog* stopping them to do wrong actions or an *adviser* providing relevant information and hints what to do next (e.g. that further connections should be entered). Further studies can be conducted to examine whether feedback would affect ontology quality (e.g. by comparing ontologies developed using ROO with or without the semantic feedback feature).

More work is needed to produce understandable and actionable feedback. Entendre provides a systematic way to understand the user's input and also facilitates the integration of existing tools for syntactic and semantic analysis. However, the study shows that it is not sufficient only to understand the users but also to *make users understand what is conveyed with the feedback*. During the study we asked participants questions testing their understanding of logical implications. KE-novices understood the logical implications described in the feedback in 69% of the cases, while KE-experts in 80%. Although there was noise in few questions, it was clear that feedback could cause confusion (10% of all cases), might be overwhelming (10%), and, in one case, was misleading. The analysis of these results points to further work required to improve the effectiveness of Entendre. The confusing terminology of the feedback messages(e.g. unsatisfiable class) can be improved and made more intuitive. Furthermore, novice ontology authors seemed to have trouble following some of the instructions provided by the feedback,

such as "adding more related facts" when an input was novel. The static feedback message generation can make novice ontology authors feel overwhelmed and do not provide enough guidance as to what to do next. These issues indicate that there is a need for more flexibility when presenting feedback to ontology authors.

In conclusion, this chapter has showed that Entendre enables a main requirement for active listening – producing an *understanding* of the speaker (in this case the user who is an ontology author). In other words, identifying what effect an axiom added by the user can have on the ontology. This understanding enables ontology authoring tools to pay more attention to the resultant ontology and be *proactive in offering timely feedback and advice*, which is beneficial to ontology authors. However, our results also show that there is a need for more fine-grained feedback control and interaction between the system and the ontology author. In the next chapter, we investigate whether we can use discourse analysis to enhance the interaction between ontology authors and the system.

Chapter 5

Perico: Dialogue-based Interaction for Ontology Authoring

In previous chapters we have investigated how ontology authoring systems can use CNLs (and their related lexical and syntax analysis) and logic-based integration analysis to support ontology authors when adding new statements to the ontology. Although the combined syntactic and semantic analyses can be used to provide beneficial feedback to novice ontology authors, we have seen some issues regarding novice authors finding some feedback overwhelming and not knowing how to perform suggested tasks. Both of these issues are related to the limited interaction that ROO uses to communicate with the ontology author: ROO waits for an input from the ontology author and provides feedback or updates the ontology based on the input. Such limited interaction means it is up to the ontology author to correctly follow any instruction or messages to provide an improved input or a new input that builds on a previous input.

This chapter proposes to extend Entendre with a new layer of input analysis to enable more advanced ontology authoring interactions. In order to make this extension generic enough to support a wide variety of ontology authoring interactions, we investigate *dialogue-discourse analysis* to track the authoring interaction. This allows us to build on an extensive body or research on dialogue systems.

The goal of this chapter is thus to formally define a generic dialogue framework, which we will call Perico for describing and executing ontology authoring interactions. Such a dialogue framework has the following requirements:

- it must model the interaction from the perspective of one of the participants in the dialogue: the *ontology authoring system participant*;
- it must be ontology-driven (i.e. any agreed knowledge should be stored in an ontology language);
- it should describe ontology authoring interactions based on a standard representation, while taking into account ontologies and their reasoning services;
- it must enable the description and execution of non-trivial ontology authoring interactions;
- it must allow ontology authoring interactions based on a CNL;
- it must extend the Entendre framework for input analysis and feedback;
- it must provide reusable and composable interactions for enabling advanced ontology authoring interactions;

The contribution of this chapter is Perico, a framework for describing dialogues for ontology authoring. In order to motivate and define Perico, we first discuss the relevant work in Section 5.1 and introduce basic terminology about dialogue systems in Section 5.2. We then present the Perico framework in Section 5.3 and validate it by formalising an existing ontology authoring interaction in 5.4. Then, in Sections 5.5 we show how the framework can be used to design, formalise and implement extensions to existing ontology authoring interactions.

5.1 Relevant Work on Dialogues for Ontology Authoring

This section presents existing work on dialogues for ontology authoring. Although much research has been done on tool support for ontology authoring as described

in Section 2.1, relatively little work has been done on exploring dialogue systems for ontology authoring. We are not aware of any dialogue-based ontology authoring systems, but there are several dialogue systems for tasks that are relevant to ontology authoring such as *knowledge formulation* and *ontology querying*. Because we are interested in supporting novice ontology authors, we also look at tutoring dialogue systems.

Knowledge Formulation A number of interactive knowledge acquisition systems have been studied [5, 22, 25, 28, 89, 140]. These systems have in common that they have been designed to help novice users formulate their knowledge in a way that can be captured by the system. In order to facilitate this process, they encode one or more strategies to (i) elicit new knowledge based on existing knowledge, (ii) give feedback about how the system has interpreted user inputs and (iii) give feedback about inconsistency issues for integrating new knowledge into the existing knowledge base. The proposed interactions vary in complexity. For example, a simple interaction strategy proposed in [89] consists of the following steps: (i) the system generates a number of unknown facts and asks the user whether they are true, false or non-sensical, (ii) the user provides an answer, (iii) the system processes the answer and repeats the interaction with new unknown facts.

Evaluation results for some of these systems have been positive, showing that the systems successfully capture knowledge from users without the need for much training [25, 89]. The description of the interaction strategies in these systems is not formalised, but only illustrated with examples that are dependent on the choices of user interface, type of knowledge elicited, etc. This makes it difficult to compare, reuse and adapt the proposed interaction strategies. In this chapter, we define a framework that can be used to formally describe such interaction strategies for knowledge formulation in terms of dialogue moves.

Dialogues for Ontology Querying

Natural language interfaces for querying knowledge bases and ontologies have also been studied. Lexical and syntactic analysis issues discussed before also occur in querying systems, causing ambiguity or generating overly large answer spaces.

To resolve such issues, simple dialogue interactions have been used to resolve ambiguity or to narrow the search space [31, 32, 85, 86]. Although these systems use similar interaction strategies they are not described in a uniform manner. This makes it difficult for ontology authoring systems to compare the proposed strategies to, for example, decide which strategy is best suited in a particular ontology authoring context. This chapter proposes a framework that will enable the description of such strategies in terms of dialogue plans.

Tutoring Dialogues

The provision of Entendre semantic feedback can be seen as a form of tutoring information that is provided during the ontology authoring task. As such, existing work on intelligent tutoring systems(ITS) is relevant. Although we are not aware of ITS that deal with ontology authoring in this way, there are ITS that aim to teach students theory and/or skills in other domains, some of which are related to "logical systems". For example, in recent years, Di Eugenio et al. have analysed various tutoring dialogue strategies (based on pedagogical literature, intuition and natural dialogue corpora) in order to determine which strategies correlate to better learning of students. Some of these strategies correspond to factors we have encountered when studying the Entendre feedback such as (i) feedback conciseness [37] (ii) expert vs. non-expert tutoring [104] and (iii) evaluating effective tutoring dialogue move sequences [38]. Most of these strategies have been formalised in terms of dialogue moves such as *direct procedural instructions* (suggesting actions to students), *positive* and *negative feedback* (confirm correctness or notify the student about errors). Currently, these tutoring strategies would have to be manually added to existing ontology authoring systems because do not support interaction in terms of dialogue moves. This chapter will define a framework that facilitate the re-use of such strategies for ontology authoring.

This section presented work on various interactive systems that can support ontology authoring tasks. We saw that for knowledge formulation and ontology querying, there is a lack of formal and reusable descriptions of such interaction strategies. The area of Intelligent Tutoring Systems has formalised and analysed

the effectiveness of various tutoring strategies which could be used to provide better ontology authoring interactions. However, a framework is needed for (i) formalising existing knowledge formulation and ontology querying interaction strategies to make them available to novice ontology authors and (ii) reusing already formalised tutoring strategies in the context of ontology authoring. This chapter will present such a framework for describing and executing ontology authoring interaction strategies in Section 5.3. In the next section, we give an overview of the topic of dialogue systems and introduce relevant terminology that we need to describe dialogues for ontology authoring.

5.2 Dialogue Systems Overview

Our aim in this chapter is to expand the interactive features of ontology authoring systems by extending Entendre with a dialogue model. This section provides an overview of dialogue systems and defines the terminology used in the research of dialogue system development that we use in the rest of the chapter.

5.2.1 Basics of Dialogue Systems

A *dialogue system* is a system that can interact with other agents (human or not) in a dialogue. The agents are called *dialogue participants* and the dialogue itself is defined as a sequence of *dialogue moves*: messages produced by said participants. In this thesis we are interested in dialogues that achieve one or more specific *dialogue tasks*, such dialogues are assumed to have specific goals, e.g. information seeking, information provision and guidance.[1]

The research on dialogues –in the context of computing and dialogue systems in particular– has a long history and dates back to work by Austin [6]. Dialogues have been studied in various research streams corresponding to different viewpoints and different goals – philosophy (for dissecting the nature of dialogues), computational linguistics (for analysing and annotating dialogues),

[1]Note that this is in contrast to *chatbots* which are able to interact in dialogues, but the resulting dialogues do not have a specific goal other than the interaction itself.

artificial intelligence (creating dialogue agents for various purposes), argumentation (analysing how dialogues are used to build up arguments), agent computing (creating agents that can communicate and cooperate with other agents), user interfaces (providing a dialogue-based interface for various tasks), etc. These different research streams have resulted in a number of different *dialogue theories*: ways to model dialogue interactions. Although these theories have some marked differences, there are some general functional and informational components that can be identified.

There is a general agreement on the functional components that a dialogue system must posses. This is captured by the general architecture for dialogue systems [15, 52], which we present below in Section 5.2.4. However, there is less of an agreement on a fixed set of informational components for a dialogue (system): although at each moment in a dialogue, there must be an immutable *dialogue state*, different dialogue theories disagree as to which *knowledge is relevant to the dialogue* as well as how this knowledge should be represented. Relevant knowledge consists at least of:

discourse knowledge , which describes the dialogue moves that participants can perform, as well as the obligations that dialogue moves create and dependencies that exist between dialogue moves. Discourse knowledge is domain and task independent.

domain knowledge describing the topics under discussion during the dialogue. This is thus knowledge which is not about the dialogue itself, which is covered by the discourse knowledge.

dialogue task knowledge describing the overarching goals (or plans) of one or more of the participants. Note that discourse knowledge often describes dialogue moves in terms of goals as well; the difference is that the dialogue task describes higher-level goals that cannot be achieved by a single dialogue move.

There is also little agreement between dialogue theories on how to (or even whether to) specify the move that a participant should perform at any given moment. To illustrate some of the varying decisions in dialogue theories we give

a couple of examples. Regarding the modelling of the *dialogue state*, some dialogue theories decide to model the dialogue in terms of *beliefs and intentions* while others ignore these issues and only represent the expressed messages [135]. While *most dialogue theories agree on representing the dialogue as a game between different participants*, there is no agreement as to what should be the basic building blocks of such a game (i.e. what constitutes a dialogue move). Some dialogue theories provide a large *taxonomy of domain-independent dialogue moves* [118],[1] while others restrict themselves to a relatively small number of dialogue moves [15]. In Section 5.2.2, we present a recent draft that aims to standardise dialogue moves. Regarding the specification of the dialogue task, some dialogue theories choose for a purely rule-based approach [98], while others argue for a probabilistic approach [66]. For more about the different choices and dialogue theories, we refer to [135].

The wide variety of dialogue theories that have been proposed means it is often difficult to compare different dialogue theories and also difficult to develop dialogue systems based on these theories. To facilitate the formal description of dialogue theories and systems three complementary views are used:

- a **computational** view that has become a standard for describing dialogue systems (Section 5.2.3);
- a **functional** view based on an agreed architecture of dialogue systems in terms of their required functional components (Section 5.2.4);
- an **informational** view based on a draft standard for dialogue annotation; the proposed standard describes one way to decompose dialogues into informational components (Section 5.2.2);

[1] Dialogue moves are also called dialogue acts, speech acts, conversational moves etc. Note that when a dialogue system performs a dialogue act, that act can be interpreted in terms of some communicative function. See Section 5.2.3 for a definition of dialogue moves.

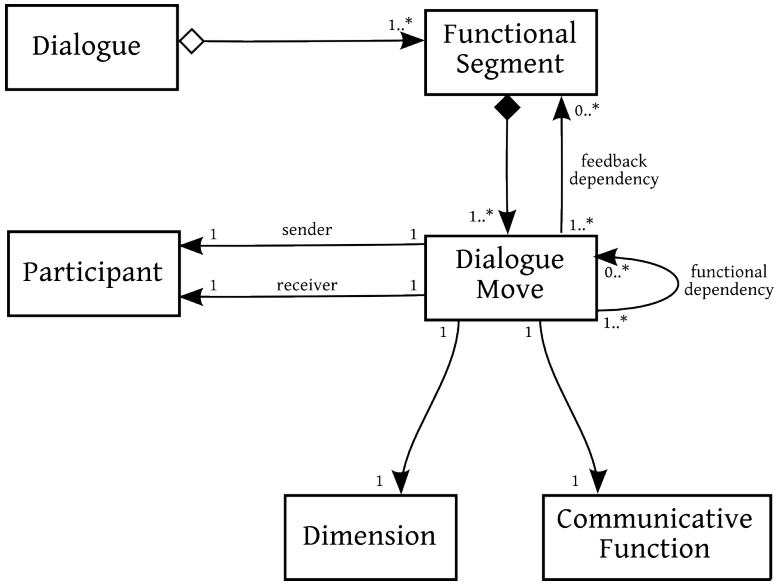

Figure 5.1: UML Class diagram showing the data model for describing the structure of a dialogue. This is a slight adaptation of the dialogue metamodel defined in [74] and [21].

5.2.2 Informational View: A Standard for Dialogue Annotation

ISO-DIS-24617-2 [74] is a recent standard, currently in draft status, created by experts on dialogue moves that takes into account much of the work on dialogue systems and dialogue annotation from the last decades. It provides a basic model of a dialogue, shown in Figure 5.1, as a sequence of **functional segments** which is the basic functional unit of the dialogue. Functional segments are used as a way to aggregate one or more **dialogue moves**.[1] At the same time, functional segments provide a more fine-grained discourse information unit than dialogue turns, which may consist of large number of sentences.[2]

Dialogue moves are the main units of discourse information, they contain

[1]The ISO standard calls them dialogue acts, but in this PhD we will call them dialogue moves to keep in line with the ISU terminology [98].

[2]Dialogue turns are not directly part of the dialogue model in the ISO standard, but they can be inferred from the sequence of functional segments.

information about:

Participants and their roles, i.e. which participant is the sender, receiver (or other) of the dialogue move.

Dimension for the main topic of the dialogue move. The ISO standard provides a set of 9 **core dimensions** for dialogue moves. Eight of the dimensions are about the dialogue itself, like *feedback*, *time management* or *discourse structuring*. The last dimension is the *task* dimension, which indicates that the dialogue move states something about the domain (i.e. it is related to the main dialogue task).

Communicative Function defines the discourse semantics of the dialogue move. The ISO standard defines a taxonomy of communicative functions – shown in Figure 5.2 – that includes variants of *information-providing functions* (such as `inform`, `answer` and `correction`), *information-seeking functions* (such as `propositional question` and `choice question`) and *action-discussion functions* (e.g. `offer`, `suggest` and `accept request`). The given taxonomy is a starting point for describing dialogues and is meant to be extensible.

Dependency Relations are used to describe relations between dialogue moves and functional segments. While communicative functions act as a dialogue schema stating that there may be a relation between a `question` and a subsequent `answer`, dependencies instantiate these relations. There are two main types of dependencies: **functional dependencies** between two dialogue moves (e.g. a `question` being `answer`ed) and **feedback dependencies** between a dialogue move and a functional segment(e.g. an `auto-negative-feedback` stating that a previous functional segment was not understood).

The ISO standard also assumes that each dialogue move is associated to some **semantic content**, which is defined as some "information, situation, action, event, or objects that the sender of the dialogue move wants to bring to the attention of the other dialogue participants" [74, Section 3.17]. The standard

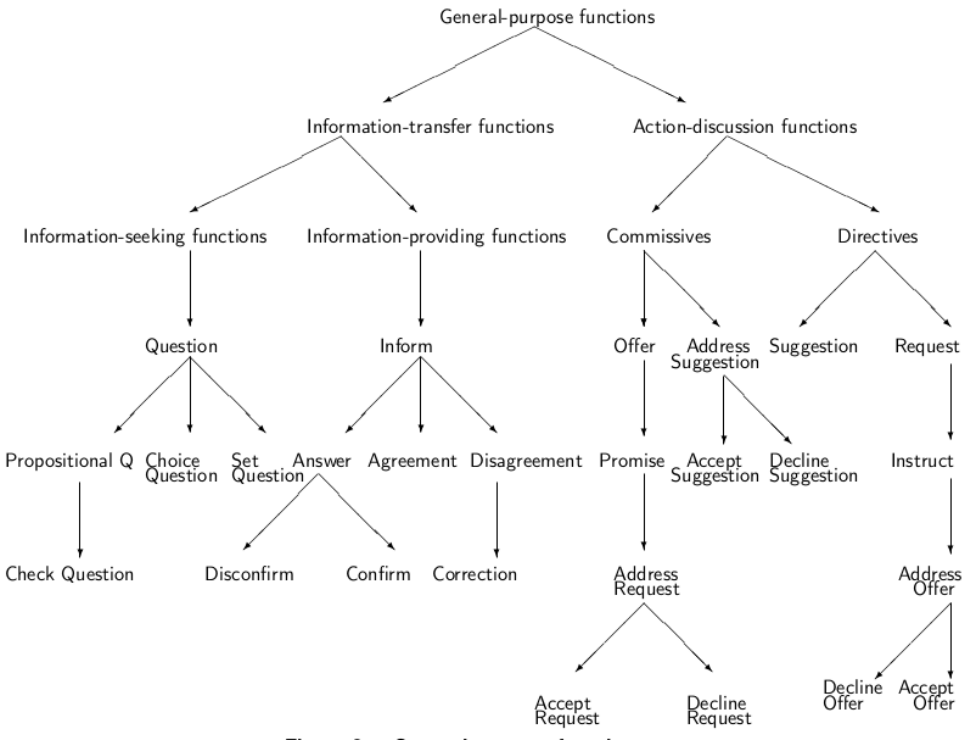

Figure 5.2: Taxonomy of general-purpose communicative functions as defined in [74].

does not require any specific representation for the semantic content, which is why it is not included in Figure 5.1.

In Section 5.3.1 we use this standard as a starting point for representing the informational components of dialogues for ontology authoring. In order to formalise dialogue interactions we will need to extend this model, which clearly focuses on the required *discourse knowledge*, with a representation for the domain knowledge and the task knowledge.

5.2.3 Computational View: Information-State-Update

The Information-State-Update (ISU) [98] view of dialogue systems provides a generic way of describing dialogue theories from a computational point of view. The idea behind ISU is that, although different dialogue theories have different

ways of modelling dialogue components, all theories must ultimately represent the **dialogue state** in some way. Also, dialogue theories must provide a way to *update* the dialogue state as the dialogue advances.

According to ISU, a dialogue theory can be expressed (and thus a dialogue can be modelled) in terms of:

- **Informational Components**: a description of what information is taken into account as being part of the information state of the dialogue.
- **Formal Representations**: a description on how to represent the chosen informational components.
- **Dialogue Moves**: a set of actions that can be performed by the dialogue system. Execution of these moves causes changes in the (representation of the) informational components.
- **Update Rules**: a set of rules that describe legal ways in which the informational components can change and which dialogue moves can be performed in the current information state.
- **Update Strategy**: determines which (or in which order) update rule should be used when multiple rules are available.

Examples of dialogue theories and systems described using ISU are given in the original paper on ISU [98]. In the rest of this chapter, we adopt ISU to describe a dialogue framework for ontology authoring (Section 5.3) and to formalise dialogue systems for ontology authoring (Sections 5.4 and 5.5).

5.2.4 Functional View: General Dialogue Pipeline

A general architecture for dialogue systems [15, 52] is shown in Figure 5.3; it describes a pipeline of functional components that any dialogue system must contain:

- **Input Recogniser**: receives inputs from other dialogue participants.

- **Input Interpreter**: converts the received input into an internal representation that the dialogue manager – the next component in the pipeline – can use. This internal representation captures the relevant *meaning* of the input, by *relating the input to the domain under discussion* (e.g. describing what the input says in terms of concepts, relations or instances in the domain) and *describing the communicative function of the input* (e.g. whether the input is seeking information, requesting information, agreeing with a previous statement or clarifying a previous statement).

- **Dialogue Manager**: updates the dialogue state by monitoring dialogue events – such as silences or valid internal representations of inputs – and combines them with the current dialogue state. Two key functions provided by the dialogue manager are (i) *dialogue planning*: deciding on whether there are actions that need to be performed by a dialogue participant and (ii) *knowledge grounding*: deciding whether the agreed domain knowledge between the dialogue participants should be updated.

- **Output Generator**: translates a semantic output message into a language that other dialogue participants can understand (usually natural language).

- **Output Presenter**: transmits the output message to other dialogue participants.

In Section 5.3.2 we follow this architecture and terminology to define a framework for describing dialogues for ontology authoring.

5.2.5 Implementation of Dialogue Systems

The research literature describes various ways to develop dialogue systems based on the architecture shown in Figure 5.3. We discuss the most common choices for implementing the components of dialogue systems.

Input Recogniser and Output Presenter Since human dialogue is often spoken, most recent research has focused on developing dialogue systems that support this mode of interaction. In such a setting, the input recogniser is a

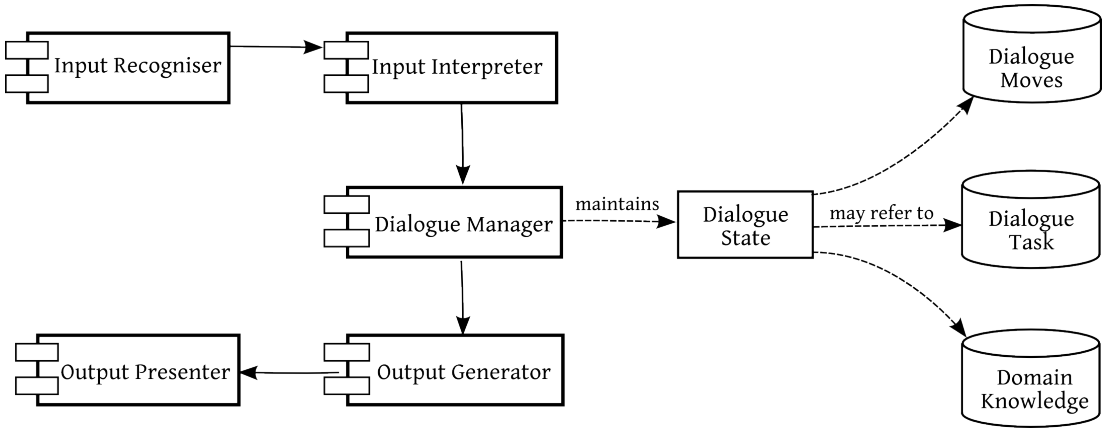

Figure 5.3: Main functional and informational components of a dialogue system.

speech recogniser and a speech synthesiser plays the role of the output presenter. Recently, there has also been work on multi-modal dialogue systems, where speech is mixed with a GUI or other visual interface [76]. Other common systems limit themselves to textual input and output [140]. The choice of input and output components is important in the sense that speech recognition and synthesis is error-prone, thus the dialogue manager needs to take into account the possibility of errors at these stages. In this thesis we limit ourselves to textual input and output as we concentrate on the input interpretation and the dialogue management components. However, as the interface is in a CNL, some elements of error correction similar to speech-based dialogue will be considered.

Input Interpreters are usually implemented by using NLP techniques to find patterns in the input that can be translated into a chosen internal representation. The internal representation in current dialogue systems can vary from simple key-value structures to logic assertions in some logical language. Since NLP techniques for deriving key-value structures are more robust, these types of internal representation is more common in current dialogue systems. In key-value structures, the keys are often pre-defined "entities" of the domain[1] and the task

[1]NLP techniques are used for finding various other types of linguistic phenomena besides named-entity matching in dialogue systems such as anaphora resolution, finding cue-phrases, dialogue moves, etc.

of the input interpreter is to determine whether the input contains a values for some of these entities. Developing input interpreters often requires gathering a corpus of possible inputs in order to use machine learning techniques or to define patterns that need to be looked for.[1] This means that input interpreters are often custom built for a specific dialogue situation, since the corpus and the target "entities" are dependent on the particular dialogue domain and goal.

To avoid having to build custom input interpreters for each dialogue domain, systems can restrict the interface: requiring users of the system to start their inputs in a particular way (e.g. "I think that..." or "I disagree with...") or requiring them to learn a specific vocabulary to interact with the system. This makes it easier for the system to recognise the role of the inputs in the dialogue.

Output Generators are usually developed using natural language generation techniques. These again depend on the internal representation of the domain and the roles of the output in the dialogue. The simplest and most common approach is to use template-based generators. This can be done in dialogues that have a well-defined domain and goal, where dialogue developers know in advance the types of outputs that the system can produce. In such cases, templates can be defined in advance that have slots that can be filled with values from the dialogue domain. When using a more expressive internal representation, the output generation can be more flexible and does not need to be defined in advance, since outputs can be generated from logical statements.

Dialogue Managers are developed based on the various different dialogue theories [98], but are also informed by the type of dialogue that the system needs to support. Historically, researchers have proposed various dialogue toolkits which aid the development of dialogue managers. These toolkits already make some decisions about representing the dialogue state (and thus restrict the number of dialogue theories that can be used with said toolkit). There are relatively simple approaches for developing dialogue managers based on execution of **finite-state** machines and completion of slots in **frames** in [97]; these approaches may not be flexible enough to represent advanced ontology authoring interactions.

[1] This is especially the case when users of the dialogue system are free to use any normal language to interact with the system.

Since we aim to be able to describe and implement complex dialogues for ontology authoring, we opt for more robust **plan-based** dialogue managers.[1] dialogue managers require that all dialogue moves define pre-conditions in order to decide which moves can be applied at a certain point in the dialogue. The use of only rules can be cumbersome for complex dialogues where there is a large number of dialogue moves. In such cases, it can be difficult for a dialogue developer to predict the execution of a dialogue and to guarantee the correctness of a dialogue plan. The user of pre-conditions can also lead to repetition: for example, to simulate phases in a dialogue, the same conditions have to be specified multiple times for various dialogue moves that are all meant to be available only at a certain phase of the dialogue. For this reason some plan-based dialogue managers provide ways to organise the dialogue plan in the form of task trees or similar structures. This can limit the options that a user of the system has (e.g. because the dialogue is not in the right phase of the plan), but makes the dialogue more predictable and easier to develop. In Section 5.3.1 we adopt a definition of task trees that we will use to describe ontology authoring dialogues.

Dialogue Engine Toolkits

Since developing dialogue systems from scratch requires considerable effort, there have been several initiatives to create *dialogue engine toolkits* which provide basic implementations of the main dialogue architecture. We have reviewed several of these toolkits [15, 35, 63, 98, 114, 117][2] to determine whether these toolkits satisfied the requirements for our framework (see introduction of this chapter). Some of these toolkits were commercial or not available for reuse, thus we focused only on those that were open source. Some toolkits only provided an API for developing ISU dialogue systems, but did not provide any reusable components [114]. Other toolkits were not stable enough to be extended [63]. Yet others were using a custom representation for discourse knowledge [15] or did not allow easy extensions for input interpretation via Entendre or representing the domain knowledge using ontologies. In conclusion, none of the evaluated toolkits was a good fit for describing ontology authoring dialogues without major changes;

[1] Also called inference-based.

[2] See also the Ontology-based Dialogue Platform, `http://www.semvox.de`

in Section 5.6 we show an implementation of a simple dialogue toolkit that is a simplified implementation of the RavenClaw dialogue manager [15], one of the evaluated toolkits.

Conclusion

This section presented basic concepts for describing dialogue systems in general. We discussed standards for describing dialogue systems that we can reuse to describe ontology authoring interactions. In particular we identified a standard for describing discourse information based on ISO-DIS-24617-2. We also identified the basic functional components for executing dialogues and identified plan-based dialogue managers as a suitable approach for representing ontology authoring interactions. Finally, we evaluated various dialogue engine toolkits, but found none that could be reused while meeting our requirements for the dialogue framework. In the next section we use these decisions to describe a framework for describing and executing dialogues for ontology authoring.

5.3 Perico: Dialogue Framework for Ontology Authoring

In this section we present Perico [1], a framework for describing and building dialogues for supporting ontology authoring. The definition of the Perico framework is guided by the requirements the we identified at the beginning of this chapter.

5.3.1 Informational View of Perico

All the dialogue information in Perico is captured in a **Dialogue State** ($\mathcal{S}$), which is a triple consisting of the following **informational components** (IC): a **Reference Ontology**, $\mathcal{O}_{\text{Reference}}$, an **Execution Agenda** and a **Dialogue History**.

[1] The name "Perico" is Spanish for parakeet.

$$\mathcal{S} : \begin{bmatrix} \mathcal{O}_{\text{Reference}} & : & \mathsf{Set}(\mathsf{Axiom}) \\ \text{Agenda} & : & \mathsf{Stack}(\mathsf{DialoguePlan}) \\ \text{History} & : & \mathsf{Seq}(\mathsf{FunctionalSegment}) \end{bmatrix} \tag{5.1}$$

Reference Ontology

The **formal representation** of $\mathcal{O}_{\text{Reference}}$ is an ontology as defined in Section 4.3.2. It represents the current knowledge of the system participant. Perico further assumes that the reference ontology corresponds to the *agreed knowledge* between the dialogue participants. Note that, the knowledge in the reference ontology does not have to be explicitly agreed upon during the dialogue: it may be assumed to be agreed. This assumption means that existing ontologies can be used as a starting reference ontology as they are assumed to contain agreed knowledge between the participants. During the course of the dialogue, it may become clear that some axioms in the reference ontology are not shared between all participants in the dialogue. In such cases, those axioms must be removed from the agreed ontology.

The reference ontology may consist of various modules, for example, a reference ontology will typically be the union of a domain ontology, which contains knowledge about the domain under discussion, and a discourse ontology, which contains knowledge about the meaning of dialogue moves.

Execution Agenda

The **Execution Agenda** contains information about the goals and plans of the system agent that are driving the current dialogue. In Perico the goals and plans are represented using a stack of dialogue plans:

Definition 5 (Dialogue Plan)**.** *A dialogue plan is a 4-tuple of:*

parameters *typed slots that can be filled with specific instances during the execution of the dialogue plan;*

pre-conditions *describe when the plan can be executed;*

effects *describe any changes to the dialogue state that occur when this plan is executed;*

subplans *a set of dialogue plans that can be executed to achieve the goal of this plan. Subplans may consist of alternative strategies to achieve the goal of this plan, or it may consist of a sequence of plans that need to be executed in order to achieve the goal of this plan.*

This definition without the subplans is widely used in the literature, see for example [98].[1] Note that dialogue plans are parameterised; the execution agenda can only contain instantiated dialogue plans where the parameters refer to specific information from the dialogue state.

Types of Dialogue Plans This definition of dialogue plans is flexible and provides a way to specify the behaviour of dialogue systems. When building dialogue systems it is useful to discern between the following types of dialogue plans[2]:

basic plans describe built-in plans in Perico that correspond to dialogue moves from ISO-DIS-24617-2. These basic plans are domain independent and serve as building blocks for more complex plans such as grounding and domain-task plans. In Section 5.3.3 we discuss the basic plans provided in Perico.

grounding plans describe how and when the reference ontology is updated. Perico provides a default set of grounding plans for the supported dialogue moves, since the discourse semantics of dialogue moves pre-determine how they should be ground. This enables dialogue engineers to focus on specifying the domain-task plans (see below); however, dialogue engineers may provide alternative grounding plans (for example, by requiring explicit confirmation of all grounded knowledge).

domain-task plans describe how dialogue moves should be combined with agreed domain knowledge to achieve some domain task. These plans are always provided by dialogue engineers.

[1] The subplans element is borrowed from [15] in order to make task trees explicit as discussed below.

[2] The types of dialogue plans are not relevant to the dialogue state; the execution agenda can contain any type of dialogue plan.

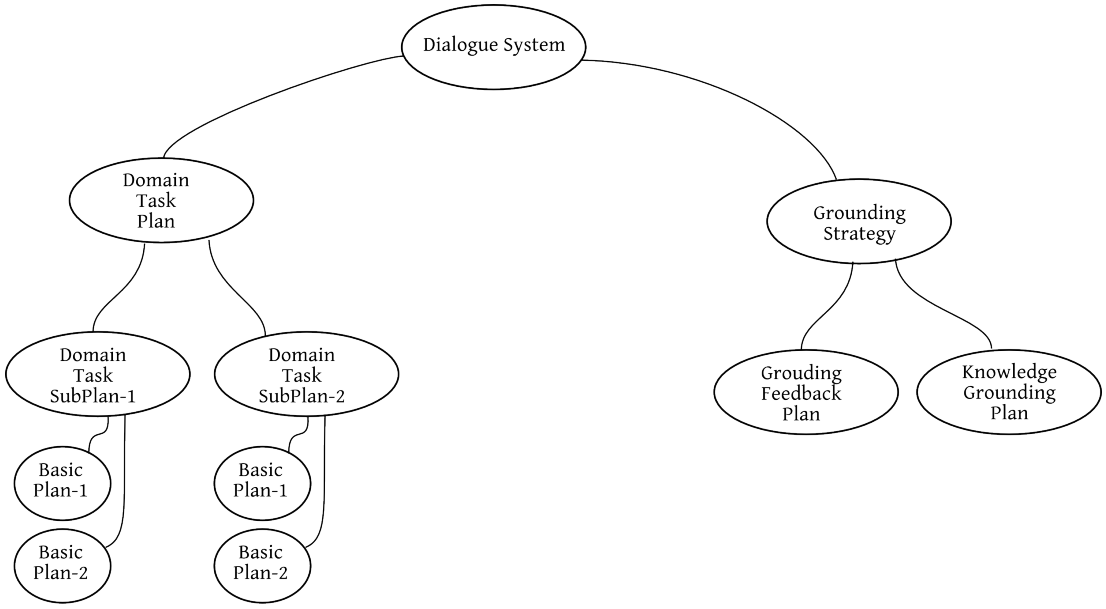

Figure 5.4: Generic task tree of a dialogue system in Perico. A dialogue system consists of one of more domain-task plans and a grounding plan, which schedule basic plans to achieve their goals.

Task Trees Dialogue plans can be composed into larger structures at design time (or at runtime) via their subplans; we call these larger structures **task tree**s. Inversely, all nodes in task trees are dialogue plans. Perico uses task trees as the main way to describe dialogue systems[1]. The task tree effectively specifies which set of dialogue moves are available at any point of the dialogue; in terms of the information-state-update view of dialogues, the task tree defines the *update rules and strategies*. Figure 5.4 presents a generic task tree for a dialogue system in Perico.

[1]This notion is based on the RavenClaw style of dialogue task specification based on task trees. However, where RavenClaw uses an agent-based terminology for the nodes of the task tree (with dialogue agencies and agents), we use the terminology of dialogue plans.

Dialogue History

The **Dialogue History** in Perico contains information about the previous *functional segments* of the current dialogue. For this, Perico reuses and refines the definition of functional segments from ISO-DIS-24617-2. The dialogue history is represented as a sequence of functional segments; as a consequence the dialogue history *provides the discourse structure* of the dialogue. Next, we describe how Perico extends the ISO-DIS-24617-2 (see Section 5.2.2) standard.

First, Perico defines two types of **participants**: the system participant, which is a Perico System Agent and other participants, which are Perico User Agents. In this thesis, we assume that every dialogue occurs between a single system participant (the dialogue authoring system) and a user participant (the ontology author).

Second, Perico extends the ISO standard by taking into account the **provenance of discourse data**. Since Perico can be used to develop dialogue systems, it prescribes two ways of generating discourse data for a dialogue. For dialogue moves produced by the system agent, the discourse data must be generated by some discourse plan (see below for a definition of discourse plans). Data about dialogue moves by user participants is generated through a *discourse reading* which is the result of some discourse analysis process. A discourse reading is defined as an extension to the notion of syntactic reading defined in Entendre:

Definition 6 (Discourse Reading)**.** *Let ι be an input string, $r^{\mathcal{L}}$ be a syntactic reading of ι over a reference ontology $\mathcal{O}$ that is part of a dialogue state $\mathcal{S}$, and $\Gamma_{\mathcal{M}}$ be the set of strategies for mapping input segments onto dialogue moves. Then a Discourse Reading of ι is an annotation graph over a set of labels $L_{\mathcal{M}}$. A label in $L_{\mathcal{M}}$ is a tuple $\langle \mathcal{M}, \mu \rangle$, where:*

- *$\mathcal{M}$ is a set of dialogue moves and*
- *$\mu \in \Gamma_{\mathcal{M}}$ is a discourse analysis strategy: it assigns one or more dialogue moves to a syntactic reading based on the dialogue state $\mathcal{S}$.*

A third extension to ISO-DIS-24617-2 is that Perico **explicitly relates dialogue moves to their "semantic content"**.[1] Perico requires each dialogue

[1] As discussed in Section 5.2.2, the ISO-DIS-24617-2 standard only gives a general definition

move to be associated to zero or one semantic content, which encodes domain-specific information provided by the dialogue move. A dialogue move has no semantic content when it is purely defined by its communicative function and its relations to previous moves. For example, an `agree`ment with a previous dialogue move has no semantic content of its own. A dialogue move has a semantic content when its communicative function requires some extra domain information: for example dialogue moves which are `question`s, must provide some domain-dependent semantic content defining what is being asked. Another example is an `inform` move where the semantic content describes what is being informed. In Perico the semantic content is represented by an axiom in an ontology language.[1]

A fourth extension is that ISO-DIS-24617-2 only defines one type of feedback dependency and one type of functional dependency. Perico introduces typed functional dependencies as binary relations between two dialogue acts. For example, the ISO standard can only state that a dialogue move `dm2` has communicative function `inform` and a functional dependency to some previous dialogue move `dm1`. Perico allows for a more fine-grained description by referring to the type of functional dependency that holds between `dm1` and `dm2`, for example: `justifies` or `rephrases`.

Finally, because the domain knowledge is represented using ontologies, Perico is able to **extend and formalise dialogue moves** from the ISO standard in terms of dialogue plans, dialogue readings, ontologies and reasoning services related to ontologies. These extensions and formalisations are presented in detail in Section 5.3.3.

Perico's extensions to the ISO dialogue metamodel (see Figure 5.1) are summarised in Figure 5.5.

of "semantic content" for a dialogue moves, but does not include it as part of the model for dialogue moves.

[1]In practice, this axiom may be accompanied by extra information to clarify whether an the axiom represents a question or to provide extra information for generating verbalisations of the axiom.

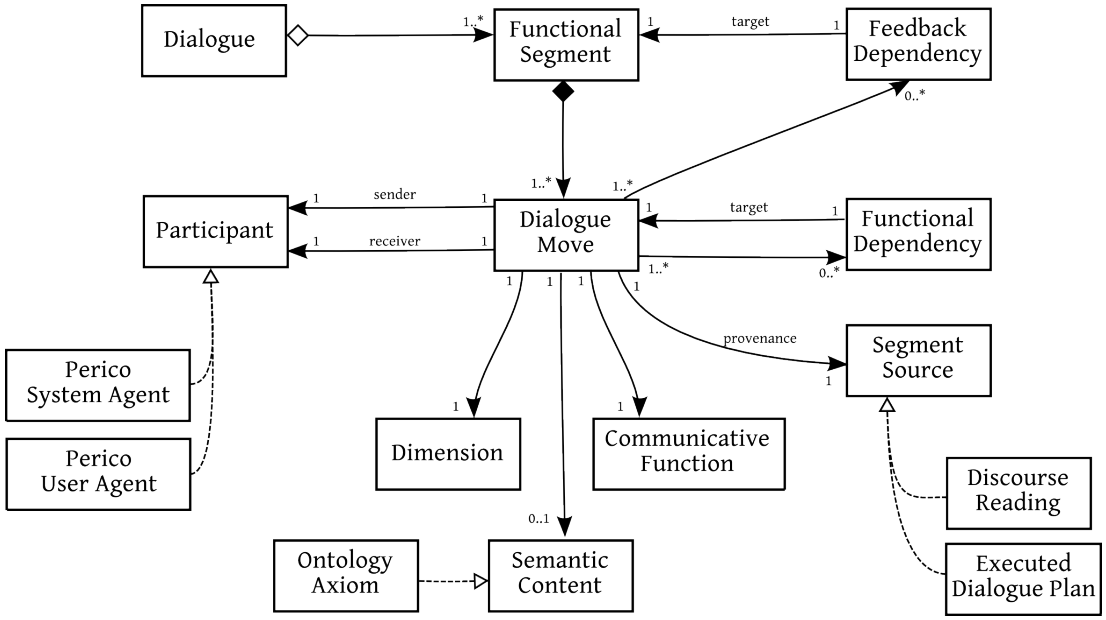

Figure 5.5: UML Class diagram showing the data model in Perico for describing the discourse structure of a dialogue. This datamodel extends the dialogue metamodel shown in Figure 5.1 by (i) specifying types of Participants, (ii) adding the Semantic Content, (iii) adding a Segment Source and (iv) modelling functional and feedback dependencies as classes rather than just relationships.

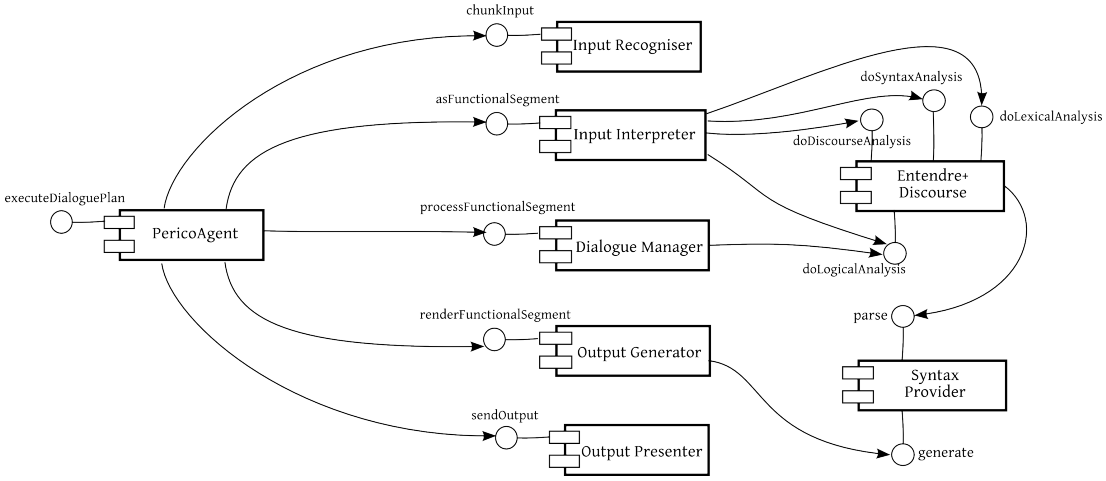

Figure 5.6: Functional view of Perico showing the main components, the services they provide and the functional dependencies between the components.

5.3.2 Functional Components

Perico follows the general architecture of dialogue systems as discussed in Section 5.2. Here, we discuss specific design decisions made in Perico to adapt dialogue systems for ontology authoring.

Perico uses a hub-like design where the dialogue is mediated by a **PericoAgent**, which communicates with the components of the dialogue pipeline as shown in Figure 5.6. The rationale for using a hub-like design is that the `PericoAgent` is the only component that controls the dialogue state. By using this design instead of strictly following the general architecture as depicted in Figure 5.3, we allow decoupling of the 5 generic components in the dialogue pipeline. These components can provide services that are independent of the current dialogue state and the pipeline components are not aware of each other.

Dialogue Pipeline Components

Perico does not impose major restrictions regarding the **Input Recogniser** and **Output Presenter**. However, since Entendre focuses on textual input analysis, in practice these two components will also be text-based.

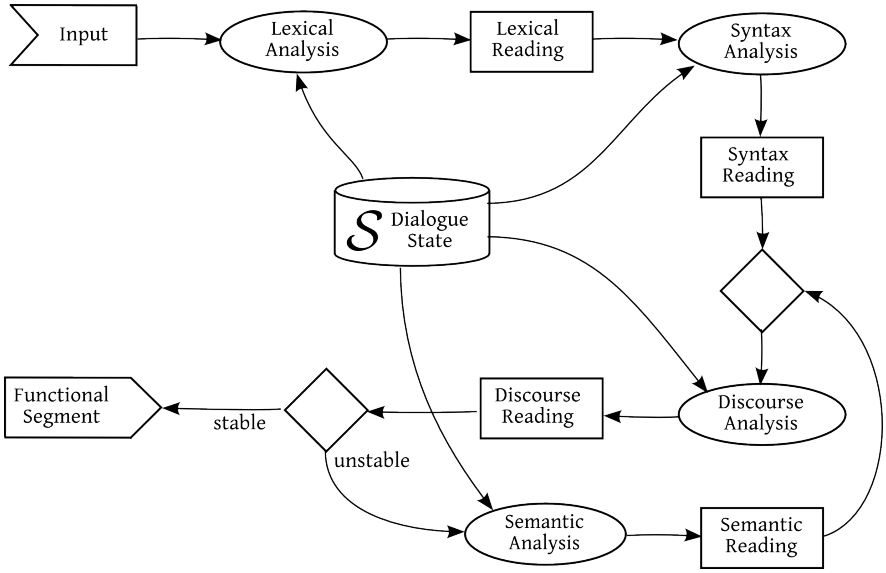

Figure 5.7: UML Activity Diagram showing the workflow for the `Input Interpreter` component in Perico. The workflow is used for analysing inputs from other dialogue participants and converting them into functional segments of the dialogue.

The **Input Interpreter** component in Perico reuses the Entendre pipeline of lexical, syntax and semantic analysers (see Chapter 4). Furthermore, in Perico we add a new type of analyser called a *Discourse Analyser*: a component that extends a syntactic reading and derives a *Discourse Reading* of the input (i.e. it attaches dialogue move annotations to the input). The output of the `Input Interpreter` is based on the dialogue moves in the discourse reading, which are aggregated into a `Functional Segment`.[1] [2] Figure 5.7 gives an overview of the input interpretation process in Perico.

The **Dialogue Manager** component in Perico consists of two subcomponents: the *knowledge grounder* and the *dialogue plan executer*. The knowledge grounder

[1]This follows the model of dialogues from the ISO-DIS-24617-2 standard discussed in Section 5.2.2.

[2]Note that some discourse readings require more than just a syntax reading; for example, to determine whether an input is an `answer` to a `set question`, Perico needs to perform semantic analysis of the semantic content associated with the dialogue move.

may use the semantic analysis from Entendre to decide whether to add or remove knowledge to the reference ontology. The knowledge grounder also can inject *grounding plans* to the execution agenda to provide feedback about how messages have been interpreted, or to request confirmation. The dialogue plan executer is responsible for determining Perico's next dialogue move based on the current dialogue state, which includes a history of functional segments and the current execution stack (the set of dialogue plans that are being executed). The output of the `Dialogue Manager` is a functional segment (consisting of one or more dialogue moves) that needs to be performed.

Finally, the **Output Generator** converts a functional segment proposed by the `dialogue manager` and converts it into an output that can be understood by the other dialogue participants. Since functional segments consist of both discourse and domain information, the `output generator` needs to be able to convert both types of information. For domain information, Perico can reuse the rendering syntax (e.g. a CNL-generator) discussed in Section 4.3.1 (on page 104). For discourse information, Perico uses simple text-based templates since we are not aware of a CNL that we can reuse for this purpose (and developing a CNL for discourse information is not in the scope of this PhD).

5.3.3 Basic Dialogue Plans in Perico

Perico defines a set of basic dialogue plans which can be composed into more complex behaviour of a dialogue system. Perico provides basic plans that produce functional segments for each of the main communicative functions defined by the ISO standard [74] and depicted in Figure 5.2. Providing a formalisation for all of these dialogue plans is not in the scope of this PhD since not all of the communicative functions are necessary for ontology authoring. This section shows how some basic dialogue plans can be defined in terms of the current dialogue state (in the precondition of the plan) and a desired dialogue state (in the effects of the plan). This section also shows how Perico also defines dialogue plans which extend the basic taxonomy of communicative functions defined by the ISO standard. These extensions rely on reasoning and input analysis services described in Chapter 4.

Basic Knowledge Grounding

The syntactic and semantic analyses from Entendre make it possible to define a couple of basic knowledge grounding strategies.

Dialogue Plan: GroundSyntacticallyCorrect

Params : fs : FunctionalSegment

Precondition :
- fs contains a dialogue move dm
- dm has an information providing communicative function
- dm provenes from a syntactic reading $r^{\mathcal{L}}$

Effect :
- **if** $r^{\mathcal{L}} \in$ **Understood**
- $\mathcal{O}_{\text{Reference}} := \mathcal{O}_{\text{Reference}} + \text{AxiomsIn}(r^{\mathcal{L}})$
- **else** Skip

Subplans : $\emptyset$

Dialogue Plan: GroundSafeAxioms

Params : fs : FunctionalSegment

Precondition :
- fs contains a dialogue move dm
- dm has an information providing communicative function
- dm provenes from a semantic reading $r^{\mathcal{C}}$

Effect :
- **if** $r^{\mathcal{C}} \in \{\mathbf{N}, \mathbf{N}+\}$
- $\mathcal{O}_{\text{Reference}} := \mathcal{O}_{\text{Reference}} + \text{AxiomsIn}(r^{\mathcal{L}})$
- **else** Skip

Subplans : $\emptyset$

Basic Information Providing

The most common dialogue move used in dialogues is to `inform` a fact to another dialogue participant. Perico provides plan `InformAx`, which takes an axiom that is entailed by the reference ontology (in the current dialogue state) and generates a functional segment that expresses this axiom. The formal definition of this dialogue plan is as follows:

Dialogue Plan: InformAx

Params : α : Axiom

Precondition : $\mathcal{O}_{\text{Reference}} \models \alpha$

Effect :
- History := History + fs
- fs contains a dialogue move dm
- dm matches [sender = `me`, sc = α, cf = `inform`, prov = `InformAx`(α)]

Subplans : $\emptyset$

Note that since `InformAx` requires an axiom as a parameter, it may not be available for all types of moves: not all information may be available or may be expressible in terms of (a module from) the reference ontology. For this reason, Perico offers a plan that will simply output a given string parameter.

Dialogue Plan: InformString

Params : msg : String
Precondition : $\emptyset$
Effect :
$$\begin{cases} \text{History} := \text{History} + \text{fs} \\ \text{fs contains a dialogue move dm} \\ \text{dm matches } [\text{sender} = \texttt{me}, \text{cf} = \texttt{inform}, \text{prov} = \texttt{InformString}(\text{msg})] \end{cases}$$
Subplans : $\emptyset$

Plan `InformAx` is preferred over `InformString` because Perico can use reasoning services to analyse α and may be able to automatically handle subsequent functional segments. For example, if a participant disagrees with the output functional segment, Perico can automatically execute a resolution strategy for determining the validity of α (and whether it should be removed from the reference ontology); this can only be done if the move is generated via the `InformAx`.

Information Provision based on Ontology Services

Ontologies and related reasoning services allow for more complex types of `inform` moves where the semantic content is not simply an axiom, but rather some analysis result based on an axiom. We distinguish between a variety of such moves based on analysis types.

Integration analysis provides information about the classification of axioms as described in the chapter on Entendre. Perico defines some plans for conveying some of the information gathered during the Entendre analysis.

First, whenever another participant produces an utterance and Perico produces a successful syntax analysis of that utterance, Perico can perform integration analysis on the read axioms from the utterance and inform the participant about whether the axiom is novel or leads to inconsistency, etc.

Dialogue Plan: InformEntendreAxiomClassification

Params : α : Axiom

Precondition :
- There exist $\mathrm{dm}_i, r^{\mathcal{M}}, r^{\mathcal{L}}, r^{\mathcal{C}}$ such that:
- dm_i is a dialogue move that occurs in History,
- dm_i `matches` [cf = `inform`, sc = α, receiver = `me`, provenance = $r^{\mathcal{M}}$],
- $r^{\mathcal{M}}$ is a discourse reading that is based on $r^{\mathcal{L}}$
- $r^{\mathcal{L}}$ is a syntactic reading that contains α in one of its labels,
- $r^{\mathcal{C}}$ states that $\alpha \in$ **N, N+, R, I** or **U**

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `inform`, sc = $r^{\mathcal{C}}$, dim = `OntAuth`,
  - fundep = `classifiesAxIn`(dm_i), prov = `InformEntendreAxiomClassification`(α)]

Subplans : $\emptyset$

When the integration analysis for an axiom is **N+** (the axiom is novel and has relevant new implication), Perico can inform other participants about which new implications were found as well as how many were found.

Dialogue Plan: InformsRelevantImplicationsCount

Params : α : Axiom

Precondition :
- There exist $\mathrm{dm}_i, r^{\mathcal{M}}, r^{\mathcal{L}}, r^{\mathcal{C}}$ such that:
- dm_i is a dialogue move that occurs in History,
- dm_i `matches` [cf = `inform`, sc = α, receiver = `me`, provenance = $r^{\mathcal{M}}$],
- $r^{\mathcal{M}}$ is a discourse reading that is based on $r^{\mathcal{L}}$
- $r^{\mathcal{L}}$ is a syntactic reading that contains α in one of its labels,
- $r^{\mathcal{C}}$ states that $\alpha \in$ **N** or **N+** with new relevant implication set $\Delta^{\mathcal{E}}_{\alpha}$

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `inform`, sc = $\{|\Delta^{\mathcal{E}}_{\alpha}| = x\}$, dim = `OntAuth`,
  - fundep = `countsRelevantImplications`(dm_i),
  - prov = `InformsRelevantImplicationsCount`(α)]

Subplans : $\emptyset$

Dialogue Plan: InformsRelevantImplications

Params : α : Axiom

Precondition :
- There exist $\mathrm{dm}_i, r^{\mathcal{M}}, r^{\mathcal{L}}, r^{\mathcal{C}}$ such that:
- dm_i is a dialogue move that occurs in History,
- dm_i `matches` [cf = `inform`, sc = α, receiver = `me`, provenance = $r^{\mathcal{M}}$],
- $r^{\mathcal{M}}$ is a discourse reading that is based on $r^{\mathcal{L}}$
- $r^{\mathcal{L}}$ is a syntactic reading that contains α in one of its labels,
- $r^{\mathcal{C}}$ states that $\alpha \in \mathbf{N}+$ with new relevant implication set $\Delta^{\mathcal{E}}_{\alpha}$

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `inform`, sc = $\Delta^{\mathcal{E}}_{\alpha}$, dim = `OntAuth`,
  - fundep = `informsRelevantImplications`(dm_i),
  - prov = `InformsRelevantImplications`(α)]

Subplans : $\emptyset$

Whenever an axiom is entailed by the reference ontology, that axiom can be justified. Perico only provides a basic move for informing other participants about one justification.[1]

Dialogue Plan: InformsOneJustificationFor

Params : α : Axiom

Precondition :
- There exist a dm_i such that:
- dm_i is a dialogue move that occurs in History,
- dm_i `matches` [cf = `inform`, sc = α],
- $\mathcal{O}_{\mathrm{Reference}} \models \alpha$,
- $\alpha \notin \mathcal{O}_{\mathrm{Reference}}$

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `inform`, sc = $\mathcal{J}(\alpha, \mathcal{O}_{\mathrm{Reference}})$, dim = `OntAuth`,
  - fundep = `informsOneJustificationFor`(dm_i),
  - prov = `InformsOneJustificationFor`(α)]

Subplans : $\emptyset$

Finally, another plan provided by Perico is that of rephrasing a previously made statement. This plan is only possible for dialogue moves that provene from some input axiom (i.e. generated through an `InformAx` plan or an $\mathcal{L}$-atomic syntactic reading). The output axiom is then equivalent to the input axiom

[1] There are various other ways to provide justifications for an axiom which we leave as future work since they are refinements of the single justification case: all the justifications for the axiom can be provided instead of only a single justification. Also, only part of a justification can be provided, for example, because some axioms in the justification are deemed to be trivial.

(according to the reference ontology), but has a different signature.

Dialogue Plan: Rephrases

Params : α : Axiom

Precondition :
- There exist dm_i, β such that:
- dm_i is a dialogue move that occurs in History,
- dm_i `matches` [cf = `inform`, sc = α],
- $\sigma(\alpha) \neq \sigma(\beta)$,
- β is equivalent to α through an entity mapping such that
- for all entities $e \in \sigma(\alpha)$ there is an entity $e' \in \sigma\beta$ such that $\mathcal{O}_{\mathrm{Reference}} \models e \equiv e'$

Effect :

Subplans : $\mathrm{InformAx}(\alpha_o)$

Other Dialogue Plans

Perico follows the ISO-DIS-24617-2 hierarchy of dialogue moves (see Figure 5.2) and provides other dialogue plans such as information seeking moves. For example, in order to produce a `propositional question`, Perico defines a dialogue plan `PropositionalQuestionAx` which takes an axiom α, and produces a dialogue move that asks whether α is true or not.

Dialogue Plan: PropositionalQuestionAx

Params : α : Axiom

Precondition : $\mathcal{O}_{\mathrm{Reference}} \not\models \alpha$

Effect :
- History := History + fs
- fs contains a dialogue move dm
- dm matches [sender = `me`, sc = α, cf = `propositionalQuestion`,
- prov = `PropositionalQuestionAx`(α)]

Subplans : $\emptyset$

The ISO-DIS-24617-2 requires for propositional questions to only be asked when the producer of the move does not know whether α is true or not. Often, dialogue systems need to check whether an α that is entailed by the reference ontology is really true; for such situations ISO-DIS-24617-2 provides the `checkQuestion` dialogue move. Perico provides plan `CheckQuestionAx` for generating such questions; it is the same as `PropositionalQuestionAx`, except for its precondition:

Dialogue Plan: CheckQuestionAx

Params : α : Axiom
Precondition : $\mathcal{O}_{\text{Reference}} \models \alpha$
Effect :
$$\begin{cases} \text{History} := \text{History} + \text{fs} \\ \text{fs contains a dialogue move dm} \\ \text{dm matches } [\text{sender} = \texttt{me}, \text{sc} = \alpha, \text{cf} = \texttt{checkQuestion}, \text{prov} = \texttt{CheckQuestionAx}(\alpha)] \end{cases}$$
Subplans : $\emptyset$

Finally, Perico offers a basic plan for asking `setQuestions`, which are questions where a specific bit of information is being asked. To model this using axioms as the semantic content of dialogue moves, we define a special set of entities $\mathcal{U}$ which represent variables being asked. The basic plan for asking `setQuestions` is:

Dialogue Plan: SetQuestionAx

Params : α : Axiom
Precondition : $\sigma(\alpha) \bigcap \mathcal{U} = u$
Effect :
$$\begin{cases} \text{History} := \text{History} + \text{fs} \\ \text{fs contains a dialogue move dm} \\ \text{dm matches } [\text{sender} = \texttt{me}, \text{sc} = \alpha, \text{cf} = \texttt{setQuestion}, \text{prov} = \texttt{SetQuestionAx}(\alpha)] \end{cases}$$
Subplans : $\emptyset$

Note that by this definition, we restrict `setQuestions` to have a single unknown variable.

Although Perico can represent information-seeking moves, implementing a query answering analysis is not in the scope of this PhD and including query answering plans such as `AnswerAx`, `ConfirmAx`, `DisconfirmAx`, `DeclineAnswer` and `AnswerUnknownAx` is left as future work.

5.3.4 Conclusion

This section presented the Perico framework for ontology authoring dialogues. The presented framework fulfils the requirements that we identified at the beginning of the chapter:

- Perico's **informational components** are such that:

  - the information is described from the perspective of one of the participants in the dialogue: the *system participant*;
  - the domain knowledge is represented using ontologies;
  - the discourse knowledge is represented by extending ISO-DIS-24617-2, taking into account ontologies and their reasoning services (see Section 5.3.1);
  - the task knowledge extends the notion of task trees from RavenClaw by taking into account the richer set of dialogue moves provided by the ISO-DIS-24617-2 standard (see Section 5.3.1);
  - a standard dialogue state is defined based on the chosen representation for domain, discourse and task knowledge;

- Perico's **functional components** follow the main basic dialogue system architecture shown in Figure 5.3, but are influenced by the choices for representing domain, discourse and task. In particular, Perico reuses and extends Entendre to describe the standard dialogue system components. A detailed description is given in Section 5.3.2.

- Perico defines key parts of the **information-state-update** view:

  - basic dialogue moves from ISO-DIS-24617-2 are defined as well as some specific dialogue moves made possible by the domain knowledge;
  - dialogue grounding is defined in terms of changes to the domain ontologies.

Perico itself does not include any dialogue systems for ontology authoring. In the next section we validate the Perico framework by formalising existing ontology authoring interactions.

5.4 Validation of Perico with a Basic Ontology Authoring Dialogue

In this section, we validate the generality of the Perico framework by using it to formalise an existing dialogue authoring interaction. We will use the following

methodology to formalise the interaction:

- we select and annotate an example dialogue between the system and an ontology author,
- we discuss which extensions are required to formalise the dialogue interaction and provide a formalisation of this dialogue system.

5.4.1 Annotating Entendre Feedback

This section shows a simulated interaction between a novice ontology author and ROO. The input sentences and the reference ontology are the same as we used on the evaluation study on Entendre (see Section 4.6). The feedback is generated by ROO (see Section 4.5). The selected interaction focuses on syntactically correct inputs that add new information to the ontology. We intentionally do not show how an existing axiom can be deleted from the reference ontology. We leave axiom deletion as future work as we did when defining Entendre feedback. To avoid repetition, we only show the annotated version of the interaction below.

In order to annotate the discourse structure of the interaction, we first chunked the feedback provided by Entendre. This was done to avoid having one big functional segment with a big set of dialogue moves. When annotating the interaction, we tried to reuse the dialogue plans defined in Perico. In the cases when no suitable dialogue plan was available, we added new discourse instances such as functional dependencies or communicative functions. In the next section we summarise which additions to Perico were required. Note also that during this annotation, we did not look for a task structure, focusing only on the discourse structure of individual functional segments. We do provide an annotation about changes to the ontology that occur.

We present the interaction as a sequence of functional segments, for each functional segment we show:

- the functional segment name, of the form **fsx**, where x is the index of the segment;
- the name of the sender (either OA or Perico);

- the input or output of the functional segment; this can be textual or a description of some action (e.g. a button click by the ontology author or button enablement by the system).
- the dialogue move information of the functional segment and
- if available, the syntactic and integration reading of the textual input.

At the beginning of the dialogue the Perico agent is initialised with some dialogue plan and the Leeds Point-of-Interest ontology, $\mathcal{O}_{\texttt{LeedsPOI}}$:

$$\mathcal{S}_0 = \begin{bmatrix} \mathcal{O}_{\text{Reference}} & = & \mathcal{O}_{\text{LeedsPOI}} \\ \text{Agenda} & = & [??] \\ \text{History} & = & \mathsf{Seq}() \end{bmatrix} \tag{5.2}$$

Dialogue Segment for Axiom Category N

fs1 OA: Every Student Accommodation is a kind of Accomodation.
[id = dm1-1, sc = α_1, dim = `LeedsPOI`, cf = `inform`][1]
$r^{\mathcal{L}}_{\text{fs1}} = \alpha_1 =$ `StudentAccommodation` $\sqsubseteq$ `Accommodation`
$r^{\mathcal{C}}_{\text{fs1}} = \alpha_1 \in \mathbf{N}$

fs2 Perico: This assertion is novel:
[id = dm2-1, sc = $r^{\mathcal{C}}_{\text{fs1}}$, dim = `autoFeedback`, feedep = fs1, cf = `autoPositive`]
[id = dm2-2, dim = `OntologyAuthoring`, fundep = `classifiesAxIn`(dm1-1), cf = `inform`]

fs3 Perico: it has not been added to the ontology yet.
[id = dm3-1, dim = `OntologyAuthoring`, fundep = `rephrases`(dm2-2), cf = `inform`]

fs4 Perico: However, no further implications could be made based on this input.
[id = dm4-1, dim = `autoFeedback`, feedep = fs1, cf = `autoPositive`]
[id = dm4-2, sc = $\{\Delta^{\mathcal{E}}_{\alpha_1} = \emptyset\}$, dim = `OntologyAuthoring`,
fundep = {`contrasts`(dm3-1), `countsRelevantImplications`(dm1-1)}, cf = `inform`]

fs5 Perico: This generally means that the terms you are using have been loosely defined in the ontology. It is also possible that there are some implications that could not be found.
[id = dm5-1, dim = `OntologyAuthoring`, feedep = fs1, fundep = `informsRelevantImplications`(dm4-2), cf = `inform`]

fs6 Perico: Advice: Add the sentence...
[id = `dm6 - 1`, dim = `OntologyAuthoring`, fundep = `instructAddInput`(dm1-2), cf = `directProceduralInstruction`]

[1]The legend for the annotations is as follows: id is the unique identifier, sc is the semantic content, dim is the dimension, cf stands for communicative function, feedep stands for feedback dependency, `fundep` stands for functional dependency. For the semantic content, we will be reusing the notation introduced in Section 4.3.2 and we assume that we have an axiom in an ontology language representing the same information.

fs7 Perico: ...and maybe add more new definitions if you want the ontology to be able to make more inferences
[id = dm7 − 1, dim = OntologyAuthoring, fundep = instructAddMoreRelated(dm1 − 2),
cf = directProceduralInstruction]

fs8 Perico: *enables OK-button*
[id = dm8 − 1, dim = OntologyAuthoring, fundep = offersAcceptOfReading(dm1-1), cf = Offer]

fs9 OA: *clicks on OK-button*

[id = dm9 − 1, dim = OntologyAuthoring, fundep = accepts(dm8-1), cf = acceptOffer]

After this interaction the dialogue state is as follows:

$$\mathcal{S}_9 = \begin{bmatrix} \mathcal{O}_{\text{Reference}} & = & \mathcal{O}_{\text{LeedsPOI}} + \alpha_1 \\ \text{Agenda} & = & [??] \\ \text{History} & = & [fs1 \ldots fs9] \end{bmatrix} \tag{5.3}$$

Dialogue Segment for Axiom Category N+

fs10 OA: Every Teaching Hospital is a kind of Hospital.
[id = dm10-1, sc = α_2, dim = LeedsPOI, cf = inform]
$r^{\mathcal{L}}_{\text{fs10}} = \alpha_2 = \texttt{TeachingHospital} \sqsubseteq \texttt{Hospital}$
$r^{\mathcal{C}}_{\text{fs10}} = \alpha_2 \in \mathbf{N}+$

fs11 Perico: This assertion is **novel**:
[id = dm11-1, dim = autoFeedback, feedep = fs10, cf = autoPositive]
[id = dm11-2, sc = $r^{\mathcal{C}}_{\text{fs10}}$, dim = OntologyAuthoring, fundep = classifiesAxIn(dm10-1), cf = inform]

fs12 Perico: it has not been added to the ontology yet.
[id = dm12-1, dim = OntologyAuthoring, fundep = rephrases(dm11-2), cf = inform]

fs13 Perico: This input **implies** 6 new things.
[id = dm13-1, dim = autoFeedback, feedep = fs10, cf = autoPositive]
[id = dm13-1, sc = $\{|\Delta^{\mathcal{E}}_{\alpha_2}| = 6\}$, dim = OntologyAuthoring, fundep = countsRelevantImplications(dm10-1),
cf = inform]

fs14 Perico: Have a look at the list of implications
[id = dm14 − 1, dim = OntologyAuthoring, fundep = inspectRelevantImplicationsFor(dm10-1),
cf = directProceduralInstruction]

fs15 Perico: to make sure you agree with the implications.
[id = dm15 − 1, dim = OntologyAuthoring, fundep = checkAgreementRelevantImplicationsFor(dm10-1), cf = directProceduralInstruction]

fs16 Perico: If you do not agree, it may be that you are using the wrong terminology.
[id = dm16 − 1, dim = OntologyAuthoring, fundep = informsRelevantImplications(dm15-1), cf = inform, modality = uncertain]

fs17 Perico: This input implies that:

- *Every Teaching Hospital has footprint a Footprint.*

- *Organisation and Teaching Hospital are mutually exclusive.*
- *Training Centre and Teaching Hospital are mutually exclusive.*
- *Every Teaching Hospital is a kind of Topographic Object.*
- *Every Teaching Hospital is a kind of Place.*
- *Teaching Hospital and University (Institution) are mutually exclusive.*

[id = dm17-1, dim = `autoFeedback`, feedep = fs10, cf = `autoPositive`]
[id = `dm17` − 2, sc = $\Delta^{\mathcal{E}}_{\alpha_2}$, dim = `OntologyAuthoring`, fundep = `informsRelevantImplications`(dm10-1), cf = `inform`]

fs18 Perico: *enables OK-button*
[id = `dm18` − 1, dim = `OntologyAuthoring`, fundep = `offersAcceptOfReading`(`dm10` − 1), cf = `Offer`]

fs19 OA: *clicks on OK-button*

[id = `dm19` − 1, dim = `OntologyAuthoring`, fundep = `accepts`(dm18-1), cf = `acceptOffer`]

At this point the dialogue state is as follows:

$$\mathcal{S}_{19} = \begin{bmatrix} \mathcal{O}_{\text{Reference}} & = & \mathcal{O}_{\text{LeedsPOI}} + \alpha_1 + \alpha_2 \\ \text{Agenda} & = & [??] \\ \text{History} & = & [fs1 \ldots fs19] \end{bmatrix} \tag{5.4}$$

Dialogue Segment for Axiom Category R

fs20 OA: Every University has part at least one Point of Interest.
[id = dm20-1, sc = α_3, dim = `LeedsPOI`, cf = `inform`]
$r^{\mathcal{L}}_{\text{fs20}} = \alpha_3 =$ `University` $\sqsubseteq \geq 1$`hasPart`.`PointOfInterest`
$r^{\mathcal{C}}_{\text{fs20}} = \alpha_3 \in \mathbf{R}$

fs21 Perico: This statement is redundant.
[id = dm21-1, dim = `autoFeedback`, feedep = fs20, cf = `autoPositive`]
[id = dm21-2, sc = $r^{\mathcal{C}}_{\text{fs20}}$, dim = `OntologyAuthoring`, fundep = `classifiesAxIn`(dm20-1), cf = `inform`]

fs22 Perico: Although the sentence itself is not included in the ontology, this sentence can be inferred from the following sentences that have been said to be true.
[id = dm22-1, dim = `OntologyAuthoring`, fundep = `rephrases`(dm21-2), cf = `inform`]

fs23 Perico: Check the justification:
[id = `dm23` − 1, dim = `OntologyAuthoring`, fundep = `inspectJustificationFor`(dm20-1), cf = `directProceduralInstruction`]

fs24 Perico: Existing Rabbit statements implying that *Every University has part at least one Point of Interest*:

- *Every University has part a Building that has purpose Education of University Students.*
- *Every Building is a kind of POI.*
- *POI and Point of Interest are equivalent.*

[id = dm24-1, dim = `autoFeedback`, feedep = fs10, cf = `autoPositive`]
[id = $\texttt{dm24} - \texttt{2}$, sc = $\mathcal{J}(\alpha_3, \mathcal{O})$, dim = `OntologyAuthoring`,
fundep = `informsOneJustificationFor`(dm20-1), cf = `inform`]

fs25 Perico: *enables OK-button*

[id = $\texttt{dm25} - \texttt{1}$, dim = `OntologyAuthoring`, fundep = `offersAcceptOfReading`($\texttt{dm20} - \texttt{1}$), cf = `Offer`]

At this point the ontology author does not click on the `OK`-button, but starts entering the following input sentence. The agreed knowledge stays thus the same (only the history changes):

$$\mathcal{S}_{25} = \left[\begin{array}{ccc} \mathcal{O}_{\text{Reference}} & = & \mathcal{O}_{\text{LeedsPOI}} + \alpha_1 + \alpha_2 \\ \text{Agenda} & = & [??] \\ \text{History} & = & [fs1 \ldots fs25] \end{array}\right] \tag{5.5}$$

Dialogue Segment for Axiom Category U

fs26 OA: Every Student Union is contained within a University(Institution).
[id = dm26-1, sc = α_4, dim = `LeedsPOI`, cf = `inform`]
$r^{\mathcal{L}}_{\text{fs20}} = \alpha_4 = \texttt{StudentUnion} \sqsubseteq \texttt{isContainedWithin.University\_Institution}$
$r^{\mathcal{C}}_{\text{fs20}} = \alpha_4 \in \mathbf{U}$
[id = dm26-2, dim = `OntologyAuthoring`, fundep = `declines`(dm25-1), cf = `declineOffer`]

fs27 Perico: This sentence makes concept `Student Union` unsatisfiable!
[id = dm27-1, dim = `autoFeedback`, feedep = fs26, cf = `autoPositive`]
[id = dm27-2, sc = $r^{\mathcal{C}}_{\text{fs20}}$, dim = `OntologyAuthoring`,
fundep = `unsatisfiableConceptIn`(dm26-1), cf = `inform`]

fs28 Perico: This means that nothing can be a `Student Union` anymore.
[id = dm28-1, dim = `OntologyAuthoring`, fundep = `rephrases`(dm27-2), cf = `inform`]

fs29 Perico: **Advice** You should not add an unsatisfiable concept to an ontology because this concept becomes practically unusable.
[id = $\texttt{dm29} - \texttt{1}$, dim = `OntologyAuthoring`, fundep = `discourageAddInput`(dm27-2),
cf = `directProceduralInstruction`]

fs30 Perico: This is especially true if you make a concept unsatisfiable and that concept was defined by somebody else, as you are probably not using the concept in the way it was intended.

fs31 Perico: Check the list of contradicting sentences:

- *Organisation and POI are mutually exclusive.*
- *Every Student Union is contained within a University (Institution).*
- *Every University (Institution) is a kind of Organisation.*
- *The relationship contains must have subject POI*

- *The relationship is contained within is the inverse of contains.*

[id = dm31-1, dim = `autoFeedback`, feedep = fs26, cf = `autoPositive`]
[id = `dm31 − 2`, sc = $\mathcal{J}$(`StudentUnion` $\sqsubseteq \bot, \mathcal{O} \cup \{\alpha_4\}$),
dim = `OntologyAuthoring`, fundep = `inspectJustificationFor`(`dm27 − 2`),
cf = `directProceduralInstruction`]
[id = `dm31 − 3`, dim = `OntologyAuthoring`, fundep = `informsOneJustificationFor`(dm27-2), cf = `inform`]

fs32 Perico: *enables OK-button*

[id = `dm32 − 1`, dim = `OntologyAuthoring`, fundep = `offersAcceptOfReading`(`dm26 − 1`), cf = `Offer`]

The dialogue state at the end of the interaction is:

$$\mathcal{S}_{32} = \begin{bmatrix} \mathcal{O}_{\text{Reference}} & = & \mathcal{O}_{\text{LeedsPOI}} + \alpha_1 + \alpha_2 \\ \text{Agenda} & = & [??] \\ \text{History} & = & [fs1 \ldots fs32] \end{bmatrix} \tag{5.6}$$

In the next section we analyse which extensions to Perico are required to describe the presented interaction and we will infer which dialogue strategies are required to execute such a dialogue.

5.4.2 Perico Extensions and Dialogue Formalisation

Based on the annotated dialogue, we now discuss which extensions to Perico were required to describe the discourse structure of the interaction. We also analyse which interaction strategy is used by Entendre. We formalise this dialogue system, which we will call BOADiS(Basic Ontology Authoring Dialogue System).

Dialogue Components for BOADiS

BOADiS supports interactions between the system and a single external dialogue participant (an ontology author). BOADiS requires the following variants of the Perico general components:

- **Textual input recogniser**: allows the ontology author to enter textual inputs and click on two GUI-buttons (labelled "OK" and "Cancel") to communicate with the dialogue system. It also allows the user to select an existing CNL sentence and click on a "Delete" button.

- **Template-based output generator**: allows the generation of pre-defined output messages based on templates. In particular, it uses the Entendre templates for axiom category feedback.
- **Textual output presenter**: shows the textual output to the ontology author.

Dimensions

The annotated dialogue presented above contains dialogue moves with three dimensions: (i) `autoFeedback` (dialogue moves that provide information about how ontology author's inputs are interpreted by ROO), (ii) `OntologyAuthoring` and (iii) `LeedsPOI`. The last two dimensions are domain-specific [1]. Since BOADiS is based on the feedback presented in Section 4.5, these dimensions follow directly from the goals of the Entendre feedback: (i) informing the ontology author about how ROO is interpreting the input and (ii) tutoring (instructing) ontology authors about ontology authoring concepts and in particular ontology defects.

Dialogue Moves

The example dialogue shows that the ontology author is limited to `inform` and `acceptOffer` dialogue moves. Although, not shown in the example interaction, the author also can click on the *Cancel*-button, which translates to an explicit `declineOffer` move. Note however, that an `inform` move following an `offer` is also interpreted as a `declineOffer` as shown in **fs26**.

The dialogue moves produced by Perico are `autoPositive`, `inform`, `offer` and `directProceduralInstruction`. The first three come directly from the ISO standard of dialogue moves. The last move is a specialised version of the `instruct` move from the ISO standard that we reuse from a schema proposed in [104]. Direct procedural instructions occur when a tutor (in our case Perico) instructs a learner about what steps the learner should take next.

Although not shown in the example interaction, the dialogue system could also produce `autoNegative` moves, when an input has not been understood.

[1] I.e. they extend the `task` dimension as defined by the ISO standard

Dialogue Plans and Functional Dependencies

The annotated dialogue contains several typed functional dependencies that specify how dialogue moves relate to each other. BOADiS uses most of the basic information-providing dialogue plans provided by Perico(see Section 5.3.3), which result in specific functional dependencies to other moves in the dialogue. However, BOADiS requires new dialogue plans for producing `directProceduralInstruction` moves:

InstructAddInput instructs another participant to add an input to the agreed ontology;

DiscourageAddInput instructs another participant **not** to add an input to the agreed ontology;

InstructAddMoreRelated instructs another participant to add more inputs that are related to some previous move;

InspectRelevantImplications instructs another participant to inspect a list of new relevant implications for a previous move;

InspectJustificationFor instructs another participant to inspect a list of existing assertions (assumed to be shared) that logically imply a statement made in a previous move;

CheckAgreementRelevantImplications instructs another participant to check whether there is agreement with a list of new relevant implications. Note that this is a more specific instruction than `inspectRelevantImplications`, which does not explicitly state why the list should be inspected;

These functional dependencies can be produced by adding dialogue plans such as:

Dialogue Plan: InstructsAddInput

Params : dm_i : DialogueMove
Precondition : $\mathrm{dm}_i \in$ History
Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `directProceduralInstruction`, dim = `OntAuth`,
  - fundep = `instructAddInput`(dm_i),
  - prov = `InstructsAddInput`(dm_i)]

Subplans : $\emptyset$

Dialogue plans for generating the other direct procedural instructions in BOADiS are similarly straightforward.

Finally BOADiS also adds a dialogue move for producing the `offer` dialogue move:

OffersAcceptOfReading offers to agree on the reading of some previous move;

Dialogue Plan: InstructsAddInput

Params : dm_i : DialogueMove
Precondition : $\mathrm{dm}_i \in$ History
Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `Offer`, dim = `OntAuth`,
  - fundep = `offersAcceptOfReading`(dm_i),
  - prov = `OffersAcceptOfReading`(dm_i)]

Subplans : $\emptyset$

Dialogue Strategy

Based on the presented discourse annotations of the Entendre feedback, we can specify dialogue strategies. We distinguish between the *domain-task plan* – capturing knowledge about points of interest in Leeds – and the *grounding strategy*[1] – providing feedback about the system's processing of inputs as well as trying to avoid ontology defects and making the author aware about such defects.

[1] We introduced these types of dialogue plans on page 164.

There is **no domain-task plan** in the analysed dialogue. The ontology author must take the initiative to provide the system with inputs containing her knowledge about the domain.[1]

Regarding the **grounding strategy** in the dialogue, we consider two separate substrategies:

Input Feedback The annotated dialogue shows that the system participant uses a feedback strategy (for unambiguously understood inputs) that is fairly uniform across all the axiom categories:

1. the system `inform`s the user about how the input relates to the agreed knowledge (generally through the `classifiesAxIn` functional dependency, but sometimes a more specific dependency is used such as `informsUnsatisfiableConcept`).
2. the system explains the meaning of the reading classification; this is done either by rephrasing the classification (i.e. the `rephrase` functional dependency) or by providing some relevant inferences that follow from the classification.
3. the system provides instruction as to what to do next (e.g. uses one or more of the applicable `directProceduralInstruction` moves).
4. if there are relevant inferences or justifications (axiom categories **N+**, **R**, **U** and **I**), the system provides these using the available Perico dialogue plans.
5. Finally, BOADiS offers the ontology author the option to accept the reading or not.

This strategy is summarised in Figure 5.8.

Knowledge Grounding BOADiS uses a cautious grounding strategy where no input is automatically accepted; instead, all inputs trigger a grounding plan

[1]Note that making the author aware about ontology defects (and terminology) can also be considered a second domain-task. However, in the given dialogue this topic is not discussed on its own right, but is rather discussed as part of the feedback to some author input. Therefore we consider this topic as being part of the grounding strategy and not as a domain-task plan.

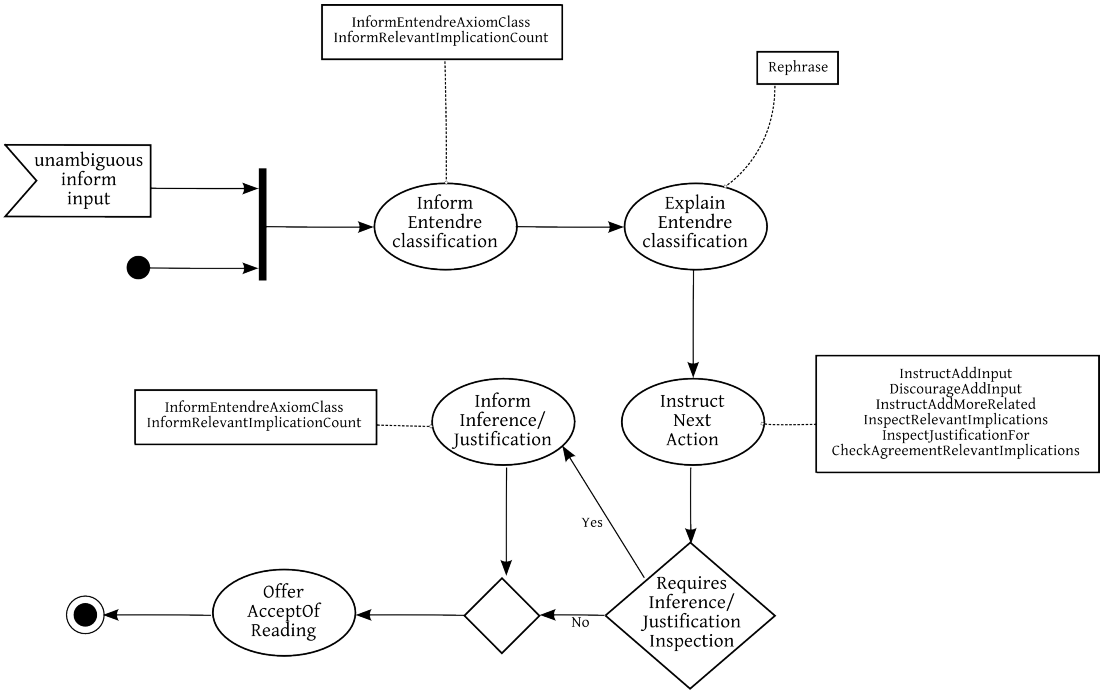

Figure 5.8: UML activity diagram depicting the uniform feedback strategy in BOADiS for inputs that have an unambiguous syntactical reading. This strategy is applied to provide feedback for axioms in 5 main Entendre axiom categories. The diagram also shows the main dialogue plans that are used at each point in the strategy.

that provides the Entendre feedback about the input as described above. The ontology author is required to explicitly accept an `offer` in order to ground the input. Note that this grounding strategy is unsafe since it allows users to accept sentences which introduce ontological defects (axiom categories **R**, **U** and **I**). Authors that accept an axiom in the **I**-category will subsequently not be able to get meaningful feedback.

The formalisation of the *update rules* and *update strategy* follows directly from the above and is summarised in Figure 5.9. The strategy for providing the input feedback is summarised in Figure 5.8. The knowledge grounding plan is defined as follows:

Dialogue Plan: GroundAfterOfferAccept

Params : fs : FunctionalSegment

Precondition :
- fs contains a dialogue move dm_1
- dm_1 matches [dim = `OntologyAuthoring`, cf = `Offer`, fundep = `accepts`(dm_2)]
- dm_2 matches [fundep = `offersAcceptOfReading`(dm_3)]
- dm_3 has an information providing communicative function
- dm_3 has semantic content α

Effect : $\mathcal{O}_{\text{Reference}} := \mathcal{O}_{\text{Reference}} + \alpha$

Subplans : $\emptyset$

5.4.3 Summary

We used Perico to describe and formalise an existing ontology authoring interaction, the provision of Entendre semantic feedback. In doing so, we validated the generality of the Perico framework and showed that it can be easily extended to simulate existing ontology authoring interactions.

The formalisation of the interaction shows clear dialogue strategies used by the system to provide feedback and to ground knowledge. We note that both of these strategies were not explicitly stated during the design and development of Entendre. This validates the approach of using Perico to describe existing ontology authoring interactions.

In the next section we further validate Perico by adapting BOADiS to improve support for novice ontology authors.

5.5 Extending Perico to Support Novice Ontology Authors

In the previous section we validated Perico by implementing a basic ontology authoring dialogue system, BOADiS. In this section we further validate Perico by using it to guide the improvement of an existing ontology authoring interaction. We aim to improve the support that BOADiS provides to novice ontology authors by following the following steps:

Figure 5.9: Dialogue task tree of BOADiS. Note the lack of a domain-task plan and the focus on providing feedback.

- we analyse known issues of the ontology authoring interaction based on the discourse structure and interaction strategy of the initial system,
- we design and formalise alternative interaction strategies to address key issues, resulting in an updated dialogue system.

5.5.1 Explaining Key Issues in BOADiS

During our evaluations of ROO and Entendre, we saw that the used interactions, while enabling the involvement of novice ontology authors, still showed some issues. Two key issues affecting novice ontology authors were:

- novice authors find some feedback overwhelming and confusing,
- novice authors are uncertain about what to do after receiving feedback.

Since BOADiS provides a formal description of the Entendre interaction, we analyse some characteristics of BOADiS and discuss how the dialogue moves and strategies used in BOADiSrelate to the identified key issues.

Monolithic Feedback : After an ontology author's input, Perico's response is a series of moves containing feedback (about axiom classification, rephrasing of that classification, instruction on what to do next, etc.). Perico always presents the full sequence of feedback as one monolithic block that the ontology author cannot interrupt. Our findings in Section 4.6 indicate that such *monolithic feedback can be overwhelming* to novice authors. This feedback strategy is closely related to the moves the ontology author can make; in BOADiS, the author can only make `inform` and `acceptOffer` moves. Thus, since the ontology author cannot request justifications, rephrasings or other moves from the system, the system has to anticipate these information-seeking moves and provide the answers by default. This has several disadvantages:

1. parts of the feedback are repetitive: an author may ask for advice on what to do the first time he enters a **U**-axiom, but may not do so after that. However, the system will continue to repeat the same advice.

2. the system participant cannot anticipate all the questions the ontology author may have. On the one hand, the system may anticipate a question that ontology authors will never ask (e.g. explain what it means that "an input is novel"? between **fs11** and **fs12** in the annotated interaction in page 181). On the other hand, allowing ontology authors to ask questions, even if the system cannot answer the question, provides data about what kind of information ontology authors are interested in.

3. some of the information provided by the system is vague or introduces conditional assumptions. Some of the instructions for next actions were vague and unclear. For example, for axiom category **N**, the suggestion is to "add more sentences if the author is expecting inferences". It is not clear which type of sentences the author should add next. It is also not clear whether the author understands what it means to "expect inferences" from the input at hand. The underlying reason for such feedback is that the system is trying to anticipate questions and has to guess conditions that may apply to the current situation. This introduces uncertainty and clutter to the feedback and may be a major contributing factor to all three key issues.

No System Initiative The current dialogue for ontology authoring simply waits until the ontology author informs the system about a new fact and then provides feedback regarding that input. Furthermore, any new input is treated the same way regardless of the dialogue history. In particular, we note that the BOADiS dialogue plan has the following characteristics:

1. It assumes that the ontology author knows what knowledge should be added to the ontology and that the author knows how to formulate that knowledge. This assumption is clear in the lack of dialogue moves to elicit specific knowledge from the ontology author. This lack of explicit guidance can result in novice authors not knowing what to do next as we found on our evaluations of ROO and Entendre.

2. The dialogue system does not follow-up on procedural instructions. Although the system participant provides procedural instructions to the ontology author, the system does not control whether the authors follow these instructions. This is related to the limited number of available moves to the ontology author. For example, after the `checkAgreementRelevantImplications` instruction, the ontology author may not inspect the implications at all. Or, if the author does inspect the implications and disagrees with some implication, the author cannot express this disagreement or request an explanation for the implication. This may directly contribute to authors not being sure on what to do next, after receiving the feedback.

5.5.2 Strategies for Improving the Interaction

We now propose two high-level strategies for addressing the key issues affecting the BOADiS interaction.

Simplifying Feedback

In order to avoid novice authors finding the feedback overwhelming and confusing, we propose to simplify the feedback strategy depicted in Figure 5.8. We propose the following three substrategies[1] to do this:

First, we propose to *avoid ontology authoring terminology when informing about the Entendre classification*. Thus, instead of introducing the Entendre axiom category (e.g. novel, redundant, inconsistent), and then explaining the category, we now aim to simply explain the integration reading.

Second, *replace direct procedural instructions with custom elicitation moves*. For example, instead of instructing to add more related sentences, the system can be more proactive in suggesting types of sentences to add next. The overall strategy for eliciting new knowledge is described in the next section.

Finally, we simplify the knowledge grounding by *skipping the explicit offer to accept the reading*. Novice authors always followed the suggestions in the

[1]These strategies follow directly from the analysis presented in Section 5.5.1

feedback, thus if an input is not immediately problematic (axiom categories N and N+), Perico can automatically add the axioms to the reference ontology.

Guiding Next Action

In order to make it clearer for novice authors what the next action should be, we propose to add various dialogue plans that give Perico the initiative in the ontology authoring dialogue. The proposed changes are:

Introduce elicitation plans. Instead of waiting until the ontology author enters a new fact about the domain, the dialogue system can be proactive and instruct the ontology author to provide a new fact. Furthermore, elicitation plans can use the dialogue state to request more specific knowledge about the domain. For example:

- Perico could use the Entendre axiom integration analysis to directly ask questions related to a previous input. Thus, instead of using generic direct procedural instruction moves, we propose to ask questions to the authors. For example, for axiom categories **N** and **N+**, the system can *ask the author for expected inferences*. For the **U** case, the system can ask the author for agreement with justifying axioms.
- Perico could use the dialogue history to ask for specific relations between previously mentioned entities. Such elicitation moves can help the author to *discover new ways to formulate knowledge* and to *maintain the focus of the dialogue*.

Introduce plans to inform the authors about what they can say next. Since we are no longer assuming that the ontology author knows what to say next, Perico needs to take the initiative and tell the author what types of dialogue moves are available at any point in the dialogue. Thus, after an elicitation move by Perico, the system can inform the author about how to respond: by, for example, providing a new fact, declining to respond or asking for help.

Introduce plans for providing hints about the CNL. Since novice authors may not know how to formulate specific knowledge, Perico needs to be able to inform authors about suitable CNL sentences. For example, if Perico is eliciting knowledge at the conceptual level, Perico can help the user by giving examples of CNL sentences that translate to T-Box axioms in the ontology language.

We presented two strategies for improving key issues in BOADiS. In the next section, we will use Perico to define an improved dialogue system called BOADiS2. The target for BOADiS2 is to simplify grounding and feedback strategies while adding elicitation moves to push the dialogue (and the ontology authoring) forward.

5.5.3 Improved Ontology Authoring Dialogue

In order to apply the changes to BOADiS discussed above, we introduce a number of dialogue-task plans for eliciting new knowledge, providing hints and providing options for how to reply to elicitation moves. Although the system has added complexity to the domain-tasks, the system's grounding plan is much simpler. The updated task tree is shown in Figure 5.10.[1]

The elicitation plans are governed by the `ElicitationScheduler`, which decides which of the available subplans should be executed next. It has specialised plans such as `ElicitExplicitInference` and `ElicitScopedTBoxAxiom` which can only be executed in specific situations. If none of the specialised plans can be executed, it uses a default elicitation plan, which simply elicits a new fact about the domain.

Dialogue Plan: ElicitationScheduler	
Params :	$\emptyset$
Precondition :	$\emptyset$
Effect :	push subplans into Agenda
Subplans :	$DefaultElicitation, ElicitExpectedInference, ElicitScopedTBoxAxiom$

[1] Compare to the original task tree from Figure 5.9.

Figure 5.10: Dialogue task tree of BOADiS2; it gives an overview of the main domain-task plan, its sub-plans and the dialogue moves used to achieve the goals of the dialogue.

Dialogue Plan: ElicitNewAxiom

Params : $\emptyset$
Precondition : $\emptyset$
Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `Instruction`, dim = `OntAuth`, prov = `ElicitNewAxiom`]

Subplans : $\emptyset$

Dialogue Plan: ElicitExpectedInference

Params : $\emptyset$
Precondition :
- latest input functional segment in History has dialogue move dm_i
- dm_i has integration analysis **N** or **N+**

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `Request`, dim = `OntAuth`, fundep = `requestExpectedInferenceFrom`(dm_i), prov = `ElicitExpectedInference`]

Subplans : $\emptyset$

Dialogue Plan: ElicitScopedTBoxAxiom

Params : dm_i : DialogueMove
Precondition :
- dm_i matches [fundep = `isExpectedInferenceFrom`(dm_2)]

Effect :
- History := History + fs,
- fs `contains` dialogue move dm_o,
- dm_o `matches` [sender = `me`, cf = `Instruct`, dim = `OntAuth`, fundep = `instructsElicitTBoxAxiomFor`(dm_i), prov = `ElicitScopedTBoxAxiom`]
- The scope σ is the signature of the axioms in dm_i and dm_2

Subplans : $\emptyset$

The `HintProvider` plan only is scheduled when the author has produced a dialogue move with communicative function `RequestHint`.[1] Subplans of the Hint Provider check whether a special type of hint is required. For example, `HintForTBoxRbtInScope` checks whether the hint request comes after an `ElicitScopeTBoxAxiom`; if this is the case, the hint provided will only suggest Rabbit sentences that translate into T-Box axioms in OWL. If none of the subplans detects the need for a special type of hint, the `HintForRbtInform` provides a default help for writing Rabbit sentences.

[1] This is a custom communicative function that inherits from the standard `Request`.

The `AnswerOptionProvider` looks whether the system has information about the possible answers that an author can return and provides that information. In our case, the `InformElicitationAnswerOptions` plan can be executed after any of the subplans of the ElicitationScheduler have executed and provides options for telling a new fact, asking for help or skipping the current elicitation plan.

5.5.4 Conclusion

In this section we showed that Perico can be used to guide and formalise changes to existing ontology authoring interactions to provide better support to novice ontology authors. We defined an ontology authoring dialogue system, BOADiS2, with improved strategies aimed to minimise the key issues described in Section 5.4.2. This showed that the Perico framework is flexible and easy to extend. To further validate Perico we next show an implementation of the framework and BOADiS2.

5.6 Implementation

As part of this PhD we implemented the various components of Perico in Scala. We have created an API that dialogue engineers can use to define task trees that can be executed by Perico.

The implemented input analyser integrates with the Entendre implementation described in the previous chapter and is able to generate discourse readings for inform moves. The implementation provides hooks for dialogue engineers to define custom discourse readings based on the current dialogue state. These hooks are used for example to correctly analyse help requests by the user, which are not valid CNL statements, but which are valid moves in the BOADiS2 dialogue.

The Perico implementation also includes a web-based interface for interacting with Perico-based dialogue systems. It provides a chat-like interface where authors can see the history of the dialogue and enter new inputs. We have chosen to implement a dialogue interface that does not extend the existing ROO interface in order to be able to study the dialogue interaction in isolation, without the interference of other GUI components. As we mentioned in a previous chapter, we want to work on a web-based interface for ROO; this interface can serve as

a starting point for such an interface. A screenshot of the interface is shown in Figure 5.11.

5.6.1 Interaction Example

This section shows a representative dialogue between Perico and a simulated ontology author. The dialogue was created by using the implemented Perico dialogue agent running BOADiS2. The example dialogue is using the Leeds Points of Interest ontology that we used to evaluated the Entendre feedback. We assume that the author has browsed through the ontology (and thus is aware of some of the entities in the ontology) and has seen some example Rabbit sentences. We further assume that the author has received the task to enter an initial fact (`Bus Line 95 has end point Bodington Hall`), and then follow the dialogue.

fs1 Perico: Tell me a new fact to add to the ontology.

fs2 Perico: You can do one of the following:

- Enter a new fact or
- Enter *Skip* if you do not want to tell a new fact or
- Enter *Help* to receive a hint.

fs2 OA: Bus Line 95 has end point Bodington Hall.

fs3 Perico: This fact is indeed new.

fs4 Perico: Normally, entering a new fact enables me to make further inferences but I cannot do this in this case.

fs5 Perico: Can you tell me something that I should be able to infer?

fs6 Perico: You can do one of the following:

- Enter a new fact or
- Enter *Skip* if you prefer not to answer this question or
- Enter *Help* to receive a hint.

fs7 OA: Bus Line 95 stops near Bodington Hall.

fs8 Perico: Actually, "Bus Line 95 stops near Bodington Hall" is a new fact.

fs9 Perico: Tell me more general facts about `Bus Route` or `Hall of Residence` or `has end point` or `stops near`.

fs10 Perico: You can do one of the following:

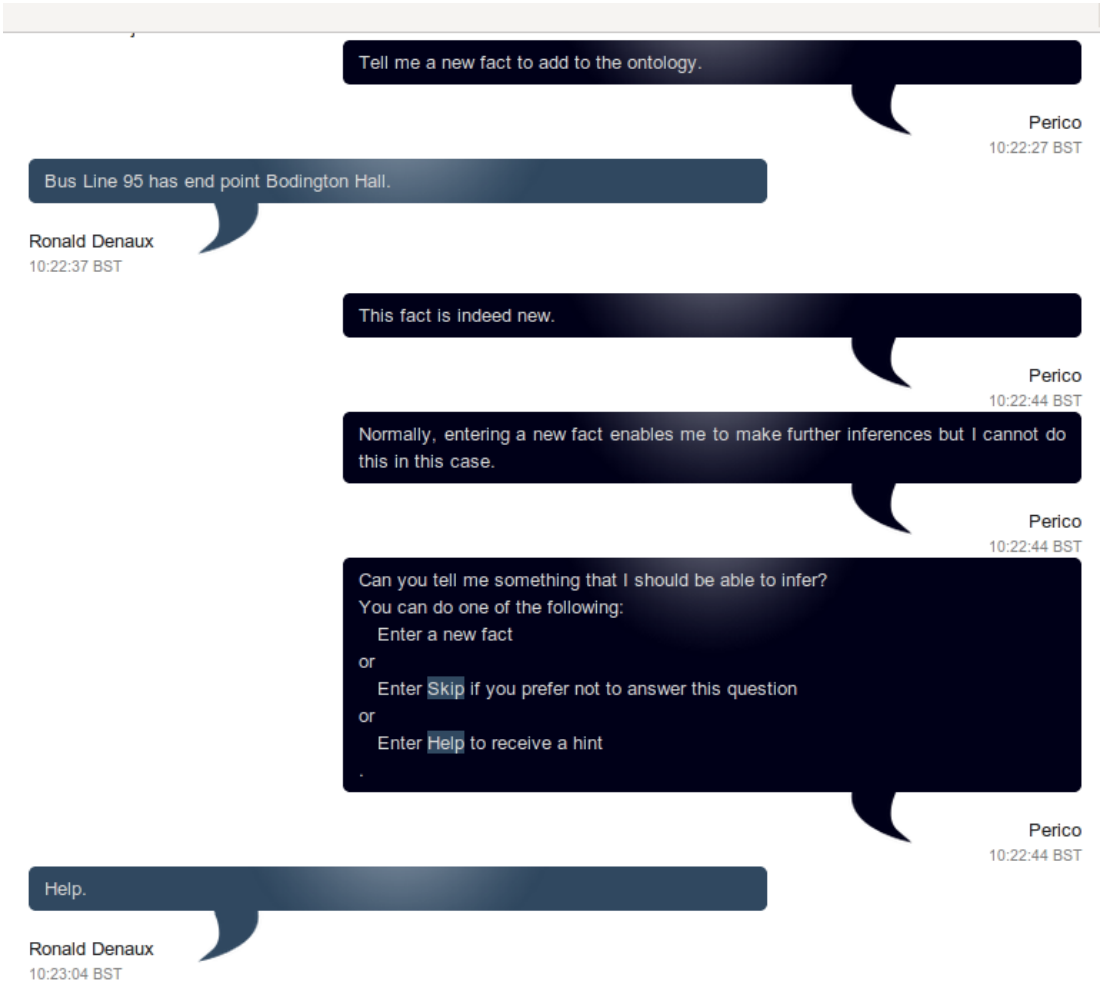

Figure 5.11: Screenshot of the web interface implementation of Perico.

- Enter a new fact or
- Enter *Skip* if you prefer not to answer this question or
- Enter *Help* to receive a hint.

fs11 OA: Help.

fs12 Perico: General facts relate concepts to each other. For example:

- Every *Car* is a kind of *Vehicle*.
- Father and Mother are mutually exclusive.
- Every School has a part a Building that has purpose Education.
- Every car has at least three Wheels.

General facts can also define relationships. For example:

- The relationship *is directly connected to* is a special type of the relationship *is connected to*.
- The relationship *contains* is the inverse of *is contained by*.
- For the relationship *eats*, nothing *eats* itself.

fs13 OA: Help.

fs14 Perico: Facts are simple sentences such as:

- Every Car is a kind of Vehicle.
- Car is a concept.
- Every car has at least three Wheels.

See for more information the documentation on Rabbit.[1]

fs15 OA: Every Student Hall has purpose Accommodation.

fs16 Perico: This general fact is new.

fs17 Perico: I can now infer that *Bodington Hall has purpose* a *Accommodation*.

fs18 Perico: Tell me more specific facts about *Bodington Hall*.

fs19 Perico: You can do one of the following:

- Enter a new fact or
- Enter *Skip* if you prefer not to answer this question or
- Enter *Help* to receive a hint.

fs20 OA: Skip.

fs21 Perico: Tell me a general fact about `stops near`.

fs22 Perico: You can do one of the following:

[1] Links to on-line documentation about the Rabbit language.

- Enter a new fact or
- Enter *Skip* if you prefer not to answer this question or
- Enter *Help* to receive a hint.

fs23 OA: Help.

fs24 Perico: I know that:

- *has end point* is a relationship.
- Every *Transport Route has end point* exactly 1 POI.

You can use general facts to further define the meaning of *has end point*. For example you could enter facts similar to:

- The relationship *has end point* is a special type of the relationship *has purpose*.
- The relationship *has end point* is the inverse of *stops near*.
- For the relationship *has end point*, nothing *has end point* itself.

fs25 OA: The relationship has end point is a special type of the relationship stops at.

fs26 Perico: This general fact is new.

Discussion

The example dialogue shows how the BOADiS2 dialogue plan provides support to novice authors in the following ways:

Various elicitation moves guide the user by narrowing down the type of knowledge that the author should enter. The dialogue starts with a default elicitation move in `fs1`. In `fs5`, the `ElicitExpectedInference` plan is executed after after the **N** axiom from `fs2`. In `fs9` and `fs21`, the `ElicitScopedTBoxAxiom` plan is executed after an expected inference is detected. Finally in `fs18`, an `ElicitScopedABoxAxiom` is executed.

The dialogue show various instances where the `InformElicitationAnswerOptions` is executed (`fs2`, `fs6`, `fs10`, `fs19` and `fs22`). This plan is always executed right after an elicitation move.

The author makes use of the available options by providing new facts, asking for help (`fs11`, `fs13` and `fs23`) or skipping an elicitation move (`fs20`).

The system provides hints to the author in `fs12`, where the `HintForTBoxRbtInScope` plan exploits the knowledge about the expected types of Rabbit sentences to only suggest T-Box sentences. In `fs14`, since the author is still asking for help, the

system falls back to the default plan for helping users formulate facts. In `fs24`, the system again provides help based on a `HintForTBoxRbtInScope`; however, note that since the scope now is a single relationship (`stopsNear`), the hint can be more focused on relationships.

Finally, the new simplified `IntegrationFeedbackProvider` results in functional segments `fs3-4`, where feedback is given about the **N** axiom from `fs2`. In `fs8`, feedback is given for the **N**-axiom from `fs7`; Perico provides an even shorter feedback in this case since it knows that `fs7` is an expected inference from `fs2`. Functional segments `fs16-17` provide feedback about an **N**+-axiom from `fs15`.

This section presented an overview of how the Perico framework was implemented, as well as an example interaction based on the BOADiS2 interaction and a discussion of how the dialogue is executed.

5.7 Conclusion

The chapter presented a framework for describing and executing ontology authoring interactions in terms of dialogue moves. This framework, called Perico, enables the use of discourse analysis (on top of syntactic and semantic analyses) to formalise ontology authoring interactions. Perico reuses standards from the area of dialogue systems and integrates them with ontology engineering concepts.

This chapter provided validation of the Perico framework by (i) using it to formalise the Entendre semantic feedback interaction; (ii) using it to design and formalise an improved interaction for ontology authoring and (iii) implementing the framework and interactions. These validations showed that the framework is flexible and can be extended to formalise existing ontology authoring interactions.

The validations showed that Perico is a promising approach for improved adaptability in ontology authoring support. Compared to existing ontology authoring systems, the use of a dialogue framework for describing ontology authoring interactions allows for the definition of fine-grained support plans. In particular, the dialogue history subcomponent of the dialogue state can be used as a trigger to provide fine-grained feedback or to start a specific elicitation plan. A main advantage of the proposed framework is thus, that it facilitates the specification, implementation and study of adaptive responses from the system.

Chapter 6

Conclusion and Future Research Directions

This thesis presented research on providing tool support to make ontology authoring more intuitive. This research is relevant to enabling domain experts to actively contribute to the process of ontology authoring. We have focused on using tool support that enables domain experts to formulate their knowledge using expressive ontology languages, enabling them to contribute to heavyweight ontologies. The research followed an explorative and iterative approach to developing principled tool support approaches. Starting with syntactic support using a Controlled Natural Language, we explored providing more holistic tool support by including support for ontology engineering methodologies. We also added support for becoming aware of logical consequences of new inputs and providing understandable tool support. Finally, we defined a framework to enable the analysis of ontology authoring interactions using dialogue systems. The main contribution of this work is the definition and development of intelligent tool support for novice ontology authors.

This chapter provides a summary of the outcomes and research contributions of this research. We also provide a discussion of immediate and long-term research directions based on this work.

6.1 Summary

This research work has explored the area of intelligent tool support for enabling the direct involvement of domain experts in the process of building ontologies. We have iteratively formalised, implemented and evaluated this tool support in order to make ontology authoring more accessible. We started from the hypothesis that holistic intelligent tool support can enable the active involvement of domain experts in ontology authoring. This hypothesis was shared by existing work which suggested the application of an ontology authoring methodology that uses a Controlled Natural Languages in order to involve domain experts in ontology authoring. This resulted in our first research question:

- *How can CNL-based tool support be integrated with support for following an ontology authoring methodology and how does such combined tool support affect ontology authoring by domain experts?*

We started to addressed this question in Chapter 3 by implementing ROO, an ontology authoring tool that was designed to cater for the needs of domain experts with little or no ontology engineering experience. We showed how to use NLP, parsing and GUI design techniques to provide appropriate tool support. This chapter presented tool support for formulating knowledge in a way that is understood by the CNL parser and tool support for adhering to the ontology authoring methodology.

The second way we addressed this question was by evaluating ROO. We performed a comparative evaluation with another CNL-based ontology authoring tool as a baseline to focus on how the holistic tool support affected domain experts. The evaluation provided empirical evidence that the implemented holistic approach enabled domain experts to build ontologies from scratch in a short period of time. Furthermore, we showed that the holistic tool support provided benefits in terms of ontology quality, ease of use and awareness about the ontology authoring process. The evaluation study and other practical experiences with ROO helped us to identify aspects that could result in improved tool support for domain experts. One of these aspects was the need for improved disambiguation

handling and feedback. Another aspect was the need to make domain engineers more aware about the logical consequences of their ontology authoring actions.

Based on these aspects we formulated a second research question:

- *How can the syntactic analysis required for understanding textual inputs (such as CNL) be formalised and integrated with semantic analysis of the inputs in order to provide understandable feedback to domain experts?*

We addressed this question in Chapter 4 by defining a framework called Entendre for facilitating the systematic analysis of ontology author's textual inputs. The first part of this framework formally defines the main syntactic analyses of an input that can be performed by an ontology authoring system and the main results of such analyses. The framework provides a categorisation of lexical and syntactic analysis results which provide a guide for performing disambiguation and for providing effective feedback.

The second part of the Entendre framework addressed the research question by defining an axiom-integration analysis strategy that categorises input axioms based on the logical effects it has on the ontology being built. This categorisation provides an overview of ontology defects that can be introduced and can be used to provide targeted feedback about semantic issues to ontology authors.

The final way we addressed the research question was by implementing the Entendre framework and evaluating the semantic feedback. We performed an evaluation with 5 novice ontology authors and 5 experienced knowledge engineers to find out what ontology authors thought of the interactive feedback, whether they understood it and whether they would know what to do after receiving the feedback. We found that both novice and experienced ontology authors liked the feedback and though it was useful and informative. The evaluation reinforced earlier results that indicated that effective interaction between the ontology authoring system and domain experts is needed to produce better feedback and to guide ontology authors actions.

This resulted in our third research question:

- *How can dialogue systems be used to formalise and improve ontology authoring interactions for better support of domain experts?*

This question was addressed in Chapter 5, where we reviewed the literature on dialogue systems for ontology authoring. We then defined Perico, a dialogue framework for ontology authoring that can be used for describing and executing ontology authoring interactions in terms of dialogue moves. The presented framework adds a layer of discourse analysis (on top of Entendre) to formalise ontology authoring interactions. We reused standards from the area of dialogue systems and integrate them with ontology engineering concepts in order to define the Perico framework. We validated Perico by using it to (i) formalise an existing ontology authoring interaction and (ii) adapt and formalise an existing ontology authoring interaction to provide better support to novice ontology authors. A final validation of the framework consisted of the implementation of the framework itself (reusing the Entendreframework) and of the formalised dialogue systems.

This thesis resulted in the following software outputs [1]:

- a library with a set of tools for the Rabbit controlled natural language including a parser, a Rabbit to OWL converter, an OWL to Rabbit renderer and GUI components for editing ontologies using Rabbit.
- an ontology authoring "Guide-dog" library, which analyses an ontology to suggest actions to perform based on the Kanga methodology and improve the quality of ontologies.
- the ROO ontology authoring tool which is based on Protégé 4 and integrates the Rabbit and Guide-dog libraries.
- the Entendre library for analysing textual ontology authoring inputs, which understands inputs in both Rabbit and Manchester Syntax. This library also performs input-axiom analysis in order to provide textual feedback about the logical consequences of adding an input to an existing ontology.
- the Perico dialogue toolkit, which provides an API for defining dialogue systems where the shared knowledge of the participants results in an ontology. The toolkit is integrated with the Entendre library to enable dialogue interactions based on Rabbit.

[1] All of these libraries and tools are open source. See Appendix A for the relevant links.

6.2 Contributions

The work presented in this thesis resulted in a number of original contributions, which we discuss in this section.

Intelligent Tool Support for Involving Domain Experts in Ontology Authoring

Various CNL-based tools have been proposed and developed to make ontology authoring more accessible to domain experts [33, 47, 80, 122, 125]. In this line, we have contributed by developing tool support for the construction of rich ontologies that is suited to people without knowledge engineering skills and who have no access to a knowledge engineer. Novel aspects of this tool support are that:

- it is based on an ontology authoring methodology. To the best of our knowledge none of the existing CNL-based tools provide guidance to make up for the lack of experience in ontology authoring of domain experts.
- it is based on an expressive CNL, while some existing CNL-based tools only provide a small subset in order to define lightweight ontologies [47].
- it provides easy to understand assistance for correcting syntactic mistakes and to perform ontology authoring steps; this is an improvement on existing CNL-based ontology authoring tools [80] which provide feedback that is harder to understand by domain experts.

Although the readability and writeability of various CNLs has been studied in order to improve the language design [41, 96], such studies are performed without tool support. Other evaluations of CNL-based tools have used a CNL with limited expressivity [47] and simple tasks. In this line, we contributed a experimental study that closely matches a realistic scenario for building an ontology from scratch.

Currently, the ROO distribution provided at sourceforge has more than 1000 downloads[1]. The tool is being used in a EU-project[2] and has been used by several

[1]The link to download ROO is available on the project web site `http://www.comp.leeds.ac.uk/confluence/`

[2]`http://imreal-project.eu/`

staff members and students at the University of Leeds to build ontologies.

A Holistic Framework for Providing Feedback about Ontology Assertions

Most existing CNL-based and traditional ontology authoring tools only provide feedback about syntactic errors. They may impose severe restrictions on the input language syntax in order to avoid dealing with ambiguity [47, 80]. [1] We contribute to this area by defining a novel formal description of the lexical and syntactic analysis of textual ontology authoring inputs. This formal description allows for the controlled integration of various lexical and syntactic analysis strategies.

Various ontology authoring tools have been extended with tools for helping knowledge engineers debug, explore and understand the logical structure of ontologies [9, 69, 82, 99]. However, most of these tools are only available to knowledge engineers and may require a firm understanding of the logical aspects of ontology languages. Our contribution in this area is the definition of an algorithm for performing axiom-integration analysis that combines existing reasoning services and result in understandable feedback. To the best of our knowledge we are the first to provide such interactive feedback to domain experts.

Another novelty of our work is that it defines and implements a holistic framework that is able to analyse a textual input both syntactically and semantically in order to provide relevant feedback.

Finally, we contributed a novel evaluation of semantic feedback provided to both novice and expert ontology authors.

A Dialogue Framework for Ontology Authoring

Dialogue-like interfaces have been used in the area of knowledge acquisition to gather simple common-sense facts [22] or extend existing knowledge bases [25, 140]. Although such dialogues are applicable to ontology authoring, no generally available system exists for describing and executing ontology authoring interactions. In this area we contribute a generic model for describing ontology author-

[1] Or allow ambiguity but do not deal correctly with it such as the Manchester Syntax in Protégé 4.

ing interactions based on well-established standards for modelling dialogues. The framework serves as a dialogue toolkit which can be applied to different ontology authoring interactions and has been validated with existing discourse patterns.

6.3 Future Work

In the previous sections we summarised the main achievements and contributions of our work. In this section we present immediate applications and extensions of our work as well as longer-term research directions that can build on our work.

6.3.1 Applications and Research Directions with Current Outputs

We presented evaluation studies of both ROO and the Entendre semantic feedback with domain experts. However, both studies were performed in controlled settings and only allowed for relatively short interactions between domain experts and ROO. We have also gathered some anecdotal evidence based on experiences from colleagues and research partners in using the tool; in these cases ROO has been used to build ontologies, but no data has been gathered.

An immediate research direction is to use ROO as a tool for teaching ontology authoring. This will allow us to monitor the interaction of novice ontology authors with the tool in more detail over a longer period of time. Such a practical study could allow us to gain insights into the development of knowledge engineering skills.

Another practical study with ROO and Perico is to use a dialogue to elicit personal conceptual models about a particular domain. Gathering such personal conceptual models would allow us to investigate whether we can measure and compare different perspectives on a domain. In particular, we are considering the *sustainability* domain, which is a relevant, but complex subject with various aspects that can affect the participants perspectives.

We showed a validation of Perico based on simulated ontology authoring interactions. We are planning to conduct a controlled study with domain experts in order to evaluate the two dialogue systems we have defined (BOADiS and

BOADiS2). The study will evaluate whether the different interaction strategies has an effect on the quality of the resultant ontology.

ROO will be used in a new research project (NETTUN) for modelling decision making knowledge. It will serve as the basis for a tool to capture tacit knowledge and to show the resulting ontology to domain experts for inspection.

Finally, we are also planning to adapt Perico to define dialogues for validating and expanding user models [1]. In this case we will treat the user model as our reference ontology and define interactions strategies to validate uncertain facts about the user. We will use existing semantic web data and ontologies as background knowledge for the dialogue interactions.

6.3.2 Long-term Research Directions

The combination of ROO, Entendre, Perico and the web enable large-scale involvement of domain experts in ontology authoring. This opens up research into crowdsourcing of ontology authoring using CNL. Due to the large number of collaborators, a framework will be necessary for comparing different perspectives about the same domain.

In the future we want to *use Entendre to further improve on the Rabbit parsing* to report and analyse ambiguities, attempt different strategies for not understood and partially understood inputs. The improved architecture that Entendre provides will make it easier to enhance the syntactic analysis of ontology authors' inputs. For example, Entendre would be a better place to keep track of and resolve ambiguity. The current implementation of the Rabbit parser works, but makes the default Rabbit parsing quite slow. By moving the ambiguity handling from the parser to Entendre-based extensions, the Rabbit parser could become much more efficient (useful for batch processing), while tools would still be able to provide robust parsing based on Entendre during ontology authoring. Other improvements include exploring how the available semantic analysis can help in syntactic disambiguation or can inform the need for an alternative syntactic analysis strategy.

We would like to extend our current work to devise strategies for helping domain experts reuse existing top ontologies or ontology design patterns when

defining their ontologies. One of the problems in this case in that top ontologies often use very abstract and unintuitive terminology (e.g. endurant, perdurant). A *framework for adapting to the level of abstraction of the domain expert* is required, such a framework could use ontology reasoning to try to reformulate assertions and questions while avoiding overly abstract or overly concrete terminology.

Appendix A

Relevant Links

This appendix provides a list of supporting information related to this PhD.

Rabbit

- The BNF for Rabbit can be found at `https://sourceforge.net/apps/mediawiki/confluence/index.php?title=Rabbit_BNF`
- The Java library defining the Rabbit API is `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence/trunk/rabbitParser`
- The Java implementation of the Rabbit parser based on Gate can be viewed at `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence/trunk/gateRabbitParser`
- The JAPE rules for Rabbit can be viewed at `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence/trunk/gateRabbitParser/src/main/gateapp/jape/`
- The Java implementation of the Rabbit to OWL conversion can be found in `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence/trunk/owlapiConverter`

ROO

- The source code repositofy for ROO can be viewed at `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence`. It contains modules for the main interface, the `guidedog`, protege 4 extensions and `rabbit-gui` components.
- The Drools rules for the guide dog can be viewed at `http://confluence.svn.sourceforge.net/viewvc/confluence/confluence/trunk/guidedog/src/main/rules/uk/co/ordnancesurvey/kanga/guidedog/NextTaskCalculator.drl?revision=1571&view=markup`
- The evaluation materials are avaliable from `http://www.comp.leeds.ac.uk/confluence/study.html`

Entendre

- The source code for Entendre can be found at `http://entendre.git.sourceforge.net/git/gitweb-index.cgi`
- The evaluation materials are avaliable from `http://www.comp.leeds.ac.uk/confluence/Entendre-Study/`

Perico

- Source code can be viewed at `http://perico.git.sourceforge.net/git/gitweb-index.cgi`

References

[1] Abel, F., Dimitrova, V., Hauff, C., Houben, G.J., Thakker, D.: Second Version of Demonstrator of Context and Learner Modelling Services. ImReal Project Deliverable D4.3. Tech. rep., University of Leeds (2012) 212

[2] Aguado, G., Bañón, A., Bateman, J., Bernardos, S., Fernández, M., Gómez-Pérez, A., Nieto, E., Olalla, A., Plaza, R., Sánchez, A.: ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In: Workshop on Applications of Ontologies and Problem Solving Methods, ECAI. vol. 98 (1998) 16

[3] Alani, H., Dupplaw, D., Sheridan, J., Darlington, J., Shadbolt, N., Tullo, C.: Unlocking the potential of public sector information with semantic web technology. The Semantic Web pp. 708–721 (2007), http://www.springerlink.com/index/986L678302785616.pdf 1

[4] Angele, J., Moench, E., Oppermann, H., Staab, S., Wenke, D.: Ontology-based query and answering in chemistry: Ontonova project HALO. In: Proceedings of the 2nd International Semantic Web Conference. pp. 913–928. Springer (2003), http://www.springerlink.com/index/ELA4PEL666UE99E9.pdf 17, 19

[5] Arinze, B.: A natural language front-end for knowledge acquisition. ACM SIGART Bulletin 1989(108), pp. 106–114 (Apr 1989), http://0-dl.acm.org.wam.leeds.ac.uk/citation.cfm?id=63266.63280 149

[6] Austin, J.L.: How to do things with words, vol. 88. Harvard University Press (1975) 151

[7] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003), http://dblp.uni-trier.de/db/conf/dlog/handbook2003.html 84

[8] Babitski, G., Bergweiler, S., Grebner, O., Oberle, D., Paulheim, H., Probst, F.: SoKNOS - Using Semantic Technologies in Disaster Management Software. In: Extended Semantic Web Conference (ESWC). pp. 183–197. Heraklion, Greece (2011) 76

[9] Bail, S., Parsia, B., Sattler, U.: Extracting Finite Sets of Entailments from OWL Ontologies. In: Description Logics (2011) 87, 210

[10] Basten, H.H.J., Vinju, J.: Parse Forest Diagnostics with Dr. Ambiguity. 4th International Conference on Software Language Engineering (Jul 2011) 79

[11] Bauer, J., Sattler, U., Parsia, B.: Explaining by Example: Model Exploration for Ontology Comprehension. Description Logics (2009), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.151.3820 90

[12] Berners-Lee, T., Hendler, J., Lassila, O., Others: The semantic web. Scientific american 284(5), pp. 28–37 (2001) 1

[13] Bernstein, A., Kaufmann, E., Fuchs, N.: Talking to the Semantic Web- A Controlled English Query Interface for Ontologies. In: 14th Workshop on Information Technologies and Systems (2004), http://www.ifi.uzh.ch/pax/uploads/pdf/publication/765/TalkingToTheSemanticWeb_WITS2004.pdf 17

[14] Bird, S., Liberman, M.: A formal framework for linguistic annotation. Speech Communication 33(1-2), pp. 23–60 (2001), http://www.sciencedirect.com/science/article/pii/S0167639300000686 94, 95

[15] Bohus, D., Rudnicky, A.: The RavenClaw dialog management framework: Architecture and systems. Computer Speech & Language 23(3), pp. 332–361 (2009), http://www.sciencedirect.com/science/article/

B6WCW-4TVJ3KG-1/2/d6bfd64173650f150219cf4a43a51a66 152, 153, 157, 161, 162, 164

[16] Bontcheva, K., Wilks, Y.: Automatic report generation from ontologies: the MIAKT approach. Natural Language Processing and Information Systems pp. 1–19 (2004) 16

[17] Boose, J.H.: A knowledge acquisition program for expert systems based on personal construct psychology. International Journal of Man-Machine Studies 23(5), pp. 495–525 (1985) 10

[18] Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. Cognitive science 9(2), pp. 171–216 (1985) 8

[19] Braun, S., Schmidt, A., Walter, A.: Ontology maturing: a collaborative web 2.0 approach to ontology engineering. Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge 273 (2007), http://www.ra.ethz.ch/cdstore/www2007/km.aifb.uni-karlsruhe.de/ws/ckc2007/papers/BraunSchmidtWalterNagypalZacharias.pdf 13

[20] Buchanan, B.G., Shortliffe, E.H.: Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley series in artificial intelligence). Addison-Wesley Longman Publishing Co., Inc. (1984) 7

[21] Bunt, H., Alex, J., Carletta, J., Choe, J.w., Fang, A.C., Hasida, K., Lee, K., Petukhova, V., Popescu-belis, A., Romary, L., Soria, C., Traum, D.: Towards an ISO standard for dialogue act annotation. Proceedings of the Seventh International Conference on Language Resources and Evaluation (2010), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.178.9209 15, 154

[22] Chklovski, T., Gil, Y.: Towards managing knowledge collection from volunteer contributors. In: Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05) (2005) 149, 210

[23] Cimiano, P., Haase, P., Heizmann, J., Mantel, M., Studer, R.: Towards portable natural language interfaces to knowledge bases The case of the ORAKEL system. Data & Knowledge Engineering 65(2), pp. 325–354 (2008), http://www.sciencedirect.com/science/article/B6TYX-4R68N7K-2/2/63cb920499ba55e3ba5ff9217d33e739 17

[24] Clark, P., Chaw, S.Y., Barker, K., Chaudhri, V., Harrison, P., Fan, J., John, B., Porter, B., Spaulding, A., Thompson, J., Others: Capturing and answering questions posed to a knowledge-based system. In: International Conference On Knowledge Capture: Proceedings of the 4 th international conference on Knowledge capture. pp. 63–70 (2007) 19

[25] Clark, P., Hayes, P., Reichherzer, T., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y.: Knowledge entry as the graphical assembly of components. In: Proceedings of the international conference on Knowledge capture - K-CAP 2001. p. 22. ACM Press, New York, New York, USA (Oct 2001), http://dl.acm.org/citation.cfm?id=500737.500745 149, 210

[26] Clark, P., Murray, W., Harrison, P., Thompson, J.: Naturalness vs. Predictability: A Key Debate in Controlled Languages. In: Fuchs, N. (ed.) Controlled Natural Language, Lecture Notes in Computer Science, vol. 5972, pp. 65–81. Springer Berlin / Heidelberg, Marettimo Island, Italy (2010), http://dx.doi.org/10.1007/978-3-642-14418-9_5 19, 80

[27] Clark, P., Porter, B.: KM–the knowledge machine: Users manual. Tech. rep., Technical report, AI Lab, Univ Texas at Austin.(http://www. cs. utexas. edu/users/mfkb/km. html) (1999) 19

[28] Costetchi, E., Ras, E., Latour, T.: Automated Dialogue-Based Ontology Elicitation. Procedia Computer Science 7, pp. 185–186 (Jan 2011), http://dx.doi.org/10.1016/j.procs.2011.09.008 149

[29] Cregan, A., Schwitter, R., Meyer, T.: Sydney OWL Syntaxtowards a Controlled Natural Language Syntax for OWL 1.1. In: Proceedings of the Workshop on OWL: Experiences and Directions (OWLED). vol. 258. CEUR-

WS vol. 258, Innsbruck, Austria (2007), http://www.ghachey.info/site_media/files/papers/2010/12/17/Sydney_OWL_Syntax.pdf 19, 32

[30] Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., Dimitrov, M., Dowman, M., Aswani, N., Roberts, I., Li, Y., Others: Developing Language Processing Components with GATE version 5 (a User Guide). User guide, University of Sheffield (2009), http://gate.ac.uk/sale/tao/tao.pdf 20, 35, 36, 43

[31] Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. 7th Extended Semantic Web Conference, ESWC 2010 pp. 106–120 (2010) 150

[32] Damljanovic, D., Agatonovic, M., Cunningham, H.: FREyA: an Interactive Way of Querying Linked Data using Natural Language. In: The Semantic Web: ESWC 2011 Workshops. pp. 125–138. Springer (2012) 150

[33] Davis, B., Iqbal, A.A.: RoundTrip ontology authoring. In Proceedings of the 7th International Semantic Web Conference, ISWC (2008) 2, 209

[34] Delugach, H.: ISO/IEC 24707 Information technology–Common Logic (CL)–A Framework for a Family of Logic-Based Languages. ISO specification (2007), http://standards.iso.org/ittf/PubliclyAvailableStandards/c039175_ISO_IEC_24707_2007%28E%29.zip 7

[35] Denaux, R., Aroyo, L., Dimitrova, V.: OWL-OLM: Interactive Ontology-based Elicitation of User Models. In: Proceedings of the Workshop on Personalisation in the Semantic Web, User Modelling, PerSWeb05. pp. 34–46. Proceedings of the Workshop on Personalisation in the Semantic Web, User Modelling, PerSWeb05 (2005), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.105.562 161

[36] Denaux, R., Dolbear, C., Hart, G., Dimitrova, V., Cohn, A.G.: Supporting domain experts to construct conceptual ontologies: A holistic approach.

Web Semantics: Science, Services and Agents on the World Wide Web 9(2), pp. 113–127 (Jul 2011), http://dx.doi.org/10.1016/j.websem.2011.02.001 137, 138

[37] Di Eugenio, B., Fossati, D., Haller, S., Yu, D., Glass, M.: Be Brief, And They Shall Learn: Generating Concise Language Feedback for a Computer Tutor. International Journal of Artificial Intelligence in Education 18(4), pp. 317–345 (2008), http://iospress.metapress.com/content/323054631t056845/ 150

[38] Di Eugenio, B., Fossati, D., Ohlsson, S., Cosejo, D.: Towards explaining effective tutorial dialogues. Annual Meeting of the Cognitive Science Society Cognitive Science Society pp. 1430–1435 (2009) 150

[39] Dolbear, C., Hart, G.: Combining spatial and semantic queries into spatial databases. In: 5th International Semantic Web Conference (Posters & Demos) (2006) 1

[40] Dolbear, C., Hart, G., Goodwin, J., Zhou, S., Kovacs, K.: The Rabbit Language: description, syntax and conversion to OWL. Technical Report IRI-0004. Tech. Rep. IRI-0004, Ordnance Survey Research Labs (2007) 2, 28

[41] Engelbrecht, P., Hart, G., Dolbear, C.: Talking rabbit: a user evaluation of sentence production. In: Proceedings of the 2009 conference on Controlled natural language. pp. 56–64. Springer (Jun 2010), http://portal.acm.org/citation.cfm?id=1893475.1893480 22, 31, 63, 69, 209

[42] Euzenat, J., Shvaiko, P.: Ontology matching. Recherche 67, p. 2 (2007) 9

[43] Evermann, J., Fang, J.: Evaluating ontologies: Towards a cognitive measure of quality. Information Systems 35(4), pp. 391–403 (Jun 2010), http://dx.doi.org/10.1016/j.is.2008.09.001 9

[44] Fernandez-Lopez, M., Gomez-Perez, A., Juristo, N.: METHONTOLOGY: from ontological art towards ontological engineering. In: Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering (1997) 9, 11

[45] Fuchs, N.E., Schwitter, R.: Attempto controlled english (ace). The First International Workshop on Controlled Language Applications (CLAW) (1996), http://arxiv.org/pdf/cmp-lg/9603003 18

[46] Funk, A., Davis, B., Tablan, V., Bontcheva, K., Cunningham, H.: Controlled Language IE Components version 2. SEKT Project Deliverable D.2.2.2. Tech. rep., University of Sheffield (2007) 15

[47] Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: Clone: Controlled language for ontology editing. Proceedings of ASWC and ISWC 2007 4825, pp. 142–155 (2007), http://www.springerlink.com/index/51q65803t4wj70v3.pdf 1, 2, 20, 35, 38, 63, 209, 210

[48] Gangemi, A.: Ontology design patterns for semantic web content. In: The Semantic WebISWC 2005. vol. 3729. Springer (2005), http://www.springerlink.com/index/F513071T4477H2W2.pdf 66

[49] Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: Modelling ontology evaluation and validation. Proceedings of the European Semantic Web Conference pp. 140–154 (2006) 54, 58

[50] Gangemi, A., Guarino, N., Masolo, C.: Sweetening ontologies with DOLCE. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web. Springer-Verlag, London, UK (2002), http://www.springerlink.com/index/5p86jk323x0tjktc.pdf 11

[51] Ghidini, C., Rospocher, M., Serafini, L., Kump, B., Pammer, V., Faatz, A., Zinnen, A., Guss, J., Lindstaedt, S.: Collaborative Knowledge Engineering via Semantic MediaWiki. In: Proceedings of the International Conference on Semantic Systems I-SEMANTICS 08. pp. 134–141 (2008) 13

[52] Ginzburg, J., Fernández, R.: Computational Models of Dialogue. In: The Handbook of Computational Linguistics and Natural Language Processing, chap. 16, pp. 429–481. Wiley-Blackwell (2010), http://dx.doi.org/10.1002/9781444324044.ch16 152, 157

[53] Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. of Artificial Intelligence Research 31(1) (2008) 76

[54] Gruber, A., Westenthaler, R., Gahleitner, E.: Supporting domain experts in creating formal knowledge models (ontologies). In: Proceedings of I-KNOW. pp. 252–260 (2006), http://knowminer.at/corpi/iknow-papers/data-2000-2010/pdf/31_SupportingDomainExperts.pdf 10

[55] Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge acquisition 5(2), pp. 199–220 (1993) 7

[56] Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. International journal of human computer studies 43(5), pp. 907–928 (1995) 1

[57] Guarino, N., Welty, C.A.: An overview of OntoClean. Handbook on ontologies pp. 201–220 (2009) 9

[58] Harrison, P., Maxwell, M.: A new implementation of GPSG. In: Proc. 6th Canadian Conf on AI. pp. 78–83 (1986) 19

[59] Hart, G., Dolbear, C., Goodwin, J.: Lege Feliciter: Using structured English to represent a topographic hydrology ontology. In: Proceedings of the 3rd OWL Experiences and Directions Workshop. vol. 258. CEUR-WS vol. 258 (2007) 29

[60] Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a control natural language for authoring ontologies. In: Proceedings of the 5th European Semantic Web Conference. pp. 348–360. Springer (2008) 14, 29, 31

[61] Hartman, J., Spyns, P., Giboin, A., Maynard, D., Cuel, R., Suárez-Figueroa, M., Sure, Y.: Methods for Ontology Evaluation. Tech. rep., Knowledge Web Deliverable, D1.2.3 (2004) 50

[62] Hartmann, J., Sure, Y., Haase, P., Palma, R., Suarez-Figueroa, M.: OMVontology metadata vocabulary. In: Workshop on Ontology Patterns for the Semantic Web at ISWC2005 (2005) 14

[63] Heinroth, T., Denich, D., Schmitt, A.: OwlSpeak - Adaptive Spoken Dialogue within Intelligent Environments. In: 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). pp. 666–671 (Mar 2010), http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5470518 161

[64] Hendler, J.: Why Semantic Web will never work (2011), http://videolectures.net/eswc2011_hendler_work/ 75

[65] Holsapple, C.W., Joshi, K.D.: A collaborative approach to ontology design. Communications of the ACM 45(2), pp. 42–47 (Feb 2002) 76

[66] Hori, C.: Statistical dialog management applied to WFST-based dialog systems. In: 2009 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 4793–4796. IEEE (Apr 2009), http://www.computer.org/portal/web/csdl/doi/10.1109/ICASSP.2009.4960703 153

[67] Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.H.: The manchester OWL syntax. In: OWL: Experiences and Directions. vol. 216. CEUR-WS vol. 216 (2006) 29

[68] Horridge, M., Bail, S., Parsia, B., Sattler, U.: The Cognitive Complexity of OWL Justifications. International Semantic Web Conference (2011) 69

[69] Horridge, M., Parsia, B., Sattler, U.: Justification oriented proofs in OWL. 9th international semantic web conference on The semantic web pp. 354–369 (Nov 2010) 87, 89, 119, 210

[70] Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., Others: The ontology inference layer OIL. URL: http://www. ontoknowledge. org/oil/papers. shtml (2000) 8

[71] Horrocks, I., Others: DAML+OIL: A Description Logic for the Semantic Web. IEEE Data Engineering Bulletin 25(1), pp. 4–9 (2002) 8

[72] Ian Horrocks, P.F.P.S.: Reducing OWL Entailment to Description Logic Satisfiability. Journal of Web Semantics pp. 17—-29 (2003), http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.4085 89, 132

[73] Isabelle, P., Bourbeau, L.: TAUM-AVIATION: its technical features and some experimental results. Computational Linguistics 11(1), pp. 18–27 (1985) 16

[74] ISO: ISO DIS 24617-2 Language resource management Semantic annotation framework, Part 2: Dialogue acts. Tech. rep., ISO, Geneva (2010), http://let.uvt.nl/research/ti/semantic-annotation/DIS24617-2.pdf 15, 154, 155, 156, 171

[75] Jiménez-Ruiz, E., Cuenca Grau, B., Horrocks, I., Berlanga, R.: Ontology Integration Using Mappings: Towards Getting the Right Logical Consequences. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) Proceedings of the 6th European Semantic Web Conference. Lecture Notes in Computer Science, vol. 5554, pp. 173–187. Springer Berlin Heidelberg, Berlin, Heidelberg (May 2009), http://dl.acm.org/citation.cfm?id=1561533.1561554 91, 111

[76] Johnston, M., Bangalore, S., Vasireddy, G., Stent, A., Ehlen, P., Walker, M., Whittaker, S., Maloor, P.: MATCH: An architecture for multimodal dialogue systems. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. pp. 376–383. Association for Computational Linguistics (2002) 159

[77] Kaljurand, K.: ACE View an ontology and rule editor based on Attempto Controlled English. 5th International Workshop on OWL Experiences and Directions (2008) 18, 32, 51, 77

[78] Kaljurand, K.: General architecture of a controlled natural language based multilingual semantic wiki. Controlled Natural Language pp. 110–120 (2012) 18

[79] Kaljurand, K., Alumäe, T.: Controlled Natural Language in Speech Recognition Based User Interfaces. Controlled Natural Language pp. 79–94 (2012) 18

[80] Kaljurand, K., Fuchs, N.: Bidirectional mapping between owl dl and attempto controlled english. In: Principles and Practice of Semantic Web Reasoning. pp. 179–189. Springer (2006) 2, 18, 209, 210

[81] Kaljurand, K.: Paraphrasing Controlled English Texts. In: Fuchs, N.E. (ed.) Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009). CEUR Workshop Proceedings, vol. 448. CEUR-WS (2009), http://ceur-ws.org/Vol-448/paper8.pdf 19, 23

[82] Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.: Debugging unsatisfiable classes in OWL ontologies. Web Semantics: Science, Services and Agents on the World Wide Web 3(4), pp. 268–293 (Dec 2005), http://portal.acm.org/citation.cfm?id=1741308.1741345 9, 132, 210

[83] Kamp, H.: Discourse representation theory. Natural Language at the computer pp. 84–111 (1988) 18

[84] Kaufmann, E., Bernstein, A.: NLP-Reduce: a "naïve" but domain-independent natural langauge interface for querying ontologies. Proceedings of the 6th European Semantic Web Conference (2007) 80

[85] Kaufmann, E., Bernstein, A.: Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. Web Semantics: Science, Services and Agents on the World Wide Web 8(4), pp. 377–393 (Nov 2010), http://dx.doi.org/10.1016/j.websem.2010.06.001 80, 150

[86] Kaufmann, E., Bernstein, A., Zumstein, R.: Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. In: 5th International Semantic Web Conference. pp. 980 – 981 (2006), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.6633 150

[87] Khalili, A., Auer, S.: User Interfaces for Semantic Content Authoring: A Systematic Literature Review. Journal of Web Semantics p. Submitted (2012), http://svn.aksw.org/papers/2011/JWS_SemanticContentAuthoring/public.pdf 9, 14, 24

[88] Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM 42(4), pp. 741–843 (Jul 1995), http://dl.acm.org/citation.cfm?id=210332.210335 7

[89] Kim, J., Gil, Y.: Acquiring Problem-Solving Knowledge from End Users: Putting Interdependency Models to the Test. In: Proceedings of the National Conference on Artificial Intelligence. pp. 223–229 (Aug 2000), http://en.scientificcommons.org/42868239 149

[90] Klischewski, R.: Ontologies for e-document management in public administration. Business Process Management Journal 12(1), pp. 34–47 (2006) 1, 2

[91] Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open development environment for semantic web applications. The Semantic Web–ISWC 2004 pp. 229–243 (2004) 9

[92] Kotis, K., Vouros, G.: Human-centered ontology engineering: The HCOME methodology. Knowledge and Information Systems 10(1), pp. 109–131 (2006) 11, 12

[93] Kovacs, K., Dolbear, C., Hart, G., Goodwin, J., Mizen, H.: A Methodology for Building Conceptual Domain Ontologies. Technical Report IRI-0002. Tech. rep., Ordnance Survey Research Labs (2006), http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2006/Methodology_for_Building_Conceptual_Domain_Ontologies_V1.pdf 2, 11, 28, 33

[94] Kozaki, K., Kitamura, Y., Ikeda, M., Mizoguchi, R.: Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of Role and Relationship. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web. pp. 213–218 (Oct 2002), http://dl.acm.org/citation.cfm?id=645362.650867 9, 14

[95] Kuhn, T.: An evaluation framework for controlled natural languages. Proceedings of the 2009 conference on Controlled natural language 5972, pp. 1–20 (Jun 2010), http://dx.doi.org/10.1007/978-3-642-14418-9_1http://portal.acm.org/citation.cfm?id=1893475.1893477 69

[96] Kuhn, T.: The Understandability of OWL Statements in Controlled English. Semantic Web Journal (2012), http://www.semantic-web-journal.net/content/understandability-owl-statements-controlled-english 22, 63, 209

[97] Larsson, S.: Issue-based dialogue management. Department of Linguistics, G{ö}teborg University (2002) 160

[98] Larsson, S., Traum, D.R.: Information state and dialogue management in the TRINDI dialogue move engine toolkit. Natural Language Engineering 6(3–4), pp. 323–340 (2000), http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=68611&fulltextType=RA&fileId=S1351324900002539 153, 154, 156, 157, 160, 161, 164

[99] Lehmann, J., Bühmann, L.: ORE-a tool for repairing and enriching knowledge bases. The Semantic Web–ISWC 2010 pp. 177–193 (2010) 210

[100] Lenat, D.B., Guha, R.V.: The evolution of CycL, the Cyc representation language. ACM SIGART Bulletin 2(3), pp. 84–87 (1991) 8

[101] Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10(8), pp. 707–710 (1966) 100

[102] Liebig, T., Noppens, O.: OntoTrack: A semantic approach for ontology authoring. Web Semantics: Science, Services and Agents on the World Wide Web 3(2-3), pp. 116–131 (Oct 2005), http://www.sciencedirect.com/science/article/pii/S1570826805000120http://dl.acm.org/citation.cfm?id=1741305.1741319 90, 91

[103] Loutas, N., Lee, D., Maali, F., Peristeras, V., Tarabanis, K.A.: The Semantic Public Service Portal (S-PSP). In: Extended Semantic Web Conference (ESWC). pp. 227–242 (2011) 76

[104] Lu, X., Di Eugenio, B., Kershaw, T., Ohlsson, S., Corrigan-Halpern, A.: Expert vs. Non-expert Tutoring: Dialogue Moves, Interaction Patterns and Multi-utterance Turns. In: Gelbukh, A. (ed.) Computational Linguistics and Intelligent Text Processing, Lecture Notes in Computer Science, vol. 4394, pp. 456–467. Springer Berlin / Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-70939-8_40 150, 185

[105] Maedche, A., Staab, S.: Ontology learning for the Semantic Web. IEEE Intelligent Systems 16(2), pp. 72–79 (Mar 2001) 9

[106] Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, pp. 165–228 (2009) 9

[107] Mueckstein, E.M.: Controlled natural language interfaces (extended abstract): the best of three worlds. In: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science. pp. 176–178. ACM (1985) 16

[108] Noppens, O., Liebig, T.: Understanding Interlinked Data – Visualising, Exploring, and Analysing Ontologies. In: Proceeding of the International Conferences on Knowledge Management and New Media Technology. pp. 341–348 (2008) 14

[109] O'Connor, M., Das, A.: SQWRL: a Query Language for OWL. OWL Experiences and Directions Workshop (OWLED) (2009) 85

[110] Ordnance Survey's GeoSemantics team: Hydrology Ontology (2009), http://www.ordnancesurvey.co.uk/oswebsite/ontology/Hydrology/v2.0/Hydrology.owl 30, 31

[111] Pask, G., Kallikourdis, D., Scott, B.C.E.: The representation of knowables. International journal of man-machine studies 7(1), pp. 15–134 (1975) 16

[112] Pereira, F.C.N., Warren, D.H.D.: Definite clause grammars for language analysisa survey of the formalism and a comparison with augmented transition networks. Artificial intelligence 13(3), pp. 231–278 (1980) 18

[113] Peroni, S., Motta, E., D'Aquin, M.: Identifying Key Concepts in an Ontology, through the Integration of Cognitive Principles with Statistical and Topological Measures. In: Asian Semantic Web Conference. Springer-Verlag, Berlin, Heidelberg (2008) 14

[114] Quintal, L., Sampaio, P.: A methodology for domain dialogue engineering with the Midiki dialogue manager. In: Text, Speech and Dialogue. pp. 548–555. Springer (2007) 161

[115] Recommendation, W.: OWL 2 Web Ontology Language Document Overview (Second Edition) (2012), http://www.w3.org/TR/owl2-overview/ 2

[116] Roberts, R.B., Goldstein, I.P.: The FRL manual. Tech. rep., MIT (1977) 8

[117] Ross, R., Bateman, J.: Daisie: Information state dialogues for situated systems. In: Text, Speech and Dialogue. pp. 379–386. Springer (2009) 161

[118] Schiffrin, A., Petukhova, V.V., Salmon-Alt, S.: Documented compilation of semantic data categories. Tech. rep., LIRICS Project, Deliverable 4.3 (2007), http://lirics.loria.fr/doc_pub/D4-3.pdf 153

[119] Schmidt, A., Hinkelmann, K., Ley, T.: Conceptual foundations for a service-oriented knowledge and learning architecture: Supporting content,

process and ontology maturing. In: Proceeding of the International Conferences on Knowledge Management and New Media Technology. pp. 369–377 (2009), http://www.springerlink.com/index/f44633v28w339p74.pdf 13

[120] Schober, D., Malone, J., Stevens, R.: Practical experiences in concurrent, collaborative ontology building using Collaborative Protégé. Nature Precedings (2009), http://hdl.handle.net/10101/npre.2009.3517.2 12

[121] Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press, Cambridge, Mass. (1999), http://books.google.com/books?id=HlXOW_1fsIEC&hl=de 11

[122] Schwitter, R.: Representing knowledge in controlled natural language: a case study. In: Knowledge-Based Intelligent Information and Engineering Systems. vol. 3213, pp. 711–717. Springer, Wellington, New Zealand (2004) 2, 77, 209

[123] Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A Comparison of three Controlled Natural Languages for OWL 1.1. In: 4th OWL Experiences and Directions Workshop (OWLED 2008 DC). Washington DC (2008) 19, 20, 21, 32

[124] Schwitter, R., Ljungberg, A., Hood, D.: Ecole: A look-ahead editor for a controlled language. In: Proceedings of the Joint Conference combining the 8th International Workshop of the European Association for Machine Translation (EAMT) and the 4th Controlled Language Application Workshop (CLAW). pp. 141–150 (2003), http://web.science.mq.edu.au/~rolfs/papers/CLAW03-ECOLE.pdf 20, 22

[125] Schwitter, R.: Creating and Querying Formal Ontologies via Controlled Natural Language. Applied Artificial Intelligence 24(1), pp. 149–174 (2010) 2, 209

[126] Siegel, S., Castellan, J.: Nonparametric Statistics for the Behavioral Scences. McGraw-Hill, 2nd edn. (1988) 54

[127] Sirin, E., Parsia, B.: SPARQL-DL : SPARQL Query for OWL-DL. OWL Experiences and Directions Workshop (OWLED) (2007) 85

[128] Spreeuwenberg, S., van Grondelle, J., Heller, R., Grijzen, G.: Design of a CNL to Involve Domain Experts in Modeling. In: CNL 2010 Second Workshop on Controlled Natural Languages. vol. 622. CEUR-WS. org, Marettimo Island, Italy (2010) 16

[129] Sterling, L., Shapiro, E., Eytan, M.: The art of Prolog, vol. 94. Wiley Online Library (1986) 7

[130] Suárez-Figueroa, M., Gómez-Pérez, A.: Towards a glossary of activities in the ontology engineering field. In: The 6th Language Resources and Evaluation Conference. Marrakech (2008), http://oa.upm.es/5211/ 9

[131] Sure, Y., Staab, S., Studer, R.: On-To-Knowledge Methodology (OTKM). In: Handbook on Ontologies, pp. 117–132. International Handbooks on Information Systems, Springer (2004) 9, 11

[132] Tablan, V., Damljanovic, D., Bontcheva, K.: A natural language query interface to structured information. In: Proceedings of the 5th European Semantic Web Conference. pp. 361–375. Springer, Tenerife, Spain (2008), http://dl.acm.org/citation.cfm?id=1789430 17

[133] Tartir, S., Arpinar, I.B., Moore, M., Sheth, A.P., Aleman-Meza, B.: OntoQA: Metric-based ontology quality analysis. In: IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources. vol. 9 (2005) 58, 59

[134] TopQuadrant: TopBraid Composer, http://www.topquadrant.com/products/TB_Composer.html 9

[135] Traum, D.R.: Twenty Questions on Dialogue Act Taxonomies. Journal of semantics 17(1), pp. 7–30 (2000) 153

[136] Tudorache, T., Noy, N., Tu, S., Musen, M.: Supporting collaborative ontology development in protege. In: Proceedings of the 7th International Semantic Web Conference (ISWC2008). vol. 7 (2008) 12

[137] Tudorache, T., Falconer, S., Noy, N.F., Nyulas, C., Üstün, T.B., Storey, M.A., Musen, M.A.: Ontology Development for the Masses: Creating ICD-11 in WebProtégé. In: EKAW 2010 - Knowledge Engineering and Knowledge Management by the Masses. Lisbon (2010) 12, 14

[138] Uschold, M., King, M.: Towards a methodology for building ontologies. In: IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal (1995) 9, 11

[139] Wilcock, G.: Talking owls: Towards an ontology verbalizer. Human Language Technology for the Semantic Web and Web Services, ISWC 3, pp. 109–112 (2003) 16

[140] Witbrock, M., Baxter, D., Curtis, J., Schneider, D., Kahlert, R., Miraglia, P., Wagner, P., Panton, K., Matthews, G., Vizedom, A.: An interactive dialogue system for knowledge acquisition in cyc. In: Proceedings of the workshop on mixed-initiative intelligent systems. pp. 138–145 (2003) 149, 159, 210

[141] Wyner, A., Angelov, K., Barzdins, G., Damljanovic, D., Davis, B., Fuchs, N., Hoefler, S., Jones, K., Kaljurand, K., Kuhn, T., Others: On controlled natural languages: Properties and prospects. Controlled Natural Language pp. 281–289 (2010) 16

[142] Zhou, L.: Ontology learning: state of the art and open issues. Information Technology and Management 8(3), pp. 241–252 (2007) 9