

Infrastructure for Mass Argumentation Support on the Semantic Web

Bita Banihashemi

Master of Science in Information Technology
Faculty of Informatics
British University in Dubai
March 2008

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(Bita Banihashemi)

To my parents and my brother Sohail

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Iyad Rahwan. His wide knowledge in the field of argumentation, his encouragement and support have been of great value to me throughout this thesis.

My sincere thanks to Dr. Chris Reed, for his input and feedback during the early stages of this thesis. I am also deeply grateful to Professor Douglas Walton, for his invaluable advice and suggestions.

Abstract

In this thesis, I present an OWL ontology for describing arguments and argument schemes. Following the same key principles of the *World Wide Argument Web (WWAW)* for building a large-scale Web of structured and inter-connected arguments, this ontology provides an infrastructure for mass argumentation support on the Web.

First, I describe the OWL ontology which is based on a new reification of the Argument Interchange Format (AIF) and structures arguments according to Walton's theory of argumentation schemes. Then, I demonstrate how this ontology enables the use of automated Description Logic reasoning over argument structures. In particular, OWL reasoning enables significantly enhanced querying of arguments through automatic scheme classifications, instance classification, inference of indirect support in chained argument structures and inference of critical questions. Finally, I present the implementation of a Web-based system for authoring and querying argument structures in RDF which utilizes the proposed OWL ontology.

Contents

| | |
|--|-----------|
| Abstract | 6 |
| 1 Introduction | 10 |
| 1.1 Argumentation and Artificial Intelligence | 10 |
| 1.2 Argumentation Support Systems and the Web | 11 |
| 1.3 Problem Statement | 11 |
| 1.4 Aims | 12 |
| 1.5 Contributions | 12 |
| 1.6 Scope | 13 |
| 1.7 Organisation of the thesis | 13 |
| 2 Argumentation on the World Wide Web | 14 |
| 2.1 Mass Argumentation Tools on Web 2.0 | 14 |
| 2.1.1 Existing Tools | 14 |
| 2.1.2 Assessing Web 2.0 Tools | 16 |
| 2.2 Argumentation and the Semantic Web | 16 |
| 2.2.1 Existing Tools | 17 |
| 2.2.2 Assessing Semantic Web based Argumentation Tools | 18 |
| 2.3 Desiderata | 18 |
| 2.4 Opportunities for further improvement | 19 |
| 2.5 Comparison of the main features of different argumentation tools | 19 |
| 2.6 Summary | 19 |
| 3 Overview of Core Argument Interchange Format | 21 |
| 3.1 Argument Network and Nodes | 21 |
| 3.2 Edges in the Argument Network | 21 |
| 3.3 Representing Conflict Among Arguments in the AIF | 22 |
| 3.4 Summary | 23 |
| 4 Reification of Argument Schemes in the AIF | 24 |
| 4.1 Overview of Argument Schemes | 24 |
| 4.2 Schemes in the Original AIF | 25 |

| | | |
|----------|---|-----------|
| 4.3 | Classification of Scheme Hierarchy | 26 |
| 4.4 | Summary | 27 |
| 5 | A New Argumentation Ontology in Description Logic | 28 |
| 5.1 | Description Logics | 28 |
| 5.2 | Representing the Main Concepts and Properties | 28 |
| 5.3 | Examples | 30 |
| 5.4 | Capturing Support Among Chained Arguments | 31 |
| 5.5 | Representing Conflicts Among Arguments | 31 |
| 5.6 | Summary | 32 |
| 6 | OWL Reasoning over Argument Structures | 34 |
| 6.1 | Inference of Indirect Support in Chained Arguments | 34 |
| 6.2 | Automatic Classification of Argument Schemes and Instances | 35 |
| 6.2.1 | General Inference Pattern | 35 |
| 6.2.2 | Examples | 36 |
| 6.3 | Inferring Critical Questions | 37 |
| 6.3.1 | General Inference Pattern | 38 |
| 6.3.2 | An Example | 38 |
| 6.4 | Summary | 39 |
| 7 | Implementation | 40 |
| 7.1 | Basic System Architecture | 40 |
| 7.2 | Tools Used | 41 |
| 7.2.1 | Ontology Editor | 41 |
| 7.2.2 | RDF Repository (Semantic Web Framework) | 42 |
| 7.2.3 | Description Logic Reasoner | 42 |
| 7.2.4 | Core Webiste | 43 |
| 7.2.5 | Web Server | 43 |
| 7.2.6 | Database | 44 |
| 7.2.7 | Client-Side Scripting | 44 |
| 7.2.8 | Style Sheets | 44 |
| 7.3 | System Features | 44 |
| 7.3.1 | System Users | 45 |
| 7.3.2 | Listing available arguments | 45 |
| 7.3.3 | Creation of new arguments | 45 |
| 7.3.4 | Attacking/Supporting existing arguments | 48 |
| 7.3.5 | Retrieving attacking/supporting arguments of a claim | 49 |
| 7.3.6 | Retrieving scheme details | 49 |
| 7.3.7 | Displaying the scheme hierarchy and the statement hierarchy | 50 |

| | | |
|----------|---|-----------|
| 7.3.8 | Creation of new schemes | 50 |
| 7.3.9 | Searching for arguments based on keywords, authors, schemes and date range | 54 |
| 7.4 | Summary | 55 |
| 8 | Evaluation | 57 |
| 8.1 | Re-visiting the Desiderata | 57 |
| 8.2 | Avicenna and current Web-based argumentation tools | 58 |
| 8.3 | Limitations | 58 |
| 8.4 | Summary | 59 |
| 9 | Conclusion and Further Work | 61 |
| A | The Semantic Web and OWL | 62 |
| A.1 | The Semantic Web | 62 |
| A.2 | Web Ontology Language (OWL) | 63 |
| B | Argument Structures | 64 |
| C | Description Logics | 65 |
| D | Scheme Definitions in Description Logic | 67 |
| E | Sample Queries | 69 |

Chapter 1

Introduction

Argumentation can be defined as “a verbal and social activity of reason aimed at increasing (or decreasing) the acceptability of a controversial standpoint for the listener or reader, by putting forward a constellation of propositions (i.e. arguments) intended to justify (or refute) the standpoint before a rational judge” [51].

The theory of argumentation is a rich interdisciplinary area of research encompassing but not exclusive to philosophy, psychology, linguistics, and communication studies [15]. Argumentation has been recognised as a key area of importance in Artificial Intelligence, specifically, over the last decade.

1.1 Argumentation and Artificial Intelligence

The study of argumentation in Artificial Intelligence is receiving increasing attention since it was first recognised that argumentation can embark upon issues of reasoning and explanation in the presence of incomplete and uncertain information; unlike the assumption underlying usage of classical methods for representation and reasoning, that information is complete, certain and consistent.

One of the most significant differences between “logical proof” and “persuasive argument” is that arguments are defeasible: the reasoning that formed a persuasive case for a claim, in the light of changes in viewpoint or awareness of information not previously available, may subsequently fail to convince. This defeasibility is never removed: an argument may cease to be challenged and so accepted, but the possibility of challenge always remains [5].

Since argumentation has emerged as an important sub-discipline of Artificial Intelligence, a number of significant contributions have been made in both theoretical and practical branches of this field. Examples include: non-monotonic reasoning [13], knowledge engineering [11], natural language processing [18], legal reasoning [4], ontology engineering [45] and recommender systems technology [14].

Multi-agent systems communication and negotiation is another area where argumentation has been influential [36]. Recently, Toronni et al [46] have proposed an architecture for supporting a high-level reasoning and argumentation-driven interaction among Semantic Web services; another recent research [6] utilizes argumentation in composition of Web services. A comprehensive recent survey on argumentation in Artificial Intelligence is reported by Bench-Capon and Dunne [5].

1.2 Argumentation Support Systems and the Web

Another area witnessing significant growth is argumentation support systems [25] that enable users to browse, visualise, search and manipulate arguments and argument structures. The World Wide Web has made the advent of large-scale argumentation support possible. The Web can be seen as an ideal platform for enhancing argumentative expression, due to its global reach and openness. A variety of opinions and arguments are presented every day on the Web, in discussion forums, blogs, news sites, debate sites, mailing lists, etc. As such, the Web acts as an enabler of large-scale argumentation, where different views are presented, challenged, and evaluated by contributors and readers [38].

However, current Web-based argumentation systems exhibit a trade-off between scalability and structure. On one hand, scalable systems such as discussion forums and blogs, lack the argumentative structure that makes the task of searching, evaluating, comparing and identifying the relationships among arguments difficult. On the other hand, highly-structured deliberation support systems such as *Parmenides* [2] usually support a small number of participants (in specific domains) and are therefore not easily scalable. Moreover, there is lack of interoperability among existing systems and integration of argument repositories across different systems is not straight forward [38, 37].

Motivated by the current limitation in Web-based argumentation and the vision of creating a highly scalable yet highly structured argument representation on the Web, Rahwan et al [38] proposed theoretical and software foundations of a *World Wide Argument Web (WWAW)*: a large-scale Web of inter-connected arguments posted by individuals on the World Wide Web in a structured manner. The theoretical foundation was an ontology based on the Argument Interchange Format [15] but extended to include Walton's account of argumentation schemes [56]. The software foundation was based on Semantic Web ontology language RDF Schema (RDFS) [9]. They also proposed a pilot system, ArgDF¹ as the first implementation of this vision.

1.3 Problem Statement

The WWAW provides a rich and novel framework for building a Web of inter-connected arguments. However, the implemented ontology of ArgDF suffers from a number of limitations, both in terms of design specification and the ontology language used.

An important aspect that is not investigated by any of the Web-based argumentation support systems is the possibility of inferring and acquiring additional information about arguments in an argument network as opposed to what is explicitly captured. Argument networks change as new arguments are added or new interaction among arguments are formed. In such a dynamic network, automated reasoning can enable inference of additional and accurate knowledge about arguments.

Another feature that is not modelled or utilized by any of the existing systems is how certain argument schemes relate to each other. Argument schemes provide patterns of presumptive reasoning (for drawing inferences from certain premises to conclusions). As stated by Walton [56], some schemes are sub-species of others and thus, are not mutually disjoint.

I believe an opportunity exists to further improve the state of the art in Web-based argumentation systems; by following the same principles of WWAW, an ontology (and a Web-based system that utilizes this ontology) can be designed that is based on open standards (to enable integration of argument repositories among different tools). This ontology will provide high scalability while representing the arguments (and their interactions) in a highly structured manner. The ontology can improve upon the underlying ontology of ArgDF and solve some of its existing problems and limitations. Moreover, classification of hierarchy of argument schemes and utilizing different ontological reasoning tasks in an argument network can improve query

¹<http://www.argdf.org/>

answering and analysis of arguments. Exploring this opportunity forms the basis of this thesis.

1.4 Aims

This thesis addresses the following research questions:

- What are the ontological primitives needed to capture the relationship of argument schemes?
- Which knowledge representation language provides the formal syntax and semantics to model the argumentation domain?
- What types of ontological reasoning tasks can be useful in specifying arguments using ontology of schemes?
- How can we exploit this representation together with existing Semantic Web tools to develop systems for rich argumentation annotation, navigation and querying?

1.5 Contributions

The contributions of this thesis are both in terms of the design of an argumentation ontology and the implementation of a Semantic Web-based system exploiting the proposed ontology. I outline these contributions next as an answer to each of the research questions identified in the previous section.

- *What are the ontological primitives needed to capture the relationship of argument schemes?*

The new ontology is based on a reification of the Argument Interchange Format (AIF) exploiting Walton's account of schemes. It defines two main concepts: *Schemes* and *Statements*. Statements represent different parts of a scheme (i.e. conclusion, premises, assumptions and exceptions). Each scheme is defined in terms of its constituent parts; the conclusion and the premises of the scheme are defined as the necessary-and-sufficient conditions of the scheme while the exceptions and assumptions are captured through the necessary conditions. This enables explicit classification of schemes themselves.

- *Which knowledge representation language provides the formal syntax and semantics to model the argumentation domain?*

The formalisation of the proposed ontology is based on Web Ontology Language (OWL) in Description Logic notation. Description Logics provide a formal syntax and formal model-theoretic semantics, and thus, can provide support for sound and complete reasoning algorithms. In particular, a specific dialect of OWL, called OWL-DL is used which is equivalent to logic *SHOIN(D)* (very expressive, yet decidable).

Another important advantage of OWL (and other Semantic Web Ontology Languages) is that they represent information in standard, machine-processable formats. Therefore, the argument repositories of the system can be shared and integrated with other tools with minimum effort.

- *What types of ontological reasoning tasks can be useful in specifying arguments using ontology of schemes?*

The proposed ontology enables the first explicit use of Description Logic-based OWL reasoning over argument structures. In particular, OWL reasoning enables significantly enhanced querying and analysis of arguments through automatic scheme classifications, instance classification, inference of critical questions and inference of indirect support in chained argument structures. This provides a seed for further work that combines traditional argument-based reasoning techniques [13] with ontological reasoning in a Semantic Web environment.

- *How can we exploit this representation together with existing Semantic Web tools to develop systems for rich argumentation annotation, navigation and querying?*

The implementation of a Semantic Web-based system that exploits the new ontology to allow for creating, manipulating, and querying complex argument structures requires a number of tools. The core Website is built on Java. Jena provides the programming environment for manipulating the ontology model. Moreover, ARQ libraries are used to provide the SPARQL query engine. Pellet, an open source description logic reasoner for OWL-DL, enables inference over the ontology model generated by Jena. The ontology and the instances are stored in a SQL Server database.

The system provides the following features: listing the available arguments, creation of new arguments, attacking/supporting existing arguments, retrieving attacking/supporting arguments of a claim, retrieving argument scheme details, displaying hierarchy of argumentation schemes, creation of new argumentation schemes and searching for arguments in both basic and advanced modes.

1.6 Scope

In the research presented in this thesis, I focus on designing an OWL ontology based on a reification of the Argument Interchange Format for description and annotation of arguments and argument schemes. Current implementation utilizes Walton's account of argumentation schemes. I explore different ways in which the expressive power of OWL and the use of description logic reasoning can improve analysis and queries over argument structures. A Web-based system is also built to exploit this ontology and allow for creation, update and querying of arguments in RDF. This system also enables extension of the underlying ontology by adding new argument schemes.

This research does not engage in other important aspects of argumentation support systems, such as argument visualisation [25], evaluating the acceptability of arguments [17], natural language processing [58], content acquisition, argument generation [12], etc. However, the proposed ontology provides an interesting opportunity that can be explored through further work to include the above mentioned features of argumentation support systems.

1.7 Organisation of the thesis

The rest of this thesis is organised as follows: In the next chapter, I discuss some of the most important current Web-based argumentation applications, and outline a number of key requirements of WWAW for large-scale argumentation on the Web. In Chapter 3, a brief overview of the Argument Interchange Format is provided. Chapter 4 provides an overview of argument schemes and compares two of their reifications in the AIF: one by the underlying ontology of ArgDF and the second by the proposed ontology. The new ontology is presented in Chapter 5 and its various new reasoning capabilities are explained in Chapter 6. Chapter 7 focuses on the architecture of the Web-based system, the tools used and the system features. In Chapter 8, the Web-based system is compared against the WWAW key requirements and the system limitations are outlined. The last chapter explains future directions of this project and the conclusion. Appendix A includes an overview on Semantic Web technologies and the Web Ontology Language (OWL). Appendix B describes the most common argument structures. A brief explanation of Description Logics is available in Appendix C. Appendices D and E include sample argument scheme definitions (in Description Logics) and SPARQL queries respectively.

Chapter 2

Argumentation on the World Wide Web

In this chapter, I compare a number of Web-based argumentation tools and identify some of their limitations with respect to a truly global scale argumentation system. Moreover, a number of key requirements for large-scale argumentation on the Web are explained. I also explore some opportunities for further improvement of large-scale argumentation on the World Wide Web.

2.1 Mass Argumentation Tools on Web 2.0

There are a number of Web 2.0 applications,¹ designed to support large-scale argumentation on the World Wide Web. In this section, I discuss some of these tools and provide a concise assessment.

2.1.1 Existing Tools

An imperative feature of a Web-based argumentation system is its ability to capture (explicitly) the structural components of each argument as well as the way different facts, opinions, and arguments relate to one another and, as such, contribute to the argument network [38].

Some of the existing argumentation tools provide a simple structure; an example of such sites is Debatepedia.² It synthesizes two critical elements: a ‘wiki’ technology and a ‘logic tree’ debate methodology. The Wiki technology enables collaborative content management on the World Wide Web by users (a significant feature of Wikis is the ease with which pages are created, edited and linked). The logic tree is a pro/con logic tree, in which the main debate topic is located at the root. In other words, every debate is driven by a main yes/no question that reflects a substantial public debate. Then, users are allowed within this logic tree and with the accompanying software, to present yes/no sub-questions that break-down the larger main question. Sub-questions provide a way of breaking down, categorizing, and simplifying a certain debate topic. It is also possible to shift sub-questions up or down a tree.

Arguments are added in free format text and do not follow any specific structure. Users support (or attack) arguments by adding their arguments under pro/yes (or con/no) sections of each question. Arguments can be re-used across debates; however, each debate tree is treated in isolation. A basic keyword search is also provided in this system.

¹Web 2.0 refers to second phase of architecture in application development on the World Wide Web. It focuses on user contribution and collaboration by tagging data (e.g. Social bookmarking), editing data (e.g. Wikis), mass publishing (e.g. Weblogs) and Web feeds (e.g. RSS).

²<http://wiki.idebate.org/index.php/>

Debatemapper³ is one of the most recent Web-based argumentation systems which adds more structure to arguments by using elements representing different argumentative constructs (e.g. *issues* open to debate, the *positions* taken, *arguments* attacking or supporting these positions and *repertoire* of possible measures and alternative policies). Every map conforms to a ‘map grammar’ which specifies a vocabulary of element types and a set of rules stipulating how they may be combined.

It is possible to link elements across debate maps and build cross relationships among them. Users can also rate an element (e.g. a position) according to a scale (e.g. 1 to 9); the average of these ratings on each element is used in sorting them visually (in descending order) in maps. Different modes of keyword search on maps are available to users. While debatemapper structures argument maps by identifying different components such as issues or positions, it does not provide a distinction among different types of arguments.

Cohere [43] is another Web-based argumentation tool that is intended to allow students and researchers make personal and collective sense of problems while working on distributed systems. The data model of Cohere is inherited from ClaiMaker [49]. Users can create (or re-use) *Ideas* and link them by means of different *Connection* types. All connections are broadly classified as positive or negative. Ideas play one or more *Roles* within a given association. Both connections and roles can be extended by users. Using the role mechanism, IBIS structure⁴ or any other argumentation scheme can be modelled in the system.

In Cohere, visualisation is provided through a tool called *Connection Net*. This tool allows different views over the argument network by filtering the graph according to certain connection types. The system also offers a simple keyword search. Cohere represents arguments in a relational database format and this data is only transformed to XML before being sent to a Web page.

Truthmapping⁵ is a public argumentation support system which exhibits an advanced argument structure; it distinguishes between premises and conclusions of an argument. Users can agree with or attack existing arguments (via critiques) and the creator of the argument can add a single rebuttal to each critique. Arguments can be chained (although supporting claims is restricted to the same team members) and can contain hyperlinks to other Websites or to premises or conclusions of other arguments. A state map visually summarizes the overall user rating of different parts of an argument. A basic keyword search is provided over categories and topics.

Although Truthmapping provides a rather advanced structure of arguments, it only provides a distinction between deductive and inductive types of arguments. The fact that cross-references (hyperlinks) exist among different arguments enables linking them in different types of formations that *resemble* more complex structures such as divergent or convergent arguments. (See Appendix B for a brief overview of argument structures). However, as argued by Rahwan [37], the cross-references among arguments have no *explicit semantics*. So it is not possible to connect multiple arguments *explicitly* in complex structures (e.g. that contain combinations of convergent arguments, divergent arguments, etc.). This limitation hinders the possibility of using meta-data about arguments to enhance the automated search and evaluation of arguments on the Web.

Capturing the structural attributes of arguments, as well as the details of different complex interactions among arguments, not only facilitates evaluation and search of arguments, it also enables far better visualisation and navigation of arguments by users or automated tools. Moreover, it improves the group’s ability to reach consensus and make higher quality decisions [19]. Such structure could also simplify the automated support for the argumentation process (e.g. discovering inconsistencies or synergies among disputants)[38].

Parmenides [2] is an example of highly-structured argument-based deliberation support sys-

³<https://debatemapper.com/sf/home.aspx>

⁴The Issue Based Information Systems (IBIS) were first introduced to enable collaborative problem identification and solving through modelling issues, positions and arguments [41].

⁵<http://www.truthmapping.com>

tems (ADSS). Parmendies is based on a formal model of argumentation and a specific inference scheme for justifying the adoption of an action. It is intended to let governments (or other groups) present users with a position justifying a particular action and give them the opportunity to critique that position by disputing various points.

After a position is put forward, the system provides a forms-based, questionnaire interface to obtain views from the user, and a fixed set of possible attacks that can be made. Once the original position has been subjected to their critique, another sequence of forms enables them to propose positions of their own; again in a way which will lead them to construct their position in the same form of the argument scheme.

A main drawback of Parmendies, similar to most ADSS, is that it is intended for a small number of participants and focused on a specific domain. Consequently, it is based on specialised approaches of interaction and decision-making, instead of a general theory of argumentation. A Web-based argumentation system must be based on a general theory of argumentation and allow for a variety of reasoning patterns to structure interactions [38].

Another limitation of existing large-scale Web-based argumentation systems indicated by Rahwan [37] is the limited (or lack of) integration between the different argument repositories. This limits the ability to provide services (e.g question answering systems) that make use of arguments from multiple mass argumentation repositories. One of the main challenges in argumentation support tools on the Web is the lack of a shared ontology (or interchange format) for representing arguments and argumentation.

2.1.2 Assessing Web 2.0 Tools

The existing Web 2.0 systems for argumentation support suffer from a number of limitations. Firstly, most of them exhibit a trade-off between structure and scalability. Highly structured systems (such as Parmenides) are intended for smaller domains and are based on specific reasoning patterns instead of general theories of argumentation, while highly scalable systems (such as Debatepedia) present very simple structures of arguments and argument networks, limiting automated querying and analysis of argument repositories.

Another drawback with Web 2.0 argumentation systems such as Truthmapping is that while arguments are structured, and cross-links and connections exist among arguments, these links carry no explicit semantics. This limitation hinders the possibility of using meta-data about arguments to enhance the automated search and evaluation of arguments on the Web.

Lastly, a shared ontology (or interchange format) for representing arguments does not exist, making it very difficult (or impossible in some cases) to provide services (such as query answering) that utilize arguments from multiple mass argumentation systems [38, 37].

2.2 Argumentation and the Semantic Web

The key feature of Semantic Web technologies (see Appendix A for a brief overview) is that they represent Web information in standard, machine-processable formats. Semantic markup enables us to explicitly annotate arguments and their different components.

Semantic Web technologies can present a solution to the integration among mass argumentation tools through two key features: Firstly, a unified argument description ontology could act as an inter-lingua between the different tools and secondly, if a standard ontology of arguments can not be achieved, then ontology mapping tools [23] can potentially provide means for the automatic translation of a variety of argument annotation languages [37].

2.2.1 Existing Tools

XML, located at the lower layer of the Semantic Web technologies (see Figure A.1), introduces structure and *syntactic* interoperability. The provided structure can be made machine-accessible through DTDs and XML Schema.

CoPe.it! [48] utilizes XML and is designed for community deliberation on the Web. CoPe.it! is based on IBIS (Issue Based Information Systems) and captures users' contributions in form of issues, positions and arguments. This system implements an incremental formalization approach (to facilitate the process of decision making) through different projections of the workspace. A projection can be defined as a particular representation of the collaboration space. For example, the 'IBIS graphs' provide an informal projection while the synthesized 'outline tree' from this graph provides a formal projection. In the 'outline tree' mode, users are able to state their preferences over different positions by ranking the strength of each position. Depending on the status of positions and preferences, the scores for different alternatives of an issue are calculated. A basic search facility is also provided with this system.

An XML interchange language called "Argument Markup Language" (AML) has been proposed for structuring arguments by annotating premises and conclusions in XML. The valid AML document structures are contained in a DTD file. AML is used in Araucaria [42]. Araucaria is a stand-alone application tool for analysing and diagramming arguments. Arguments can be authored (diagrammed) using alternative sets of argumentation schemes such as Walton [54], Pollock [33], and Katzav and Reed [24]. This tool also provides the facility to design one's own argumentation schemes. Araucaria supports labels (tags) that are used to indicate user's evaluation for each part of an argument (nodes or arrows). These labels do not carry any semantics and are simply used to document user's preferences.

Once arguments have been analysed, they can be uploaded to AraucariaDB, which is an online repository of arguments. It provides a search engine,⁶ which allows advanced searches based on combination of different parameters such as argument schemes, argument creation date range, argument analyst or source, etc. Both CoPe.it! and Araucaria enable search over their online argument repositories using XPath queries.

These various attempts at providing argument markup languages share a limitation: they are built on XML, and standard XML does not provide any means of talking about *semantics* (meaning of the data); each particular language is designed for use with a specific tool; as a consequence, the semantics of arguments specified in these languages is tightly coupled with particular schemes to be interpreted in a specific tool and according to a specific underlying theory. Thus, for example, arguments in CoPe.it! are to be interpreted in relation to the theory of issue-based information systems [37].

Unlike XML, Semantic Web ontology languages such as RDF Schema (RDFS) [9] and OWL [28] can offer a unified ontology for describing and annotating arguments as they contain machine-processable *semantics*. RDFS is a vocabulary for describing properties and classes of RDF-based [27] resources, with semantics for hierarchies of such properties and classes. OWL adds more vocabulary for describing properties and classes to RDFS and enables reasoning about asserted concepts to infer new concepts.

DiscourseDB⁷ is an argumentation system based on Semantic Wiki [53] technology. This system collects the opinions of the world's journalists and commentators about ongoing political events and issues. Using this tool, users can post arguments and other users can have "for", "against" or "mixed" positions on those arguments. It provides the facility to export content into OWL/RDF format for use by other Semantic Web applications.

In addition to simpler keyword searches, this system offers a semantic search module in which users can use Semantic Media Wiki's query language to write queries. The only type of inference available over the argument network is sub-class hierarchy (e.g. a query to list instances of a specific topic theme returns instances under all the topics that are sub-classes

⁶<http://araucaria.computing.dundee.ac.uk/search.php>

⁷http://discoursedb.org/wiki/Main_Page

of that specific topic theme). While semantic search enables users to run different queries and retrieve the needed information, this feature can only be used by users who are familiar with the special Semantic Media Wiki’s query language.

Another drawback with DiscourseDB is that the arguments posted are in free format text and do not follow any specific structures or adhere to any explicit schemes. Moreover, it is not possible to form complex structures of multiple inter-connected arguments (e.g. convergent, divergent, etc). Capturing argument structures as well as the details of different interactions among arguments is essential for evaluation and querying of arguments by users or automated tools.

ArgDF is a pilot system presented by Rahwan et al [38] following the theoretical and software foundations of *World Wide Argument Web* (see Section 2.3 for a brief overview). ArgDF is based on a RDFS ontology that models the Argument Interchange Format (AIF) specification and extends it to include Walton’s account of argumentation schemes.

In ArgDF, users can author new arguments that adhere to any of the available schemes; they can also attack or support existing arguments (although the process of support is constrained in some ways). Users can also extend the underlying ontology by adding new argumentation schemes; the new schemes are added as new instances of the scheme related concepts (classes) in the ontology. A semantic based keyword search facility is also offered by the system that returns the supporting/attacking arguments of a claim containing a specific keyword.

ArgDF implements an interchange format and is based on open standards, and therefore resolves many problems related to current Web-based argumentation systems. However, the underlying ontology of ArgDF suffers from number of limitations, both in terms of design specification and the ontology language used.

A core argumentation ontology developed in OWL is reported by Verheij [52]. He suggests that each argumentation format should use the argumentation core ontology as its starting point and provide a translation back into the core ontology. In this case, translations between argumentation formats are optional and can be developed whenever considered useful. The core ontology is meant to provide the glue. At the time of writing, no Web-based system has been reported that utilizes this ontology.

2.2.2 Assessing Semantic Web based Argumentation Tools

As explained in the previous section, XML merely provides syntactic interoperability. Therefore, in systems based on XML (such as Araucaria and CoPe.it!), each particular language is designed for use with a specific tool; as a consequence, the semantics of arguments specified in these languages is tightly coupled with particular schemes to be interpreted in a specific tool and according to a specific underlying theory.

ArgDF utilizes a RDFS ontology which offers machine-processable semantics and therefore a unified ontology for describing and annotating arguments; however, RDFS does not offer the strength and expressivity that other ontology languages such as OWL can provide.

2.3 Desiderata

Motivated by the limitations in current Web-based argumentation systems, Rahwan et al [38] proposed the theoretical and the software foundations of a *World Wide Argument Web* (WWAW): a large-scale Web of structured and inter-connected arguments. In their paper, they have listed the following key requirements that are important in order to allow for large-scale argument annotation on the Web:

- The WWAW must support the storage, creation, update and querying of argumentative structures;

- The WWAW must have Web-accessible repositories;
- The WWAW language must be based on open standards, enabling collaborative development of new tools;
- The WWAW must employ a unified, extensible argumentation ontology;
- The WWAW must support the representation, annotation and creation of arguments using a variety of argumentation schemes;

2.4 Opportunities for further improvement

In this thesis, following the same principles as WWAW, I present an OWL ontology that reflects the argumentation domain in a more comprehensive way than the original ontology underlying ArgDF. This ontology is based on a new reification of the Argument Interchange Format (AIF). In addition, Web Ontology Language(OWL) offers richer features than RDFS and presents the potential for automated inference over argument structures, such as inference based on Description Logic [3]. As an example, reasoning can be used to infer the classification of hierarchy of argumentation schemes.

2.5 Comparison of the main features of different argumentation tools

Table 2.1 summarizes the main features of the argumentation tools discussed in this chapter. In this table, ‘argument representation’ denotes the main technology used to capture the structure of arguments. ‘Micro structure’ defines structure of a single argument while ‘macro structure’ represents the complex structures (e.g. divergent, convergent, serial) of multiple inter-connected arguments. Supporting or attacking claims refer to the ability of users to support or attack arguments posted by other users of the system.

2.6 Summary

In this chapter, I compared some of existing Web 2.0 argumentation systems and highlighted their main features. The main limitations with these systems were identified as a trade-off between scale and argument structure, lack of semantics in cross links among arguments and lack of a shared ontology or interchange format. I also presented the argumentation systems utilizing the Semantic Web technologies and discussed two of their drawbacks: the tight coupling between XML based systems and the utilized XML languages as well as limited expressivity of RDFS and consequently, the system utilizing it (ArgDF). A number of key requirements of a large-scale argument annotation of the Web were explained and finally some opportunities for further improvement were identified.

| <i>System Features</i> | <i>Debatepedia</i> | <i>Cohere</i> | <i>Debatemapper</i> | <i>Parmenides</i> | <i>Truthmapping</i> | <i>CoPe.it!</i> | <i>Araucaria</i> | <i>DiscourseDB</i> | <i>ArgDF</i> |
|-------------------------------------|--------------------|--------------------------------------|---|-------------------------------|---|--------------------------|---|--|---|
| Underlying Theory | Pro/Con Logic Tree | Connected Ideas and Roles | Proprietary | Persuasion Over Action | Proprietary | IBIS | Argumentation Schemes | Proprietary | AlF, W. Schemes ^a |
| Domain | General | Learning | General | Deliberative Democracy | General | General | Education | Politics | General |
| Argument Representation | Wiki, Logic tree | SQL DB, XML | SQL DB | SQL DB | SQL DB | XML | AML | Semantic Media Wiki | RDF/RDFS |
| Visualisation | - | Connection Net | Debate Maps | - | Statement Maps | IBIS Graphs | Argument Diagrams | - | - |
| Evaluation | - | - | Rating | - | Rating | Rating | (tags) | - | - |
| Micro Argument Structure | Pro/Con Position | Idea, Role, Connection | Pro/Con Argument, Issue, Position, etc. | Persuasion Over Action Scheme | Premise, Conclusion, Critique, 1 Rebuttal | Issue, Position Argument | Scheme Based ^b | For, Against, Mixed Item On a Position For a Topic | W. Schemes |
| Macro Argument Structure | - | Linked, Serial Divergent, Convergent | Linked, Serial Divergent, Convergent | - | Linked, Serial | I.N.A. ^c | Linked, Serial, Divergent, Convergent Schemes | - | Linked, Serial, Divergent, Convergent Schemes (Instances) |
| Extensible Ontology | - | - | - | - | - | - | - | - | - |
| Inference Over Argument Network | - | - | - | - | - | - | - | Subclass Hierarchy | - |
| Creation of new arguments | Yes | Yes | Yes | Restricted (Alternatives) | Yes | Yes | Yes | Yes | Yes |
| Re-use of existing claims | Yes | Yes | Yes | - | Yes | Yes | - | Yes | Yes |
| Supporting claims | Yes | Yes | Yes | (Yes/No Answers To Questions) | Team Members | Yes | - | Yes | Restricted |
| Attacking claims | Yes | Yes | Yes | (Yes/No Answers To Questions) | Yes | Yes | - | Yes | Yes |
| Automatic classification of schemes | - | - | - | - | - | - | - | - | - |
| Search | Keyword Search | Keyword Search | Keyword Search | - | Keyword Search (Topic, Category) | Keyword Search | Advanced Search (Scheme, date, author, etc.) | Semantic Search | Semantic Search (Keyword) |

Table 2.1: Comparison of Main Features of Argumentation Systems

^aWalton Schemes [56]

^bThe argument is structured according to the scheme; for example, if argument is structured according to a Walton scheme, it contains premise(s), conclusion, assumption(s) and exception(s)

^cInformation is not available.

Chapter 3

Overview of Core Argument Interchange Format

This chapter explains the current state of the Argument Interchange Format, which is the most recent general ontology for describing arguments and argument networks.

3.1 Argument Network and Nodes

The Argument Interchange Format(AIF) represents a core ontology of argument-related concepts. Its specification can be extended to capture different argumentation formalisms and schemes. The AIF core ontology assumes that argument entities can be represented as nodes in a directed graph called an *argument network*. A node can also have a number of internal attributes, denoting things such as title, creator (author), creation date, certainty degree, acceptability status, etc. Figure 3.1 depicts the original AIF ontology reported by Chesñevar et al [15].

Information nodes relate to content and are used to represent *passive* information contained in an argument, such as a claim, premise or data that depend on domain of discourse. On the other hand, S-nodes capture the application of *schemes* (i.e. patterns of reasoning). Such schemes may be considered as domain-independent patterns of reasoning, which resemble rules of inference in deductive logics but broadened to include non-deductive inference. The schemes themselves belong to a class of schemes and can be classified further into: *rule of inference scheme*, *conflict scheme*, and *preference scheme* etc.

The AIF specialises S-nodes further into three (disjoint) types of scheme nodes, namely *rule of inference application nodes* (RA-node), *preference application nodes* (PA-node) and *conflict application nodes* (CA-node). The word ‘application’ on each of these types was introduced in the AIF to emphasise the fact that these nodes function as instances, not classes, of possibly generic inference rules. Intuitively, RA-nodes capture nodes that represent (possibly non-deductive) rules of inference, CA-nodes capture applications of criteria (declarative specifications) defining conflict (e.g. among a proposition and its negation, etc.), and PA-nodes are applications of (possibly abstract) criteria of preference among evaluated nodes.

3.2 Edges in the Argument Network

The argument network contains edges that connect different nodes. For example, an edge named “uses” connects a S-node to the scheme it exploits. The AIF core specification does not type its edges. Edge semantics can be inferred from the types of nodes they connect. The informal semantics of edges are listed in Table 3.1. Basically there are two types of edges: the

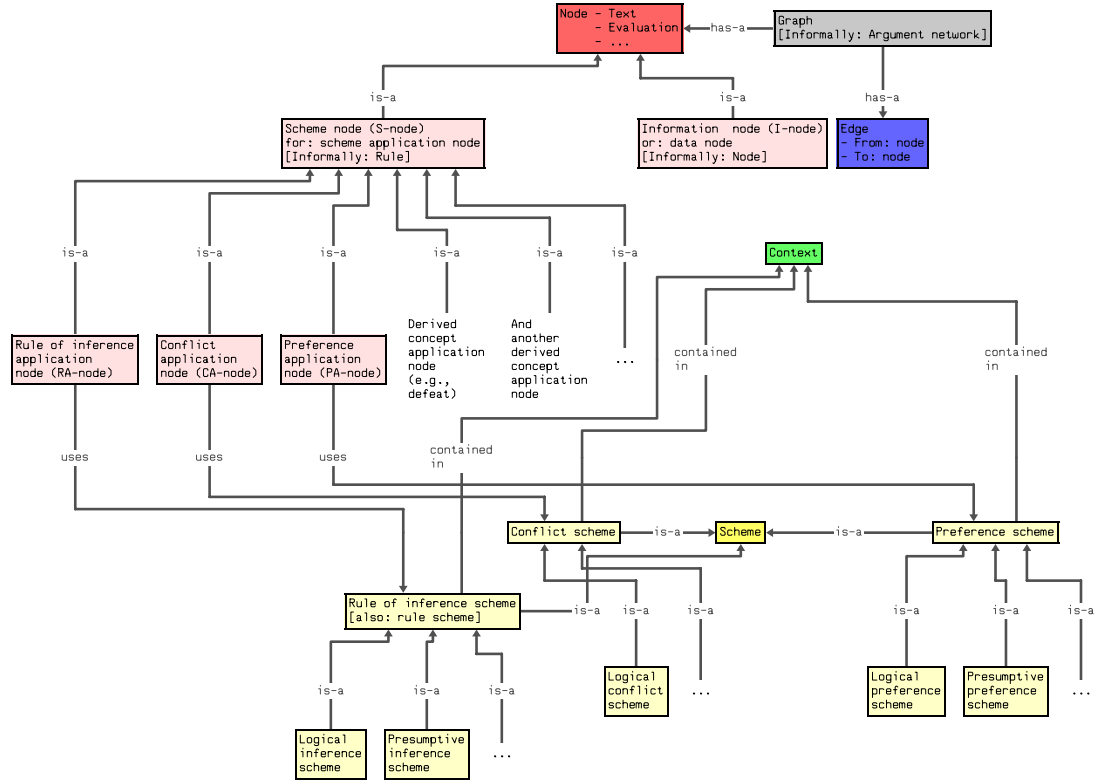


Figure 3.1: Original AIF Ontology [15]

scheme edges that emanate from S-nodes and are meant to support conclusions that follow from the S-node (these conclusions may either be I-nodes or S-nodes); and the *data edges* that emanate from I-nodes ending in S-nodes and are meant to supply data, or information to scheme applications. One of the restrictions imposed by the AIF is that no outgoing edge from an I-node can be connected directly to another I-node. This ensures that the relationship between two pieces of information must be specified explicitly via an intermediate S-node.

| | to I-node | to RA-node | to PA-node | to CA-node |
|--------------|-------------------------------------|--|--|--|
| from I-node | | I-node data used in applying an inference | I-node data used in applying a preference | I-node data in conflict with information in node supported by CA-node |
| from RA-node | inferring a conclusion (claim) | inferring a conclusion in the form of an inference application | inferring a conclusion in the form of a preference application | inferring a conclusion in the form of a conflict definition application |
| from PA-node | preference over data in I-node | preference over inference application in RA-node | meta-preferences: applying a preference over preference application in supported PA-node | preference application in supporting PA-node in conflict with preference application in PA-node supported by CA-node |
| from CA-node | incoming conflict to data in I-node | applying conflict definition to inference application in RA-node | applying conflict definition to preference application in PA-node | showing a conflict holds between a conflict definition and some other piece of information |

Table 3.1: Informal semantics of untyped edges in core AIF [15]

A simple argument in propositional logic is depicted in Figure 3.2(a). The S-nodes are distinguished from I-nodes graphically by drawing the former with a slightly thicker border. The node marked MP_1 denotes an application of the modus ponens inference rule.

3.3 Representing Conflict Among Arguments in the AIF

An attack or a conflict from one information or scheme node to another is captured through a CA-node, which captures the type of conflict. An asymmetric attack represents a state where one node (e.g. I-node) attacks another node (e.g. I-node) through a CA-node. On the

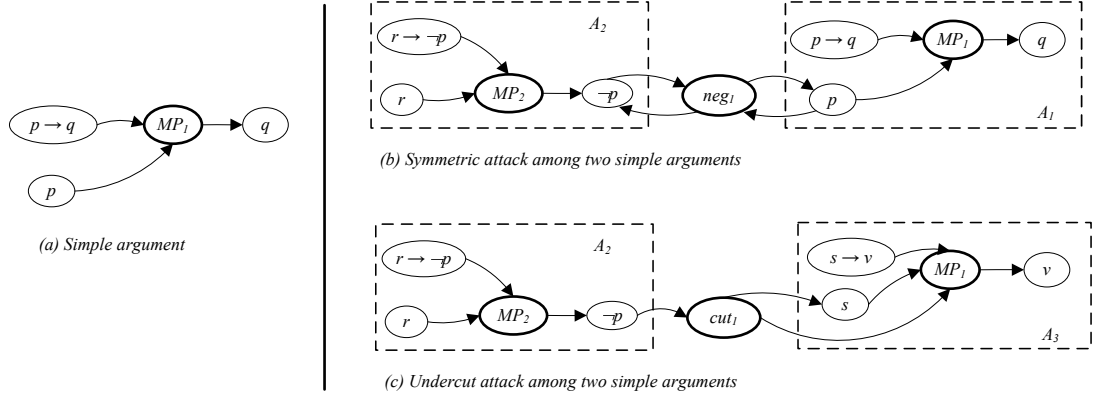


Figure 3.2: Examples of simple arguments and conflicts among them

other hand, in symmetric attacks, two nodes (e.g. I-nodes) attack each other simultaneously through a CA-node. Figure 3.2(b) depicts a symmetric conflict between two simple arguments (commonly known as a rebut attack in the literature). The node marked neg_1 denotes conflict as propositional negation.

Figure 3.2(c) illustrates a situation where a rule of inference node (RA-node) is attacked by an I-node through a CA-node. An attack on an inference application is often referred to as an undercut [32].¹ The node cut_1 represents conflict as an undercut.

3.4 Summary

The Argument Interchange Format represents a core ontology of argument-related concepts. It assumes that argument entities are represented as nodes in a directed graph called the argument network. Nodes are divided into two main groups : I-nodes that represent passive information (e.g. claims, premises, etc.) and S-nodes that capture application of schemes (i.e. patterns of reasoning). S-nodes are further specialised into different types of scheme nodes (RA-node, CA-node and PA-node) to capture different types of application of schemes. The argument network also contains edges that connect these nodes. The edges in the AIF are not typed and their semantics can be inferred from the types of nodes they connect. Different types of conflicts among arguments (e.g. asymmetric attacks, symmetric attacks) can also be represented in the AIF.

¹In some literature, asymmetric attacks by a CA-node on an I-node are also referred to as undercuts; for example, as explained by Prakken and Sartor [34], an argument A undercuts another argument B if A proves(claims) what was assumed unprovable by B .

Chapter 4

Reification of Argument Schemes in the AIF

This chapter describes argument schemes and explains their formalisation in the underlying ontology of ArgDF and suggests an enhancement to this formalisation. Moreover, classification of argument schemes in a hierarchy is discussed.

4.1 Overview of Argument Schemes

Argumentation schemes are forms of argument that capture stereotypical patterns of reasoning. They might represent the deductive or inductive forms of argument as well as forms of argument that are presumptive in nature [40]. These schemes are referred to as *presumptive* inference patterns, in the sense that if the premises are true, then the conclusion may *presumably* be taken to be true.

Structures and taxonomies of schemes have been analyzed and proposed by many theorists such as Perelman and Olbrechts-Tyteca [31], Eemeren et al [50], and Katzav and Reed [39]. But it is Walton's exposition [56] that has been most influential in computational work.

Each *Walton scheme* has a name, conclusion, set of premises and a set of critical questions. Critical questions enable contenders to identify the weaknesses of an argument based on the particular scheme, and potentially attack the argument. Here is an example:

Scheme 1. (*Scheme for Argument from Position to Know*)

- Position to know premise: *E is in a position to know whether A is true (false).*
- Assertion Premise: *E asserts that A is true (false).*
- Conclusion: *A may plausibly be taken to be true (false).*

Many types of different schemes are explained by Walton [56]; examples of which include: *argument from sign*, *argument from analogy*, *appeal to expert opinion*, etc. Actual arguments are *instances* of schemes.

Argument 1. (*Instance of Argument from Position to Know*)

- Premise: *Allen is in a position to know whether Brazil has the best football team.*
- Premise: *Allen says Brazil has the best football team.*
- Conclusion: *Brazil has the best football team.*

It is possible that premises may not always be stated, in which case it is said that a given premise is *implicit* [56]. One of the benefits of argument classification is that it enables analysts to uncover the hidden premises behind an argument, once the scheme has been identified.

One way to evaluate arguments is through *critical questions*, which serve to inspect arguments based on a particular argument scheme. For example, Walton [56] identified the following critical questions for “argument from position to know” :

Critical Questions 1. (*Critical Questions for Argument from Position to Know*)

1. Knowledge: *Is E in a position to know whether A is true(false)?*
2. Trustworthiness: *Is E an honest (trustworthy, reliable) source?*
3. Opinion: *Did E assert that A is true(false)?*

It can be seen that critical questions 1 and 3 merely question two of the explicit premises. In this thesis, the critical questions of such nature are omitted; thus, the only critical question focused on with regard to “argument from position to know” is the “Trustworthiness” question.

As discussed by Gordon et al [22], critical questions are not all alike. Some questions may refer to *assumptions* required for the inference to go through, while others may refer to *exceptions* to the rule, and correspond to Toulmin’s *rebuttal* [47]. The contemporary view is that the main difference between assumptions and exceptions lies in the *burden of proof*. The proponent of the argument has the burden of proof to answer questions about assumptions, while with exceptions the burden *shifts* to the questioner.

4.2 Schemes in the Original AIF

Let us now consider how schemes may be formalised in the AIF. The initial AIF specification separates the classification of nodes from the classification of schemes (see Figure 3.1). Both nodes and schemes are independently classified upper-level concepts. S-nodes are classified into nodes that capture inference, conflict, preference, etc. Likewise, schemes are classified into similar sub-schemes such as inference schemes, conflict schemes and so on. S-nodes are linked to schemes via a special edge *uses*.

It should be noted that the original AIF represents an “abstract model”, allowing a number of different concrete reifications to be made. The reification of the AIF in ArgDF ontology defines two types of classes for representing schemes and nodes. Moreover, Rahwan et al [38] introduced a new type of class, *Form node (F-node)*, to capture the generic form of statements (e.g. assumptions, premises) that constitute presumptive arguments. For example, *PremiseDescriptor* is a sub-class of F-node that captures the generic form of premises used in arguments.

In ArgDF ontology, the actual arguments are specified by instantiating nodes, while actual schemes are created by instantiating the “scheme” class. Then, argument instances (and their constituent parts) are linked to scheme instances (and their part descriptors) in order to show what scheme the argument follows.

Figure 4.1 shows an argument network for “an argument from position to know” using the underlying ontology of ArgDF. Here, each node in the actual argument (unshaded nodes) is explicitly linked, via a special-purpose property, to the form node it instantiates (shaded nodes). These special-purpose properties (e.g. *fulfilsScheme*) are particular reifications of the “*uses*” relation (between S-nodes and schemes) in the original AIF specification.

From the above, it is clear that ArgDF’s reification of the AIF causes some redundancy at the instance level. Both arguments and schemes are described with explicit structure at the instance level. As a result, the property “*fulfilsScheme*” does not capture the fact that a S-node represents an instantiation of some generic *class of arguments* (i.e. scheme). Having such relationship expressed explicitly can enable reasoning about the classification of schemes.

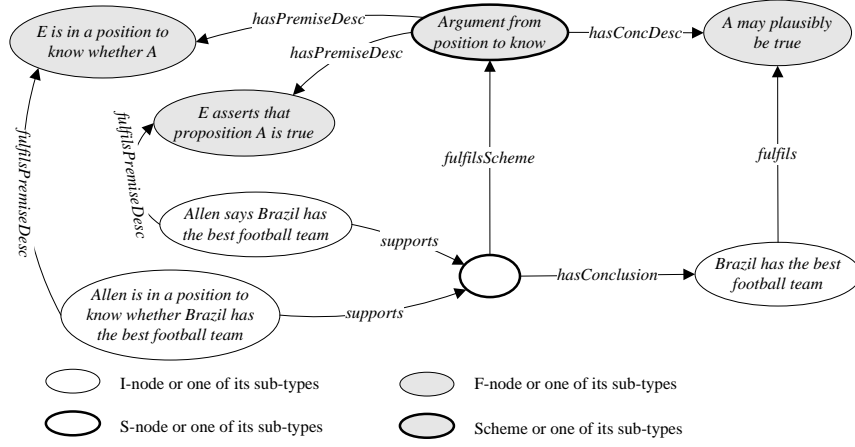


Figure 4.1: An argument network linking instances of argument and scheme components

In fact, the ontology presented in this thesis captures this relationship explicitly; presenting a simpler and more natural ontology of arguments. The AIF model is reified by interpreting schemes as classes and S-nodes as instances of those classes; in this case, the semantics of the “uses” edge can be interpreted as “instance – of”. The design of the new ontology is discussed in detail in Chapter 5.

4.3 Classification of Scheme Hierarchy

A notable aspect of schemes, receiving relatively little attention in the literature, is that they do not merely describe a *flat* ontology of arguments. Consider the following scheme:

Scheme 2. (*Scheme for Appeal to Expert Opinion*)

- Expertise premise: *Source E is an expert in domain D containing proposition A.*
- Assertion premise: *E asserts that proposition A is true (false).*
- Conclusion: *A may plausibly be taken to be true (false).*

It is clear that this scheme *specialises* the scheme for argument from position to know. Apart from the fact that both schemes share the conclusion and the assertion premise, the statement “Source *E* is an expert in domain *D* containing proposition *A*” is clearly a specialisation of the statement that “*E* is in a position to know (things about *A*).” Having expertise in a field causes one to be in a position to know things in that field.¹

Consider also the critical questions associated with the scheme for appeal to expert opinion [56] (again, here the Walton’s “field” and “opinion” questions are omitted since they merely question two of the explicit premises). Notice that the trustworthiness question is repeated, while additional expertise-related questions are added.

Critical Questions 2. (*Critical Questions for Appeal to Expert Opinion*)

1. Expertise: *How credible is expert E?*
2. Trustworthiness: *Is E reliable?*
3. Consistency: *Is A consistent with the testimony of other experts?*

¹Indeed, there may be other reasons to be in a position to know *A*. For example, if *E* is taken to refer to society as a whole, then the argument from position to know becomes “argument from popular opinion.”

4. Backup Evidence: *Is A supported by evidence?*

Thus, schemes themselves have a hierarchical ontological structure, based on a classification of their constituent premises and conclusions. The underlying ontology of ArgDF does not classify schemes according to this level of detail, but as whole entities.

As another example, let us consider legal formulation of “appeal to expert opinion”; this particular form of reasoning is of central importance in law as it relates to evidence [57]. This scheme is called “legal argument from expert opinion” and has the following structure:

Scheme 3. (*Scheme for Legal Argument from Expert Opinion*)

- Competence premise: *E is an expert in knowledge domain D (containing A).*
- Statement premise: *E said (asserted or implied) the sentence A*.*
- Interpretation premise: *A is a reasonable interpretation of A*.*
- Conclusion: *A may plausibly be taken to be true (false).*

As can be seen, this scheme includes the premises and conclusion of the “appeal to expert opinion” scheme, and in addition, it contains another specialisation premise, namely the interpretation premise. Therefore, this scheme specialises the “appeal to expert opinion” scheme.

The following critical questions are associated with this scheme. As can be seen, some of the critical questions of “appeal to expert opinion” are repeated while some new questions are added:

Critical Questions 3. (*Critical Questions for Legal Argument from Expert Opinion*)

1. Expertise: *How credible is expert E?*
2. Careful Analysis: *is E’s testimony A* based on his own careful analysis of evidence in this case?*
3. Other Experts: *Is A consistent with the testimony of other experts?*
4. Depth of Knowledge: *Is the knowledge of E about D deep enough to know about A?*

Capturing such structures (and in general, capturing the hierarchical ontological structure of different argumentation schemes) presents an opportunity to enhance analysis and querying of arguments in argument networks.

4.4 Summary

Argument schemes are forms of arguments that capture stereotypical patterns of reasoning. Each Walton scheme has a name, conclusion, premises and set of critical questions. The contemporary view on critical questions categorizes them under assumptions and exceptions, based on the burden of proof. In reification of the AIF by ArgDF, structure of arguments and structure of schemes are represented separately; this presents a redundancy in the ontology which can easily be eliminated. Moreover, the schemes themselves have a hierarchical ontological structure, based on classification of their constituent premises and conclusions. The ArgDF ontology does not classify schemes according to this level of detail, but as whole entities.

Chapter 5

A New Argumentation Ontology in Description Logic

In this chapter I describe the formalisation of the new argumentation ontology based on a new reification of the AIF in Description Logic.

5.1 Description Logics

Description Logics (DLs)[3] are a family of logical formalisms that have initially been designed for the representation of conceptual knowledge in Artificial Intelligence (see Appendix C for a short overview of DL). Description Logic knowledge representation languages provide means for expressing knowledge about concepts composing a terminology (TBox), as well as knowledge about concrete facts (i.e. objects instantiating the concepts) which form a world description (ABox). Since Description Logics are provided with a formal syntax and formal model-theoretic semantics, sound and complete reasoning algorithms can be formulated.

The formalisation of the new argumentation ontology is done using the Web Ontology Language OWL [28] in Description Logic (DL) notation. OWL is partially mapped on a Description Logic. The ontology is designed using a particular dialect of OWL, called OWL-DL, which is equivalent to logic $\mathcal{SHOIN}(D)$ [3]. While very expressive,¹ $\mathcal{SHOIN}(D)$ is still decidable, therefore enabling the use of efficient reasoning support.

5.2 Representing the Main Concepts and Properties

At the highest level, three concepts are identified: *statements* that can be made, *schemes* that describe arguments made up of statements,² and *authors* of those statements and arguments. All these concepts are disjoint.

$$\begin{aligned} \textit{Scheme} &\sqsubseteq \textit{Thing} \\ \textit{Statement} &\sqsubseteq \textit{Thing} \\ \textit{Author} &\sqsubseteq \textit{Thing} \\ \textit{Author} &\sqsubseteq \neg \textit{Scheme} \\ \textit{Author} &\sqsubseteq \neg \textit{Statement} \\ \textit{Statement} &\sqsubseteq \neg \textit{Scheme} \end{aligned}$$

¹ $\mathcal{SHOIN}(D)$ allows expression of basic DL, transitive roles, nominals, role hierarchy, inverse roles and number restrictions.

²In this thesis, the terms “scheme” and “class of arguments” are used interchangeably.

As with the original AIF, different specialisations of scheme are identified; for example the rule scheme (which describes the class of arguments), conflict scheme, preference scheme etc.

RuleScheme \sqsubseteq *Scheme*
ConflictScheme \sqsubseteq *Scheme*
PreferenceScheme \sqsubseteq *Scheme*

Each of these schemes can be further classified. For example, a rule scheme may be further specialised to capture such deductive or presumptive arguments. The same can be done with different types of conflicts, preferences, and so on.

DeductiveArgument \sqsubseteq *RuleScheme*
InductiveArgument \sqsubseteq *RuleScheme*
PresumptiveArgument \sqsubseteq *RuleScheme*
LogicalConflict \sqsubseteq *ConflictScheme*
PresumptivePreference \sqsubseteq *PreferenceScheme*
LogicalPreference \sqsubseteq *PreferenceScheme*

A number of properties (or *roles* in DL terminology) are defined, which can be used to refer to additional information about instances of the ontology, such as authors of arguments, the creation date of a scheme, and so on. The domains and ranges of these properties are restricted appropriately and described below.³

$\top \sqsubseteq \forall \text{creationDate}. \text{Date}$
 $\top \sqsubseteq \forall \text{creationDate}^-. \text{Scheme}$
 $\top \sqsubseteq \forall \text{argTitle}. \text{String}$
 $\top \sqsubseteq \forall \text{argTitle}^-. \text{RuleScheme}$
 $\top \sqsubseteq \forall \text{authorName}. \text{String}$
 $\top \sqsubseteq \forall \text{authorName}^-. \text{Author}$
 $\text{Scheme} \sqsubseteq \forall \text{hasAuthor}. \text{Author}$
 $\text{Scheme} \sqsubseteq = 1 \text{creationDate}$
 $\text{RuleScheme} \sqsubseteq = 1 \text{argTitle}$

To capture the structural relationships between different schemes, their components should first be classified. This is done by classifying their premises, conclusions, assumptions and exceptions into different *classes of statements*. For example, at the highest level, we may classify statements to declarative, comparative, and imperative, etc.⁴

DeclarativeStatement \sqsubseteq *Statement*
ImperativeStatement \sqsubseteq *Statement*
ComparativeStatement \sqsubseteq *Statement* ...

Actual statement instances have a property that describes their textual content.

$\top \sqsubseteq \forall \text{claimText}. \text{String}$
 $\top \sqsubseteq \forall \text{claimText}^-. \text{Statement}$

When defining a particular RuleScheme (i.e. class of arguments), I capture the relationship between each scheme and its components. Each argument has exactly one conclusion and at least one premise (which are, themselves, instances of class “Statement”). Furthermore, presumptive arguments may have assumptions and exceptions.

³The counterpart of data types in Description Logics are called *concrete* domains which allow integration of concrete qualities such as numbers and strings [26] into the formalism.

⁴In this thesis, an extensive ontological discussion of all types of statements is deliberately avoided. The interest is in a (humble) demonstration of how a given classification of argument *parts* may be useful for automated ontological reasoning about argument *types*. How individual parts get categorised into classes (e.g. using natural language processing techniques, or manual tagging) is beyond the scope of this thesis.

$RuleScheme \sqsubseteq \forall hasConclusion.Statement$
 $RuleScheme \sqsubseteq = 1hasConclusion$
 $RuleScheme \sqsubseteq \forall hasPremise.Statement$
 $RuleScheme \sqsubseteq \geq 1hasPremise$
 $PresumptiveArgument \sqsubseteq \forall hasAssumption.Statement$
 $PresumptiveArgument \sqsubseteq \forall hasException.Statement$

5.3 Examples

With this in place, it becomes possible to further classify the above statement types to cater for a variety of schemes. For example, to capture the scheme for “argument from position to know,” the following classes of declarative statements need to be defined (each class is listed with its property `formDescription`⁵ that describes its typical form).

$PositionToHaveKnowledgeStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “E is in position to know whether A is true (false)”
 $KnowledgeAssertionStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “E asserts that A is true(false)”
 $KnowledgePositionStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “A may plausibly be taken to be true(false)”
 $LackOfReliabilityStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “E is not a reliable source”

Now it is possible to fully describe the scheme for “argument from position to know.” Following are the necessary as well as the necessary-and-sufficient conditions for an instance to be classified as an argument from position to know.

$ArgFromPositionToKnow \equiv (PresumptiveArgument \sqcap$
 $\exists hasConclusion.KnowledgePositionStmnt \sqcap$
 $\exists hasPremise.PositionToHaveKnowledgeStmnt \sqcap$
 $\exists hasPremise.KnowledgeAssertionStmnt)$
 $ArgFromPositionToKnow \sqsubseteq \exists hasException.LackOfReliabilityStmnt$

Now, for the “appeal to expert opinion” scheme, we only need to define one additional premise type, since both the conclusion and the assertion premise are identical to those of “argument from position to know.”

$FieldExpertiseStmnt \sqsubseteq PositionToHaveKnowledgeStmnt$
`formDescription` : “source *E* is an expert in subject domain *D* containing proposition *A*”

Similarly, one of the exceptions of this scheme is identical to “argument from position to know.” The remaining assumptions and exception are added as follows:

$ExpertiseInconsistencyStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “A is not consistent with other experts assertions”
 $CredibilityOfSourceStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “E is credible as an expert source”
 $ExpertiseBackUpEvidenceStmnt \sqsubseteq DeclarativeStatement$
`formDescription` : “E’s assertion is based on evidence”

Likewise, the necessary and necessary-and-sufficient conditions of “appeal to expert opinion” are:

⁵formDescription is an annotation property in OWL-DL. Annotation properties are used to add meta-data about classes.

$$\begin{aligned}
AppToExpertOpinion &\equiv (PresumptiveArgument \sqcap \\
&\exists hasConclusion.KnowledgePositionStmnt \sqcap \\
&\exists hasPremise.FieldExpertiseStmnt \sqcap \exists hasPremise.KnowledgeAssertionStmnt) \\
AppToExpertOpinion &\sqsubseteq \exists hasException.LackOfReliabilityStmnt \\
AppToExpertOpinion &\sqsubseteq \exists hasException.ExpertiseInconsistencyStmnt \\
AppToExpertOpinion &\sqsubseteq \exists hasAssumption.CredibilityOfSourceStmnt \\
AppToExpertOpinion &\sqsubseteq \exists hasAssumption.ExpertiseBackUpEvidenceStmnt
\end{aligned}$$

Other argument schemes (e.g. argument from analogy, argument from sign, etc.) can be defined in the same way. Appendix D includes sample definitions of argument schemes in Description Logic.

5.4 Capturing Support Among Chained Arguments

Arguments can support other arguments by supporting their premises. This results in argument *chaining* where a claim acts both as a premise of one argument and as a conclusion of another. A transitive property named *supports* was added to the ontology, to allow linking the supporting argument to the supported argument in a chain:

$$RuleScheme \sqsubseteq \forall supports.RuleScheme$$

5.5 Representing Conflicts Among Arguments

Conflict among arguments are captured through different specialisations of *ConflictScheme* such as *GeneralConflict* and *ExceptionConflict*.

$$\begin{aligned}
ExceptionConflict &\sqsubseteq ConflictScheme \\
GeneralConflict &\sqsubseteq ConflictScheme
\end{aligned}$$

GeneralConflict instances capture simple symmetric and asymmetric attacks among arguments while *ExceptionConflict* instances represent exceptions to the general rule of inference. The definition of *ConflictScheme* and *Statement* classes have been extended to include the appropriate restrictions on properties used to represent attacks among different arguments.

$$\begin{aligned}
ConflictScheme &\sqsubseteq \forall confAttacks.(Statement \sqcup RuleScheme) \\
ConflictScheme &\sqsubseteq \forall isAttacked.Statement \\
ConflictScheme &\sqsubseteq \forall underMinesAssumption.Statement \\
Statement &\sqsubseteq \forall attacks.ConflictScheme \\
Statement &\sqsubseteq \forall confIsAttacked.ConflictScheme
\end{aligned}$$

Figures 5.1 (a to d) illustrate how instances of conflict scheme and the related properties are used to represent 4 different types of conflicts among arguments; namely asymmetric attacks (a), symmetric attacks (b), undermining assumptions (c) and attacking by supporting the existing exceptions (d).

In these figures, argument instances are denoted by Arg_n , premises are denoted by PX_n , conclusions by CX , assumptions by $AsmX_n$, exceptions by $ExcpX_n$ and instances of general conflict and exception conflict as GC_n and EC_1 respectively where $X = \{A, B, C, \dots\}$ and n represents the set of natural numbers $\{1, 2, 3, \dots\}$.

Figure 5.2 shows the different parts of an argument instance $A1$, which is of type “appeal to expert opinion.” The conclusion of this argument is experiencing a symmetric attack by the

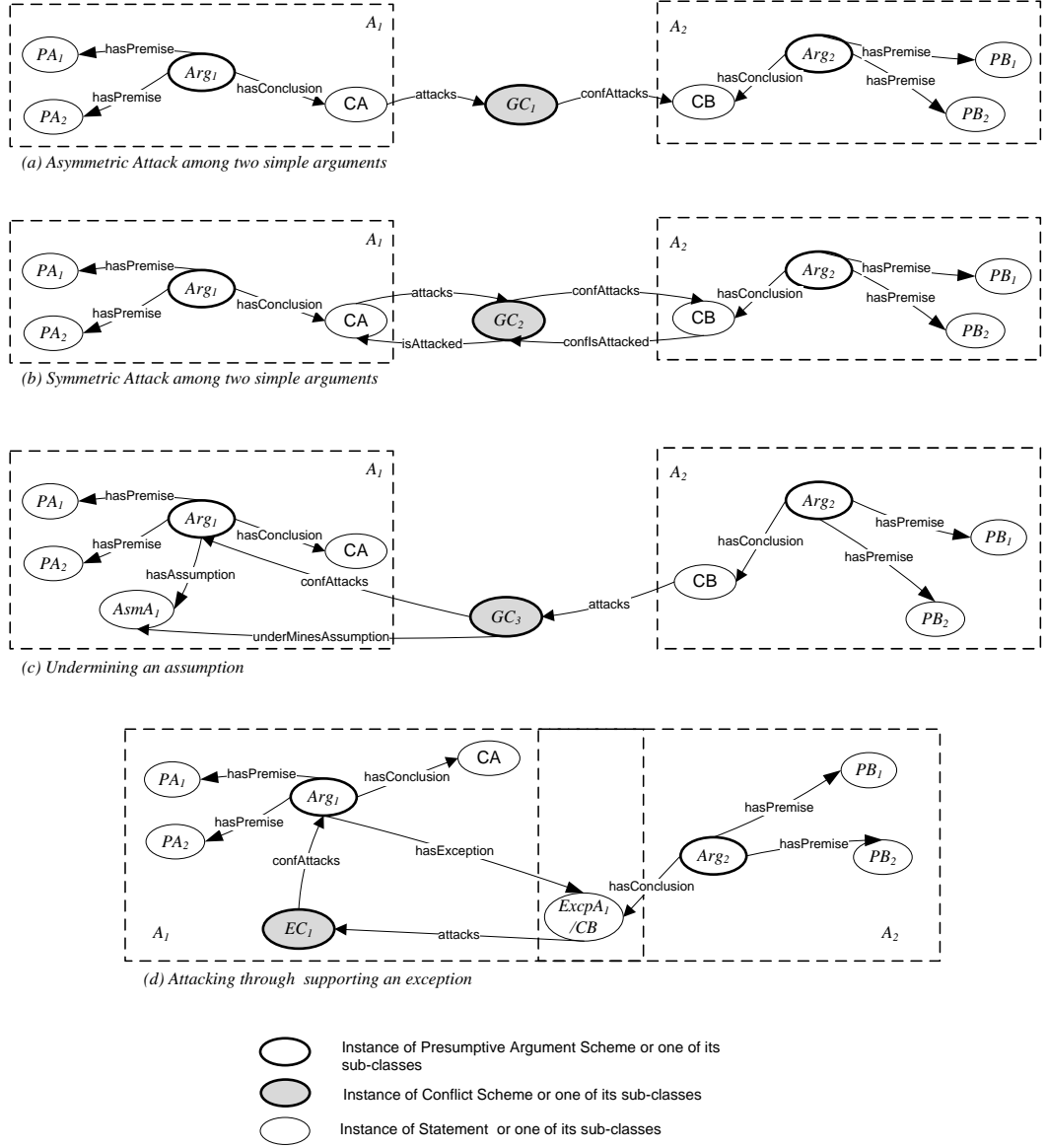


Figure 5.1: Representation of different types of attack among arguments

conclusion of argument instance A_2 (which is an instance of “argument from analogy”). An instance of “argument from popular opinion”, A_3 , has attacked argument A_1 by supporting one of its existing exceptions. For the purpose of clarity, the complete details of arguments A_2 and A_3 are not shown.

5.6 Summary

In this chapter, I presented the formalisation of the new argumentation ontology (based on a new reification of the AIF) using OWL-DL in Description Logic notation ($SHOIN(D)$). Two main concepts were identified in this ontology: *Scheme* class and the *Statement* class. The *Statement* specialisations represent the constituent parts (conclusion, premises, assumptions and exceptions) of different *Scheme* sub-classes. The conclusion and premises are defined through necessary-and-sufficient condition on a scheme while assumptions and exceptions are captured through necessary conditions. The formalisation of different types of conflicts and

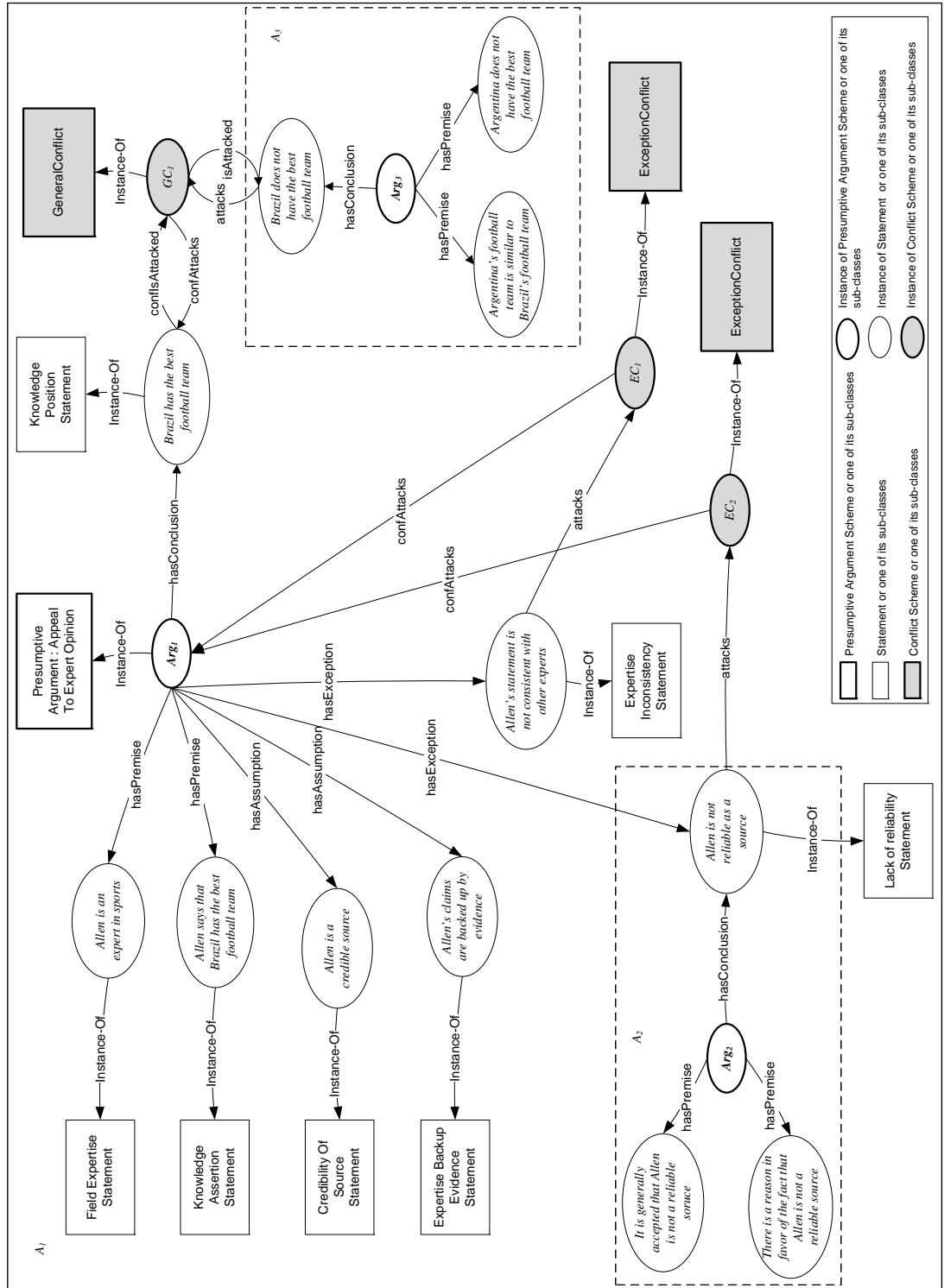


Figure 5.2: Different parts of an argument instance and its interaction with two other instances

support among arguments were also presented in this chapter.

Chapter 6

OWL Reasoning over Argument Structures

In this chapter, I describe a number of ways in which the expressive power of OWL and its support for reasoning can be used to enhance user interaction with arguments.

6.1 Inference of Indirect Support in Chained Arguments

One of the advantages of OWL over RDF Schema is that OWL supports inference over transitive properties. In other words, if $r(X, Y)$ and $r(Y, Z)$, then OWL reasoners can infer $r(X, Z)$. This can be used to enhance argument querying.

Arguments can support other arguments by supporting their premises. This results in argument *chaining* where a claim acts both as a premise of one argument and as a conclusion of another. This situation is illustrated in Figure 6.1. In Argument 1, premises PA1 and PA2 have the conclusion CA which is used at the same time as premise PB1 of the argument 2. Premises PB1 and PB2 have the conclusion CB which is used at the same time as premise PC1 of argument 3; PC1 and PC2 have the conclusion CC. Here, we can say that Argument 1 *indirectly* supports Argument 3.

A user may wish to retrieve all arguments that directly or indirectly support conclusion CC. RDF Schema does not provide straightforward support for retrieving this information. By using the transitive edge *supports*, and the description logic reasoner, small and efficient queries can retrieve the desired information.

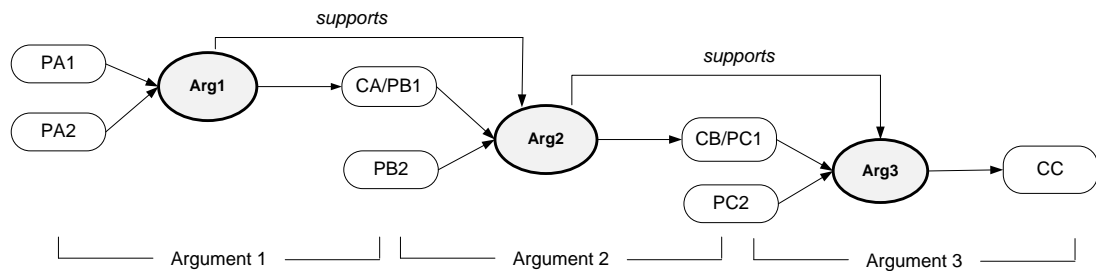


Figure 6.1: Support among chained arguments

6.2 Automatic Classification of Argument Schemes and Instances

In this section, I describe the general inference pattern behind classification of argument schemes (and their instances). This inference is based on the statement hierarchy and the conditions defined on each scheme. Two examples of this inference are also provided.

6.2.1 General Inference Pattern

Let us consider two specialisations (sub-classes) of *PresumptiveArgument* : PresScheme1 and PresScheme2. The first scheme, PresScheme1, can have an instance of CA class as its conclusion and it has premises from classes (PA1, PA2, ..., PAn). Classes CA and (PA1, PA2, ..., PAn) are specialisations of the class *Statement*. Similarly, PresScheme2 has members of CB class as its conclusion and its premises are from classes (PB1, PB2, ..., PBm) where CB and (PB1, PB2, ..., PBm) are specialisations of *Statement* and $m \geq n$.

Moreover, we assume that a relationship exists between CA and CB: They could be referring to the same class or CB is a specialisation of CA:

$$CB \equiv CA$$

Or :

$$CB \sqsubseteq CA$$

We also assume a relationship exists among the premises of these two schemes in a way that *for every* premise class of PresScheme1, there is a corresponding premise class in PresScheme2 that is either equal to or is a specialisation of that premise class in PresScheme1 (the opposite does not hold as PresScheme2 could have greater number of premises than PresScheme1).

$$PBx \equiv PAy$$

Or :

$$PBx \sqsubseteq PAy$$

Where $x = \{1, 2, \dots, m\}$ and $y = \{1, 2, \dots, n\}$

The necessary-and-sufficient conditions on PresScheme1 are defined as:

$$\begin{aligned} \text{PresScheme1} &\equiv (\text{PresumptiveArgument} \sqcap \\ &\exists \text{hasConclusion.CA} \sqcap \\ &\exists \text{hasPremise.PA1} \sqcap \\ &\exists \text{hasPremise.PA2} \sqcap \\ &\exists \text{hasPremise}(\dots) \sqcap \\ &\exists \text{hasPremise.PAn}) \end{aligned}$$

Similarly, the necessary-and-sufficient conditions on PresScheme2 are defined as:

$$\begin{aligned} \text{PresScheme2} &\equiv (\text{PresumptiveArgument} \sqcap \\ &\exists \text{hasConclusion.CB} \sqcap \end{aligned}$$

$\exists hasPremise.PB1 \sqcap$
 $\exists hasPremise.PB2 \sqcap$
 $\exists hasPremise.(...) \sqcap$
 $\exists hasPremise.PBm)$

Considering the statement hierarchy and the necessary-and-sufficient conditions defined on each class, PresScheme2 is inferred by the description logic reasoner as the sub-class of PresScheme1 in case the number of premises in PresScheme2 is greater than number of premises in PresScheme1 (i.e. $m > n$). In case the number of premises are the same (i.e. $m=n$), and at least one of the premises of PresScheme2 is a specialisation of a premise in PresScheme1 and/or the conclusion CB is a specialisation of CA, PresScheme2 is also inferred as the sub-class of PresScheme1.

6.2.2 Examples

Following the above explanation, due to the hierarchy of specialisation among different descriptors of scheme components (statements) as well as the necessary-and-sufficient conditions defined on each scheme, it is possible to infer the classification hierarchy among schemes.

Example 1. (*Inferring Appeal To Expert Opinion as the Specialisation of Argument From Position to Know*)

The necessary-and-sufficient conditions of scheme for “argument from position to know” are specified as:

$ArgFromPositionToKnow \equiv (PresumptiveArgument \sqcap$
 $\exists hasConclusion.KnowledgePositionStmnt \sqcap$
 $\exists hasPremise.PositionToHaveKnowledgeStmnt \sqcap$
 $\exists hasPremise.KnowledgeAssertionStmnt)$

Likewise, the necessary-and-sufficient conditions of “appeal to expert opinion” are:

$AppToExpertOpinion \equiv (PresumptiveArgument \sqcap$
 $\exists hasConclusion.KnowledgePositionStmnt \sqcap$
 $\exists hasPremise.FieldExpertiseStmnt \sqcap$
 $\exists hasPremise.KnowledgeAssertionStmnt)$

The following classes of declarative statements are used in definition of necessary-and-sufficient conditions on the above schemes.

$PositionToHaveKnowledgeStmnt \sqsubseteq DeclarativeStatement$
 $\text{formDescription} : \text{“E is in position to know whether A is true (false)”}$
 $KnowledgeAssertionStmnt \sqsubseteq DeclarativeStatement$
 $\text{formDescription} : \text{“E asserts that A is true(false)”}$
 $KnowledgePositionStmnt \sqsubseteq DeclarativeStatement$
 $\text{formDescription} : \text{“A may plausibly be taken to be true(false)”}$
 $FieldExpertiseStmnt \sqsubseteq PositionToHaveKnowledgeStmnt$
 $\text{formDescription} : \text{“source E is an expert in subject domain D containing proposition A”}$

Following from the statements and scheme definitions of appeal to expert opinion and argument from position to know outlined above, the reasoner infers that the former is a sub-class of the latter.

Example 2. (*Inferring the Fear Appeal Argument as the Specialisation of Argument From Negative Consequences*)

Similar inferences can be undertaken over other classes. A more elaborate example involves inferring the “fear appeal argument” scheme as sub-class of “argument from negative consequences.” Consider the specification of these two argument schemes:

Scheme 4. (*Scheme for Argument From Negative Consequences*)

- Premise: *If A is brought about, bad consequences will plausibly occur.*
- Conclusion: *A should not be brought about.*

Scheme 5. (*Scheme for The Fear Appeal Argument*)

- Fearful situation premise: *Here is a situation that is fearful to you.*
- Conditional premise: *If you carry out A, then the negative consequences portrayed in this fearful situation will happen to you.*
- Conclusion: *You should not carry out A.*

The necessary-and-sufficient conditions of the “argument from negative consequences” are detailed as:

$$\begin{aligned} ArgNegativeConseq &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.ForbiddenActionStmnt \sqcap \\ &\exists hasPremise.BadConsequenceStmnt) \end{aligned}$$

Likewise, the necessary-and-sufficient conditions of the “fear appeal argument” are detailed as:

$$\begin{aligned} FearAppealArg &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.ForbiddenActionStmnt \sqcap \\ &\exists hasPremise.FearfulSituationStmnt \sqcap \\ &\exists hasPremise.FearedBadConsequenceStmnt) \end{aligned}$$

The statements are defined below. Note that the “Feared Bad Consequence” statement is a specialization of “Bad Consequence” statement, since it limits the bad consequence to those portrayed in the fearful situation.

$$\begin{aligned} BadConsequenceStmnt &\sqsubseteq DeclarativeStatement \\ \text{formDescription} &: \text{“If A is brought about, bad consequences will plausibly occur”} \\ ForbiddenActionStmnt &\sqsubseteq DeclarativeStatement \\ \text{formDescription} &: \text{“A should not be brought about”} \\ FearfulSituationStmnt &\sqsubseteq DeclarativeStatement \\ \text{formDescription} &: \text{“Here is a situation that is fearful to you”} \\ FearedBadConsequenceStmnt &\sqsubseteq BadConsequenceStmnt \\ \text{formDescription} &: \text{“If you carry out A, then the negative consequences portrayed in this fearful situation will happen to you”} \end{aligned}$$

As a result of classification of scheme hierarchies, instances belonging to a certain scheme class will also be inferred to belong to all its super-classes. For example, if the user queries to return all instances of “argument from negative consequences,” the instances of all specializations of the scheme, such as all argument instances from “fear appeal arguments” are also returned.

6.3 Inferring Critical Questions

In this section I describe the general inference pattern behind inference of critical questions from an argumentation scheme’s super-classes and provide an example.

6.3.1 General Inference Pattern

In the previous section I described an assumption about two specialisations of *PresumptiveArgument*, PresScheme1 and PresScheme2 and the fact that PresScheme2 was inferred to be the sub-class of PresScheme1. Each of these schemes might have different assumptions and exceptions defined on their classes. For example, PresScheme1 has AsmA1 and AsmA2 as its assumptions and ExcA1 as its exception. PresScheme2 has AsmB1 and ExcB1 as its assumption and exception respectively. AsmA1, AsmA2, AsmB1, ExcA1 and ExcB1 are specialisations of *Statement* class. The the necessary conditions defined on classes PresScheme1 and PresScheme2 are:

```

PresScheme1  $\sqsubseteq$   $\exists$ hasAssumption.AsmA1
PresScheme1  $\sqsubseteq$   $\exists$ hasAssumption.AsmA2
PresScheme1  $\sqsubseteq$   $\exists$ hasException.ExcA1

PresScheme2  $\sqsubseteq$   $\exists$ hasAssumption.AsmB1
PresScheme2  $\sqsubseteq$   $\exists$ hasException.ExcB1

```

Since PresScheme2 has been inferred by the reasoner as the specialization (sub-class) of PresScheme1, a query to the system to return all assumptions and exceptions of PresScheme2, is able to return all those explicitly defined on the scheme class (i.e. AsmB1 and ExcB1) as well as those defined on any of its super-classes (in this case: AsmA1, AsmA2 and ExcA1).

6.3.2 An Example

As explained through the above example, since the schemes are classified by the reasoner into a hierarchy, if certain assumptions or exceptions are not explicitly stated for a specific scheme but are defined on any of its super-classes, the system is able to infer and add those assumptions and exceptions to instances of that specific scheme class. Since critical questions enable evaluation of an argument, inferring additional questions for each scheme will enhance the analysis process.

Consider the critical questions for “fear appeal argument” and “argument from negative consequences” described below.

Critical Questions 4. (*Critical Questions for Fear Appeal Argument*)

1. *Should the situation represented really be fearful to me, or is it an irrational fear that is appealed to?*
2. *If I don't carry out A, will that stop the negative consequences from happening?*
3. *If I do carry out A, how likely is it that the negative consequences will happen?*

Critical Questions 5. (*Critical Questions for Argument From Negative Consequences*)

1. *How strong is the probability or plausibility that these cited consequences will (may, might, must) occur?*
2. *What evidence, if any, supported the claim that these consequences will (may, might, must) occur if A is brought about?*
3. *Are there consequences of the opposite value that ought to be taken into account?*

These critical questions are represented in the ontology through the following statements:

```

IrrationalFearAppealStmnt  $\sqsubseteq$  DeclarativeStatement
formDescription : "It is an irrational fear that is appealed to"

```

PreventionOfBadConsequenceStmnt \sqsubseteq *DeclarativeStatement*

formDescription : “If A is not carried out, this will stop the negative consequences from happening”

OppositeConsequencesStmnt \sqsubseteq *DeclarativeStatement*

formDescription : “There are consequences of the opposite value that ought to be taken into account”

StrongConsequenceProbabilityStmnt \sqsubseteq *DeclarativeStatement*

formDescription : “There is a strong probability that the cited consequences will occur.”

ConsequenceBackUpEvidenceStmnt \sqsubseteq *DeclarativeStatement*

formDescription : “There is evidence that supports the claim that these consequences will occur if A is brought about.”

The necessary conditions on “argument from negative consequences” that define these critical questions are :

ArgNegativieConseq \sqsubseteq \exists *hasException.OppositeConsequencesStmnt*

ArgNegativieConseq \sqsubseteq \exists *hasAssumption.StrongConsequenceProbabilityStmnt*

ArgNegativieConseq \sqsubseteq \exists *hasAssumption.ConsequenceBackUpEvidenceStmnt*

Likewise, the necessary conditions on “fear appeal argument” are:

FearAppealArg \sqsubseteq \exists *hasException.IrrationalFearAppealStmnt*

FearAppealArg \sqsubseteq \exists *hasAssumption.PreventionOfBadConsequenceStmnt*

FearAppealArg \sqsubseteq \exists *hasAssumption.StrongConsequenceProbabilityStmnt*

“Fear appeal argument” is classified as a sub-class of “argument from negative consequences.” The critical questions 2 and 3 of “argument from negative consequences” have not been explicitly defined on “fear appeal argument”, but can be inferred through reasoning.

6.4 Summary

In this chapter, I described three different ways in which the expressive power of OWL and its support for reasoning can be used to enhance user interaction with arguments. The first section showed how adding a transitive property among chained arguments can enhance queries to return arguments that directly or indirectly support another argument in a chain. The second section illustrated how the hierarchy of specialisation among statements as well as the necessary-and-sufficient conditions defined on each scheme, enables the inference of hierarchy of argument schemes and facilitates classification of argument instances. The last section detailed the process of inference of additional critical questions for each scheme due to the scheme hierarchy and the necessary conditions defined on them.

Chapter 7

Implementation

In this chapter, I explain the basic architecture of the implemented Web-based system *Avicenna*.¹ A comparison among different tools/technologies for building this system as well as the reasons for choosing each tool/technology is also provided. Moreover, the main features of this system are highlighted and briefly explained.

7.1 Basic System Architecture

The basic system architecture is illustrated in Figure 7.1. It consists of three main tiers: the data tier, the middle tier and the client tier. The argumentation ontology (including both the TBox and the ABox) is stored in form of RDF statements (triples) in the back-end database which constitutes the data tier .

The middle tier is made up of two main components: the first component is the application server which contains the database driver, the RDF Repository API and the description logic reasoner; the second component is the Web server which includes all the server side technologies (e.g. Java Servlets, JSTL, etc.) and is able to generate static content (HTML) from dynamic pages (e.g. JavaServer Pages). These two components reside on the same server.

In the application server, with the help of the RDF Repository API and the appropriate database driver, the stored RDF statements in the database are uploaded to the statement repository. Description logic reasoner, through collaborating with the RDF Repository API, uses these asserted statements (triples) to infer additional statements which are then added to the statement repository.

The website acts as the application interface of the ontology and is hosted on a Web server. The Web server receives the RDF statements (results) from the RDF Repository API and uses the server side technologies and the dynamic pages to produce the static pages with desired content and sends these pages to the client browser.

The client tier contains the static pages generated by the Web server as well as the client side scripts. These scripts are run by the browser and mainly used to check for and limit user errors before the pages are posted back to the server. While users browse the site and request to view (or author) different statements, the request parameters are sent back to the Web server which in turn communicates these parameters to the RDF Repository API. The appropriate queries are constructed and the results are reported back to the Web server by the RDF Repository

¹Avicenna was a Persian polymath and the foremost physician and Islamic philosopher of his time. He developed an early theory on hypothetical syllogism, which formed the basis of his early risk factor analysis. In addition to developing an early theory on propositional calculus and an original theory on temporal modal syllogism, he also developed his own system of logic known as “Avicennian logic” as an alternative to Aristotelian logic. Avicenna also contributed inventively to the development of inductive logic, being the first to describe the methods of agreement, difference and concomitant variation which are critical to inductive logic and the scientific method. (Source: Wikipedia)

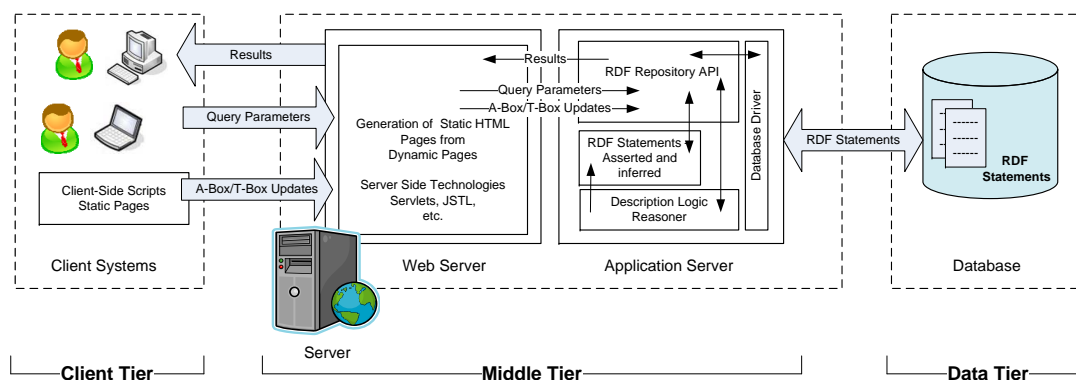


Figure 7.1: Basic System Architecture

API. Using these results, the Web server generates pages with requested content and sends them to client's browser.

The users might update the ABox by adding new arguments or through supporting or attacking existing arguments. They might also update the TBox by introducing new argument schemes that extend the underlying ontology. These updates are transferred by the Web Server to the RDF Repository API which in turn updates the back-end database.

7.2 Tools Used

7.2.1 Ontology Editor

Ontology editors are applications designed to assist in the creation or manipulation of ontologies. A Number of different ontology editors are available. Some editors are specialized in certain domains; an example is OBO-Edit² which is focused on domain of molecular biology. Other editors such as WebODE³ and Ontolingua⁴ are not an isolated tool for the development of ontologies; rather, they are offered in form of Web services, accessible to users through their Web browsers.

A number of stand-alone tools exist for design and development of ontologies. Protégé⁵ is by far the most widely used open source tool. It is built on Java, and provides an extensible platform that contains a suite of tools to construct domain models and knowledge-based applications with ontologies (i.e. RDFS, OWL). It provides a DIG⁶ interface with available description logic reasoners to facilitate reasoning tasks for OWL DL ontologies. SPARQL [35] queries can be authored and run from the environment. Different third party plug-ins have been developed for this editor. OWLViz⁷ and GraphViz⁸ plug-ins can be used for visualisation of the ontology. Protégé has an extensive community of users and forums.

OntoStudio⁹ is another powerful editor based on KAON2 [29] utilizing its built-in description logic reasoner. TopBraid Composer¹⁰ is another editor with advanced features which is very similar to Protégé. However, both of these editors are only available through commercial licenses. Other non-commercial based editors like CampTools editor¹¹ exist, but none can match

²<http://geneontology.sourceforge.net/>

³<http://webode.dia.fi.upm.es/WebODEWeb/index.html>

⁴<http://www.ksl.stanford.edu/software/ontolingua/>

⁵<http://protege.stanford.edu/>

⁶<http://dl.kr.org/dig/>

⁷<http://www.co-ode.org/downloads/owlviz/>

⁸<http://www.graphviz.org/>

⁹http://www.ontoprise.de/content/e1171/e1249/index_eng.html

¹⁰<http://www.topbraidcomposer.com/features.html>

¹¹<http://cmap.ihmc.us/coe/>

the extensive features and community support of Protégé. Therefore, Protégé was chosen as the ontology editor for this project. Comparisons among different ontology editors reported by Gomez-Perez et al [21] also highlights the advanced features of Protégé as the editor of choice.

7.2.2 RDF Repository (Semantic Web Framework)

RDF Repositories provide a framework for storage, update, inferencing and querying of RDF data. Jena¹² is an open source Semantic Web framework and grown out of work with the HP Labs Semantic Web Programme. It provides a programmatic environment in Java for RDF, RDFS and OWL and includes a rule-based inference engine environment for manipulating the ontology model. ARQ¹³ is an implementation of SPARQL[35] query language for Jena. ARQ libraries provide programmatic API access to issue SPARQL queries to the underlying ontology. Jena provides different levels of reasoning facilities as well as enabling external description logic reasoner plug-ins through DIG or direct library calls. Jena also provides persistent storage of RDF data in relational databases.

Another powerful RDF Repository is Sesame.¹⁴ Similar to Jena, it is an open source Java based RDF repository and provides persistent storage of RDF data in a back-end database. The query language used in Sesame is SeRQL[10]. A limitation of this tool is that the only available reasoner plug-in for Sesame is OWLIM.¹⁵ The expressive power and reasoning capabilities of OWLIM are inferior to OWL-DL.

Corese¹⁶ is another Semantic Web framework which offers a programmatic API to manipulate and query ontologies using SPARQL; however, it is not able to incorporate a description logic reasoner at the current time. Since the core ontology of this project is based on OWL-DL, and needs to utilize richer capabilities of description logic reasoners, Jena was selected as the suitable choice of RDF repository.

7.2.3 Description Logic Reasoner

Since Description Logics are provided with a formal syntax and formal model-theoretic semantics, sound and complete reasoning algorithms can be formulated to infer additional facts about the underlying ontology. FaCT++,¹⁷ KAON2[29], Pellet[44] and RacerPro¹⁸ are four of the most widely used OWL/DIG reasoners. A brief description of these reasoners is provided next.¹⁹

FaCT++ is a free open-source C++ based reasoner for $\mathcal{SHOIQ}(D)$ with simple data types. It implements a tableau-based decision procedure for general TBoxes and incomplete support of ABoxes(retrieval).

KAON2 is a free Java reasoner for $\mathcal{SHIQ}(D)$ and implements a resolution-based decision procedure for general TBoxes and ABoxes.

Racer Pro also provides a reasoner for $\mathcal{SHIQ}(D)$ with simple data types; it is lisp-based and is available under commercial license. Similar to FaCT++, it implements a tableau-based decision procedure for general TBoxes and ABoxes.

Pellet is a free open-source Java-based reasoner for $\mathcal{SHOIN}(D)$ with complete support of all OWL-DL and has recently been extended to $\mathcal{SROIN}(D)$ to support the new proposed features of OWL 1.1[30]. Similar to FaCT++ and Racer Pro, it implements a tableau-based decision procedure and supports general TBoxes and ABoxes. It also provides other features

¹²<http://jena.sourceforge.net/>

¹³<http://jena.sourceforge.net/ARQ/>

¹⁴<http://www.openrdf.org/>

¹⁵www.ontotext.com/owlim/

¹⁶<http://www-sop.inria.fr/edelweiss/wiki/wakka.php?wiki=Corese>

¹⁷<http://owl.man.ac.uk/factplusplus/>

¹⁸<http://www.sts.tu-harburg.de/f.f.moeller/racer/>

¹⁹<http://www.cs.man.ac.uk/~sattler/reasoners.html>

such as Ontology Analysis and Repair, Ontology Debugging, etc.

Extensive comparison among these reasoners is beyond the scope of this thesis; a benchmarking for these reasoners is reported by Gardiner et al [20]. Considering the level of expressivity and support provided by each reasoner, and the fact that Jena group recommends Pellet as the reasoner of choice (due to its robustness and scalability), Pellet was chosen as the description logic reasoner.

7.2.4 Core Webiste

With decisions about Semantic Web framework (RDF repository) and description logic reasoner in place, the choice of the most suitable technology to build the Website had to be consistent with and supportive of those decisions. Since both Jena and Pellet are based on Java, JEE²⁰ technologies were chosen to build the system. Using Java technologies allows direct interaction with both Jena and Pellet libraries with no special plug-ins required. The following JEE technologies were used in building the Website.

JavaServer Pages²¹ (JSP) technology provides a way to create dynamic web content using Java libraries and a technique called Scriptlets. It also enables development of Web based applications that are server and platform independent.

Servlet²² is a Java technology based Web component, managed by a container, that contains the application logic and generates dynamic content. Like other Java-based components, servlets are platform independent Java classes and are run by a Java enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container.

JavaServer Pages Tag Library²³ (JSTL) encapsulates as simple tags the core functionality common to many Web applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.

Java Database Connectivity²⁴ (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of SQL databases.

7.2.5 Web Server

A Web server is a program that responds to an incoming TCP connection and provides a service to the requester. There are many varieties of Web server software that support hosting a Java based Website. Tomcat²⁵ is a free Web server from Apache. It is a JSP/Servlet container, supporting numerous large-scale Web applications. Java Application Server²⁶ is not *just* a JSP/Servlet container; it is a full JEE server providing support for different layers of JEE technologies while providing a more robust Http server than Tomcat. Due to its advanced features, Java Application Server was used as the Web server in this project.

²⁰<http://java.sun.com/javaee/>

²¹<http://java.sun.com/products/jsp/>

²²<http://java.sun.com/products/servlet/>

²³<http://java.sun.com/products/jsp/jstl/>

²⁴<http://java.sun.com/javase/technologies/database/>

²⁵<http://tomcat.apache.org/>

²⁶<http://www.sun.com/software/products/appsrvr/index.xml>

7.2.6 Database

Jena facilitates creating and accessing persistent models on different databases such as MySQL,²⁷ HSQLDB,²⁸ Apache Derby,²⁹ PostgreSQL,³⁰ Oracle,³¹ and Microsoft SQL Server.³² HSQLDB and Apache Derby are more suited for small scale persistence for a single application. The other databases, as explained by Jena, all work efficiently to store and access RDF triples.

SQL Server is an enterprise level relational database management system which provides secure and reliable storage for data and provides a solid back-end for different Web based applications. It also provides services such as reporting, analysis, integration and notification. In this project, SQL Server 2005 was chosen as the back-end database for three main reasons; firstly, it is a secure and reliable database for Web based applications; secondly, as the system is hosted on a Windows platform, SQL Server provides a more stable back-end which can benefit from integrated features of the operating system for different types of services used; and the last reason was my long-time familiarity with this product which helped me in managing and fine-tuning the database to achieve optimum results while interacting with Jena.

7.2.7 Client-Side Scripting

A client-side script is a program that may accompany an HTML document and executes on the client's machine (Web browser). Scripts offer developers a means to extend HTML documents in highly active and interactive ways; for example, scripts may accompany a form to process input as it is entered. In this way, they ensure that input data conforms to predetermined ranges of values, that fields are mutually consistent, etc. Common scripting languages include JavaScript³³ and VBScript.³⁴

Both JavaScript and VBScript are powerful scripting languages; however, unlike VBScript, JavaScript is supported by almost all browsers. Therefore, JavaScript was selected as the scripting language.

7.2.8 Style Sheets

Cascading Style Sheets³⁵ is a W3C Candidate Recommendation. It is a stylesheet language used to describe the presentation of a document written in a markup language. By separating the presentation style of documents from the content of documents, CSS simplifies Web authoring and site maintenance. Its most common application is to style Web pages written in HTML and XHTML, but the language can be applied to any kind of XML document. CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice, etc. CSS is used in this project to define presentation style of JavaServer Pages.

7.3 System Features

Avicenna is a Web-based system that exploits the OWL ontology explained in Chapters 5 and 6 to enable creating arguments using different argumentation schemes and query the argument network using SPARQL. Manipulation of existing arguments is also handled through re-using existing claims and attacking/supporting available arguments. Several techniques (such as

²⁷<http://www.mysql.com/>

²⁸<http://hsqldb.org/>

²⁹<http://db.apache.org/derby/>

³⁰<http://www.postgresql.org/>

³¹<http://www.oracle.com/>

³²<http://www.microsoft.com/sql/default.aspx>

³³http://developer.mozilla.org/en/docs/About_JavaScript

³⁴<http://msdn2.microsoft.com/en-us/library/t0aew7h6.aspx>

³⁵<http://www.w3.org/Style/CSS/>

transaction-based database updates, error handling through scripts, etc) are utilized to ensure the integrity and consistency of the argument repository is maintained through different operations. In this section, the main features of this system are explained.

7.3.1 System Users

Users of the system are categorized under three main profiles : *Guests*, *Authors* and *Administrators*. Each profile has a predefined set of user credentials. Users who use the system as *Guests* are able to view the existing arguments (as well as any attacking or supporting arguments they might have) and search for any argument or scheme information they require. *Authors*, in addition to viewing and searching for arguments are able to author new arguments and/or attack or support existing arguments. *Administrators* have all the *Author* credentials and are also able to extend the ontology by adding new argument schemes. The navigation links and contents of the Web pages are implemented in a way to reflect these credentials.

New users can sign up for accounts from the system interface. The default profile assigned to each new user is the *Author* profile; obtaining administrative credentials is only possible through direct contact with the system administrator. While registering new users, validation checks are performed by the system and the appropriate error messages are displayed in case fields are empty, password has not been retyped correctly, the user name is already in use and so on.

7.3.2 Listing available arguments

The system lists the available arguments by listing their titles as displayed in Figure 7.2. These titles are in form of hyperlinks and can be used to navigate to a page where the details of the argument (its scheme, author, conclusion, premises and critical questions) are listed for further exploration. Figure 7.3 displays the details of argument ‘Tipping lowers self esteem’ which is an instance of “argument from expert opinion.”

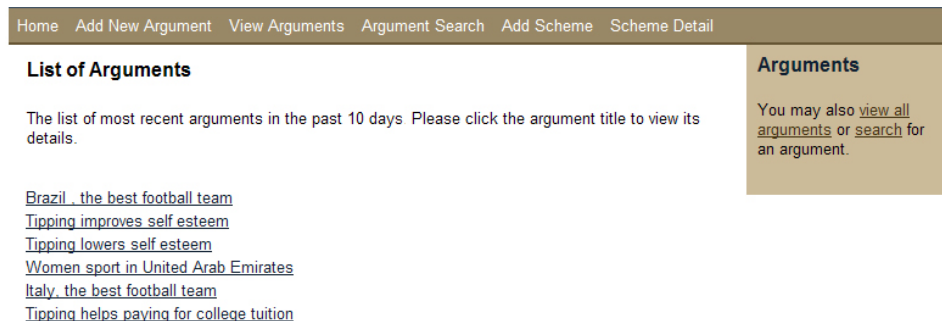


Figure 7.2: List of arguments

By choosing *View Arguments* from main navigation menu, the user can view the most recent threads of arguments (from the past 10 days and up to the current date) in the database. The user can also view the entire list of arguments or a specific set as filtered through different search modes (see Section 7.3.9). The last two options are available through a menu on the right hand side of the page.

7.3.3 Creation of new arguments

New semantically annotated arguments can be authored using new claims or claims already existing as part of available arguments in the system. By selecting *Add New Argument* from the navigation menu, the user has the choice of creating a new argument that adheres to any of the existing argument schemes. Figure 7.4 shows the list of available schemes in the system.

Avicenna
 Argumentation in OWL

[Home](#) [Add New Argument](#) [View Arguments](#) [Argument Search](#) [Add Scheme](#) [Scheme Detail](#)

Details of argument Tipping lowers self esteem

This argument is an instance of Appeal to Expert Opinion and has been created by Mona Haidar.

Premise:

Dr. Phil says that tipping lowers self esteem

Premise:

Dr. Phil is an expert in Psychology

Assumption:

Dr. Phil is a credible expert source

Assumption:

Dr. Phil assertion is backed up by evidence

Exception:

The fact that tipping lowers self esteem is not consistent with other expert assertions

Exception:

Dr. Phil is not an honest source

Scheme:

Appeal to Expert Opinion

Conclusion:

Tipping lowers self esteem

Legend

- ✔ Support
- ✖ Asymmetric Attack
- ✖ Symmetric Attack
- ✖ Use Exception to Attack
- ✖ Undermine Assumption
- 🔍 View Attacks
- 🔍 View Supporting Arguments
- 🔍 View Attacks Through Exceptions
- 🔍 View Attacks on Assumptions

About This Project | Argumentation | Walton Schemes | ArgDE | British University in Dubai | Contact Us
Copyright © 2007. All Rights Reserved.

Figure 7.3: Argument Details

After a specific argument scheme is chosen by the user, its constituent parts (the conclusion, the premises, the assumptions and exceptions) are extracted by running a query on the different restrictions defined on the scheme. If certain assumptions or exceptions are not explicitly defined on a scheme but are defined on any of its deduced super-classes, the description logic reasoner is able to infer them and add them to the list of asserted assumptions and exceptions. Since Pellet treats the anonymous restrictions defined in an OWL ontology as syntactic expressions and does not return them as answers to any query,³⁶ a work-around to solve this problem was implemented to query the default instance of each scheme class, instead of querying the class itself.

The user is then forwarded to a page with a form containing place holders for the different parts of the argument to be filled. Beside each place holder, a brief description of the claim format is provided. The textual contents of assumptions and exceptions are already filled in respective placeholders and only require minimal change by the user. Figure 7.5 illustrates the page for authoring a new argument instance of “fear appeal argument.”

The user may enter new claims or may choose to use any of the existing claims. He can access the existing claims by clicking a link to access a page (see Figure 7.6) that displays all the

³⁶<http://pellet.owldl.com/faq/different-results/>

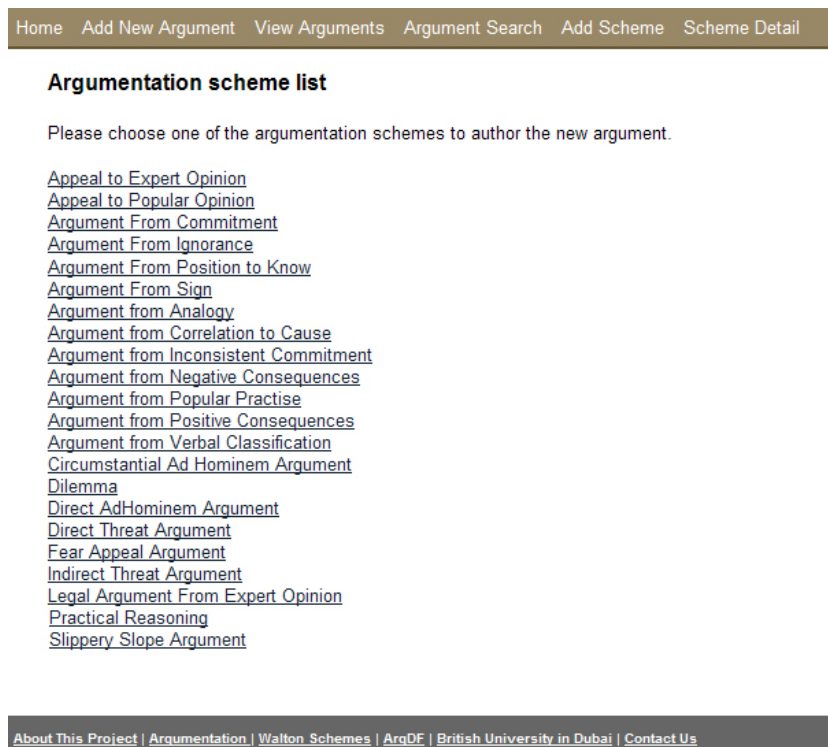


Figure 7.4: List of argument schemes

Figure 7.5: Adding New Arguments

available claims in the system. This list can be searched and filtered as required. The paging technique is implemented (through code) to limit the number of claims displayed in the page at any given time.

Validation checks on the page are available to make sure the different parts of an argument are filled before the user confirms addition of the new argument to the system. If validation check

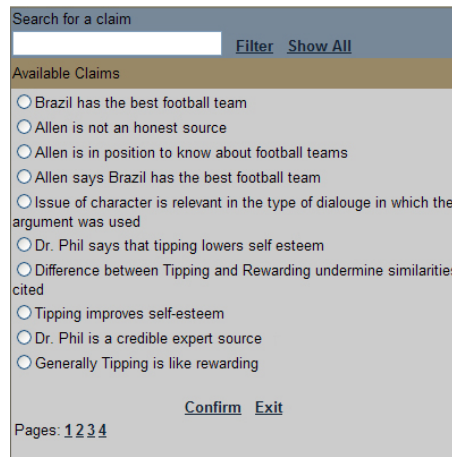


Figure 7.6: Re-using existing claims

passes, a module is ran to update the system with the new instance. The argument counter is increased and its value is used as part of the new name that is used to create the appropriate URI for the new instance. The new argument instance is created under the appropriate scheme class and is annotated with details such as author, date of creation, title, etc.

The instances of the conclusion and the premises are created under the appropriate statement classes and linked to the argument instance through *hasConclusion* and *hasPremise* properties. Instances of assumptions and exceptions (if available) are also created under the appropriate statement classes. Instances of assumptions represent a set of implicit premises of the argument and are connected to the argument instance through *hasAssumption* property. Instances of exceptions are connected to the argument instance through instances of *ExceptionConflict* and *hasException* property; such exceptions will not undercut the presumptive argument instance unless they are supported by further statements.

7.3.4 Attacking/Supporting existing arguments

Users can attack or support existing arguments. The available operations on each claim making up the argument are accessible through different icons on the right side of each claim as illustrated in Figure 7.3. Users can perform symmetric or asymmetric attacks (see Section 3.3) on a conclusion or a premise.

Users can also choose to support the conclusion or premises of an argument. As explained in Section 6.1, if the supported claim is the premise of an argument, this claim is both the conclusion of the supporting argument and the premise of the supported argument; thus creating a chain of arguments. Users can also undercut (see Section 3.3) an argument by undermining an existing assumption or supporting an existing exception of an argument.

After the user has specified the claim and the type of action (attack or support) he wants to perform on the argument, he is forwarded to the next step to add a new argument in order to facilitate the desired action. Creation of the argument follows the same routine explained in the previous section: the users are presented with a list of argument schemes to choose from and then fill out the conclusion, premises, assumptions and exceptions that adhere to that scheme. If the user has chosen to support a claim, or support an exception to create a new attack on the argument, that claim will automatically be designated as the conclusion of the argument and can not be changed. Upon confirming the addition of the new argument, a new argument instance is added under the related scheme class as explained in previous section; moreover, in case of attacks, the appropriate conflict related properties (*attacks*, *isAttacked*, *confAttacks*, *confIsAttacked* and *underMinesAssumption*) or in case of support, the (*supports*) property are used to associate the original argument and the newly created instance according to Figures

5.1 (a to d) and 6.1.

As the system allows re-using of existing claims while authoring new arguments as well as supporting and attacking different claims that are part of an existing argument, interlinked and dynamic argument networks are created - a central feature provided by the underlying ontology design.

7.3.5 Retrieving attacking/supporting arguments of a claim

Viewing different types of attacks on a premise or conclusion is also available through set of icons displayed on the right hand side of the claim. Users can view the different claims that are attacking / or being attacked by the claim and then view the arguments that those claims are part of. When the user chooses to view the attacks made against a certain claim, he has the option of choosing among three different options: to view the claims that this claim attacks, or the claims that attack this claim and finally the claims that attack this claim and are attacked by it at the same time (symmetric attack). These options are illustrated in Figure 7.7.

Conflict Categories

Please choose the type of conflict between "Brazil has the best football team" and other conflicting claims:

[The claim attacks these claims](#)
[The claims that attack this claim](#)
[Symmetric Attacks](#)

Figure 7.7: Conflict Category

The result of choosing any of the options is displaying the list of claims that are involved in a conflict relationship with the initial claim. Figure 7.8 shows the list of claims that are attacking the claim 'Brazil has the best football team.' By clicking each claim, the users can view the arguments in which the claim is utilized.

Claims List

You may click each claim to view the arguments it is utilized in.

These claims attack the claim "Brazil has the best football team":

[Italy has the best football team](#)
[England has the best football team](#)

Figure 7.8: Claims attacking 'Brazil has the best football team'

Users can also query the network to view different arguments that support a certain premise or conclusion; the transitivity feature of the *supports* property is used to return all the arguments supporting the claim whether directly or indirectly (as depicted in Figure 6.1). Figure 7.9 shows the arguments that support the claim 'Tipping lowers self esteem'; the argument 'Dr. Phil is one of the best Psychiatrists' indirectly supports the claim (since it supports an argument which in turn supports a premise of the argument with conclusion 'Tipping lowers self esteem').

The undercut attacks on each argument (through undermining assumptions or supporting exceptions) can also be queried in the same way.

7.3.6 Retrieving scheme details

Users can view the details of each of the existing scheme classes by accessing *Scheme Detail* from the navigation menu. Figure 7.10 illustrates the details of "appeal to expert opinion." A

List of Arguments

Arguments supporting "Tipping lowers self esteem"

Please click the argument title to view its details.

[Dr. Phil is one of the best Psychiatrists](#)

[Tipping lowers self esteem](#)

[Research shows that tipping lowers self esteem](#)

Figure 7.9: Claims supporting 'Tipping lowers self esteem'

number of queries are used to retrieve the information which consists of the scheme conclusion, premises, asserted assumptions and exceptions as well as its inferred super-classes and sub-classes in the scheme hierarchy.

Home Add New Argument View Arguments Argument Search Add Scheme Scheme Detail

Scheme Details of Appeal To Expert Opinion

Conclusion Descriptor:
A may plausibly be taken to be true(false)

Premise Descriptor(s):
Source E is an expert in subject domain D containing proposition A.
E asserts that A is true (false)

Asserted Assumption Descriptor(s):
E assertion is backed up by evidence
E is a credible expert source

Asserted Exception Descriptor(s):
E is not an honest source
A is not consistent with other expert assertions

Superclass(es):
Argument From Position to Know

Subclass(es):
Legal Argument From Expert Opinion

[About This Project](#) | [Argumentation](#) | [Walton Schemes](#) | [ArgDE](#) | [British University in Dubai](#) | [Contact Us](#)

Figure 7.10: Scheme Details of Argument From Position to Know

7.3.7 Displaying the scheme hierarchy and the statement hierarchy

The system is also able to display the inferred scheme hierarchy (see Figure 7.11) and the inferred statement hierarchy (see Figure 7.12 for a partial snapshot of the hierarchy). As can be seen from both figures, Pellet infers *Nothing* (or \perp) to be the sub-class of all classes.

7.3.8 Creation of new schemes

Users with administrative privileges are able to extend the underlying ontology by adding new argument scheme classes (accessible through *Add Scheme* option on the navigation menu). In creation of the new scheme definition they might re-use existing statement classes, or define new ones as specialisation or generalization of existing statement classes.

Addition of the new scheme is carried out in several stages. At the first stage (Figure 7.13) the user is asked to enter a name for the scheme. A validation check is performed to make sure the name is unique and not existing in the database. As an example, a new argument scheme named "Argument from Appearance" [55] will be added to the repository.

Classification Hierarchy of Presumptive Argument

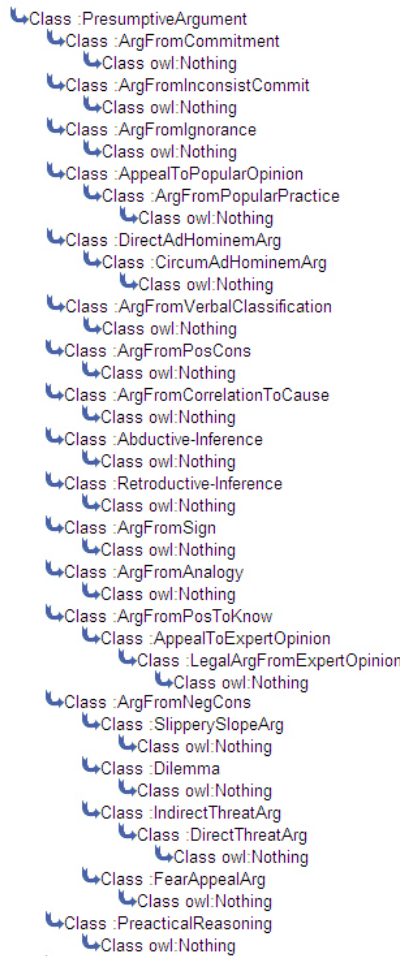


Figure 7.11: Presumptive Argument Hierarchy

In the next stage the user should define the conclusion of the argument scheme. As explained earlier, the user has the choice of utilizing any of the already defined statements, or defining a new one. In case of “argument from appearance,” the conclusion statement is already available in the system and does not need to be created. Figure 7.14 depicts this stage.

In the subsequent stage the user should define the premises. “Argument from appearance” has a single premise which is not defined in the system. Therefore, a new statement class should be created and its name, description, sub-classes and super-classes should be defined accordingly. To define all the sub-classes (or super-classes) of a new statement class, a list of available statements are provided. The user can check all the statements that are specialisations (or generalizations) of the new statement. Figure 7.15 shows this process for defining sub-classes (the same process is used for defining super-classes). In this example, this new statement class does not have any specialisations or generalizations. Therefore the user only defines the name and the description. This is illustrated in Figure 7.16.

In the next stages, the user is presented with forms to create the assumptions and exceptions. “Argument from appearance” contains two exceptions (and no assumptions) which are defined using the exceptions forms.

At each stage, the system enables the user to edit/delete his previous choices or add a new statement class. The user can access his previous choices through a small menu displayed on the right side of each form. Figure 7.17 shows this menu from exception definition form. Figure

Classification Hierarchy of Declarative Statement

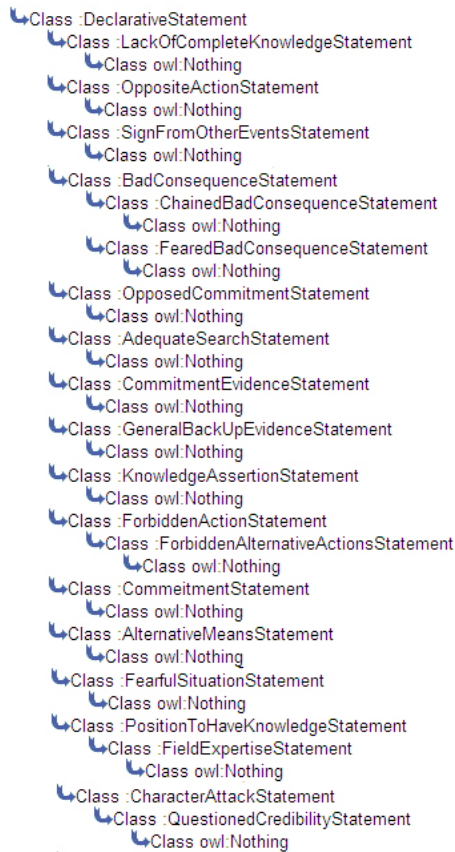


Figure 7.12: Declarative Statement Partial Hierarchy

Defining a new scheme

The process of adding a new scheme consists of 6 steps:

- Step 1: Defining the new scheme name
- Step 2: Defining the conclusion of the new scheme
- Step 3: Defining the premise(s) of the new scheme
- Step 4: Defining the assumption(s) of the new scheme
- Step 5: Defining the exception(s) of the new scheme
- Step 6: Review and Confirmation

At different stages of the process, it is possible to edit/delete or add to the previous choices made by accessing the menu of previous choices on the top right of the page.

Step 1 of 6 - Defining the scheme name

New Scheme Name: [Go To Next Stage](#)

Figure 7.13: First stage in creating a new scheme

7.18 shows this menu in a larger view.

After the user has finished defining different parts of the argument, a summary of those parts are displayed as can be seen in Figure 7.19. He also has the ability to edit/delete any of his previous options from this screen or extend the definition of the argument by adding new parts (statements).

Defining new scheme : Argument From Appearance

Step 2 of 6 - Defining the conclusion of the new scheme

To define the conclusion of the new scheme, you may choose any of the existing statements, or define a new statement. When defining a new statement, you need to specify:

1. The name (for example: Forbidden Action Statement)
2. The description (for example: A should not be brought about)
3. The superclasses of the statement; i.e. the statements that are generalizations of this statement
4. The subclasses of the statement; i.e. the statements that are specializations of this statement

☒ Use Existing
Contains Property Statement

☐ Create New
Statement Name
Statement Description

The Conclusion's super classes are :

☐ Lack Of Complete Knowledge Statement - The knowledge is not complete enough to support the argument adequately

☐ Opposite Action Statement - A (Undesired Action) is the converse of C (Desired Action)

☐ Sign From Other Events Statement - There are other events that more reliably account for the sign

Figure 7.14: Defining the conclusion of a new scheme using an existing statement

The Premise's sub classes are :

☒ Lack Of Complete Knowledge Statement -- The knowledge is not complete enough to support the argument adequately

☐ Sign-Event Correlation Statement -- There is a strong correlation between the sign and the event signified

☐ Lack Of Reliability Statement -- A is not an honest source

☒ Resolving Inconsistency Statement -- The practical inconsistency can be resolved by further dialogue.

☐ Knowledge Position Statement -- A may plausibly be taken to be true(false)

☐ Bad Consequence Statement -- If A is brought about, bad consequences (B) will plausibly occur

Figure 7.15: Defining sub-classes of a new statement class

☐ Use Existing
Lack Of Complete Knowledge Statement

☒ Create New
Statement Name: Property Appearance Statement
Statement Description: It appears that object has property F

The Premise's super classes are :

☐ Lack Of Complete Knowledge Statement -- The knowledge is not complete enough to support the argument adequately

☐ Sign-Event Correlation Statement -- There is a strong correlation between the sign and the event signified

☐ Lack Of Reliability Statement -- A is not an honest source

☐ Resolving Inconsistency Statement -- The practical inconsistency can be resolved by further dialogue.

Figure 7.16: Defining a new statement class (Property Appearance Statement)

Upon confirming the addition of the new scheme to the ontology, the new statement classes (if existing) are created and annotated with their names and descriptions. After that their appropriate super-classes and sub-classes are defined. The argument scheme class is then created with all the annotations (author, creation date, scheme name, etc.) and the necessary and necessary-and-sufficient conditions using the related statements. A default instance of this scheme is also created to assist in reasoning purposes (as explained in Section 7.3.3)

Home
Add New Argument
View Arguments
Argument Search
Add Scheme
Scheme Detail

Defining new scheme : Argument From Appearance

Step 5 of 6 - Defining the exception(s) of the new scheme

To define the exceptions of the new scheme, you may choose any of the existing statements, or define a new statement. When defining a new statement, you need to specify:

1. The name (for example: Forbidden Action Statement)
2. The description (for example: A should not be brought about)
3. The superclasses of the statement; i.e. the statements that are generalizations of this statement
4. The subclasses of the statement; i.e. the statements that are specializations of this statement

You may add additional exceptions by choosing "Save and Add Exception".

☐ Use Existing

Lack Of Complete Knowledge Statement

☒ Create New

Statement Name

Misleading Apperance Statement

Statement Description

The object appearance as having property F is misleading

Previous Statements

[Conclusion](#)
[Remove](#)

[Add New Premise](#)

[Premise 1](#)
[Remove](#)

[Add New Assumption](#)

Figure 7.17: Accessing previous choices from exception definition form

Previous Statements

[Conclusion](#)
[Remove](#)

[Add New Premise](#)

[Premise 1](#)
[Remove](#)

[Add New Assumption](#)

Figure 7.18: Accessing and editing previous statements in the process of new scheme definition

New scheme : Argument From Appearance - Summary

| | | |
|-------------------------------------|---|---|
| Conclusion Descriptor: | | |
| Contains Property Statement | object has property F | Remove Edit |
| Premise Descriptor(s): | Add New Premise | |
| Property Appearance Statement | It appears that object has property F | Remove Edit |
| Assumption Descriptor(s): | Add New Assumption | |
| Exception Descriptor(s): | Add New Exception | |
| Misleading Apperance Statement | The object appearance as having property F is misleading | Remove Edit |
| Unjustified Property Classification | There are grounds that object (more correctly) has property R | Remove Edit |

Add New Scheme

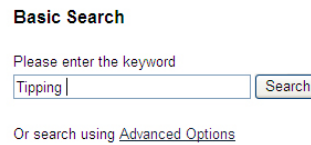
You may also [Cancel](#) the addition of the new argument scheme.

Figure 7.19: Summary of new argument scheme: Argument From Appearance

7.3.9 Searching for arguments based on keywords, authors, schemes and date range

By choosing *Argument Search* from the navigation menu, the user is able to search for specific arguments. Two modes of basic and advanced search are available in the system. In basic

search, users can query the argument network based on a keyword existent in a claim. This option is shown in Figure 7.20.



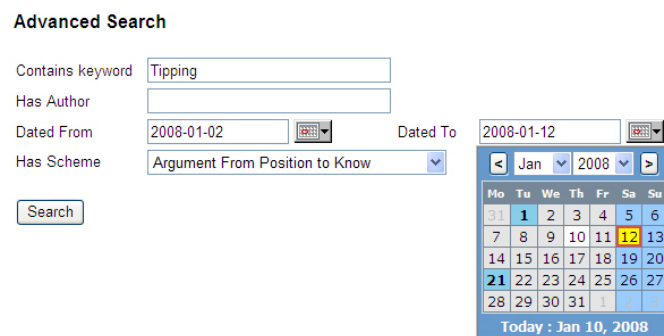
Basic Search

Please enter the keyword

Or search using [Advanced Options](#)

Figure 7.20: Basic keyword search

In advanced mode, users are able to query the network using any combination of keyword, argument author, date range and argument scheme. When searching for arguments of a specific scheme type, inference is used to return all the arguments that are instances of that specific scheme as well as instances that belong to any of its sub-classes. This is a key new feature provided by the underlying ontology and the inference mechanism of the description logic reasoner. Figure 7.21 illustrates the advance search screen.



Advanced Search

Contains keyword

Has Author

Dated From Dated To

Has Scheme

Calendar: Jan 2008

| Mo | Tu | We | Th | Fr | Sa | Su |
|----|----|----|----|----|----|----|
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | | | |

Today : Jan 10, 2008

Figure 7.21: Advanced Search Mode

The result of both modes of searches is a list of arguments. Figure 7.22 shows the results of an advanced search with the following parameters: To find all arguments which contain the keyword tipping, dated between January 2nd to January 12th, 2008 and are of scheme type “argument from position to know.” Two arguments - one of type “appeal to expert opinion” and the other of type “legal argument from expert opinion” - are returned; the details of each argument can be viewed by clicking each argument title.

List of Arguments

Your search has returned the following results. Please click the argument title to view its details.

[Mandatory tipping is not legal.](#)
[Tipping lowers self esteem](#)

Figure 7.22: Advanced Search Results

7.4 Summary

This chapter explained the architecture of the Web-based system utilizing the new proposed ontology. I outlined a comparison of the different available tools required for each part of this architecture and explained the reasons for choosing Protégé as the ontology editor, Jena as the RDF repository, Pellet as the description logic reasoner, Java as the core language of the Website and SQL server as the database. The main features of the system were also

highlighted in this chapter. These features include: listing available arguments, creation of new arguments, attacking/supporting arguments, listing the attacking/supporting arguments, displaying scheme details and scheme hierarchy, basic and advanced search modes and creation of new schemes.

Chapter 8

Evaluation

This chapter re-visits a number of key requirements for building large-scale argumentation systems on the Web to assess whether the implemented ontology and the Web-based system (Avicenna) fulfils those requirements. I also compare the main features of the system with other argumentation tools discussed in Chapter 2. Finally, some of the limitations of the system are discussed.

8.1 Re-visiting the Desiderata

Section 2.3 outlined a number of features deemed essential by Rahwan et al [38] to allow for development of a large-scale argument annotation on the Web. These features are re-visited again and the subsequent parts of this section explain how each feature is realized in the system.

The WAW must support the storage, creation, update and querying of argumentative structures:

Avicenna supports storage of arguments in persistent RDF storage (back-end database). It is possible to create new arguments by introducing new claims or using any of the existing claims in the system. While creating new arguments, the inferred critical questions on that scheme are also added to the argument instance.

Apart from re-using existing claims while authoring arguments, the argument network can be updated by attacking or supporting different claims. Attacks could be any of the following types: Symmetric attacks, Asymmetric attacks, undermining assumptions or supporting an exception. All of these different types of argument interactions are explicitly modelled in the system.

Avicenna implements and uses queries in different tasks; for example: displaying the different parts of an argument instance, displaying the details of an argumentation scheme and searching for arguments in both basic and advanced modes (advanced search queries the argument network based on different parameters; in case of searching for instances of a specific argumentation scheme, inference is used to return the instances of inferred sub-classes of that argumentation scheme as well). Queries are also utilized to return attacking or supporting arguments of a given claim; searching for supporting arguments of a claim retrieves arguments that support the claim both directly and indirectly.

The WAW must have Web-accessible repositories:

Arguments are stored in a persistent RDF storage and can be accessed and queried on the Web and by using different standard RDF query languages.

The WAW language must be based on open standards, enabling collaborative development of new tools:

Avicenna is based on OWL and annotates arguments in RDF. Both OWL and RDF are open standards and recommendations by the W3C. A variety of software development tools can be used for exploiting this either by using the ontology directly or using any of the available ontology mapping techniques.

The WAW must employ a unified, extensible argumentation ontology:

The ontology represents a new reification of the Argument Interchange Format, which is the most recent general specification for describing arguments and argument networks.

The WAW must support the representation, annotation and creation of arguments using a variety of argumentation schemes:

Avicenna not only preserves the AIFs emphasis on scheme-based reasoning patterns, but also classifies schemes based on their constituent premises and conclusions; thus enabling automatic classification of hierarchy of argument schemes. The system allows extension of the underlying ontology by addition of new schemes from the interface.

8.2 Avicenna and current Web-based argumentation tools

Table 8.1 highlights the main features of Avicenna in comparison to other Web based argumentation systems discussed in Chapter 2. In particular, the Description Logic-based OWL reasoning over argument structures is exploited in number of system features to enhance analysis and querying of arguments. An example of such features is automatic classification of schemes which is unique to Avicenna. The system also enables addition of new argument schemes to the underlying ontology. Semantic search is another feature utilizing Description logic-based OWL reasoning for instance classification and inference of indirect support in chained arguments.

The design of the underlying ontology not only facilitates the supporting and attacking of different arguments, but also enables formation of complex structures (e.g. divergent, convergent, serial) among multiple inter-connected arguments.

8.3 Limitations

The proposed ontology and the Web-based system (Avicenna) are a work in progress; there are several limitations and areas for future development. The following provides a list of some of the points through which the work can be further enhanced.

The hierarchy of argument topic categories (such as sports, education, law, etc.) has not been implemented in the ontology. The topic categories can offer a better structure to the argument network and provide broader query options.

The current approach to definition of argument schemes (and therefore, the inference of hierarchy of schemes) is based on the necessary-and-sufficient conditions on *hasPremise* and *hasConclusion* properties of each scheme (as explained in detail in Section 6.2). Certain argument schemes exhibit somewhat complex structures; for example, “circumstantial ad hominem argument” is a chain of argumentation based on combining “argument from inconsistent commitment” with the “direct ad hominem argument” [56].

In this scheme, an intermediate conclusion is based on a conclusion of one scheme (argument from inconsistent commitment), while its final conclusion is based on the conclusion of another scheme (direct ad hominem argument); in this case, should this argument be considered as several distinct arguments that are simply chained? Or it should be considered as a single argument scheme? If this scheme is considered as a single argument scheme, should it be inferred as a specialisation of all the different schemes it is made of? Or just the scheme with which its final conclusion is shared (or a specialisation of)? Considering the different types of argument schemes that the ontology must incorporate and the expected classification hierarchy results, the necessary-and-sufficient conditions on each scheme might be re-defined by using a

new property, *hasPart*, that stands for both *hasPremise* and *hasConclusion* properties, and properties *hasPremise* and *hasConclusion* will become part of the necessary conditions.

A current limitation in the SPARQL query language in case of transitive properties is that it is not possible to limit the depth of application of transitive properties. In current system, when user queries to view the supporting arguments of a claim, it is not possible to limit the returned results to those triples that provide indirect support up to two levels back (or any arbitrary number specified by the user).

Content acquisition forms another future direction. Moreover, the integration of arguments from other repositories into the system should be explored. Work is also needed on integrating effective argument visualisation techniques [25], which can help in acquisition as well as interaction with the argument repository.

Another direction of future research is integrating reasoning about the acceptability of arguments [17] based on various semantic criteria. However, in an open environment like the Web, it is unlikely for traditional rigid approaches to be successful and/or scalable. New, more approximate approaches are needed to reason about argument acceptability at such a large scale.

8.4 Summary

In this chapter, I re-visited the WVAW requirements for development of large scale argumentation on the Web and showed how Avicenna with support of its underlying ontology fulfils these requirements and enhances them through reasoning. Some of the limitations of the system were discussed and its main features were highlighted in comparison to other Web-based argumentation systems.

| <i>System Features</i> | <i>Debatepedia</i> | <i>Cohere</i> | <i>Debatemapper</i> | <i>Parmenides</i> | <i>Truthmapping</i> | <i>CoPe-ii!</i> | <i>Anuclaria</i> | <i>DiscourseDB</i> | <i>ArgDF</i> | <i>Avicenna</i> |
|-------------------------------------|----------------------------|--------------------------------------|--|---|---|--------------------------|--|--|--|--|
| Underlying Theory | Pro/Con Logic Tree General | Connected Ideas and Roles Learning | Proprietary General | Persuasion Over Action Deliberative Democracy | Proprietary General | IBIS General | Argumentation Schemes Education | Proprietary Politics | AIF, W. Schemes ^a General | AIF, W. Schemes General |
| Domain | | | | | | | | | | |
| Argument Representation | Wiki, Logic tree | SQL DB, XML | SQL DB | SQL DB | SQL DB | XML | AML | Semantic Media Wiki | RDF/RDFS | RDF/OWL |
| Visualisation | - | Connection Net | Debate Maps | - | Statement Maps | IBIS Graphs | Argument Diagrams | - | - | - |
| Evaluation | - | - | Rating | - | Rating | Rating | Rating (tags) | - | - | - |
| Micro Argument Structure | Pro/Con Position | Idea, Role, Connection | Pro/Con Argument, Issue, Position etc. | Persuasion Over Action Scheme | Premise, Conclusion Critique 1 Rebuttal | Issue Position, Argument | Scheme Based ^b | For, Against, Mixed Item On a Position For a Topic | W. Schemes | W. Schemes |
| Macro Argument Structure | - | Linked, Serial Divergent, Convergent | Linked, Serial Divergent, Convergent | - | Linked Serial | I.N.A. ^c | Linked, Serial Divergent, Convergent Schemes | - | Linked, Serial Divergent, Convergent Schemes (Instances) | Linked, Serial Divergent, Convergent Schemes (Classes) |
| Extensible Ontology | - | - | - | - | - | - | - | - | - | - |
| Inference Over Argument Network | - | - | - | - | - | - | - | Subclass Hierarchy | - | DL Inference |
| Creation of new arguments | Yes | Yes | Yes | Restricted (Alternatives) | Yes | Yes | Yes | Yes | Yes | Yes |
| Re-use of existing claims | Yes | Yes | Yes | - | Yes | Yes | - | Yes | Yes | Yes |
| Supporting claims | Yes | Yes | Yes | (Yes/No Answers To Questions) | Team Members | Yes | - | Yes | Restricted | Yes |
| Attacking claims | Yes | Yes | Yes | (Yes/No Answers To Questions) | Yes | Yes | - | Yes | Yes | Yes |
| Automatic classification of schemes | - | - | - | - | - | - | - | - | - | Yes |
| Search | Keyword Search | Keyword Search | Keyword Search | - | Keyword Search (Topic, Category) | Keyword Search | Advanced Search (Scheme, date, author, etc.) | Semantic Search | Semantic Search Keyword | Semantic Search Advanced (Scheme, date, author, etc.) |

Table 8.1: Comparison of Main Features of Argumentation Systems

^aWalton Schemes [56]

^bThe argument is structured according to the scheme; for example, if argument is structured according to a Walton scheme, it contains premise(s), conclusion, assumption(s) and exception(s)

^cInformation is not available.

Chapter 9

Conclusion and Further Work

In this thesis I introduced the design and implementation of an ontology that exploits the Web Ontology Language (OWL) for creating, navigating and manipulating complex argument structures. This ontology is based on a new reification of the Argument Interchange Format [15], exploits Walton's account of argumentation schemes [56] and follows the same key principles as WVAW [38]. The various sections in the thesis outlined how this ontology enables the use of automated description logic reasoning over argument structures. In particular, OWL reasoning enables significantly enhanced querying of arguments through automatic scheme classifications, instance classification, inference of indirect support in chained argument structures and inference of critical questions. This provides a seed for further work that combines traditional argument-based reasoning techniques [13] with ontological reasoning in a Semantic Web environment. I also presented the implementation of a Web-based system for authoring and querying argument structures in RDF which utilizes the developed OWL ontology.

The ontology introduced and the Web-based system presented in this thesis are a work in progress; there are several areas for future development in the work undertaken in this thesis. One of the most important areas through which the work can be further enhanced is acquisition of content and integration of arguments from other repositories. Argument visualisation [25] is another significant advancement. But perhaps the most interesting direction of future research is integrating reasoning about the acceptability of arguments [17] based on various semantic criteria; specially since in an open environment like the Web, traditional approaches are unlikely to be successful and new and more approximate approaches are needed to reason about argument acceptability at such a large scale.

Appendix A

The Semantic Web and OWL

A.1 The Semantic Web

The Semantic Web is at the center of Tim Berners-Lee’s vision for the future of the World Wide Web. In his Roadmap document [7], Berners-Lee contrasts the Semantic Web with the existing, merely human readable Web: “the Semantic Web approach instead develops languages for expressing information in a machine processable form.” This is perhaps the best way of summing up the Semantic Web: technologies for enabling machines to make more sense of the Web, with the result of making the Web more useful for humans [16]. The Semantic Web is intended to create a universal medium for information exchange by giving semantics to the content of documents on the Web by means of defining ontologies and individuals.

The development of the Semantic Web proceeds in steps, each step building a *layer* on top of another. Figure A.1 shows the “layer cake” of the Semantic Web. The purpose of each layer is explained next [1, 28].

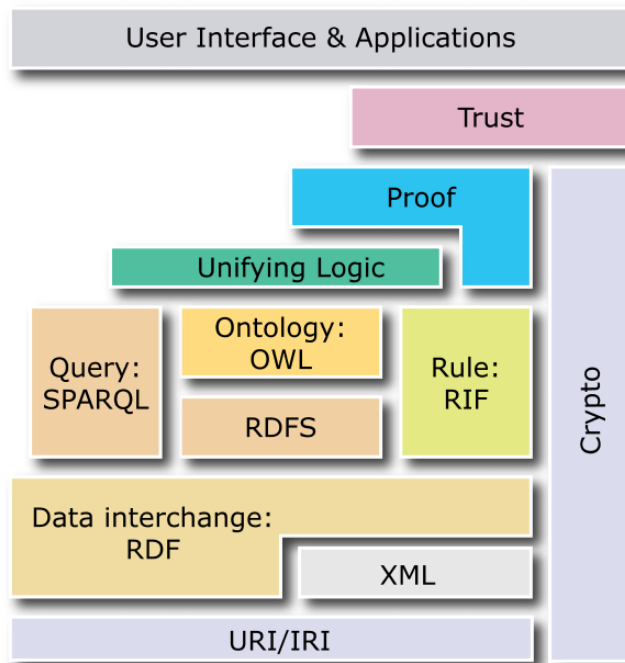


Figure A.1: Semantic Web Layer Cake [8]

URI/IRI (Uniform Resource Identifiers/Internationalized Resource Identifiers) provide an

identification scheme for entities on the World Wide Web. *XML* provides customized tagging schemes for content structure within documents, but without associating any semantics with the meaning of the content. *XML Schema* is a language for describing and restricting the structure and content of elements contained within XML documents.

RDF is a data model for describing objects (“resources”) and their relationships. Its basic building block is an object-attribute-value triple, called a statement. A RDF-based model can be represented in XML syntax. Since RDF does not make any assumptions about any particular application domain or define the semantics of a domain, *RDF Schema* is used to describe properties and classes of RDF-based resources, with semantics for hierarchies of such properties and classes.

OWL improves the expressivity of RDFS by adding more vocabulary for describing properties and classes; for example: local scope of properties, disjointness of classes, cardinality (e.g. “exactly one”), boolean combination of classes, richer typing of properties and characteristics of properties (e.g. symmetry), and enumerated classes.

SPARQL is a query language for Semantic Web data sources (in RDF format). Recent standardization efforts include *Rule Interchange Format (RIF)* as a format for interchange of rules in rule-based systems on the Semantic Web. The *Logic* layer is used to enhance the ontology language further and to enable the writing of application-specific declarative knowledge.

The *Proof* layer involves the actual deductive process as well as the representation of proofs in Web languages (from lower levels) and proof validation. Finally the *Trust* layer will emerge through the use of digital signatures and *Cryptography*. As this project is built on *OWL*, the next section provides a more detailed description of this ontology language.

A.2 Web Ontology Language (OWL)

The Web Ontology Language (OWL) [28] (formerly known as DAML-OIL) is a language for defining and instantiating Web ontologies; it is a W3C recommendation. OWL ontology includes descriptions of classes, properties and their instances. OWL improves the expressivity power of RDFS in several different ways, including: local scope of properties (restrictions on range), disjointness of classes, Boolean combination of classes (building new classes through union, intersection and complement of existing classes), cardinality restrictions (placing restrictions on how many distinct values a property can take) and special characteristics of properties (such as transitivity, symmetry, uniqueness, etc.).

OWL offers a set of formal semantics that are well-established in the domain of mathematical logic; thus enabling reasoning to be performed about knowledge. Reasoning support allows for checking consistency of the ontology and the knowledge, finding un-intended relationships between classes and automatically classifying instances in classes.

OWL is (partially) mapped on a Description Logic (See Appendix C for a brief description). W3C’s Web Ontology Working Group has defined OWL as three different sub-languages, each to fulfil a different aspect of the set of requirements:

OWL Full: Supports maximum expressiveness and syntactic freedom of RDF, but with no computational guarantees. For example, in OWL Full, a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

OWL DL (Description Logic): Supports the maximum expressiveness while retaining computational completeness and decidability. OWL DL includes the entire OWL Full language constructs under certain restrictions. This feature enables efficient reasoning support.

OWL Lite: An even further restriction limits OWL DL to a subset of the language constructors; for example, OWL Lite excludes enumerated classes, disjoint statements and arbitrary cardinality.

Appendix B

Argument Structures

There are several types of argument structures that can be identified, depending on how the inference is linked together in a chain of reasoning in a given case. Some of the most common types of these argument structures are discussed next according to the descriptions provided by Walton [56] and Rahwan [37].

The simplest argument structure is the *single* argument which has only one premise used as basis for inferring a conclusion. Figure B.1(a) depicts this argument structure. Single arguments are a special case of *linked* arguments in which a set of premises, together, lead to the conclusion (see Figure B.1(b)).

Often, arguers present complex structures of multiple inter-connected arguments. For instance, one may present multiple (individual) arguments in support of the same conclusion. The resulting argument structure is referred to as *convergent* argument and is illustrated in Figure B.1(c). The convergent argument is different from linked argument in that in convergent arguments, each premise alone supports the conclusion (i.e., together with the conclusion, it constitutes a single argument by itself).

Figure B.1(d) depicts another important structure of arguments, namely the *divergent* argument structure. In this type of structure, a single statement acts as a premise to support multiple conclusions. The final argument structure explained in this section is the *serial* argument. In this structure, the conclusion of one argument acts as a premise of another, whose conclusion could also form a premise of another argument, and so on; therefore, resulting in a chain of arguments as shown in Figure B.1(e).

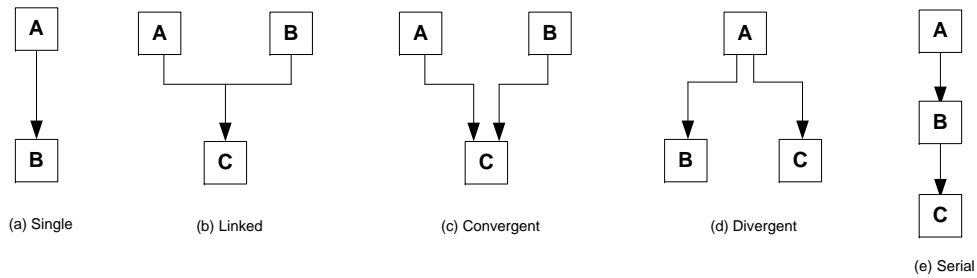


Figure B.1: Common Argument Structures

Appendix C

Description Logics

Description Logics (DLs) [3] are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain. The idea is to define complex concept hierarchies from basic (atomic) concepts, and to define complex roles (or properties) that define relationships between concepts.

Table C.1 shows the syntax and semantics of common concept and role constructors. The letters A, B are used for atomic concepts and C, D for concept descriptions. For roles, the letters R and S are used and non-negative integers (in number restrictions) are denoted by n , m and individuals (i.e. instances) by a, b . An *interpretation* \mathcal{I} consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

A DL knowledge base consists of a set of terminological axioms (often called *TBox*) and a set of assertional axioms or assertions (often called *ABox*). A finite set of definitions is called a *terminology* or *TBox* if the definitions are unambiguous, i.e., no atomic concept occurs more than once as left hand side.

| Name | Syntax | Semantics |
|--|----------------------------|--|
| Concept & Role Constructors | | |
| Top | \top | $\Delta^{\mathcal{I}}$ |
| Bottom | \perp | \emptyset |
| Concept Intersection | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| Concept Union | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| Concept Negation | $\neg C$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| Value Restriction | $\forall R.C$ | $\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$ |
| Existential Quantifier | $\exists R.C$ | $\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$ |
| Unqualified | $\geq nR$ | $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \geq n\}$ |
| Number | $\leq nR$ | $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \leq n\}$ |
| Restriction | $= nR$ | $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} = n\}$ |
| Role-value-map | $R \subseteq S$ $R = S$ | $\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow (a, b) \in S^{\mathcal{I}}\}$ $\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \leftrightarrow (a, b) \in S^{\mathcal{I}}\}$ |
| Nominal | I | $I^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ with $ I^{\mathcal{I}} = 1$ |
| Universal Role | U | $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| Role Intersection | $R \sqcap S$ | $R^{\mathcal{I}} \cap S^{\mathcal{I}}$ |
| Role Union | $R \sqcup S$ | $R^{\mathcal{I}} \cup S^{\mathcal{I}}$ |
| Role Complement | $\neg R$ | $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$ |
| Role Inverse | R^{-} | $\{(b, a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\}$ |
| Transitive Closure | R^{+} | $\bigcup_{n \geq 1} (R^{\mathcal{I}})^n$ |
| Role Restriction | $R c$ | $R^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \times C^{\mathcal{I}})$ |
| Identity | $id(C)$ | $\{(d, d) \mid d \in C^{\mathcal{I}}\}$ |
| Terminological Axioms | | |
| Concept Inclusion | $C \sqsubseteq D$ | $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ |
| Concept Equality | $C \equiv D$ | $C^{\mathcal{I}} = D^{\mathcal{I}}$ |
| Role Inclusion | $R \sqsubseteq S$ | $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ |
| Role Equality | $R \equiv S$ | $R^{\mathcal{I}} = S^{\mathcal{I}}$ |

Table C.1: Some Description Logic Role Constructors, Concept Constructors, and Terminological Axioms

To give examples of what can be expressed in DLs, we suppose that *Person* and *Female* are atomic concepts. Then $Person \sqcap Female$ is a DL concept describing, intuitively, those persons that are female. If, in addition, we suppose that *hasChild* is an atomic role, we can form the concept $Person \sqcap \exists hasChild$, denoting those persons that have a child. Using the bottom concept, we can also describe those persons without a child by the concept $Person \sqcap \forall hasChild. \perp$. These examples show how we can form complex descriptions of concepts to describe classes of objects.

The *terminological axioms* make statements about how concepts or roles are related to each other. It is possible to single out definitions as specific axioms and identify terminologies as sets of definitions by which we can introduce atomic concepts as abbreviations or names for complex concepts.

An equality whose left-hand side is an atomic concept is a *definition*. Definitions are used to introduce symbolic names for complex descriptions. For instance, by the axiom $Mother \equiv Woman \sqcap \exists hasChild. Person$, we associate to the description on the right-hand side the name *Mother*. Symbolic names may be used as abbreviations in other descriptions. If, for example, we have defined *Father* analogously to *Mother*, we can define *Parent* as $Parent \equiv Mother \sqcup Father$. Table C.2 shows a terminology with concepts concerned with family relationships.

| <i>Name</i> | <i>DL Syntax</i> | <i>Example</i> |
|----------------------------|----------------------------------|---|
| Constructor / axiom | | |
| Concept Intersection | $C \sqcap D$ | $Woman \equiv Person \sqcap Female$ |
| Concept Union | $C \sqcup D$ | $Parent \equiv Mother \sqcup Father$ |
| Concept Negation | $\neg C$ | $Man \equiv Person \sqcap \neg Woman$ |
| Existential Quantifier | $\exists R.C$ | $Mother \equiv Woman \sqcap \exists hasChild. Person$ |
| Value Restriction | $\forall R.C$ | $MotherWithoutSons \equiv Mother \sqcap \forall hasChild. Woman$ |
| MinCardinality | $\geq nR$ | $MotherWithAtLeastThreeChildren \equiv Mother \sqcap \geq 3 hasChild$ |
| Cardinality | $= nR$ | $FatherWithOneChild \equiv Father \sqcap = 1 hasChild$ |
| Bottom | \perp | $PersonWithoutAChild \equiv Person \sqcap \forall hasChild. \perp$ |
| Transitive Property | $R^+ \sqsubseteq R$ | $ancestor^+ \sqsubseteq ancestor$ |
| Role Inverse | $R \equiv S^-$ | $hasChild \equiv hasParent^-$ |
| Concept Inclusion | $C \sqsubseteq D$ | $Woman \sqsubseteq Person$ |
| Disjoint with | $C \sqsubseteq \neg D$ | $Man \sqsubseteq \neg Woman$ |
| Role Inclusion | $R \sqsubseteq S$ | $hasDaughter \sqsubseteq hasParent$ |
| Range | $\top \sqsubseteq \forall R.C$ | $\top \sqsubseteq \forall hasParent. Person$ |
| Domain | $\top \sqsubseteq \forall R^-.C$ | $\top \sqsubseteq \forall hasParent^-. Person$ |

Table C.2: A terminology (TBox) with concepts about family relationships

Appendix D

Scheme Definitions in Description Logic

In this section, the definition of some of the argument schemes in Description Logic is provided.

Argument From Positive Consequences

$$\begin{aligned} ArgFromPosCons &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.EncouragedActionStmnt \sqcap \\ &\exists hasPremise.PositiveConsequenceStmnt) \\ ArgFromPosCons &\sqsubseteq \exists hasException.OppositeConsequencesStmnt \\ ArgFromPosCons &\sqsubseteq \exists hasAssumption.ConsequenceBackUpEvidenceStmnt \\ ArgFromPosCons &\sqsubseteq \exists hasAssumption.StrongConsequenceProbabilityStmnt \end{aligned}$$

Argument From Correlation To Cause

$$\begin{aligned} ArgFromCorrelationToCause &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.CausalStmnt \sqcap \\ &\exists hasPremise.CorrelationStmnt) \\ ArgFromCorrelationToCause &\sqsubseteq \exists hasException.OtherCausalFactorsInvolvedStmnt \\ ArgFromCorrelationToCause &\sqsubseteq \exists hasAssumption.LackOfCoincidenceStmnt \end{aligned}$$

Argument From Sign

$$\begin{aligned} ArgFromSign &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.PresumedCauseStmnt \sqcap \\ &\exists hasPremise.ExistenceOfSignStmnt \sqcap \\ &\exists hasPremise.IndicationBySignStmnt) \\ ArgFromSign &\sqsubseteq \exists hasException.SignFromOtherEventsStmnt \\ ArgFromSign &\sqsubseteq \exists hasException.SignEventCorrelationStmnt \end{aligned}$$

Direct Threat Argument

$$\begin{aligned} DirectThreatArg &\equiv (PresumptiveArgument \sqcap \\ &\exists hasConclusion.ForbiddenActionStmnt \sqcap \end{aligned}$$

$\exists \text{hasPremise.BadConsequenceStmnt} \sqcap$
 $\exists \text{hasPremise.OppositeActionStmnt} \sqcap$
 $\exists \text{hasPremise.ThreatCommitmentStmnt})$
 $\text{DirectThreatArg} \sqsubseteq \exists \text{hasAssumption.ThreatRelevancyStmnt}$
 $\text{DirectThreatArg} \sqsubseteq \exists \text{hasAssumption.CredibleThreatStmnt}$

Indirect Threat Argument

$\text{IndirectThreatArg} \equiv (\text{PresumptiveArgument} \sqcap$
 $\exists \text{hasConclusion.ForbiddenActionStmnt} \sqcap$
 $\exists \text{hasPremise.BadConsequenceStmnt} \sqcap$
 $\exists \text{hasPremise.OppositeActionStmnt})$
 $\text{IndirectThreatArg} \sqsubseteq \exists \text{hasAssumption.ThreatRelevancyStmnt}$
 $\text{IndirectThreatArg} \sqsubseteq \exists \text{hasAssumption.CredibleThreatStmnt}$

Practical Reasoning

$\text{PracticalReasoning} \equiv (\text{PresumptiveArgument} \sqcap$
 $\exists \text{hasConclusion.EncouragedActionStmnt} \sqcap$
 $\exists \text{hasPremise.GoalStmnt} \sqcap$
 $\exists \text{hasPremise.GoalPlanStmnt})$
 $\text{PracticalReasoning} \sqsubseteq \exists \text{hasException.OppositeConsequencesStmnt}$
 $\text{PracticalReasoning} \sqsubseteq \exists \text{hasException.AlternativeMeansStmnt}$
 $\text{PracticalReasoning} \sqsubseteq \exists \text{hasException.ConflictingGoalsStmnt}$
 $\text{PracticalReasoning} \sqsubseteq \exists \text{hasException.RealisticGoalStmnt}$

Argument From Analogy

$\text{ArgFromAnalogy} \equiv (\text{PresumptiveArgument} \sqcap$
 $\exists \text{hasConclusion.BaseOutcomeStmnt} \sqcap$
 $\exists \text{hasPremise.BaseStmnt} \sqcap$
 $\exists \text{hasPremise.SimilarityOfCasesStmnt})$
 $\text{ArgFromAnalogy} \sqsubseteq \exists \text{hasException.DifferencesUndermineSimilarityStmnt}$
 $\text{ArgFromAnalogy} \sqsubseteq \exists \text{hasException.ExceptionSimilarityCaseStmnt}$

Appendix E

Sample Queries

This section contains a number of SPARQL [35] queries used in the Web-based system Avicenna. A brief explanation is provided before each query.

The following query is used to display the list of available arguments; except the default instance (main), the rest of instances will be returned.

Query 1. (*Retrieving list of argument instances*)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?schemInd ?title
WHERE
{
    ?schemInd kb:argTitle ?title.
    ?schemInd rdf:type kb:PresumptiveArgument.
    OPTIONAL { ?schemInd rdfs:comment ?comment. FILTER (?comment = 'Main') }
    FILTER(!bound(?comment))
}
```

Similarly, the next query is used to display the list of available arguments; however, it returns only those arguments added after a specific date specified by

Query 2. (*Retrieving list of argument instances from a specific date onwards*)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT ?schemInd ?title
WHERE
{
    ?schemInd kb:argTitle ?title.
    ?schemInd rdf:type kb:PresumptiveArgument.
    ?schemInd kb:creationDate ?date.
    OPTIONAL { ?schemInd rdfs:comment ?comment. FILTER (?comment = 'Main') }
    FILTER(!bound(?comment))
    FILTER (?date >= dateString ^^xsd:date)
}
```

The next query retrieves the list of existing argument schemes.

Query 3. (Retrieving list of argument schemes)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?scheme ?text
WHERE
{
    ?scheme rdfs:subClassOf kb:PresumptiveArgument.
    ?scheme kb:schemeName ?text.
}
ORDER BY ?text.
```

When adding a new argument or displaying the details of each argument scheme, it is necessary to retrieve the conclusion, premise, assumption and exception classes defined as restrictions on a specific scheme. In subsequent queries, **ArgSchemVar** is a variable that could represent any of the available schemes such as “argument from analogy”, “argument from negative consequences”, etc. The **MainInstance** is also a variable that holds the default instance of an argument scheme class.

Query 4. (Retrieving the premises defined on an argument scheme)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX list:<http://jena.hpl.hp.com/ARQ/list#>

SELECT DISTINCT ?preform ?pretext
WHERE
{
    ArgSchemVar owl:equivalentClass ?equiv.
    ?equiv owl:intersectionOf ?collection.
    ?collection list:member ?member1.
    ?member1 owl:onProperty kb:hasPremise.
    ?member1 owl:someValuesFrom ?preform.
    ?preform kb:formDescription ?pretext.
}
```

Query 5. (Retrieving assumptions defined on an argument scheme)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?presform ?text
WHERE
{
    MainInstance rdf:type ?class.
    ?class rdfs:subClassOf ?restriction.
    ?restriction owl:onProperty kb:hasAssumption.
    ?restriction owl:someValuesFrom ?presform.
    ?presform kb:formDescription ?text.
}
```

The next 4 queries are used to retrieve information about conflicting claims. The variable **ClaimInstance** is used to represent a specific claim instance and the variable **ArgInstance**

symbolizes the argument instance.

Query 6. (Retrieving claims in symmetric attack with a specific claim instance)

PREFIX kb:<http://www.ArgOWL.org#>

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
SELECT DISTINCT ?claimid ?text
WHERE
{
  { ClaimInstance kb:attacks ?conf.
    ?conf kb:confAttacks ?claimid.
    ?claimid kb:confIsAttacked ?conf.
    ?conf kb:isAttacked ClaimInstance.
    ?claimid rdf:type kb:Statement.
    ?claimid kb:claimText ?text. }
UNION
  { ?claimid kb:attacks ?conf.
    ?conf kb:confAttacks ClaimInstance.
    ClaimInstance kb:confIsAttacked ?conf.
    ?conf kb:isAttacked ?claimid.
    ?claimid rdf:type kb:Statement.
    ?claimid kb:claimText ?text. }
}
```

Query 7. (Retrieving claims that are attacked by a specific claim)

PREFIX kb:<http://www.ArgOWL.org#>

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
SELECT DISTINCT ?claimid ?text
WHERE
{
  { ClaimInstance kb:attacks ?conf.
    ?conf kb:confAttacks ?claimid.
    ?claimid rdf:type kb:Statement.
    ?claimid kb:claimText ?text. }
UNION
  { ClaimInstance kb:confIsAttacked ?conf1.
    ?conf1 kb:isAttacked ?claimid.
    ?claimid rdf:type kb:Statement.
    ?claimid kb:claimText ?text. }
}
```

Query 8. (Retrieving claims that attack a specific claim)

PREFIX kb:<http://www.ArgOWL.org#>

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
SELECT DISTINCT ?claimid ?text
WHERE
{
  { ?claimid kb:attacks ?conf.
    ?conf kb:confAttacks ClaimInstance.
    ?claimid rdf:type kb:Statement.
    ?claimid kb:claimText ?text. }
UNION
  { ?claimid kb:confIsAttacked ?conf.
    ?conf kb:isAttacked ClaimInstance.
```

```

?claimid rdf:type kb:Statement.
?claimid kb:claimText ?text. }
}

```

Query 9. (Retrieving claims that undermine an argument by attacking a specific assumption)

```

PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```

```

SELECT DISTINCT ?claimid ?text
WHERE
{
  ?claimid kb:attacks ?conf.
  ?conf kb:underMinesAssumption ClaimInstance.
  ?conf kb:confAttacks ArgInstance.
  ?claimid rdf:type kb:Statement.
  ?claimid kb:claimText ?text.
}

```

The following query is used to retrieve supporting arguments (direct and indirect) of a specific claim (represented by **ClaimInstance**).

Query 10. (Retrieving arguments that support a claim)

```

PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```

```

SELECT DISTINCT ?schem2 ?title
WHERE
{
  { ?schem1 kb:hasConclusion ClaimInstance.
    ?schem2 kb:supports ?schem1.
    ?schem2 kb:argTitle ?title. }
  UNION
  { ?schem2 kb:hasConclusion ClaimInstance.
    ?schem2 kb:argTitle ?title. }
}

```

The next query retrieves the list of arguments that a claim (**ClaimInstance**) is utilized in.

Query 11. (Retrieving the list argument that a claim is utilized in.)

```

PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```

```

SELECT DISTINCT ?arg ?title
WHERE
{
  { ?arg kb:hasConclusion ClaimInstance.
    ?arg kb:argTitle ?title. }
  UNION
  { ?arg kb:hasPremise ClaimInstance.
    ?arg kb:argTitle ?title. }
}

```

The subsequent queries retrieve the list of sub-classes and super-classes of an argument scheme represented by **ArgSchem**.

Query 12. (Retrieving the list of super-classes of an argument scheme)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT DISTINCT ?super ?superschemeName
WHERE
{
    ArgSchem rdfs:subClassOf ?super.
    ?super rdfs:subClassOf kb:PresumptiveArgument.
    ?super kb:schemeName ?superschemeName.
    FILTER (?super != ArgSchem )
}
```

Query 13. (Retrieving the list of sub-classes of an argument scheme)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT DISTINCT ?sub ?subschemeName
WHERE
{
    ?sub rdfs:subClassOf ArgSchem
    ?sub rdfs:subClassOf kb:PresumptiveArgument.
    ?sub kb:schemeName ?subschemeName.
    FILTER (?sub != ArgSchem )
}
```

The subsequent query performs a basic keyword search and retrieves any arguments that contain that keyword (represented by **SelKey**) in their premises or conclusions.

Query 14. (Basic Search - Retrieving arguments based on keyword)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT DISTINCT ?schem ?title
WHERE
{
    { ?schem kb:argTitle ?title.
      ?schem kb:hasConclusion ?conc.
      ?conc kb:claimText ?text.
      OPTIONAL ?schem rdfs:comment ?comment.
      FILTER(!bound(?comment))
      FILTER regex(?text, SelKey, 'i') }
    UNION
    { ?schem kb:argTitle ?title.
      ?schem kb:hasPremise ?prem.
      ?prem kb:claimText ?text.
      OPTIONAL ?schem rdfs:comment ?comment.
      FILTER(!bound(?comment))
      FILTER regex(?text, SelKey, 'i') }
}
```

Advanced search mode offers the facility to search for any combination of keyword, argument scheme, date range and author of an argument. Total of 15 queries are required for this mode.

The next 4 queries are selected from the total and detailed below. Variables **SelKey**, **SelDateFrom**, **SelDateTo**, **SelAuthor** and **SchemeClass** represent placeholders for Keyword, date from, date to, author name and argument scheme respectively.

Query 15. (Advanced Search according to date range and argument scheme)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT ?schem ?title WHERE
{
    ?schem kb:argTitle ?title.
    ?schem rdf:type SchemeClass.
    ?schem kb:creationDate ?date.
    OPTIONAL { ?schem rdfs:comment ?comment. }
    FILTER(!bound(?comment))
    FILTER (?date <= SelDateTo ^^xsd:date && ?date >= SelDateFrom ^^xsd:date)
}
```

Query 16. (Advanced Search according to author and keyword)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?schem ?title
WHERE
{
    {
        ?schem kb:argTitle ?title.
        ?schem kb:has-Author ?author.
        ?author kb:authorName ?name.
        ?schem kb:hasConclusion ?conc.
        ?conc kb:claimText ?text.
        OPTIONAL { ?schem rdfs:comment ?comment. }
        FILTER(!bound(?comment))
        FILTER regex(?name, SelAuthor , 'i')
        FILTER regex(?text, SelKey , 'i') }
    UNION
    {
        ?schem kb:argTitle ?title.
        ?schem kb:has-Author ?author.
        ?author kb:authorName ?name.
        ?schem kb:hasPremise ?prem.
        ?prem kb:claimText ?text.
        OPTIONAL { ?schem rdfs:comment ?comment. }
        FILTER(!bound(?comment))
        FILTER regex(?name, SelAuthor , 'i')
        FILTER regex(?text, SelKey + , 'i') }
}
```

Query 17. (Advanced Search according to date range, argument scheme and keyword)

```
PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?schem ?title
```

```

WHERE
{
  { ?schem kb:argTitle ?title.
    ?schem rdf:type SchemClass.
    ?schem kb:creationDate ?date.
    ?schem kb:hasConclusion ?conc.
    ?conc kb:claimText ?text.
    OPTIONAL { ?schem rdfs:comment ?comment. }
    FILTER(!bound(?comment))
    FILTER (?date <= SelDateTo ^xsd:date && ?date >= SelDateFrom ^xsd:date)
    FILTER regex(?text, SelKey , 'i') }
UNION
{ ?schem kb:argTitle ?title.
  ?schem rdf:type SchemClass.
  ?schem kb:creationDate ?date.
  ?schem kb:hasPremise ?prem.
  ?prem kb:claimText ?text.
  OPTIONAL { ?schem rdfs:comment ?comment. }
  FILTER(!bound(?comment))
  FILTER (?date <= SelDateTo ^xsd:date && ?date >= SelDateFrom ^xsd:date)
  FILTER regex(?text, SelKey , 'i') }
}

```

Query 18. (Advanced Search according to date range, argument scheme, author and keyword)

```

PREFIX kb:<http://www.ArgOWL.org#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

```

```

SELECT DISTINCT ?schem ?title
WHERE
{
  { ?schem kb:argTitle ?title.
    ?schem rdf:type SchemClass.
    ?schem kb:creationDate ?date.
    ?schem kb:has-Author ?author.
    ?author kb:authorName ?name.
    ?schem kb:hasConclusion ?conc.
    ?conc kb:claimText ?text.
    OPTIONAL { ?schem rdfs:comment ?comment. }
    FILTER(!bound(?comment))
    FILTER regex(?name, SelAuthor , 'i')
    FILTER regex(?text, SelKey , 'i')
    FILTER (?date <= SelDateTo ^xsd:date && ?date >= SelDateFrom ^xsd:date) }
UNION
{ ?schem kb:argTitle ?title.
  ?schem rdf:type SchemClass.
  ?schem kb:creationDate ?date.
  ?schem kb:has-Author ?author.
  ?author kb:authorName ?name.
  ?schem kb:hasPremise ?prem.
  ?prem kb:claimText ?text.
  OPTIONAL { ?schem rdfs:comment ?comment. }
  FILTER(!bound(?comment))
  FILTER regex(?name, SelAuthor , 'i')
  FILTER regex(?text, SelKey , 'i')
}

```

```
FILTER (?date <= SelDateTo ^^xsd:date || ?date >= SelDateFrom ^^xsd:date) }  
}
```

Bibliography

- [1] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, Cambridge, MA, USA, 2004.
- [2] Katie Atkinson, Trevor J. M. Bench-Capon, and Peter McBurney. PARMENIDES: facilitating deliberation in democracies. *Artificial Intelligence and Law*, 14(4):261–275, 2006.
- [3] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, Cambridge, UK, 2003.
- [4] Trevor J. M. Bench-Capon. Argument in Artificial Intelligence and Law. *Artificial Intelligence and Law*, 5(4):249–261, 1997.
- [5] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.
- [6] Jamal Bentahar, Zakaria Maamar, Djamel Benslimane, and Philippe Thiran. An Argumentation Framework for Communities of Web Services. *IEEE Intelligence Systems*, 22(6):75–83, 2007.
- [7] Tim Berners-Lee. Semantic web road map. Technical report, World Wide Web Consortium (W3C), 1998.
- [8] Steve Bratt. Semantic Web, and Other W3C Technologies to Watch. Presentation (talks), World Wide Web Consortium (W3C), 2007.
- [9] Dan Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation REC-rdf-schema-20040210, World Wide Web Consortium (W3C), 2004.
- [10] Jeen Broeskstra and Arjohn Kampman. SrRQL: A Second Generation RDF Query Language. Technical report, SWAD-Europe Workshop on Semantic Web Storage and Retrieval, 2003.
- [11] Daniela V. Carbogim, David Robertson, and John Lee. Argument-based applications to knowledge engineering. *Knowledge Engineering Review*, 15(2):119–149, 2000.
- [12] Giuseppe Carenini and Johanna D. Moore. Generating and evaluating evaluative arguments. *Artificial Intelligence*, 170(11):925–952, 2006.
- [13] Carlos I. Chesñevar, Ana G. Maguitman, and Ronald P. Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, 2000.
- [14] Carlos I. Chesñevar, Ana G. Maguitman, and Guillermo R. Simari. Argument-Based Critics and Recommenders: A Qualitative Perspective on User Support Systems. *Data & Knowledge Engineering*, 59(2):293–319, 2006.
- [15] Carlos I. Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott. Towards an Argument Interchange Format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.

- [16] Edd Dumbill. The Semantic Web: A Primer. Technical report, O'REILLY xml.com, 2000.
- [17] Phan M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.
- [18] Michael Elhadad. Using Argumentation in Text Generation. *Journal of Pragmatics*, 24:189–220, 1995.
- [19] Shelly Farnham, Harry R. Chesley, Debbie E. McGhee, Reena Kawal, and Jennifer Landau. Structured Online Interactions: Improving the Decision-Making of Small Discussion Groups. In *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, pages 299–308. ACM Press, New York, USA, 2000.
- [20] Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov. Automated Benchmarking of Description Logic Reasoners. In *Proceedings of the 2006 Description Logic Workshop (DL-2006)*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
- [21] Asuncion Gomez-Perez, Oscar Corcho, and Mariano Fernandez-Lopez. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer Berlin Heidelberg, Germany, 2004.
- [22] Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10–15):875–896, 2007.
- [23] Yannis Kalfoglou and Marco Schorlemmer. Ontology Mapping: The state of the art. *Knowledge Engineering Review*, 18(1):1–31, 2003.
- [24] Joel Katzav and Chris Reed. A classification system for arguments. Technical Report, Department of Applied Computing - University of Dundee, 2004.
- [25] Paul Kirschner, Simon Buckingham Shum, and Chad Carr, editors. *Visualizing Argumentation : Software Tools for Collaborative and Educational Sense-Making*. Computer Supported Cooperative Work. Springer-Verlag, London, UK, January 2003.
- [26] Carsten Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics Volume 4*, pages 265–296. King's College Publications, London, UK, 2003.
- [27] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation REC-rdf-primer-20040210, World Wide Web Consortium (W3C), 2004.
- [28] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation REC-owl-features-20040210, World Wide Web Consortium (W3C), 2004.
- [29] Boris Motik and Ulrike Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer, USA, 2006.
- [30] Peter F. Patel-Schneider and Ian Horrocks. OWL 1.1 Web Ontology Language Overview. W3C Member Submission SUBM-owl11-overview-20061219, World Wide Web Consortium (W3C), 2006.
- [31] Chaim Perelman and Lucie Olbrechts-Tyteca. *The New Rhetoric: a treatise on argumentation*. University of Notre Dame Press, Notre Dame, IN, USA, 1969.
- [32] John L. Pollock. Defeasible Reasoning. *Cognitive Science*, 11(4):481–518, 1987.
- [33] John L. Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person*. The MIT Press, Cambridge, MA, USA, 1995.

- [34] Henry Prakken and Giovanni Sartor. Argument-Based Extended Logic Programming with Defeasible Priorities. *Journal of Applied Non-Classical Logics*, 7(1), 1997.
- [35] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation REC-rdf-sparql-query-20080115, World Wide Web Consortium (W3C), 2008.
- [36] Iyad Rahwan. Guest Editorial: Argumentation in Multi-Agent Systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 11(2):115–125, 2005.
- [37] Iyad Rahwan. Mass Argumentation and the Semantic Web. *Journal of Web Semantics*, 6(1), 2008.
- [38] Iyad Rahwan, Fouad Zablith, and Chris Reed. Laying the Foundations for a World Wide Argument Web. *Artificial Intelligence*, 171(10–15):897–921, 2007.
- [39] Chris Reed and Joel Katzav. On argumentation schemes and the natural classification of arguments. *Argumentation*, 18(2):239–259, 2004.
- [40] Chris Reed and Douglas Walton. Towards a formal and implemented model of argumentation schemes in agent communication. In Pavlos Moraitis Iyad Rahwan and Chris Reed, editors, *Argumentation in Multi-Agent Systems: Proceedings of the First International Workshop (ArgMAS'04): Expanded and Invited Contributions*, volume 3366 of *Lecture Notes in Artificial Intelligence*, pages 19–30. Springer-Verlag, Berlin, Germany, 2005.
- [41] Horst W. J. Rittel. Second Generation Design Methods. In Nigel Cross, editor, *Developments in Design Methodology*, pages 317–327. J. Wiley & Sons, Chichester, New York, USA, 1984.
- [42] Glenn Rowe, Fabrizio Macagno, Chris Reed, and Douglas N. Walton. Araucaria as a Tool for Diagramming Arguments in Teaching and Studying Philosophy. *Teaching Philosophy*, 29(2):111–124, 2006.
- [43] Simon Buckingham Shum. Cohere: Towards web 2.0 argumentation. In Anothony Hunter, editor, *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA)*. IOS Press, Amsterdam, The Netherlands, 2008.
- [44] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL Reasoner. *Web Semantics*, 5(2):51–53, 2007.
- [45] Christoph Tempich, Elena Simperl, Markus Luczak, Rudi Studer, and H. Sofia Pinto. Argumentation-Based Ontology Engineering. *IEEE Intelligent Systems*, 22(6):52–59, 2007.
- [46] Paolo Torroni, Marco Gavanelli, and Federico Chesani. Argumentation in the Semantic Web. *IEEE Intelligence Systems*, 22(6):66–74, 2007.
- [47] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.
- [48] Manolis Tzagarakis, Nikos Karousos, Giorgos Gkotsis, Vasilis Kallistros, Spyros Christodoulou, Christos Mettouris, Panagiotis Kyriakou, and Dora Nousia. From Collecting to Deciding: Facilitating the Emergence of Decisions in Argumentative Collaboration. In *Proceedings of the 2nd International Workshop on Building Technology Enhanced Learning Solutions for Communities of Practice*, volume 308 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
- [49] Victoria Uren, Simon Buckingham Shum, Michelle Bachler, and Gangmin Li. Sensemaking Tools for Understanding Research Literatures: Design, Implementation and User Evaluation. *International Journal of Human-Computer Studies*, 64(5):420–445, 2006.
- [50] Frans H. van Eemeren and Rob F. Grootendorst. *Argumentation, Communication and Fallacies: A Pragma-Dialectical Perspective*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1992.

- [51] Frans H. van Eemeren, Rob F. Grootendorst, and Francisca S. Henkemans, editors. *Fundamentals of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Applications*. Lawrence Erlbaum Associates, Hillsdale NJ, USA, 1996.
- [52] Bart Verheij. An argumentation core ontology as the centerpiece of a myriad of argumentation formats. *Agentlink Technical Forum Group*, 2005.
- [53] Max Völkel, Markus Krötzsch, Denny Vrandečić, Heiko Haller, and Rudi Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006*, pages 585–594. ACM Press, New York, NY, USA, 2006.
- [54] Douglas N. Walton. *Argumentation Schemes for Presumptive Reasoning*. Erlbaum, Mahwah NJ, USA, 1996.
- [55] Douglas N. Walton. Argument from Appearance: A New Argumentation Scheme. *Logique et Analyse*, 195:319–340, 2006.
- [56] Douglas N. Walton. *Fundamentals of Critical Argumentation*. Cambridge University Press, New York, USA, 2006.
- [57] Douglas N. Walton. Visualization Tools, Argumentation Schemes and Expert Opinion Evidence in Law. *Law, Probability and Risk*, 6(1–4):119–140, 2007.
- [58] Ingrid Zukerman and Sarah George. A Probabilistic Approach for Argument Interpretation. *User Modeling and User-Adapted Interaction*, 15(1):5–53, 2005.