

# Visualisation and Evaluation of Arguments

BEng Individual Project Report

Chris Pinnick

*Supervisor:* Dr. Francesca Toni

*Second marker:* Prof. Marek Sergot

June 16, 2009

*[www.doc.ic.ac.uk/~cp06/project](http://www.doc.ic.ac.uk/~cp06/project)*

## Abstract

Tools to aid in sensemaking and the analysis of arguments have been growing in number and sophistication over the past decade. At the same time an increasing acknowledgement of the usefulness of argumentation in real-world problems has led to the discovery of applicable methods of evaluating arguments. Until now little has been done to bring these areas together.

Here we present ‘ArgumentSpace’, a collaborative argument visualisation and evaluation tool, combining a simple visual framework for argument design, with the ability to evaluate dialectic validity using the CaSAPI evaluation program developed at Imperial College.

We also explore the use of schemes in argumentation and implement a means of allowing the construction of arguments through informal abductive reasoning schemes. Schemes are defined in a common XML format, and can be dynamically added to ArgumentSpace, providing interoperability with other argumentation tools.

The use of ontologies for shared knowledge representation has increased in popularity in recent years. ArgumentSpace provides the ability to link argument statements to ontological evidence, and evaluates the ontology to check for agreement. To facilitate this, an ontological query engine has been developed for ArgumentSpace which exploits ontology semantics as well as syntax. The results of ontological evaluation are integrated with arguments to produce a statement of validity to user, taking into account both dialectic structure and ontological content.

We perform case studies on the use of ArgumentSpace in two application areas; a problem in the climate change debate, and a legal reasoning scenario involving German family law. Through using ArgumentSpace in these two extended problems, and in conjunction with usability and scalability analysis, we identify both the merits of the system, and areas of further work.

## **Acknowledgements**

I would like to thank my supervisor, Dr. Francesca Toni, for her constant guidance and support during the project. Her motivation and enthusiasm helped me develop an interest in argumentation and maintain focused throughout the project.

I would also like to acknowledge Prof. Marek Sergot, my second marker, and Dorian Gaertner, co-author of CaSAPI, for their constructive comments and assistance.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Motivation . . . . .	5
1.2	Overview of the Report . . . . .	7
<b>2</b>	<b>Background to Argumentation</b>	<b>9</b>
2.1	Argumentation . . . . .	9
2.2	Argumentation Frameworks . . . . .	10
2.3	Notions of Validity . . . . .	13
2.4	CaSAPI . . . . .	14
<b>3</b>	<b>ArgumentSpace</b>	<b>17</b>
3.1	System Introduction . . . . .	17
3.2	A Walkthrough Example . . . . .	18
3.3	ArgumentSpace Concepts . . . . .	21
3.4	Brief Technical Overview . . . . .	22
<b>4</b>	<b>Knowledge Representation and Validity</b>	<b>24</b>
4.1	Argument Representation . . . . .	24
4.2	Debate Evaluation . . . . .	28
4.3	Transformation to ABA . . . . .	30
<b>5</b>	<b>Argument Schemes</b>	<b>35</b>

5.1	Background to Schemes . . . . .	35
5.2	Enhancements to Walton’s Schemes . . . . .	42
5.3	Argument Scheme XML Definitions . . . . .	43
<b>6</b>	<b>Ontologies</b>	<b>44</b>
6.1	Background to Ontologies . . . . .	44
6.2	Ontology Representation and OWL . . . . .	46
6.3	Ontological Reasoning . . . . .	48
6.4	Query Language . . . . .	51
6.5	Combining Ontological Reasoning with the Argument . . . . .	51
<b>7</b>	<b>Issues in Software Engineering</b>	<b>56</b>
7.1	Design Structure . . . . .	56
7.2	CaSAPI as a Server . . . . .	57
7.3	Concurrency and Synchronisation . . . . .	59
7.4	Multithreading . . . . .	61
7.5	Security . . . . .	61
<b>8</b>	<b>Evaluation</b>	<b>64</b>
8.1	Case Study: A Problem in Legal Reasoning . . . . .	64
8.2	Case Study: An Argument on Climate Change . . . . .	72
8.3	Usability Evaluation . . . . .	75
8.4	Scalability and Performance Evaluation . . . . .	82
8.5	Comparison to Existing Tools . . . . .	88
<b>9</b>	<b>Conclusions</b>	<b>97</b>
9.1	Achievements and Contributions of the Project . . . . .	97
9.2	Current Weaknesses and Further Work . . . . .	98

<b>Bibliography</b>	<b>104</b>
<b>A Ontological Query Engine Proofs</b>	<b>105</b>
A.1 Individuals . . . . .	105
A.2 Classes . . . . .	106
<b>B Ontological Query Language</b>	<b>108</b>
<b>C XML Scheme: Argument for Financial Support</b>	<b>109</b>

# Chapter 1

## Introduction

### 1.1 Motivation

Public debate of matters such as climate change, civil liberties and foreign policy decision has been a staple part of society for a long time. People enjoy discussing and arguing about topics which they feel passionately about. In more serious contexts, business analysts, scientific researchers and legal experts use arguments and debate to provide an effective way of making decisions.

Often the topics debated are complex, involving large amounts of information, inferences and general knowledge. Making sense of these arguments and deciding on the reliability of a claim is often very difficult.

Argument visualisation tools help structure arguments in order to make sense of the information by distinguishing statements and connections between them in a visual manner. The user is then more able to follow the arguments made and come to a judgement on any particular claim.

In the following example, an argument is shown in its original dialogue, and visualised (split into statements, inferences and attacks):

John (1): The government should protect its citizens from harm. Smoking tobacco is extremely harmful to the smokers' health, therefore the government should ban smoking.

Chris (2): The government also has an obligation to protect its citizens' freedom of choice, banning smoking would be a breach of this.

John (1): Banning smoking would not take away freedom of choice, since smoking is often not a choice, since nicotine is addictive, once people begin smoking they lose the ability to choose freely.

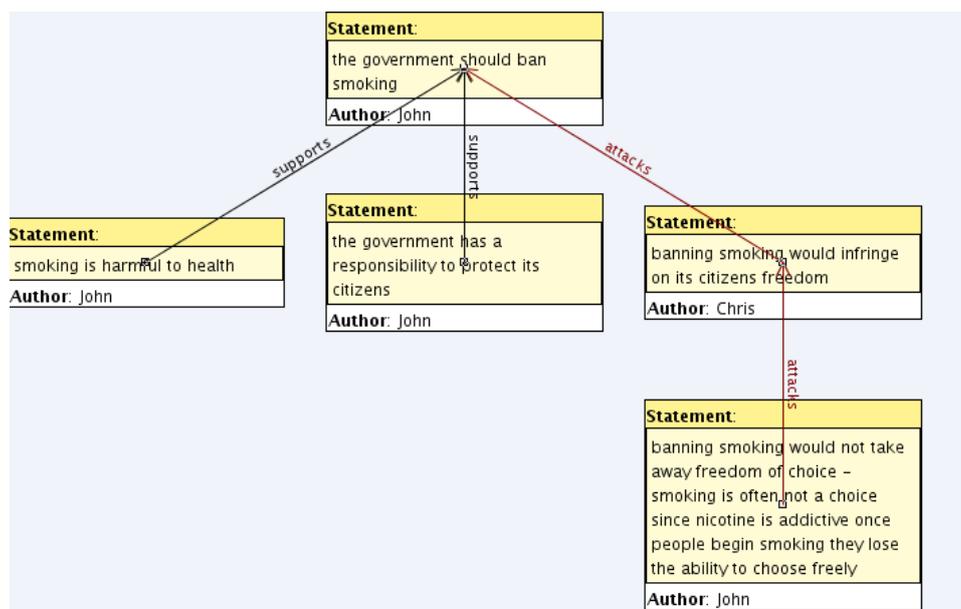


Figure 1.1: Smoking example visualised

It is clear that visualising the argument can assist in understanding it and reasoning about it.

The climate change debate is an area which has gained much recent attention. Despite a near unanimous consensus in the scientific community that global warming is a human driven issue, there still exists a vocal sceptical opinion. This is largely fuelled by sceptic think tanks funded by petroleum

related industry, who have an obvious interest in playing down the effect of CO<sup>2</sup> emissions. The use of sensemaking tools to both educate and aid in decision making on the climate change challenge has been pushed forward recently with the upcoming Essence conference [13]. This conference aims to bring together those involved in climate change issues with the designers of sensemaking tools in preparation for the UN Climate Change Conference in Copenhagen, December 2009. Conferences such as this demonstrate the growing interest and importance of argumentation tools.

In many public and media debates the conclusions made are incorrect, sometimes by accident and later retracted, sometimes intentionally. Often opposing sides of an argument have interests which will cause them to be poorly trusted as judges of the conclusion. For example, when General Motors argues that pollution is not a problem, it is not very credible. It would be very useful to be able to draw conclusion algorithmically, that is, to determine the indisputable validity of an initial claim, given the available knowledge and arguments. A number of argument mapping tools have been developed to assist in sensemaking [18, 19, 26] and much work has been done on determining the validity of arguments [15, 11]. However little has been done to bring these two areas together.

This report presents ArgumentSpace, a distributed multi-user platform for developing and evaluating arguments. ArgumentSpace combines the product of recent research in argument evaluation; the ‘Credulous and Sceptical Argumentation (Prolog Implementation)’ [15] system, developed by D. Gaertner and F. Toni; with a visual argument editor. ArgumentSpace integrates argument schemes [29] as a means of argument design and analysis. ArgumentSpace also uses ontologies in argumentation as a method of supporting statement validity, to achieve this a simple ontological query engine was developed.

## 1.2 Overview of the Report

The report is organised as follows. In chapter 2 we give a brief background on argumentation theory, which is core to ArgumentSpace. In chapter 3 we use describe the ArgumentSpace tool, giving an overview of its components and functionality. In chapter 4 we focus on the visualisation of arguments and their evaluation using CaSAPI, serving as a backend to ArgumentSpace. In chapter 5 we describe the role of schemes in argumentation and how ArgumentSpace implements and extends the concept of schemes into the tool. In chapter 6 we discuss the use of ontologies in argumentation, and the functionality provided by ArgumentSpace in evaluating ontological evidence. In chapter 7 we discuss issues of software engineering which be-

came apparent throughout the design of ArgumentSpace. In chapter 8 ArgumentSpace is evaluated with extended problems in two application areas, climate change and legal reasoning. In addition, the scalability and usability of ArgumentSpace are assessed, and the system is compared to existing tools. Finally in chapter 9 we summarise the achievements of the project and discuss areas in which ArgumentSpace could be improved.

## Chapter 2

# Background to Argumentation

This chapter presents some background on argumentation (section 2.1), focusing on two frameworks of computational argumentation in artificial intelligence, namely abstract argumentation (section 2.2.1) and assumption-based argumentation (section 2.2.2). It also explores several alternative notions of validity of arguments in the two frameworks (section 2.3). It then summarizes the CaSAPI argumentation tool (section 2.4) which is a core component of ArgumentSpace as we will see in chapter 4.

### 2.1 Argumentation

Argumentation is a field of study focusing in the interaction of parties pleading for and against some conclusion. In computer science it is largely concerned with the construction, evaluation and interaction of these arguments automatically. Argumentation has natural links with logic and finds applications in artificial intelligence and multi-agent systems, for example [4, 7]. Argumentation aids in issues of negotiation and resolution of conflict between agents, perhaps possessing differences of opinion [21]. More generally, it provides a means of representing and reasoning with a potentially incomplete and inconsistent knowledge base. As well as in artificial intelligence, argumentation has natural applications in real world problems such as legal reasoning [3] and policy decision making [32].

Argumentation frameworks provide an argument structure based on logical reasoning, formalisms for assessing whether arguments are in fact valid (whether a rational reasoner would accept a given claim, supported or op-

posed by arguments), and provide means to prove a claims validity. Numerous conferences exist driving the field of argumentation. Of particular note are ArgMAS<sup>1</sup> (Argumentation in Multi-Agent Systems), CMNA<sup>2</sup> (Computational Models of Natural Argument) and COMMA<sup>3</sup> (Computational Model of Arguments).

## 2.2 Argumentation Frameworks

### 2.2.1 Abstract Argumentation

Based around the principle that understanding and formalising the structure and acceptability of arguments was essential in leading to computer exchanges of arguments, Dung [11] proposed one of the first argumentation frameworks, which is now referred to as the abstract argumentation framework.

Abstract argumentation is a framework based around atomic arguments, where an argument is not considered to have any internal structure. In abstract argumentation what matters is “conflicts” between arguments, expressed by an attack relationship.

Formally, an abstract argumentation framework is a pair  $AF = \langle AR, attacks \rangle$  where  $AR$  is a set of arguments, and  $attacks$  is a binary relation on arguments.

For example:

$AR = \{ \text{“It will rain tomorrow”}, \text{“It won’t rain for the next 3 days”} \}$   
 $attacks = \{ (\text{“It will rain tomorrow”}, \text{“It won’t rain for the next 3 days”}) \}$

It should be noted that arguments do not necessarily carry any semantic meaning, and could be of a purely abstract form, as in:

$AR = \{ a, b, c \}$   
 $attacks = \{ (b, a), (c, b) \}$

Formally we say:

$x \in AR$  attacks  $y \in AR$  whenever  $(x, y) \in attacks$   
 $x \in AR$  attacks  $S \subseteq AR$  whenever  $x$  attacks some  $y \in S$   
 $S' \subseteq AR$  attacks  $S \subseteq AR$  whenever some  $x \in S'$  attacks  $S$ .

Dung described the principle of validity of arguments in abstract argumentation as “The one who has the last word laughs best” [11]. More formally,

<sup>1</sup>ArgMAS: [homepages.inf.ed.ac.uk/irahwan/argmas](http://homepages.inf.ed.ac.uk/irahwan/argmas)

<sup>2</sup>CMNA: [www.cmna.info](http://www.cmna.info)

<sup>3</sup>COMMA: [www.csc.liv.ac.uk/~comma](http://www.csc.liv.ac.uk/~comma)

in abstract argumentation a set of arguments is deemed acceptable (under admissible belief semantics) if it is able to counter attack all arguments attacking it and it does not attack itself.

In the example above, the set of arguments  $\{a, c\}$  is acceptable.

Criticism of abstract argumentation is based around the lack of structure of its arguments, and in particular the problem this brings in how to use shared content of different abstract arguments and how to identify attacks on an abstract argument.

### 2.2.2 Assumption Based Argumentation

Assumption based argumentation (ABA) [6, 12, 16] was built using ideas of abstract argumentation, but aims to avoid its problems by structuring arguments into a more logical form, based on rules, assumptions and defining attacks based on contrary assumptions. In the assumption based argumentation framework, arguments and attacks are built from:

**Assumptions:** Statements, on which the argument relies, which may be disproved, for example “it will rain”, or of an abstract form, for example  $p, q$ , etc.

**Contraries:** definitions of statements which are in conflict for example the contrary of “it will rain” may be “it won’t rain”.

**Rules:** logical inferences, for example “if it will rain you should take an umbrella” or more formally: “you should take an umbrella”  $\leftarrow$  “it will rain”

Then, arguments and attacks are obtained as follows:

**Arguments:** An argument in favour of a claim  $x$  is supported by a set of assumptions  $X$ , obtained through backwards deduction from  $x$  to  $X$  using the set of rules.

**Attacks:** Attacks within the framework are made by attacking an assumptions on which the argument relies. Specifically by obtaining a counter-argument whose claim is a contrary of an original assumption in support of the argument. Any attack is itself an argument, and so counter attacks follow the same structure.

For example, given the *rules*:

“you should take an umbrella”  $\leftarrow$  “it will rain”.

“it will rain”  $\leftarrow$  “the news forecast rain”.

“the news forecast sunshine”  $\leftarrow$  “the presenter was in a good mood”.

*Assumptions*: “the news forecast rain”, “the presenter was in a good mood”

*Contraries*: “the news forecast rain” is the contrary of “the news forecast sunshine”

An argument in favour of “you should take an umbrella” supported by the assumptions “the news forecast rain” could be obtained by applying the first two rules backwards.

An attack on the argument above would be the argument in favour of “the news forecast sunshine”, supported by the assumption “the presenter was in a good mood”, obtained in a similar fashion.

More formally: ABA frameworks are tuples  $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{C} \rangle$ . where:

- $(\mathcal{L}, \mathcal{R})$  is a deductive system with a language  $\mathcal{L}$  and a set of rules  $\mathcal{R}$  of the form  $x \leftarrow y_1, \dots, y_n$ .
- $\mathcal{A} \subseteq \mathcal{L}$  is a non-empty set of assumptions.
- $C$  is a total mapping from  $\mathcal{A}$  to the power set  $\mathcal{P}(\mathcal{L}) - \{\{\}\}$ , where, for any  $\alpha \in \mathcal{A}$ ,  $C(\alpha)$  is the non-empty set of contraries of  $\alpha$ .

This framework is quite general, allowing proponent-opponent arguments to be structured and evaluated.

An argument  $X \vdash x$  attacks an argument  $Y \vdash y$  if and only if  $x$  is a contrary of some assumption in  $Y$ <sup>4</sup>.

For example, given the following set of rules  $\mathcal{R}$ :

$a \leftarrow b$

$c \leftarrow d$

contrary(b,c)<sup>5</sup>

contrary(d,e)

$\mathcal{A} : \{b, d, e\}$

$\{d\} \vdash c$  attacks  $\{b\} \vdash a$  (“the argument for  $c$ , based on the assumption  $d$ , attacks the argument for  $a$ , based on the assumption  $b$ ”) because contrary(b,c) (“ $c$  is the contrary of  $b$ ”).

---

<sup>4</sup> $X \vdash x$  means the argument with conclusion  $x$ , supported by the set of assumptions  $X$ , obtained using inference rules in  $\mathcal{R}$  backwards.

<sup>5</sup> $C(\alpha) = S$  is represented here and later as contrary( $\alpha, \beta$ ) for each  $\beta \in S$ .

A set of assumptions is **admissible** if it does not attack itself and it counter attacks every set of assumptions attacking it.

A set of assumptions  $A$  attacks a set of assumptions  $B$  whenever there is an argument  $A' \vdash a$  where  $A' \subseteq A$  such that  $a$  is the contrary of an assumption in  $B$ . For example, in the previous framework, the set of assumptions  $\{b, e\}$  is admissible, whereas  $\{b\}$  is not.

## 2.3 Notions of Validity

Both abstract argumentation and ABA can be equipped with other notions of validity of arguments, in addition to the notion of admissibility we have seen before.

A set of assumptions/arguments is:

**Ideal** iff it is admissible and is a subset of all maximally admissible <sup>6</sup> sets.

**Complete** iff it is admissible and it contains all arguments/assumptions (respectively) it can defend by counter attacking all attacks against them.

**Grounded** iff it is minimally complete (it is the smallest set of arguments/assumptions which is complete).

The grounded notion is the most sceptical, ideal is still sceptical, but less so than grounded, and the admissible notion is credulous.

The following example, an abstract argumentation framework, illustrates how admissibility is credulous:

$AR : \{a, b, g, d\}$

$attacks = (a, a), (a, b), (b, a), (g, d), (d, g)$

Seen visually (where edges represent the attacks relation):



Here *admissible* sets are:  $\emptyset, \{b\}, \{d\}, \{g\}, \{b, d\}$  and  $\{b, g\}$ .

Therefore the maximally admissible sets are:  $\{b, d\}$  and  $\{b, g\}$ .

The *ideal* sets (which are admissible and a subset of both maximally admissible sets) are  $\emptyset, \{b\}$ .

---

<sup>6</sup>A set of assumptions/arguments is maximally admissible (also known as ‘preferred’) if no additional item can be added to the set without it loosing admissibility.

The only *grounded* set (minimally complete), is  $\emptyset$ .

To deem a claim acceptable, we need to identify a set of assumptions that is “consistent” (namely that it does not attack itself), includes a core support (for the initial claim) as well as assumptions that “defend” that support, and is acceptable under the chosen semantics.

Based on the three semantics; Grounded Belief semantics (GB), Ideal-Belief semantics (IB) and Admissible-Belief semantics (AB), different computational mechanisms (in the form of dispute derivations) can be defined.

### **Grounded-Belief semantics**

The proponent is not prepared to take any chances and is completely sceptical in the presence of seemingly equivalent alternatives. For a claim to be deemed valid, a *grounded* set of assumptions for the claim (a GB-dispute derivation) must be found.

### **Ideal-Belief Semantics**

The proponent is somewhat sceptical, wary of alternatives, but is prepared to accept common ground between them. For a claim to be deemed valid, an *ideal* set of assumptions for the claim (an IB-dispute derivation) must be found.

### **Admissible-Belief semantics**

The proponent would adopt any alternative that is capable of counter-attacking all attacks without attacking itself. For a claim to be deemed valid, an *admissible* set of assumptions for the claim (an AB-dispute derivation) must be found.

## **2.4 CaSAPI**

The ‘Credulous and Sceptical Argumentation, Prolog Implementation’ (CaS-API) [15, 14] is a Prolog system which computes three types of dispute derivations; GB, IB and AB (as above) under the ABA framework. Users of the system define an ABA framework and a claim which is to be evaluated, CaSAPI then computes a valid set of assumptions (a dispute derivation) under the chosen semantics (if any such set exists).

The process of computing a dispute derivation is performed through a kind of game played by two fictional players: a claim’s proponent, and an opponent trying to undermine the claim.

A simple example follows, where we wish evaluate the acceptability of  $p$ , given the ABA framework below:

$p \leftarrow a$   
 $\neg a \leftarrow b$   
 $\neg b \leftarrow c$

*Assumptions* :  $\{a, b, c\}$

*Contraries* :  $\{\text{contrary}(a, \neg a), \text{contrary}(b, \neg b), \text{contrary}(c, \neg c)\}$

The way in which CaSAPI would compute the AB-dispute derivation can be seen below in the table, which shows at each step  $i$  the sentences held by the proponent and opponent,  $P$  and  $O$  respectively, the set of assumptions  $A$  the proponent is using to support the claim and defend himself, and the set of assumptions  $C$  in the opponents argument which the proponent has chosen to counter-attack. The logic of the game can be understood through the ‘player thoughts’ column.

$i$	$P_i$	$O_i$	$A_i$	$C_i$	Player thoughts
0	$\{p\}$	$\{\}$	$\{\}$	$\{\}$	(prop) I must prove $p$
1	$\{a\}$	$\{\}$	$\{a\}$	$\{\}$	(prop) to prove $p$ , I will assume $\{a\}$
2	$\{\}$	$\{\{\neg a\}\}$	$\{a\}$	$\{\}$	(opp) to undermine the claim, I need to attack $a$ by proving $\neg a$
3	$\{\}$	$\{\{b\}\}$	$\{a\}$	$\{\}$	(opp) which I will do by assuming $b$
4	$\{\neg b\}$	$\{\}$	$\{a\}$	$\{b\}$	(prop) to prove $a$ I need to attack $b$ by proving $\neg b$
5	$\{c\}$	$\{\}$	$\{a, c\}$	$\{b\}$	(prop) which I will do by assuming $c$
6	$\{\}$	$\{\{\neg c\}\}$	$\{a, c\}$	$\{b\}$	(opp) to undermine the claim, I need to attack $c$ by proving $\neg c$
7	$\{\}$	$\{\}$	$\{a, c\}$	$\{b\}$	(opp) which I have no way of doing, proponent wins

By assuming  $\{a, c\}$ , the proponent cannot be defeated, and the claim  $p$  is deemed acceptable under the admissible belief semantics. The set  $\{a, c\}$  is termed the ‘defence set’.

The CaSAPI system has been developed through various versions, with increasing levels of functionality.

- Version 2 computed dispute derivations under GB, IB and AB semantics, but did not compute a dialectical structure (arguments, attacks, and counter arguments).
- Version 3 works with AB semantics, computes a dialectical structure,

but is restricted to use with patient selection functions <sup>7</sup>.

- Version 4 removes the restriction on selection functions of the previous version and works with AB and GB semantics <sup>8</sup>.

Overall CaSAPI provides a means to evaluate arguments described in the ABA framework. Since arguments designed visually in ArgumentSpace can be transformed into the ABA framework (see section 4.3), CaSAPI also provides a means of evaluating arguments described in my framework.

---

<sup>7</sup>Once a player has chosen a potential argument, a sentence from that argument's premises must be selected, which premise is chosen is determined by the selection function.

<sup>8</sup>As of version 4.5.

## Chapter 3

# ArgumentSpace

Models for computational argumentation are not easy to use for non expert people. However, the ability to construct, amend and evaluate arguments, for example in debates, could be useful to many. In this chapter we present ArgumentSpace, an argument visualisation and evaluation tool, through a simple use case which utilises its various features.

### 3.1 System Introduction

ArgumentSpace is a Java based tool providing four main areas of functionality, namely the ability to:

1. Design arguments visually, assisting in the design of well formed arguments with claims and supports.
2. Utilise argument schemes such as ‘Argument through expert opinion’ for the design and analysis of arguments.
3. Support statements with ontological evidence and check the evidence through an ontological query engine.
4. Evaluate the claims of arguments created through any combination of the above and present detailed information on the result of evaluation.

We will describe 2 and 3 in chapters 5 and 6 respectively. 1 and 4 are supported by the assumption based argumentation (ABA) framework described in chapter 2, and are described in chapter 4. In this chapter, we give an overview of the system.

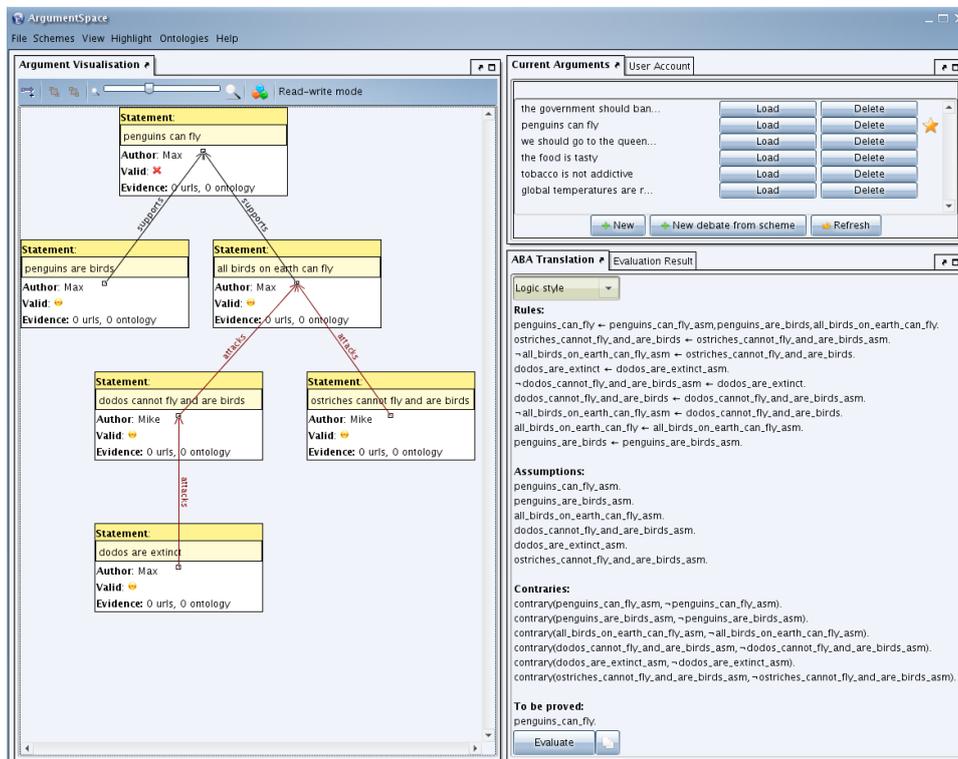


Figure 3.1: ArgumentSpace user interface

Figure 3.1 shows the ArgumentSpace graphical user interface. To the left hand side of the system is the main argument visualisation window ‘Argument Visualisation’, in which a series of statements are shown, connected by arcs showing which statements support or attack others. The window in the upper right hand corner ‘Current Arguments’ shows the debates which can be loaded into the visualisation window, by clicking on the associated load button. In the bottom right ‘ABA translation’ panel, the logical translation of the argument is shown. How these sections of the system are used will be explained throughout the report.

## 3.2 A Walkthrough Example

The best way to become familiar with the overall aims and achievements of the system is to walk through a simple example. Suppose the following argument is put forward:

“We should go to the Queen concert in the park next week because the band is really good, and I know it will be a sunny day because they said so on the BBC news”

This sentence contains a number of assertions, some of which are explicit and others implicit, seen below:

- X: “We should go to the Queen concert”
- Y: “The band is really good”
- Z: “It will be a sunny day”
- Z1: “They said so on the BBC news”
- Z2: “BBC news is a credible expert in meteorology”
- Z3: “Weather forecasts are in the domain meteorology”

As seen Z2 and Z3 are not stated in the quote, they are implicit assertions which are presumed to be agreed. By using a visualisation tool such as ArgumentSpace, the arguer is encouraged to make these assumptions explicit, so that others can identify and challenge them if necessary.

### 3.2.1 Argument design

In ArgumentSpace an argument is designed by making a series of statements, and stating the links between these statements as arcs in the graph. The result as visualised in ArgumentSpace can be seen in figure 3.2.

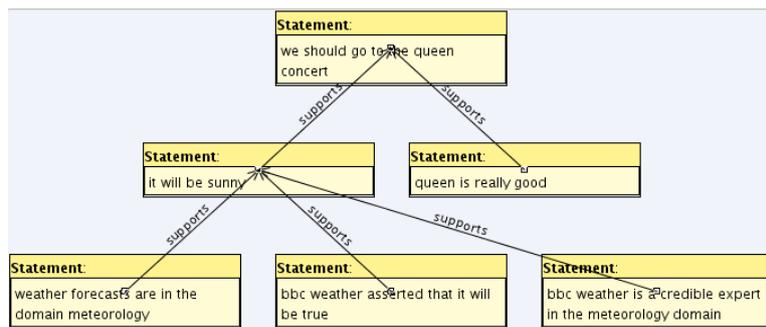


Figure 3.2: Example argument in ArgumentSpace

### 3.2.2 Argument schemes

Statements Z,Z1,Z2 and Z3 form a typical argument scheme, ‘argument through expert opinion’ (as can be seen from figure 3.3), supporting a claim

through a statement from some agreed expert source. Human arguments typically follow common patterns of reasoning, this program allows arguments to be constructed through such schemes (discussed further in chapter 5).

Input through argument Scheme	
<b>Conclusion:</b>	A1 may (plausibly) be taken to be true
<b>Premise:</b>	E1 is an expert of type D1
<b>Premise:</b>	An expert of type D1 should know A1
<b>Premise:</b>	E1 asserts that A1 is known to be true

Figure 3.3: ‘Argument through expert opinion’ viewed in ArgumentSpace

### 3.2.3 Ontologies

Statement Z2, that the BBC news is an expert in meteorology, is one of many types of statement that could be supported by, and evaluated against some agreed ontology. An ontology is just a formalism for holding knowledge, as can be seen from figure 3.4.

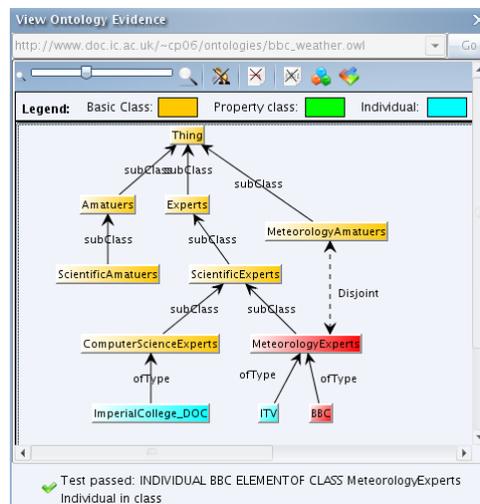


Figure 3.4: Example ontology

As well as visualising ontologies stored in the web ontology format (OWL), ArgumentSpace provides an ontological query engine for checking statements against the ontology. The statement Z2: “BBC weather is a credible expert in the meteorology domain”

Would be equivalent to: “INDIVIDUAL BBC ELEMENT OF CLASS MeteorologyExperts (that is, the BBC is a member of the set of all experts in meteorology).

This statement is then checked against the ontology using the query engine, and the result presented to the user (the development of the ontological query engine is discussed in section 6.3).

### 3.2.4 Validity

Finally, to evaluate the argument as a whole, and the validity of the initial claim (that we should go to the concert), the statement is first converted into a series of logical statements (taking into the account the result of evaluating any statements supported through ontologies), and then passed to the Prolog evaluator, namely an adapted version of CaSAPI.

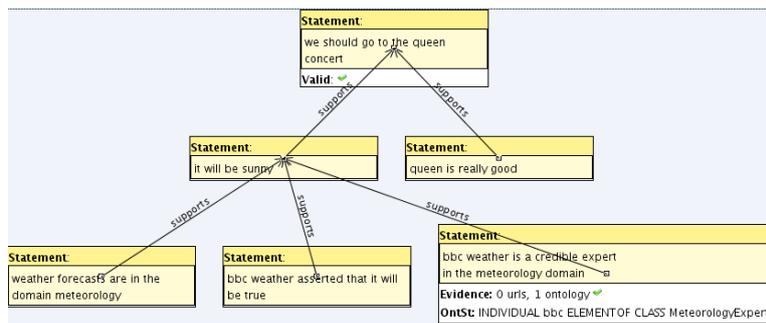


Figure 3.5: Argument evaluation result

The ABA definition of acceptability can be put simply as ‘an argument is valid if it can defend itself from all attacks’. We save the formal discussion of the logic of computing acceptability in *ArgumentSpace* for a later chapter (4). Intuitively, in figure 3.5 there is no counter argument to the statements which form the argument, and the ontological statement is deemed valid (as indicated by the tick in the evidence field). Thus, the initial claim evaluates to true, indicated by the tick next to the ‘valid’ field of the root statement, “we should go to the queen concert”.

## 3.3 ArgumentSpace Concepts

The concepts involved in this project: arguments, schemes, ontologies, ABA representation and acceptability, will be discussed in turn through the fol-

lowing chapters and may be introduced though the following diagram.

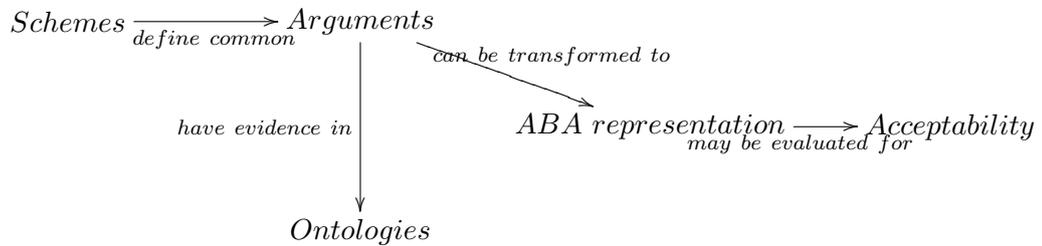


Figure 3.6: ArgumentSpace concepts

### 3.4 Brief Technical Overview

One of the goals of ArgumentSpace is to provide a tool which can be effectively used around the world for collaboration and argument, as such it is designed for distributed use. Java was chosen as the primary implementation language in part due to its platform independence, as well as its support for database and Java-Prolog inter-language communication.

In order that arguments be accessed and collaborated on remotely, they are stored both on the client machine and remotely on a Postgre SQL database server. Any updates to the local argument are reflected immediately in the database, and in any other parties also accessing the argument.

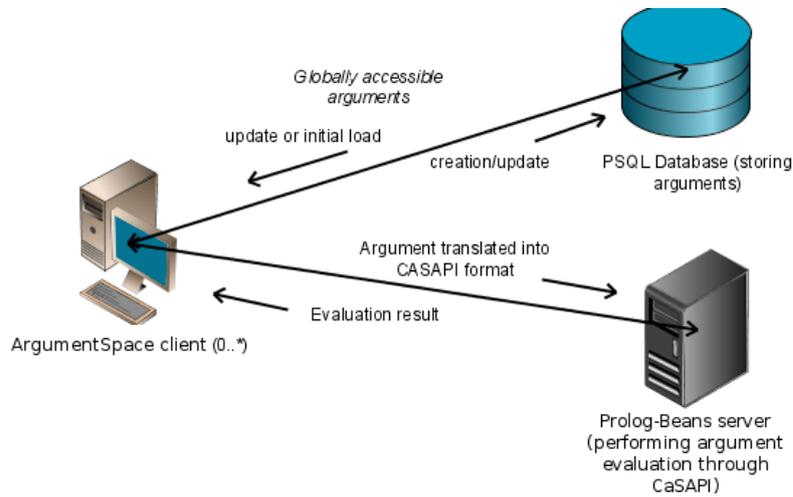


Figure 3.7: System diagram

Debates are evaluated using the CaSAPI Prolog system, to support the Java-Prolog communication the PrologBeans library [25] is used. As well as supporting inter-communication between the two languages, it provided the added advantage of allowing users not to carry Prolog with them, but to instead connect to a single server running the Prolog code.

Visualisation of arguments was developed with use of the JGraph library [1], which although not without its problems (e.g. disconnection from Java Swing, lack of use of typing), did speed up the time to first prototype.

The resulting system uses a combination of technologies and libraries, utilising the advantages of Prolog for logical reasoning, Java for interface design and databases for efficient long-term reliable data storage. Issues of software design as discussed in greater detail in chapter 7.

## Chapter 4

# Knowledge Representation and Validity

In this chapter we discuss the basic argument representation format in `ArgumentSpace`, and how arguments are transformed into the assumption based argumentation (ABA) framework for evaluation. In section 4.1, the visual representation of arguments in `ArgumentSpace` is explained, before discussing in section 4.2 how the acceptability of arguments is evaluated. In section 4.3 we discuss how arguments designed visually are transformed into the ABA framework for evaluation by `CaSAPI`.

### 4.1 Argument Representation

As seen in the background section of the report, there are a number of argumentation frameworks - formalisms for describing and evaluating arguments, including ABA and abstract argumentation.

In line with assumption based argumentation, `ArgumentSpace` allows the use of two types of relationship between statements: ‘supports’ and ‘attacks’. Note that there are other tools that, like `ArgumentSpace` have the ability to visualise statements and the relationships between them. These tools include `Cohere` [18] and `Compendium` [19], which provide an array of different relations between statements. However many of these are analogous, such as ‘challenges’, ‘is inconsistent with’ and ‘refutes’.

These links can be broadly split into just two types, those which show support and those which show disagreement. As such `ArgumentSpace` uses just two types of relation ‘supports’ and ‘attacks’.

Existing tools also often provide neutral relations which describe neither support nor attack of a statement but show ‘some link’. Since the operand relation has no impact on the argument, I decided that the relation was not necessary. Superfluous information in an argument should be avoided, since it can cloud reasoning.

In assistance of the understanding of further sections, we will now formally define the semantics of ArgumentSpace.

- Arguments consist of statements, which are linked by the relations ‘supports’ and ‘attacks’.
- Potential issues with arguments are identified by means of critical questions.

We now demonstrate the concepts of supports, attacks and critical questions in detail.

If **supports(A,b)** where *A* is a set of statements which support statement *b*:

- The proponent believes the validity of statement *b* depends on every statement in *A*, that is, all statements in *A* must be deemed valid in order that statement *b* be deemed valid.
- If we accept all the statements in *A*, this should be sufficient for us to accept *b*.

For example consider supports(A,b) where *b* is “food is tasty”.

*A* is {“food is well cooked”, “food is hot”} states that the food must be both well cooked, and hot, in order to be deemed tasty. This is represented in ArgumentSpace as in figure 4.1a.

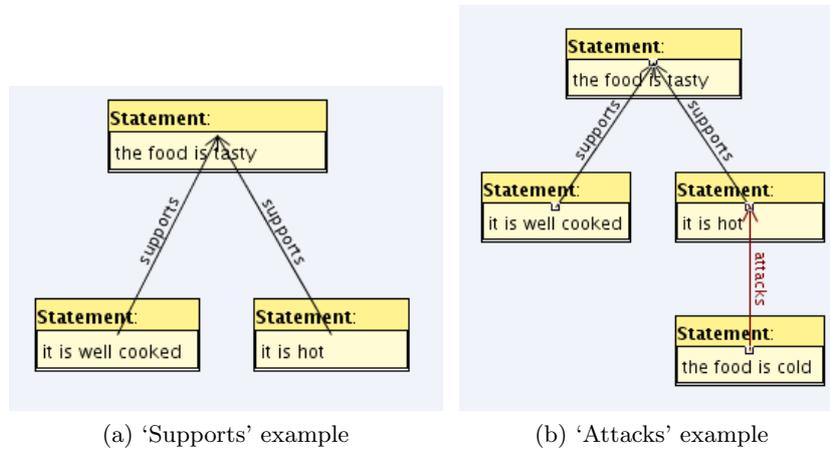


Figure 4.1: Supports and Attacks relation example

If **Attacks(a,b)** where *a* and *b* are arbitrary statements

- Statement *a* is in contradiction with statement *b*, or *a* being true is a reason for *b* not to be deemed valid.

The attacks relation can be seen in figure 4.1b.

Note that attacks in Argumentation can be of two types:

**Undercuts:** Where a statement is in contradiction with an arguments premise. For example given:

argument *A*: “We should go out because it is sunny” the statement: “It is not sunny” is an undercutting attack on argument *A*.

**Rebuttals:** Where a statement is in contradiction with the conclusion, but not any specific premise of an argument, for example:

“We should not go out because I don’t feel well” is a rebuttal attack on argument *A* given earlier.

ABA allows only undercuts. One option in representation in ArgumentSpace allows was to similarly only allow undercuts and force users to model situations requiring rebuttals through the use of an additional premise, as done in 4.3; however this is a much less intuitive solution to those from a non-argumentation background than to allowing direct rebuttals, and as such the attacks relation can be used for both types of attack in ArgumentSpace.

The last type of relationship is that of **Critical questions**. This holds between an additional statement type, also called with an abuse of terminology, critical question, and an ordinary statement. Critical questions have

no effect on the arguments formal acceptability. They are produced when designing an argument from a scheme (see chapter 5) and can also be added by the user. This can be seen as a method of finding weaknesses in an argument, which may be later turned into an attack by the opponent, or prompt the proponent to add an additional premise. Figure 4.2 illustrates the use of a critical question in an argument.

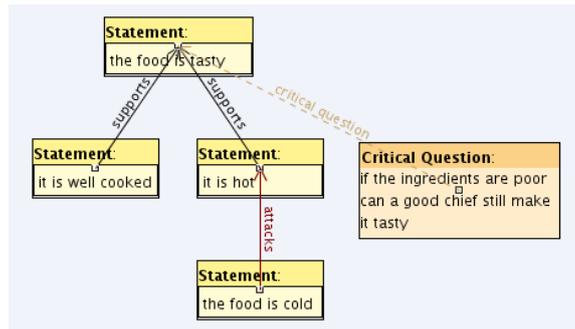


Figure 4.2: Example illustrating the three relation types

#### 4.1.1 Argument visualisation metadata

As has seen through the various examples shown so far, each statement in the argument visualisation can be displayed with varying amounts of metadata. At times the user may wish to display less than the full set of metadata, so that that argument visualisation is more compact. Statements views can be altered for a single statement (as shown in figure 4.3), or all statements.

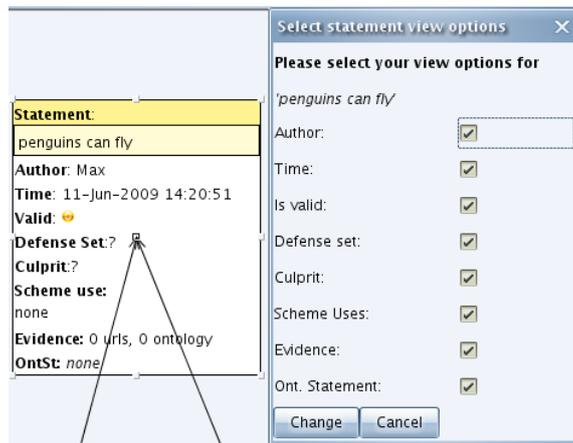


Figure 4.3: View options

The meaning of each of the data items is as follows:

**Author:** The username of the person who created the statement, or last edited it, if any editing has occurred.

**Time:** The time that the statement was created, or last edited, if any editing has occurred.

*The following items are explained in greater detail in later chapters, however for reference their meaning is given:*

**Is Valid:** If the statement has been evaluated this field will show a tick or cross indicating whether the statement is deemed acceptable or not.

**Defense set:** If some claim has been evaluated, this indicates whether or not the statement is required to prove the claims' acceptability.

**Culprit:** If some claim has been evaluated, this field indicates whether or not the statement has been targeted by the proponent for counter-attack, in order to defend the claims' acceptability

**Scheme Uses:** If the statement forms part of an argument scheme, this will be displayed here, such as 'Conclusion of argument from expert opinion'.

**Evidence:** The number of items of evidence which have been attached to support the statement are shown here, and if an ontology and ontological statement has been provided, a tick or cross indicates whether the ontology verifies the ontological statement.

**Ont. Statement:** If an ontological equivalent statement has been associated with the statement, this will be displayed here.

## 4.2 Debate Evaluation

Statements and the relations between them can be visualised in ArgumentSpace as seen previously. It would be useful to be able to determine whether or not, given the statements put forward, a particular statement is acceptable.

When dealing with arguments, there is often uncertainty and inconsistency in the knowledge base, statements put forward cannot be completely trusted, and are often contradictory - from different points of view. To exemplify this point, consider the arguments in figure 4.4.

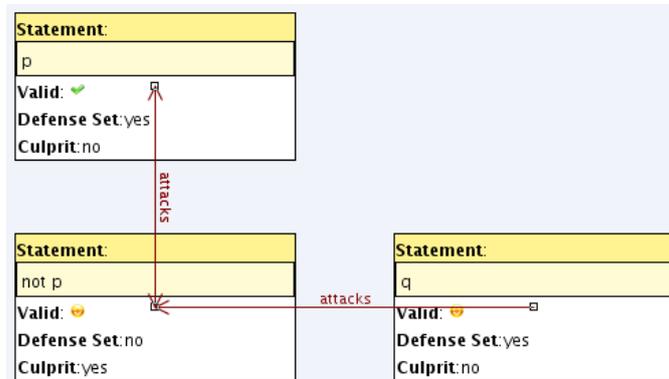


Figure 4.4: Conflicting statements

Should we wish to evaluate the acceptability of  $p$ , traditional first order logic is not suitable; since we have  $p$  and  $\neg p$ , we have  $\perp$ , and from  $\perp$  we have anything. Thus, we can be certain of nothing, and so cannot reason whether  $p$  is acceptable or not.

Since first order logic is not suitable, a new standard of acceptance is needed, argumentation provides this - a means of reasoning with an incomplete and inconsistent knowledge base. In argumentation, arguments are only acceptable on a conditional basis, subject to additional knowledge. Therefore an acceptable argument is often termed a defeasible argument; one in which the conclusion can be accepted tentatively in relation to the evidence known so far in a case, but may need to be retracted as new evidence comes in (or as the other party in the argument responds).

The concept of defeasible arguments is obviously less rigorous than validity in logic. However, it is important in everyday reasoning, and is prominent in legal reasoning, where laws are described in generalisations with exceptions. An argument may be accepted tentatively, until new evidence is seen which shows that a case is in fact an exception to the rule, and the old argument no longer holds.

In argumentation a number of different standards of acceptability exist, even within a particular framework. The standard of acceptance used by ArgumentSpace is that based on admissible belief semantics described in section 2.3. Under the representation framework used in ArgumentSpace, this essentially means that (in the absence of ontological evidence, described later) an argument is acceptable if every attack (or some premise of the attack) is counter attacked.

In figure 4.4, using argumentation, we could reasonably deduce that  $p$  is acceptable and  $\neg p$  is not (if we are willing to assume  $q$ ), since by assuming

$q$ , we should deduce that  $\neg p$  is not acceptable, and so we have no reason not to accept  $p$ . This is computed formally using CaSAPI (described in section 4.2) and the ABA concept of admissibility.

CaSAPI provides a means to evaluate arguments described in the ABA framework. Since arguments described visually in ArgumentSpace can be transformed into ABA frameworks (as we will see in the next section), CaSAPI provides a means of evaluating arguments in ArgumentSpace.

### 4.3 Transformation to ABA

In order to evaluate arguments defined using my visualisation tool, I have implemented a mapping from my visual framework to ABA.

As described previously (in section 2.2.2), an ABA framework is described in terms of rules, assumptions and definitions of contraries of assumptions. The mapping from visual arguments in ArgumentSpace to ABA frameworks is explained initially through a simple example:

#### Example 1: Supports

Assume that the support relation in figure 4.5 is given:

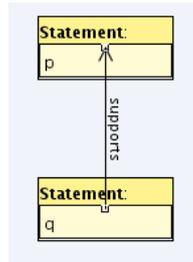


Figure 4.5:  $p$  is supported by  $q$

This can be mapped to the following ABA framework:

ABA Rules:

$p \leftarrow q, p\_asm$  (1)

$q \leftarrow q\_asm$  (2)

Since  $q$  is required to justify  $p$ , we start with rule (1)

Since  $q$  in general may itself require supports (although in this

case it does not), we define another rule (2).

Every statement's translated rule includes a defeasible condition in the form of a corresponding assumption ( $p$  relies on  $p\_asm$ ,  $q$  relies on  $q\_asm$  etc). These defeasible conditions allow us to turn rebuttal attacks described in the visual framework into undercutting attacks in ABA, as we will see in example 2, below. These assumptions can be thought of as saying 'presuming nothing falsifying this rule is found'.

Thus we have the two rules shown above.

Assumptions:  $p\_asm, q\_asm$

Contraries:  $\text{contrary}(q\_asm, \neg q\_asm)$  ,  $\text{contrary}(p\_asm, \neg p\_asm)$

Assumptions and contraries must also be defined. Assumptions are statements which may be subject to disproof.

Contraries define how assumptions can be disproved. We will leave out the assumptions and contraries from future examples, since it is clear from our notation which assumptions and contraries we have - all assumptions end in  $\_asm$  and  $\neg x$  is always contrary to  $x$ .

## Example 2: Attacks

The following example extends the previous, in that an attack is now made on statement  $q$ , through statement  $r$  - as shown in figure 4.6.

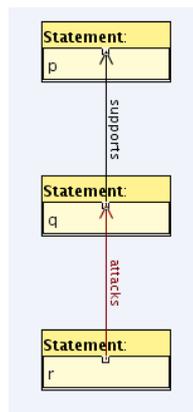


Figure 4.6: An attack is introduced on  $q$

This can be mapped to the following ABA framework:

Rules:

$p \leftarrow q, p\_asm$  (1)

$q \leftarrow q\_asm$  (2)

$\neg q\_asm \leftarrow r$  (3)

$r \leftarrow r\_asm$  (4)

The new attack is represented by the introduction of a new rule (3) whose head is in contradiction with  $q$ 's defeasible condition.

This rule has a body consisting of  $r$ , the attacking statement.

$r$ , like any statement, is defined in terms a rule (4) with all its supports (in this case none), and a defeasible condition to allow counter-attack.

### Formal mapping rules:

Formally, every node (providing support or attack) maps to  $conclusion \leftarrow support_1, \dots, support_n, conclusion\_assumption$

every attacking arc from 'attack' to 'node' maps to:

$\neg conclusion\_asm \leftarrow attack$

Where  $support_1, \dots, support_n$  and  $attack$  are themselves nodes, which are defined recursively as above.

*Critical question nodes and arcs are ignored in this transformation, as they have no effect on the evaluation.*

This is the basic mapping framework. We now present a real-world example:

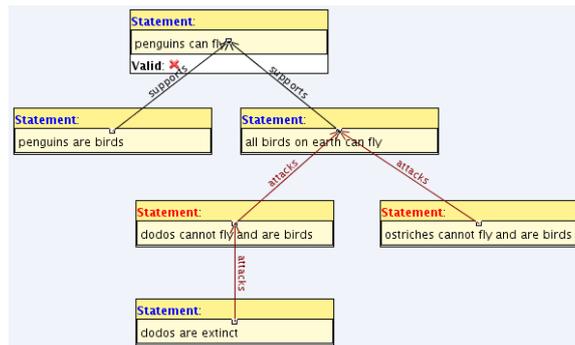


Figure 4.7: Example debate

Above is a debate designed in ArgumentSpace. After evaluation using CaS-API, statements can be highlighted based on whether their assumptions were taken up by the proponent (of the root claim) or opponent. Proponent

statements are blue. Opponent statements are shown in red.

**The translated ABA framework:**

*penguins\_can\_fly* ← *all\_known\_birds\_can\_fly*, *penguins\_are\_birds*,  
*penguins\_can\_fly\_asm*.

*penguins\_are\_birds* ← *penguins\_are\_birds\_asm*.

*all\_known\_birds\_can\_fly* ← *all\_known\_birds\_can\_fly\_asm*.

*¬all\_known\_birds\_can\_fly\_asm* ← *ostriches\_cant\_fly\_and\_are\_a\_bird*.

*ostriches\_cant\_fly\_and\_are\_a\_bird* ←  
*ostriches\_cant\_fly\_and\_are\_a\_bird\_asm*.

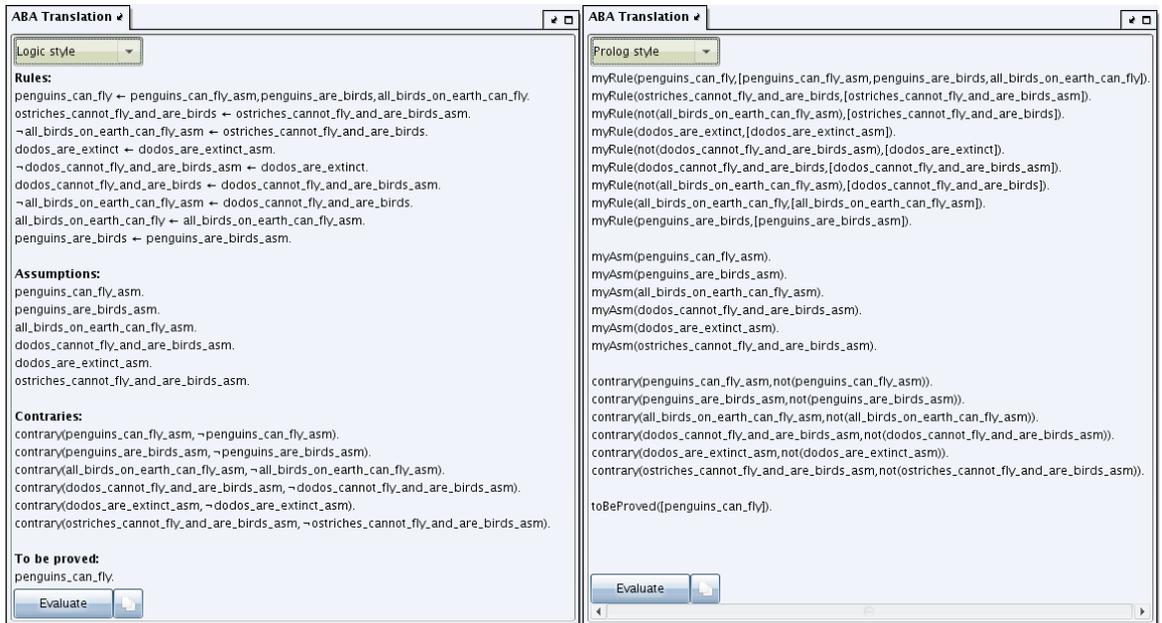
*¬all\_known\_birds\_can\_fly\_asm* ← *dodos\_cant\_fly\_and\_are\_a\_bird*.

*dodos\_cant\_fly\_and\_are\_a\_bird* ← *dodos\_cant\_fly\_and\_are\_a\_bird\_asm*.

*¬dodos\_cant\_fly\_and\_are\_a\_bird\_asm* ← *dodo\_are\_extinct*.

*dodo\_are\_extinct* ← *dodo\_are\_extinct\_asm*.

To evaluate a claim, the user clicks on the statement to be evaluated and then clicks the evaluate button. The argument is then transformed into the ABA framework (as above), which can be viewed in either a logical style (see figure 4.8a) or in its verbatim Prolog code (see figure 4.8b) within ArgumentSpace’s ‘ABA translation’ panel. The results from CaSAPI are sent back to the ArgumentSpace client, and displayed in the argument visualisation and in the ‘Evaluation result’ panel (see figure 4.9). In this case the initial claim is not acceptable.



(a) Logical view of ABA framework

(b) Prolog view of ABA framework

Figure 4.8: ABA translation shown in ArgumentSpace

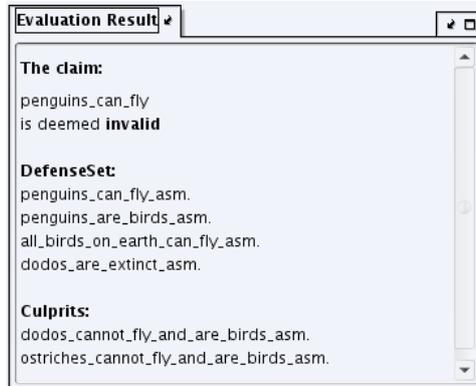


Figure 4.9: Evaluation result panel

## Chapter 5

# Argument Schemes

In this chapter we discuss the use of Argument Schemes (commonly used argument structures) in ArgumentSpace. In section 5.1 we introduce the concept and uses of argument schemes, specifically the identification of enthymemes (‘missing premises’) in section 5.1.1, and the use of critical questions for finding argument weaknesses in section 5.1.2. In section 5.2 the enhancements made to argument scheme definitions specifically in ArgumentSpace are discussed. Finally in section 5.3 the ability to define and load new schemes is explained.

### 5.1 Background to Schemes

Argument schemes are forms of argument, structures of inference which are commonly used in everyday discourse, as well as in more specialist areas such as legal reasoning. Since the publication of D. Walton’s 1996 book [29], they have gained increasing attention and recognition for their merits. Argument schemes are useful for two main reasons:

- They aid in developing arguments backed by sound and commonly accepted reasoning.
- They provide analytical tools with which to analyse arguments and evaluate them critically, by helping identify implicit assumptions and providing common ‘critical questions’ with which to probe the proponents argument for weaknesses.

For example the scheme ‘Argument from Expert Opinion’ is especially important in legal reasoning, where experts in medicine, ballistics and other

areas are often called upon to make statements which are given great importance in the outcome of a case.

Argument from expert opinion

**Premise:** Source E is an expert in domain S.

**Premise:** An expert in domain S should know whether A is true.

**Premise:** E asserts that A is true.

**Conclusion:** A may plausibly be taken to be true.

**Critical question:** Is E credible as an expert source?

**Critical question:** Is A consistent with what other experts assert?

ArgumentSpace provides the ability to browse known argument schemes, and create new arguments using them. Figure 5.1 shows the scheme viewer inspecting the scheme ‘argument from expert opinion’.

The screenshot shows the 'Manage/Inspect Schemes' interface. A dropdown menu lists several argument schemes, with 'Argument from Expert Opinion' selected. Below the menu is a diagram showing a central circle connected to two smaller circles below it. To the right, the scheme's components are listed:

- Conclusion:** a1 may plausibly be taken to be true
- CQ:** is a1 consistent with what other experts of type d1 say
- Premise:** e1 is an expert of type d1  
*[Ontological St: INDIVIDUAL E1 ELEMENT OF CLASS D1]*
- CQ:** is e1 a genuine expert of type d1
- Premise:** an expert of type d1 should know a1
- CQ:** is a1 relevant to the knowledge of d1
- Premise:** e1 asserts that a1 is known to be true
- CQ:** is e1s assertion based on evidence
- CQ:** did e1 really assert that a1 is known to be true

At the bottom, there is a button labeled 'Upload new schemes from XML'.

Figure 5.1: Viewing schemes in ArgumentSpace

The idea of premises and conclusion should be intuitive and is common with the previously mentioned ABA framework. Critical questions will be discussed later.

### 5.1.1 Enthymemes

The term enthymeme refers to an argument in which one or more statements that are part of the argument are not explicitly stated, although in some cases it is the conclusion that is missing [10]. Enthymemes are loosely termed ‘missing premises’.

“The corporate income tax should be abolished; it encourages waste and high prices”

Formally:

Premise: The corporate income tax encourages waste and high prices.

Conclusion: The corporate income tax should be abolished.

In the example above, the missing premises are “In general, whatever encourages waste and high price should be abolished”.

In order to assess the validity of the argument, we need to test whether we agree with each of the premises, but if a premise is missing, then it is much more difficult for a reasoner to assess the argument. For example, we may disagree that high prices always bad, believing in the case of alcohol that high prices are good, because they discourage excessive drinking.

Using the scheme browser in ArgumentSpace, we can identify when a scheme is being used, and so identify missing premises, which can be added to the argument. The above example fits the scheme ‘**Argument from negative consequence**’, seen in figure 5.2.

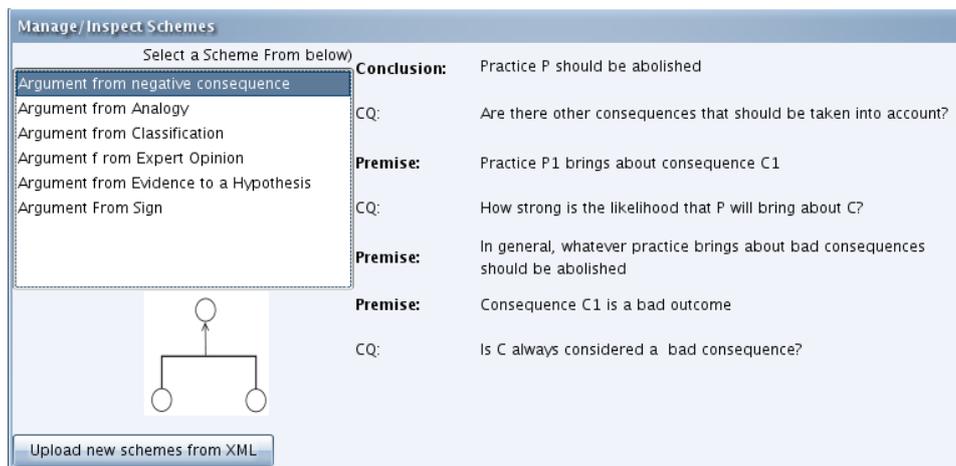


Figure 5.2: Argument from negative consequence

### 5.1.2 Critical Questions

One of the most useful aspects of argument schemes are their associated critical questions. Once a scheme has been identified we can use critical questions to assist the user in accessing whether an argument is valid, whether there is a weakness in a premise or a missing premise which would not hold.

As an example of the use of critical questions:

“James W Johnston, CEO of R.J. Reynolds Tobacco Company, testified before Congress that tobacco is not an addictive substance and that smoking cigarettes does not produce any addiction. Therefore we should believe him and conclude that smoking does not in fact lead to any addiction.”

This is an example of argument from expert opinion (above). There is a weakness in the argument, which can be probed by asking critical questions, suggested by the Argument Scheme:

**Critical question:** Is E credible as an expert source?

**Critical question:** Is A consistent with what other experts asset?

The important critical question here is whether the president of a tobacco company is a credible expert source on issues of health. Although he may be knowledgeable on how to sell tobacco, it is likely he is not a medical expert

and, just as importantly, it is likely he is biased. As such these critical questions need be assessed and, if appropriate, used as an attack on the argument.

In ArgumentSpace, arguments can be inputted from schemes. A scheme is selected from the list of currently available schemes in ArgumentSpace, and the user is provided with the premises, conclusions and critical questions associated with the scheme. The user must input the variables used in the scheme and, once complete, the argument is added to the visual display. Figure 5.3 shows the selection and defining of variables to input an argument from scheme. Figure 5.4 shows the resultant argument.

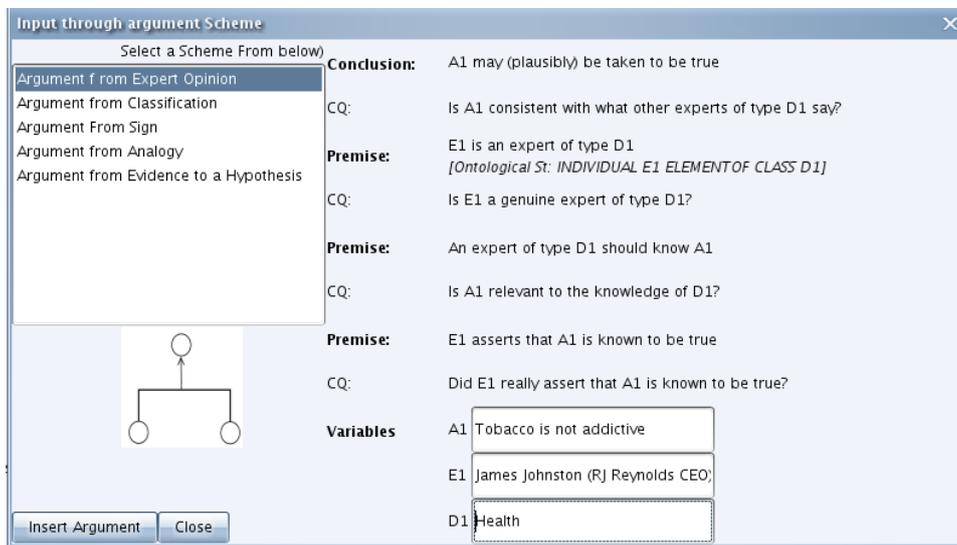


Figure 5.3: Creating the tobacco argument from scheme

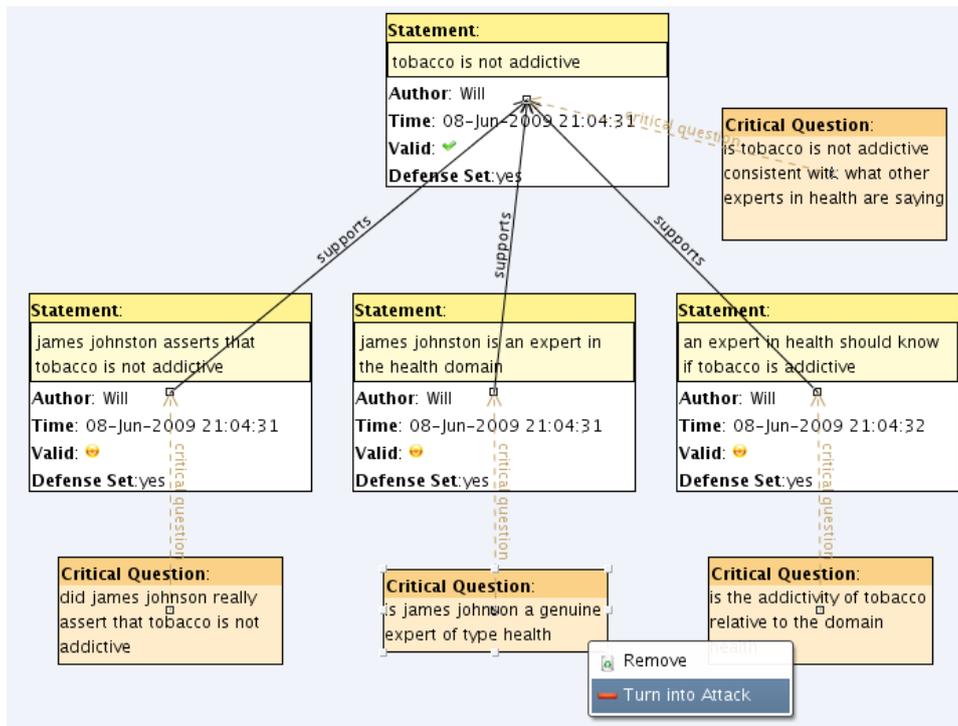


Figure 5.4: Resultant tobacco argument

As well as the conclusion and premise nodes being added as statements to the argument, critical questions are automatically added, as shown in figure 5.4. Critical questions can be characterised into two types:

Those for which the **burden of proof is placed with the opponent** - critical questions which are to be turned into an attack and require pro-activity on behalf of the attacker. For example, consider the critical question: ‘is the author biased’. To turn this into an attack, some reasonable justification would be required (as seen above in figure 5.4). In ArgumentSpace this kind of critical question, where the opponent believes the critical question is relevant and can be justified, can be converted into an attack. In the tobacco example, this conversion is shown in figure 5.5.

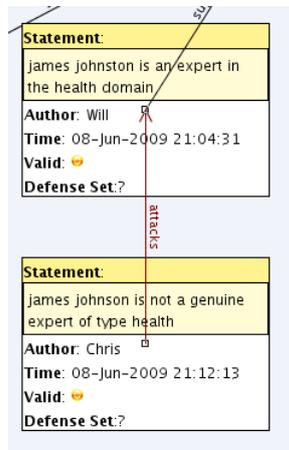


Figure 5.5: Critical question turned into attack

Those for which the **burden of proof is placed with the proponent** - critical questions which essentially ask for an additional premise by the opponent to justify his statement. For example, consider the critical question: ‘is the statement based on any evidence?’. This kind of question should be responded to by the proponent answering the question, justifying himself with an additional premise. In ArgumentSpace this kind of critical question, where the proponent believes the critical question is relevant, can be replaced with an additional premise, as seen in figure 5.6.

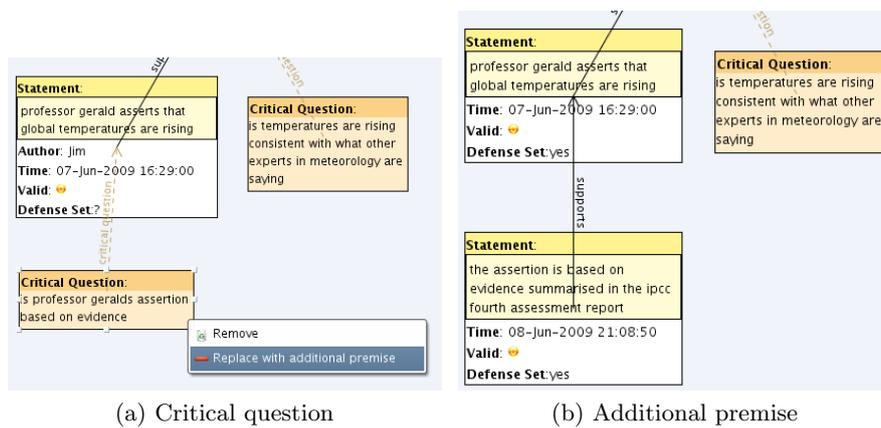


Figure 5.6: Critical question converted into an additional premise

Some might argue that critical questions should simply be another premise in the scheme in the first place, in fact Walton has presented versions of his schemes in which this is the case [10]. However, it is argued that a bal-

ance is needed between having enough core premises so that arguments are plausible and a size of scheme which makes it usable and easy to remember. There are often many exceptions to the rule represented by a scheme, and critical questions highlight exceptions which may or may not be appropriate. In many cases they can be removed from the argument since there is no justification for an attack, or need for an additional premise, as is the final option in ArgumentSpace.

## 5.2 Enhancements to Walton's Schemes

### Critical question association

In the schemes proposed by Walton, critical questions are associated with the whole argument. But it is clear that critical questions are generally either associated with either a particular premise, or the conclusion (in a similar way to attacks in ABA being either on an assumption, or the conclusion through the use of an undercut on the defeasible condition). As such the critical questions found in ArgumentSpace schemes are classified by premise or conclusion. For example in 'Argument from Expert Opinion', the critical question 'Is E credible as an expert source?' is associated with the premise 'Source E is an expert in subject domain S containing proposition A'.

### Critical question classification by burden of proof

As seen in section 5.1.2, critical questions can convey a loose burden on either the proponent or the opponent. The schemes provided in ArgumentSpace are classified into these two types, with different options appropriately; if on the proponent an option to 'replace with additional premise' is presented, if on the opponent an option to 'convert to attack' is presented.

### Ontological equivalent statements

Ontological reasoning (see chapter 6) is a key component of ArgumentSpace. Premises which may be verified through an ontology as evidence come with an associated ontological statement in the query language devised for ArgumentSpace. This encourages the use of ontologies and aids the user in forming ontological statements.

For example:

Premise: E is an Expert in Domain D1

Ontological equivalent statement:  
“INDIVIDUAL E ELEMENTOF CLASS D1”

The ontological statement will likely need adjustment to meet the exact syntax ontological terms, however the suggestions for ontological statement are useful as a starting point for the user.

### 5.3 Argument Scheme XML Definitions

ArgumentSpace has the ability to load new schemes defined in an XML format into the program and onto the database server (an example of this is seen in figure 5.7), including an XML parser for this purpose. Araucaria [26] is another argumentation program designed at the university of Dundee, and also provides the facility to load new XML schemes. ArgumentSpace’s XML format is compatible with that of Araucaria, aiding the sharing of schemes.

In this project, additions were made to the scheme definitions proposed by Walton and used by Araucaria (see section 5.2). The XML format is designed to provide backwards compatibility whilst incorporating these new features.

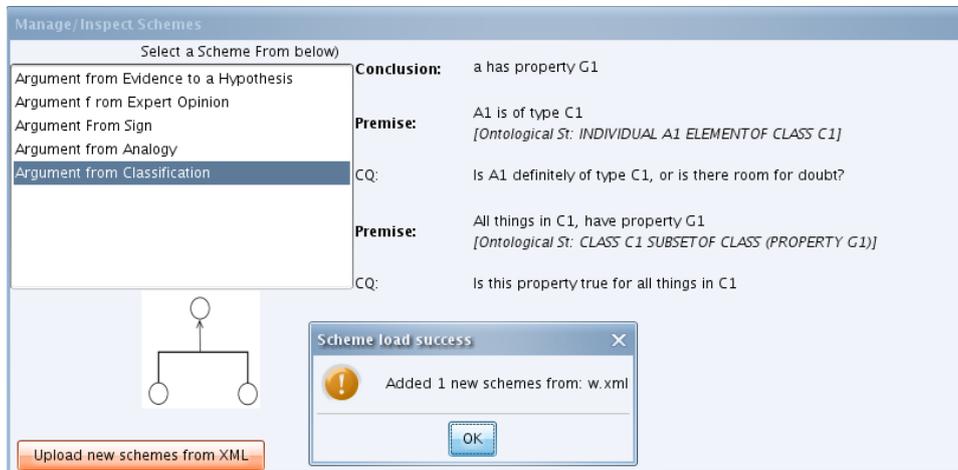


Figure 5.7: User informed of result of scheme load

## Chapter 6

# Ontologies

In this chapter we discuss the use of ontologies in ArgumentSpace. Ontologies and the web ontology language are introduced in section 6.1 and 6.2 respectively. The design of an ontological query engine is discussed in section 6.3. The query language defined for making verifiable statements about ontologies which can be evaluated in ArgumentSpace is discussed in section 6.4. Finally, the way in which results from ontological evaluation are combined within arguments is discussed in section 6.5.

### 6.1 Background to Ontologies

The use of ontologies in argumentation is a relatively new phenomenon. Ontologies, like argumentation, are a form of knowledge representation, used to capture knowledge about some domain of interest. The concept of ontologies has existed since the ancient Greeks, originating from philosophy and metaphysics and referring to the study of nature and the organisation of reality. The computer science definition of ontology is more precise, deriving from a desire to share and reuse information among different people and software. An ontology describes the concepts in a domain and also the relationships that hold between those concepts (such as ‘is a sub-type of’, ‘is an instance of’ etc). An example ontology can be seen in figure 6.1.

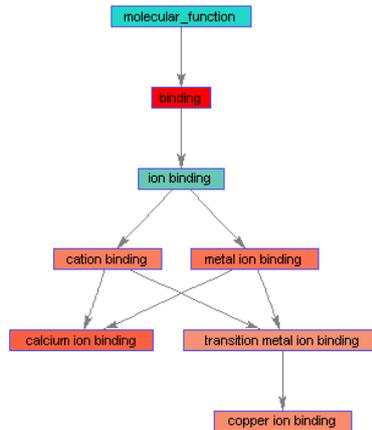


Figure 6.1: Section of the Gene Ontology database

In contrast to argumentation knowledge bases, ontologies must be consistent. If the ontology says classes A and B are distinct, and somewhere else declares an individual which is an instance of both classes A and B, the ontology is inconsistent, and should not be consulted. However, should a consistent ontology exist and be agreed on, it could act as a source of evidence. Statements by the proponent or opponent in an argument, described using argumentation frameworks such as ABA, could be verified against the ontology.

To explain how this might work, we present the following example:

A: Feline cancer of the liver is indicated by the CACNA1G gene; this gene is a T-type calcium channel gene, and other cancers indicated by this gene class have responded well to treatment with the drug Melequine, therefore feline cancer of the liver is also likely to respond to this treatment.

B: No, CACNA1G is a H-type calcium channel gene, so the treatment is unlikely to work.

In this case A's argument is defeated, and would not be deemed acceptable. However, without further information there is no way of knowing whether B's attack is actually correct. If a medical ontology existed (such as in figure 6.1) holding information on gene types, it could be queried to check the result, and either validate or invalidate the statement, as is possible in ArgumentSpace.

## 6.2 Ontology Representation and OWL

Over recent years there has been a move towards a single format for describing ontologies: Web Ontology Format (OWL). Large scale ontologies have been developed in OWL, for example the NASA sweet ontologies<sup>1</sup> and the breast cancer research ontology<sup>2</sup>; however, a variety of other ontology formats are still in use. One of the largest real-world ontologies, the Gene Ontology<sup>3</sup> (containing over 50,000 terms), uses a specific medical format, the OBO (Open Biomedical Ontologies) format, although efforts are underway to provide a mapping between OBO and OWL.

Many modern ontology languages (including OWL) derive their semantics from those of description logic<sup>4</sup>. Description logic is a subset of first-order logic and although various description logics exist, they all share a common set of features: concepts, binary predicates, and constants (no variables are allowed) [2].

OWL is considered one of the fundamental technologies underpinning the Semantic Web, and has attracted both academic and commercial interest. It is a specialisation of XML, and an extension of RDF (the resource description framework), another W3 formalism. RDF specialises XML by standardising meanings for: class, subclass, property, subproperty, domain, range, etc. OWL is a further specialization of RDF; it adds standard meaning for: cardinality, inverse properties, synonyms, and more. An example of an OWL statement is shown below, declaring that the class Men is a subclass of the class People.

```
<rdf:Description rdf:about="#Men">
  <rdfs:subClassOf rdf:resource="#People"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>
```

OWL is actually a family of knowledge representation languages for authoring ontologies including OWL Lite, OWL DL<sup>5</sup>, and OWL Full. OWL light and OWL DL are the main languages, the latter providing a greater degree of expressibility. OWL Full is quite a different language, designed to provide backward compatibility with RDF but possessing slightly different semantics - it is rarely used. In ArgumentSpace we provide support for OWL Lite

---

<sup>1</sup>NASA SWEET ontologies: [sweet.jpl.nasa.gov/ontology](http://sweet.jpl.nasa.gov/ontology)

<sup>2</sup>Cancer Research UK, clinical ontology for breast cancer: [acl.icnet.uk/~mw](http://acl.icnet.uk/~mw)

<sup>3</sup>The Gene Ontology: [www.geneontology.org](http://www.geneontology.org)

<sup>4</sup>A notable exception is CycL, based on first order logic.

<sup>5</sup>The 'DL' is an acronym for description logic.

and OWL DL, although only a subset of their features are used by the query engine we define for validating ontological statements.

ArgumentSpace uses the OWL API [23] developed primarily at the University of Manchester, this provides a parser for OWL DL, outputting an object-based representation of the ontology. ArgumentSpace turns this object representation into a graph for visual inspection by the user. This ontology viewer works in conjunction with the reasoner described in the following section.

Once an ontology is loaded from an internet URL, the viewer (figure 6.2) provides the ability to restrict the information included in the ontology graph (as this information can become quite complex), by removing arcs representing the disjoint property (if many classes are disjoint) or to show only classes and not individuals. Users can also type in the name of a specific class or individual to highlight in the viewer, to help find information in a large ontology (for example in figure 6.2 the user has chosen to highlight the class *ChangesInRisk*, highlighted red in the ontology viewer).

The user might use the viewer to inspect an ontology and decide whether it may support his arbitrary natural language statements, and determine the relevant classes for use in these statements.

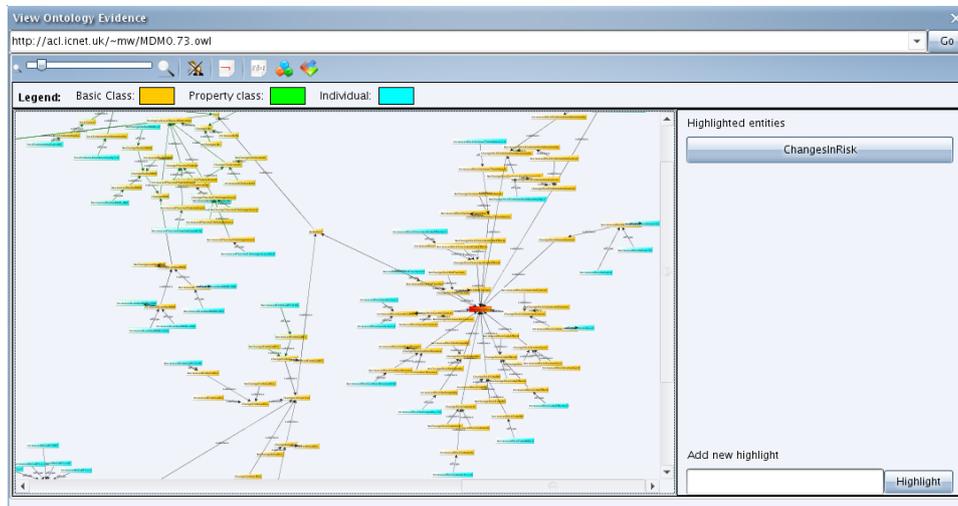


Figure 6.2: Inspecting a complex ontology

After a statement about the ontology has been made, the viewer can be used to provide visual evidence of the evaluation result.

For example, consider the case with the user stating (believing the ontology

to satisfy) that the classes *SociablePerson* and *HardWorkingPerson* are disjoint. The ontological query engine (see section 6.3) may determine that this is incorrect, due to the existence of *fred* belonging to both classes. The ontology viewer in this case will highlight *fred*, so that the user can be assured as to the result (figure 6.3).

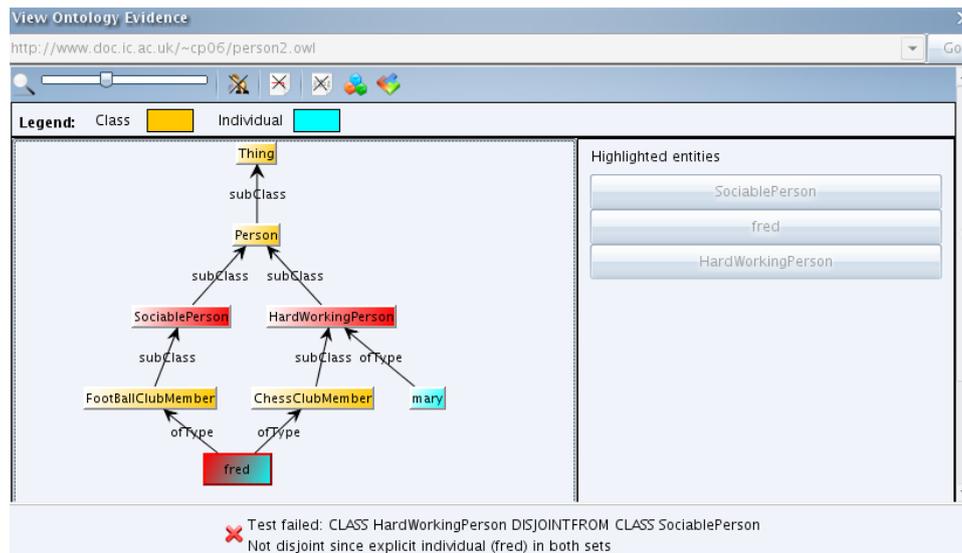


Figure 6.3: Reasoner results and visual proof

## 6.3 Ontological Reasoning

As a result of deriving its semantics from description logic, OWL is based on open world semantics (in contrast to SQL and Prolog which adopt the closed world assumption). Under the open world assumption, if a statement cannot be proved to be true using current knowledge, we cannot draw the conclusion that the statement is false.

For example, if the only assertion about class *ChildrenOfBen* is:

$peter \in ChildrenOfBen$

under the closed world assumption we would infer that Peter is the only child of Ben. However, under the open world assumption, under OWL, we can only infer that Peter is a child of Ben, and there may or may not be other children of Ben that we do not know about.

This allows the progressive development of ontologies, not requiring that all knowledge is present before we begin using the ontology for reasoning. For example, the Gene ontology has been under development since 1998, and is still being added to with new discoveries in bioinformatics.

Since ontologies are not assumed to be complete, the impact on reasoning is:

- Three valued logic is required - a statement may be proved, disproved, or no conclusion may be made (should there be insufficient information to draw conclusion).
- Since ontologies are often incomplete and only contain fragments of information, if for example we wish to know whether two sets A and B are disjoint, we can start by looking to see if 'A disjoint B' has been explicitly declared. But, if it has not, we cannot stop there. There may be other ways to work out whether A and B are disjoint (is an individual declared a member of both? Is one a subset of the other and non-empty?). As we can see there may be a variety of ways of disproving an assumption. And simply checking that no individuals are declared in both (as the definition of disjoint-ness requires), is often not enough.

In order to check the validity of statements about OWL ontologies, an ontological reasoner and query engine was needed. OWL reasoners (such as Pellet [9]) provide a means of traversing the ontology tree, whilst query engines (such as ARQ<sup>6</sup>) provide a means of asking direct questions on the ontology. Pellet was used for visualising ontologies, however after considering the currently available query engines, I decided instead to write a simple engine myself, since those already available were too complex and poorly documented for simple integration with ArgumentSpace.

The requirements for the query engine were:

- Provide the ability to pose statements which could be evaluated to true/false/unknown, based around basic questions of sub-classing, type, and properties.
- Execute on a query language which is simple to use, and could be understood by non-expert users.
- Evaluate queries against OWL ontologies taking into account the semantics of OWL, using disproof techniques as well as looking for explicit assertions.

---

<sup>6</sup>ARQ, a SPARQL query engine for OWL ontologies: [jena.sourceforge.net/ARQ](http://jena.sourceforge.net/ARQ)

As an example of why the third point is needed, we might state in an argument that Diane is happy.

We could check this argument against an ontology by asking: “Is Diane in the class of Happy people?” or in the ArgumentSpace query language: “INDIVIDUAL diane ELEMENTOF CLASS Happy”.

A simple syntactic checker might look for an assertion that *diane* is in the class *Happy*, see that this is not the case, and return unknown. However, using the ontology in figure 6.4, because there is an assertion that *diane* is in the complement class of *Happy*, and we know that nobody is in both *X* and *complement(X)*, we can infer that she is not in class *Happy* and declare the statement false.

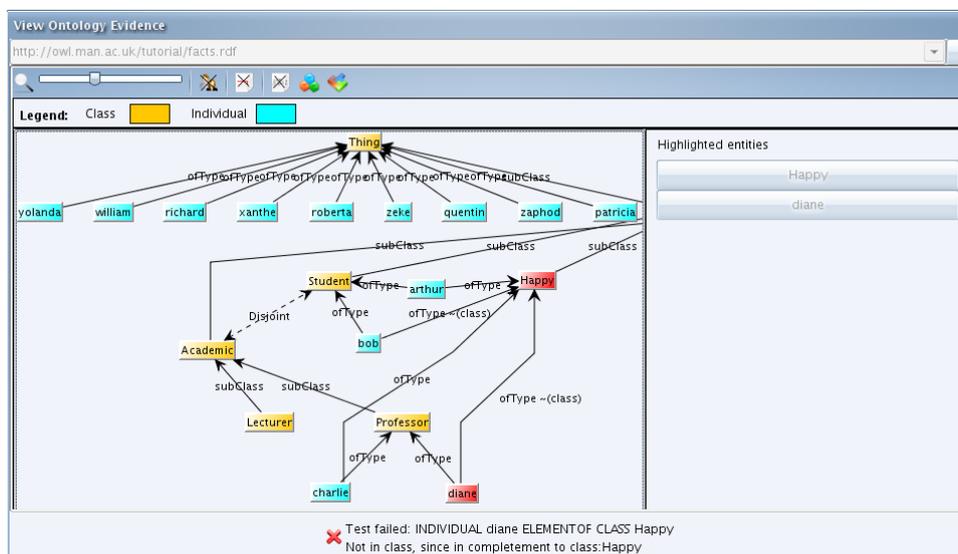


Figure 6.4: Reasoner results and visual proof for a simple ontology

OWL ontologies are often incomplete, and provide situations where we are able to prove or disprove statements by exploiting OWL’s semantics and inferring additional knowledge that is not explicitly stated in the ontology. As such our query engine was developed to work on top of the Pellet reasoner, using various proof rules described in the appendix A.

## 6.4 Query Language

After developing a query engine, a means of describing queries it was needed. At first restricting the options to a selection of common queries was used. But we decided that a query language would be more flexible and allow for future expansion, as well as compound queries constructed using conjunction and disjunction.

Although existing formalisms for querying ontologies exist, in particular SPARQL (a query language for RDF), it was decided that the language is too complex for the needs of ArgumentSpace. For example, asking whether *fred* is in both the classes *HardWorkingPerson* and *SociablePerson*, in SPARQL, would be as follows:

```
ASK { #fred rdf:type #HardWorkingPerson . #fred rdf:type #SociablePerson }
```

Since we believe that the language did not meet the requirement to be easily human readable, a new language was devised.

As an example, the equivalent expression to that above is:

```
INDIVIDUAL fred ELEMENTOF CLASS HardWorkingPerson AND  
INDIVIDUAL fred ELEMENTOF CLASS SociablePerson
```

This simple query language was designed to provide the level of expressibility needed to pose commonly used queries to the reasoner. The parser for this was written using the Antlr compiler generator suite.

The exact syntax of my new query language is defined in appendix B, but in summary, the following operators are supported:  
{*is subset of, is member of, is disjoint from, is empty set, not, and, or*}

## 6.5 Combining Ontological Reasoning with the Argument

The practice of combining ontological reasoning with argumentation is relatively novel, for example see [32].

One difference between existing solutions, such as [32], and mine is that their ‘argument statements’ are statements of description logic proposed automatically, and so could be evaluated on the ontology directly. ArgumentSpace is meant to be used by humans, and the statements are in natural language

and so need to be turned into description logic by the user. Also the ontology they are using as a knowledge base is hand crafted for their purposes and so is relatively complete, meaning the query engine does not need to do so much work to prove or disprove statements. ArgumentSpace’s query engine aims to validate statements wherever possible in an incomplete ontology (see section 6.3).

Although requiring the user to perform the mapping from English to description logic is undesirable, computational mapping from English to logic is as yet an unsolved problem. By making the query language as simple as possible I have aimed to make the translation by hand a simple process for humans.

My approach to including ontology results is simple:

For every node which includes an equivalent ontological statement and some ontologies, each ontological statement is checked in turn:

- If all ontologies validate the statement, or the result of evaluating the statement against the ontology is ‘unknown’, no change to the overall argument is made.
- If any ontology invalidates the statement, a contrary to the statement is added to the overall argument which cannot be countered. Thus, if the overall claim relies on the statement, the initial claim will not be deemed acceptable in the evaluation.

As an example, we present a simple argument, initially with no ontological evidence:

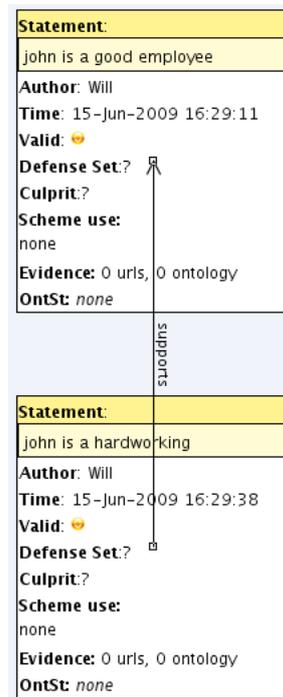


Figure 6.5: Without any ontological evidence, the argument is trivially accepted

This argument can be mapped into ABA, as suggested in chapter 4, as follows:

Rules:

$john\_is\_a\_good\_employee \leftarrow john\_is\_hardworking, john\_is\_a\_good\_employee\_asm$   
 $john\_is\_hardworking \leftarrow john\_is\_hardworking\_asm$

Assumptions:

$john\_is\_hardworking\_asm$   
 $john\_is\_a\_good\_employee\_asm$

Contrary definitions:

$contrary(john\_is\_a\_good\_employee\_asm, not(john\_is\_a\_good\_employee\_asm))$ .  
 $contrary(john\_is\_hardworking\_asm, not(john\_is\_hardworking\_asm))$ .

Given this framework, the claim  $john\_is\_a\_good\_employee$  evaluates to true, as shown by the tick in the valid field on the root claim in figure 6.5.

If we knew of the existence of an ontology which gave evidence to the statement, “john is hardworking” it could be added to the statement, as has been done in figure 6.7. The statement is translated by the user to:

“INDIVIDUAL john ELEMENT OF CLASS HardWorkingPerson”, which is then evaluated separately.

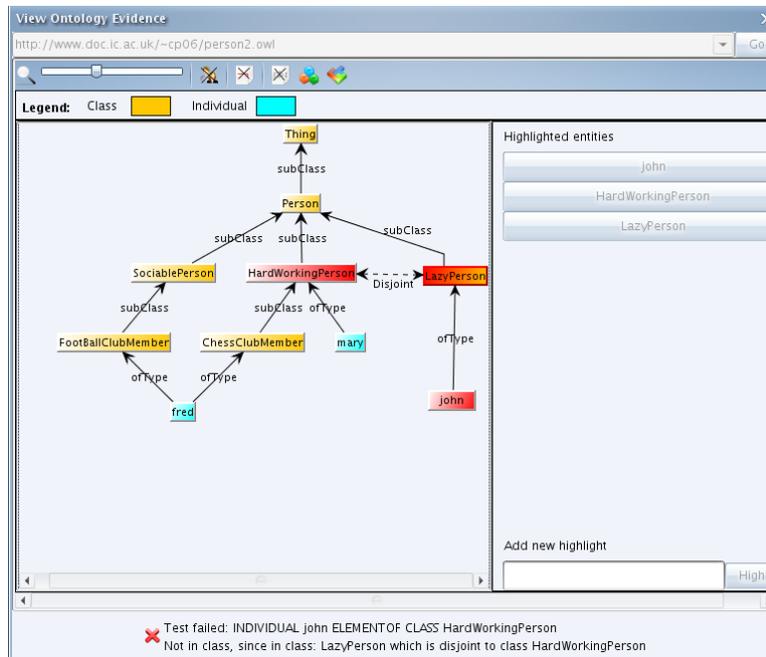


Figure 6.6: Ontological reasoning

However the ontology invalidates the claim, since instead it declares john as a lazy person, and we know there is nobody who is both lazy and hardworking. As a result, an additional rule is added to the earlier ABA translation of the argument:

$$\text{not}(\text{john\_is\_hardworking\_asm}) \leftarrow \text{failed\_ont\_john\_is\_hardworking.}$$

$$\text{failed\_ont\_john\_is\_hardworking.}$$

On evaluation, no admissible set of assumptions can be found by CaSAPI since there is no way in which the ‘failed ontology’ fact can be counter-attacked. The arguments initial claim is therefore not accepted, as shown in figure 6.7.

This simple approach scales well to arguments containing multiple statements backed by ontological support.

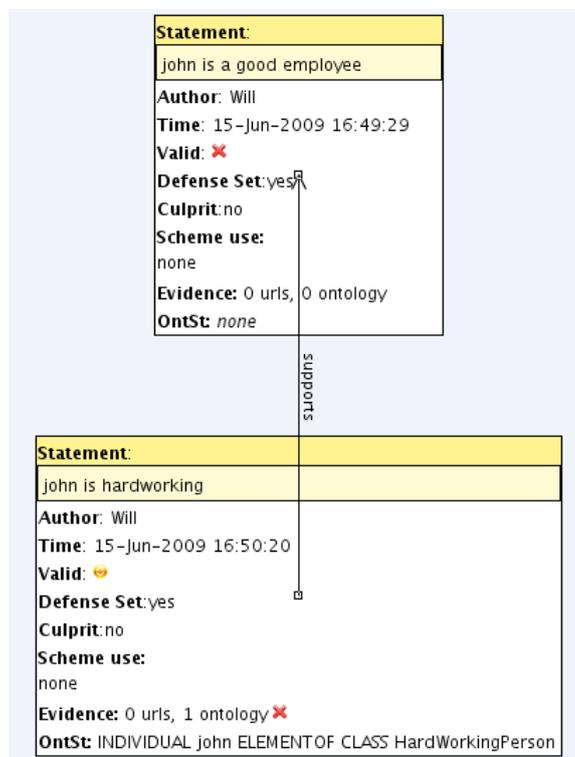


Figure 6.7: Demonstrating the result of additional ontology evidence

## Chapter 7

# Issues in Software Engineering

This chapter discusses some of the design choices and software engineering issues tackled during implementation of the ArgumentSpace system. In section 7.1 we discuss decisions with regard to the program structure. Using the CaSAPI Prolog program as a component of ArgumentSpace provided particular challenges, discussed in section 7.2. The distributed nature of ArgumentSpace provided required concurrency and synchronisation considerations, discussed in section 7.3. The use of multithreading is discussed in 7.4, and finally security issues are discussed in section 7.5.

### 7.1 Design Structure

ArgumentSpace was designed with extensibility and modularity in mind. Thus the functionality of the program was broadly split into different areas and designed with as little coupling as possible to aid modularity.

Since it was clear early on that the argument model would change little throughout development, but the graphic representation might and did change quite radically as new ideas were formed, the design was broadly based on a model-view-controller design pattern. Separating interface considerations from the argument logic stopped alterations in either significantly affecting the other.

Further decoupling was achieved due to the distributed nature of the program (database server, Prolog server, and Java client), this provided natural separation and interface points for different code segments.

Thus the main packages formed were:

- The argument representation model - the internal representation of the currently loaded argument.
- The argument visual view - graphic representation of the argument through an adaptation of the JGraph library.
- Prolog communication code - responsible for translation from the ArgumentSpace internal representation to a series of Prolog clauses, and also handling communication with the Prolog sever.
- Ontological query engine - responsible for parsing and evaluating queries against an ontology.
- Database communication code - responsible for fetching and updating arguments held on the Postgre SQL database.

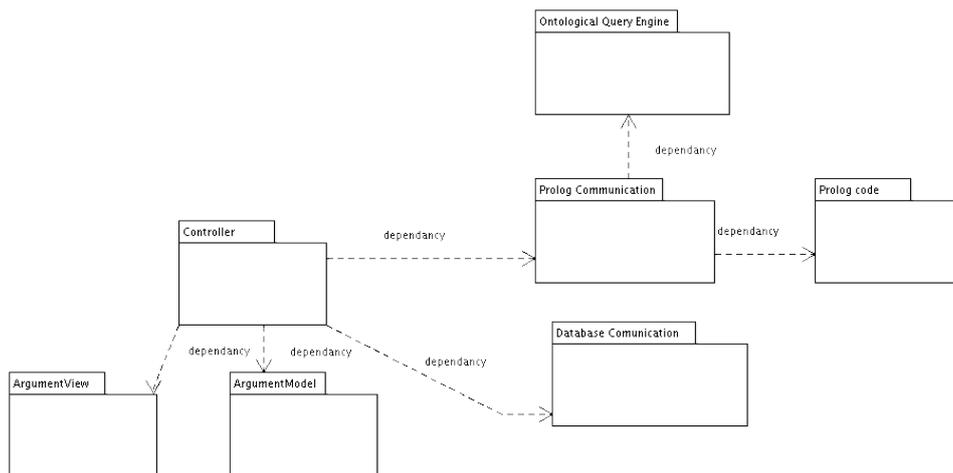


Figure 7.1: UML Package diagram

The aim was to maintain minimal dependencies between these different packages, and this has largely been achieved. Thus the reasoning code, argument or ontology based, was kept separate and only called from a single point.

## 7.2 CaSAPI as a Server

For the CaSAPI evaluation program to be used as a server by distributed ArgumentSpace clients, some modification was required. Sicstus Prolog-

Beans [25] (a Prolog and Java library) was used in order to facilitate Java-Prolog communication. Using this, Prolog queries can be made with bound or unbound variables to a Prolog program, and both the result of the query (pass/fail) and the resultant variable bindings returned to Java.

An alternative was explored with the use of the Sicstus Jasper library [24], which essentially simulates the Prolog interpreter in Java. If this method were used the entire CaSAPI codebase would have to be included with ArgumentSpace. It was decided that this was an inferior solution, since separating the ArgumentSpace and CaSAPI server code provided a number of benefits:

- It reduced the memory and processor footprint required on each client using ArgumentSpace (since the Prolog CaSAPI code would be held separately).
- By separating the dialectic evaluation code from the rest of the code, any changes or improvements could be made to the evaluation code without requiring action by the user.

In order to move from stand-alone program to server, suitable for use in ArgumentSpace, a number of modifications had to be made:

- Previously CaSAPI loaded ABA frameworks from file. CaSAPI had to be changed so that frameworks could be loaded and removed on the server for each client query. This was dealt with through the use of dynamic Prolog clauses.
- Previously CaSAPI would output the intermediary defence sets, culprit nodes etc, when evaluating a claim to screen rather than return these in a variable as part of the Prolog query. This alteration was needed so that results could be returned to Java.
- Previously if a claim was not deemed acceptable, CaSAPI would simply return 'no'. This was altered so that information on the assumptions taken by the proponent and opponent are returned to ArgumentSpace, even in the event that a claim is deemed invalid. This was achieved using extra variable in the main 'evaluate' predicate which was bound to either 'pass' or 'fail' depending on the original CaSAPI predicate result (using negation as failure to complete the Prolog call in the case where the original CaSAPI predicate fails and the claim is not deemed valid).

Although changes were necessary, the aim was to keep these changes as separate as possible from the main CaSAPI code, ensuring any future versions of CaSAPI could be used with ArgumentSpace without major work. This has been achieved, the new 'server' code is held in a new single Prolog module.

### 7.3 Concurrency and Synchronisation

From the onset, a desired requirement of ArgumentSpace was to provide remote collaboration on arguments, allowing one person to contribute to an argument, another to come along and add an attack, a further person to add evidence to support a statement etc. Indeed, this is important to support the sharing of information and to provide a platform of debate.

This presented issues in concurrency and synchronisation:

- If one user edits an argument, all others viewing the argument should see that change (synchronisation).
- Protection was required to stop the situation in which two persons edit an argument simultaneously, one person may not see his update and wonder why (concurrency).

It was decided the simplest solution to meeting the requirements and avoiding the problems, was a course grained locking mechanism on arguments, whereby only a single user could open and edit an argument at any one time, but any number of users may view the argument. The rules implemented were:

- To enter an argument in read-write mode you require a write-lock on the argument. Only a single user can hold the write-lock at a given time.
- When you enter an argument, if no one else has the write-lock, you acquire it. When you switch to another argument or exit the program you loose the write-lock.
- If you enter an argument and someone else already has the write-lock, you can choose to enter in read-only mode. In this mode the argument is periodically updated to reflect any changes made by the owner of the write-lock.
- To prevent users from holding the lock on an argument indefinitely, a time-out is implemented, whereby if a user holding a write-lock on an argument is inactive for 30 minutes, he is informed and switched to read-only mode.

Figures 7.2, 7.3 and 7.4 illustrate how the system interacts with users to provide collaboration and deal with the issues described above.

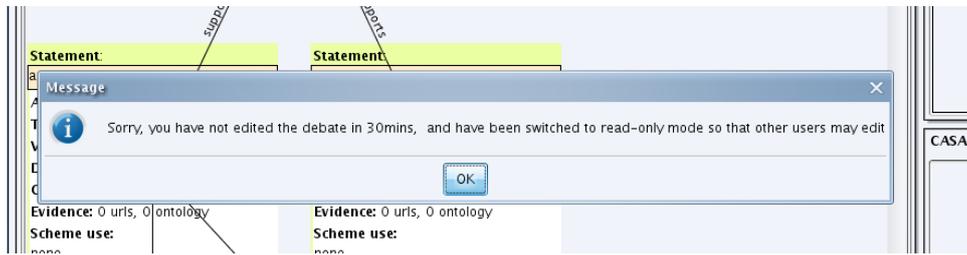


Figure 7.2: User informed of inactivity timeout

Although this mechanism could be improved, since it is not envisaged that the time between different users collaborating is instantaneous, we believe the solution's complexity is appropriate to the needs of the system.



Figure 7.3: Opening a locked argument

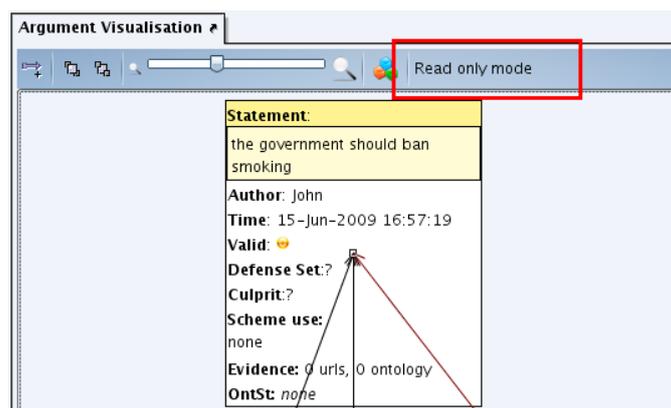


Figure 7.4: Read-only mode indicator

## 7.4 Multithreading

The evaluation of arguments either using CaSAPI, or using the ontological query engine is a computationally intensive process, and in the case of CaSAPI suffers from network delays. To avoid ‘freezes’ to the user when conducting evaluation or when viewing evidence (which involves viewing evaluation results for evidence), the use of multithreading was employed, described visually in figure 7.5.



Figure 7.5: Thread diagram

Evaluation code is conducted on a separate thread, so that the rest of the program can maintain responsiveness, and a load indicator is used to inform the user that evaluation is in progress (this is shown in figure 7.6).

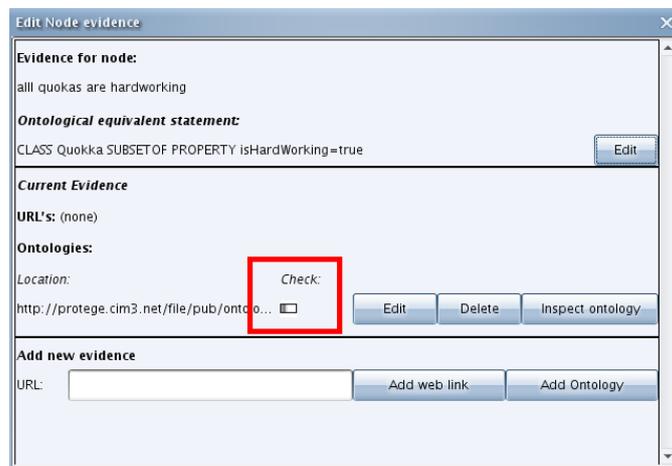


Figure 7.6: Evaluation in progress indicator - whilst evaluation occurs in separate thread

## 7.5 Security

Two security issues were considered in the design of ArgumentSpace: the confidentiality of information discussed, and the authentication of users, as a means to ensuring the non-repudiation of statements.

It is envisaged that ArgumentSpace may be used by people who do not wish for the contents of their arguments to be available to anyone logging into the system. To address this issue ArgumentSpace users can decide on the database server storing their arguments, and the location of the CaSAPI server where arguments are sent for evaluation (see figure 7.7). Should a group of users wish to use ArgumentSpace to discuss arguments confidentially, they can simply use their own database server, and set ArgumentSpace to connect this. In this setup, outside users would be unable to access the information discussed, without knowing the address, username and password of the database.

The alternative would have been to provide security restrictions on individual arguments within ArgumentSpace. However, since security can never be absolute, placing the burden of security on the owner of the database server was the preferred option. Thus, the individual user, or group of users, can put in place the physical and virtual security restrictions to suit their needs.

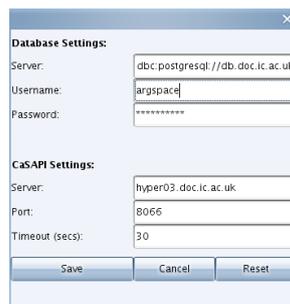


Figure 7.7: Server settings

The second security concern was with regard to ensuring non-repudiation of information, that is, being able to trust who wrote a particular statement, and that person not being able to deny it. The result is that in ArgumentSpace, on each statement, the time and user who last edited it is recorded and displayed, as shown in figure 7.8. To provide this, authentication of users was required.

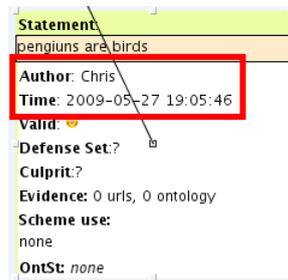


Figure 7.8: Author information

When a user registers to use ArgumentSpace, they provide a unique username and associated password (see figure 7.9), and every time they login to view or edit an argument these credentials must be supplied for authentication. In order to minimise security risks, passwords are never transmitted in plaintext form. The Secure Hash Algorithm (SHA) [20] is used to encrypt passwords at both the database server and client.

- On registration, the password is encrypted at the clientside, before being sent in encrypted form to the database server where it is stored encrypted.
- When the user attempts to authenticate himself, the supplied password is encrypted again with SHA, sent encrypted, and then checked for equality with the encrypted version stored on the database server.

This ensures the password is never transferred unencrypted, in a form which if intercepted can easily be identified and re-used.

A screenshot of a web browser window showing a registration form. The browser's address bar shows "Community Arguments" and "Connection Panel". The form is titled "Register your details" and contains three input fields: "Username:", "Password:", and "Email, optional - for password resets:". Below the input fields are two buttons: "Register" and "Cancel Registration, Login".

Figure 7.9: Registration

## Chapter 8

# Evaluation

In this chapter we evaluate the project and ArgumentSpace to determine its successes and weaknesses, with a consideration for existing work. In section 8.1 we assess ArgumentSpace through an extended example in the legal domain. In section 8.2 we present a debate around the issue of climate change, and discuss the ability of ArgumentSpace to express arguments in this context. Usability issues relevant to ArgumentSpace are evaluated in section 8.3, and the scalability of ArgumentSpace is assessed in section 8.4. Finally in section 8.5 the system is compared to existing tools.

### 8.1 Case Study: A Problem in Legal Reasoning

One particular area where argumentation is of special importance is legal reasoning [5, 3]. A legal case typically centres on a conflict between two parties and is resolved by each side producing arguments in an effort to persuade the judge that their side is right. Modelling legal reasoning is to a large extent analogous to modelling arguments.

In this section we will evaluate ArgumentSpace by considering its ability to represent knowledge in a legal context, specifically by exploring an example adapted from that presented in [3], centered around a fragment of German Family Law. We present the relevant legal statutes and background to the case in section 8.1.1. Then we show how the arguments of the case could be modelled and evaluated in ArgumentSpace in section 8.1.2 and evaluate the model in section 8.1.3. Next we attempt to define an argument scheme of a valid argument by the proponent for the case in section 8.1.4, and evaluate the scheme in section 8.1.5.

### 8.1.1 Legal source

The example presented is a case involving whether or not an individual is obliged to support a needy family member financially. The law is based around the principle that family should support the needy before the state does. The German law under consideration defines when a family member is obliged to support another family member.

#### **The statutes**

The statutes which define this area of law are:

- 1601 (Support Obligations) Relatives in direct lineage are obliged to support each other.
- 1589 (Direct Lineage) A relative is in direct lineage if he is a descendant or ancestor. For example, children, grandchildren, parents, and grandparents are in direct lineage.
- 1602 (Neediness) Only needy persons are entitled to support by family members. A person is needy only if unable to support himself.
- 1603 (Capacity to Provide Support) A person is not obligated to support relatives if he does not have the capacity to support others, taking into consideration his income and assets as well as his own reasonable living expenses.
- 1611a (Neediness Caused by Own Immoral Behavior) A needy person is not entitled to support from family members if his neediness was caused by his own immoral behavior, such as gambling, alcoholism, drug abuse or an aversion to work.

#### **The case summary**

John has been unemployed for 3 months, and wishes to claim state benefits. The government benefits agency has identified Antony, John's grandfather, as his only living relative. Antony has been ordered to support John; however Antony no longer talks to John, and takes the case to court, arguing that he should not be obliged to provide support.

### 8.1.2 The arguments modelled

We show below how the government agency (referred to as ‘the agency’ from here-on) and Antony take turns to put forward their arguments in the case, and how an equivalent model is developed in ArgumentSpace.

#### The agency’s initial case

The agency (1): *“It is clear that from statutes 1601,1589, 1603 and 1611 that a person is obliged to support his relatives in direct lineage, if they are in need, and the person has the capacity to provide support. Antony is the grandfather of John, who is clearly in need of support due to his desire to claim benefits, and there is no evidence that John’s neediness has been caused by immoral behaviour. Therefore we argue that Antony is obliged to support John.”*

This can be visualised in figure 8.1.

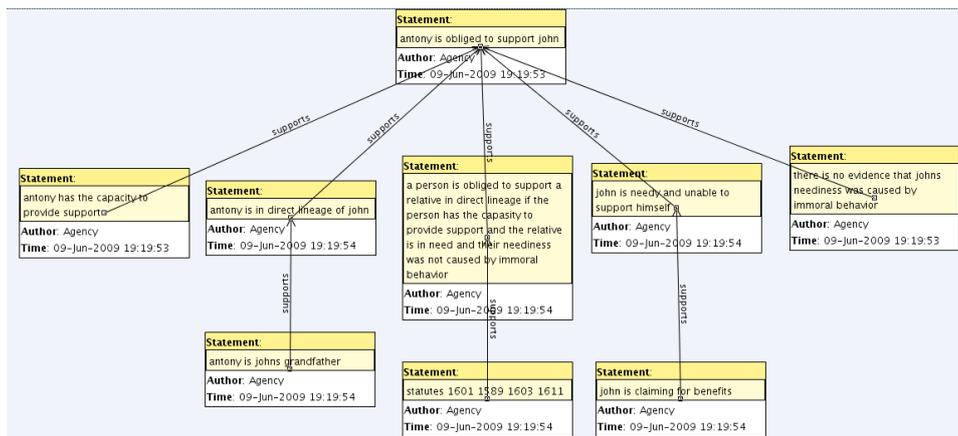


Figure 8.1: The agency’s initial argument visualised in ArgumentSpace

The conclusion of the agency’s argument is clearly that ‘Antony is obliged to support John’. This has been supported by a number of premises; a generalisation justified by the various relevant legal statutes, as well as the premises which enable the generalisation to apply.

## Antony's defense

Antony (1): *“John’s state was caused by alcoholism and gambling, this immoral behaviour makes him disqualified from my support”*

Antony (2): *“Regardless, I do not have the capacity to provide support, my pension is very little”*

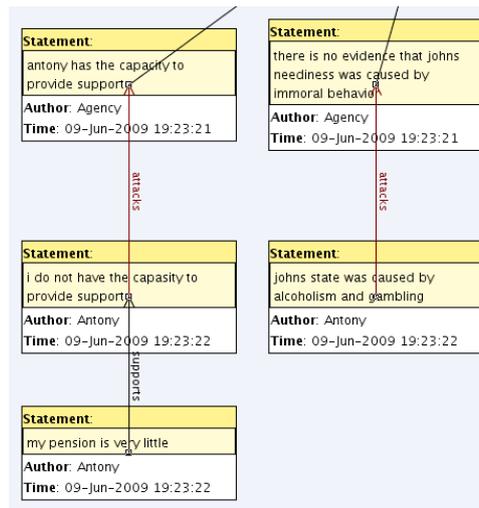


Figure 8.2: Antony's defense visualised (only partial argument graph shown)

We model Antony's defense as two new statements, each of which is an attack on a premise of the agency's argument (an undercutting attack). The second statement that he does not have the money to help John is supported by the premise that his 'pension is very little'. This is visualised in figure 8.2.

## The agency's response

The agency (2): *“Antony does have the capacity to provide support, he drives a Maserati showing he has plenty of money”*

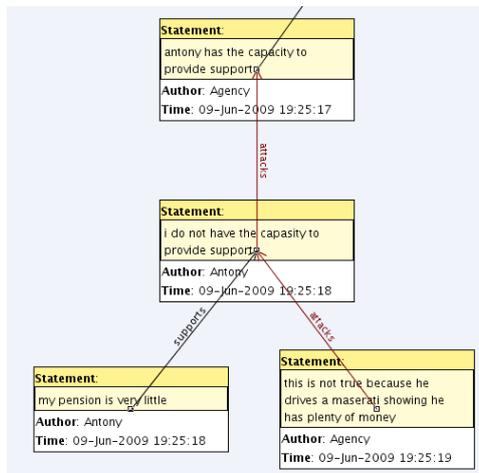


Figure 8.3: The agency’s counter argument visualised

We model the agency’s final response as a new attack, a rebuttal on the claim that Antony does not have the capacity to provide support (see figure 8.3).

Alternatively, this could have been modelled as two statements, ‘he does have capacity to support’, supported by ‘he drives a Maserati’. The validity would have been equivalent and so for conciseness only a single statement was modelled.

### 8.1.3 The resulting model and evaluation

Figure 8.4 shows the full debate and resulting evaluation.

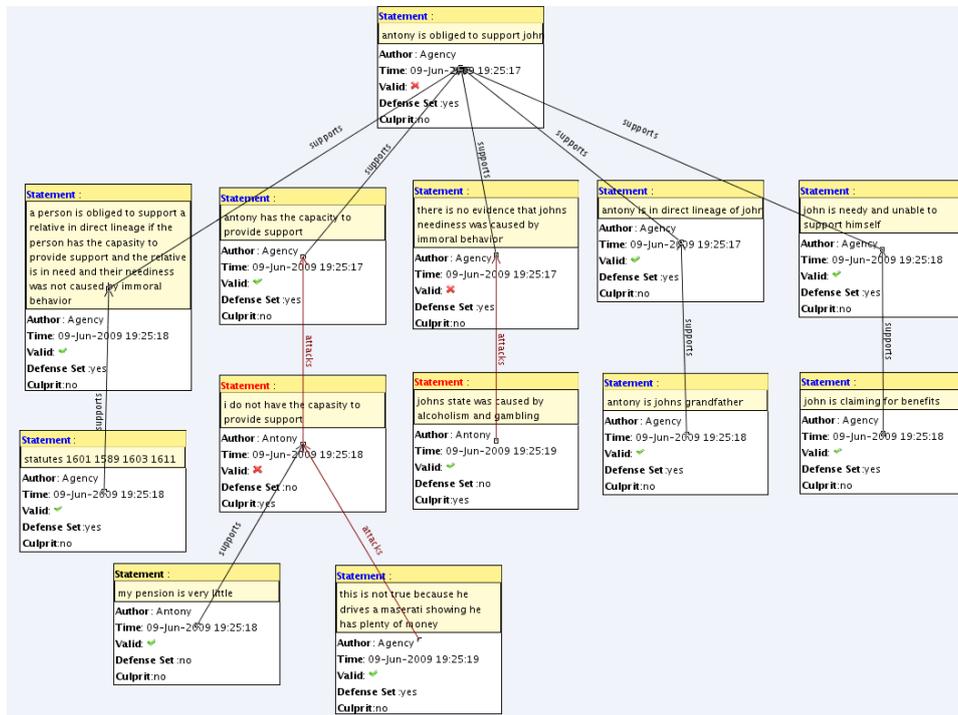


Figure 8.4: The resulting debate visualised

The decision by the judge is: *“My verdict is that Antony is not obliged to support John, although I believe he has the capacity to provide support, John’s immoral behaviour disqualifies him from support through family members.”*

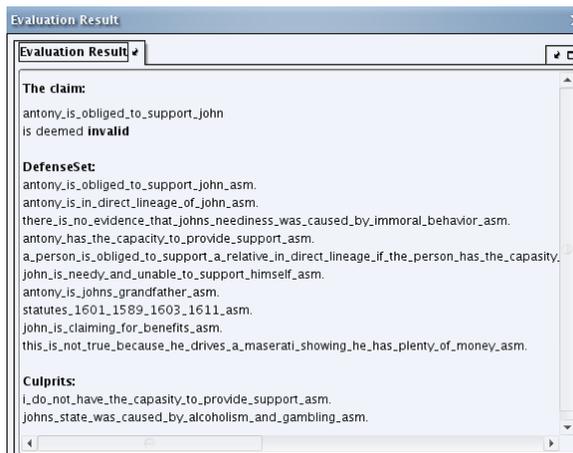


Figure 8.5: Result of evaluation in ArgumentSpace

The judge's verdict is also found through evaluation of the ArgumentSpace model, as seen by the valid field in the root statement in figure 8.4 and in the detailed results window (figure 8.5). Intuitively this is the correct conclusion.

Although the agency succeeds in counter-attacking the claim that Antony does not have enough money to support John, they have no way of counter-attacking the claim that John's poor state arose from immoral behavior. Thus, an attack is left unanswered on the initial claim that John is obliged to provide support, and the claim is deemed unacceptable.

The statement that Antony is the grandfather of John, or that Antony's pension is very little, could have been held and verified in an ontology (although this was not the case in our example) by ArgumentSpace. However the key claim that John's state was caused by immoral behaviour is a kind of claim which would be very difficult to verify automatically. In reality it might require an analysis of John's character and interviews with witnesses in order to arise at a subjective judgement, a complex task likely beyond the realm of any current computer program.

Overall, ArgumentSpace has provided an effective means of modelling and evaluating the argument in this case. It should be noted however that a judgement would have required more than assessment of dialectic validity, but analysis of the validity of the underlying premises, something ArgumentSpace attempts to assist with through the use of ontologies. Whilst acknowledging that many situations cannot be verified with ontologies, verifying an arbitrary statement is well beyond the scope of this project.

#### 8.1.4 A valid case argument as a scheme

Based on the argument used by the proponent (the Agency) in the previous example, and the attacks which eventually lead to the case being rejected, a scheme can be developed. The scheme will represent a valid argument leading to the conclusion that an individual should pay another support (under German law).

##### **'Argument for financial support':**

Conclusion: A is obliged to support B.

Premises:

A is in direct lineage of B.

*CQ: How are they in direct lineage?*

B is needy, he is unable to support himself.

*CQ: What proof is there that B is unable to support himself?*

A has the capacity to provide support.

*CQ: Are there any factors which show he is unable to provide support?*

The neediness of B was not caused by immoral behavior

*CQ: Was B's neediness caused by alcoholism or other immoral behavior?*

This scheme states that A is obliged to support relative B, if B is needy, A has the capacity to provide support, they are in direct lineage, and B's state was not caused by immoral behaviour.

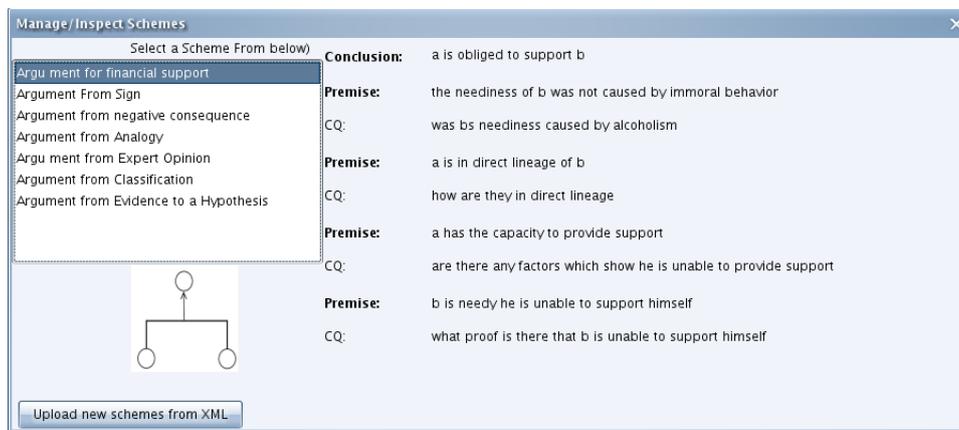


Figure 8.6: The scheme loaded into ArgumentSpace

This scheme can then be defined in an XML format, as in appendix C, before being loaded into ArgumentSpace, as seen in figure 8.6.

### 8.1.5 Legal scheme evaluation

One of the weaknesses identified through this example is a deficiency in the current format of schemes in ArgumentSpace. In the example above, it would be desirable to state that certain premises require nested premise in support, for example in the case of “A is in direct lineage of B”, stating how they are related. However currently only a single level of premise is supported in the XML definition. To cover this problem in the example, critical questions are used (e.g. “How are they in direct lineage?”), which can be turned into additional premises. However, critical questions are weaker than premises, in that they are not used in the evaluation, and cannot be attacked, therefore a desirable enhancement to ArgumentSpace would be support for multi-level schemes.

Further, should we have the ability to represent nested premises, we could have justified ‘A is in direct lineage of B’ with ‘A is an ancestor of B or A is a descendant of B’. Since this is obviously the disjunction of two statements, it would be desirable to maintain this logical structure both in graphic representation, and for evaluative purposes. However ArgumentSpace currently does not support this.

## 8.2 Case Study: An Argument on Climate Change

In this section we present an example debate about climate change, taken from the Royal Society<sup>1</sup> publication ‘Climate change controversies: a simple guide’. We present the argument in its original textual form (section 8.2.1), which, although well written, is complex and not easy to understand and analyse. We show how it can be represented visually in ArgumentSpace (section 8.2.2). Finally we discuss the problems identified through this example in section 8.2.3.

### 8.2.1 Source argument

Climate change critic:

*“Observations of temperatures taken by weather balloons and satellites do not support the theory of global warming.”*

The Royal Society:

*“It is true that in the early 1990s initial estimates of temperatures in the lowest part of the earth’s atmosphere, based on measurements taken by satellites and weather balloons, did not mirror the temperature rises seen at the earth’s surface. However these discrepancies have been found to be related to problems with how the data was gathered and analysed and have now largely been resolved. Our understanding of global warming leads us to expect that both the lower atmosphere - the troposphere where most greenhouse gases are found - and the surface of the earth should warm as a result of increased levels of greenhouse gases in the atmosphere. At the same time, the lower stratosphere - the part of the atmosphere above the greenhouse gas ‘blanket’ - should cool.*

*Some have argued that climate change, as a result of human activities, isn’t happening because early measurements taken from satellites and weather balloons seemed to show that virtually no warming was happening in the troposphere. However, this has been found to be due to errors in the data. Satel-*

---

<sup>1</sup>The Royal Society: [www.royalsociety.org](http://www.royalsociety.org)

lites were found, for example, to be slowing and dropping in orbit slightly, leading to inconsistencies in their measurements. Variations between the instruments onboard different satellites also led to discrepancies - a problem that has also been found with weather balloons. Furthermore, a mathematical error in one of the original analyses of satellite data meant that it showed less warming in the troposphere. However, once adjustments are made to take account of these and other issues, the warming in the troposphere is shown to be broadly consistent with the temperature trends we see at the earth's surface."

### 8.2.2 Visual representation

The above argument is clearly quite complex, and representing it in ArgumentSpace (as in figure 8.7) can help understand the inferences, points of agreement, and disagreement. For example, through representing the source text in a graphical form we can instantly see there is no disagreement over rising temperatures at surface level, and that it is argued that there are three sources of error in 1990's satellite data.

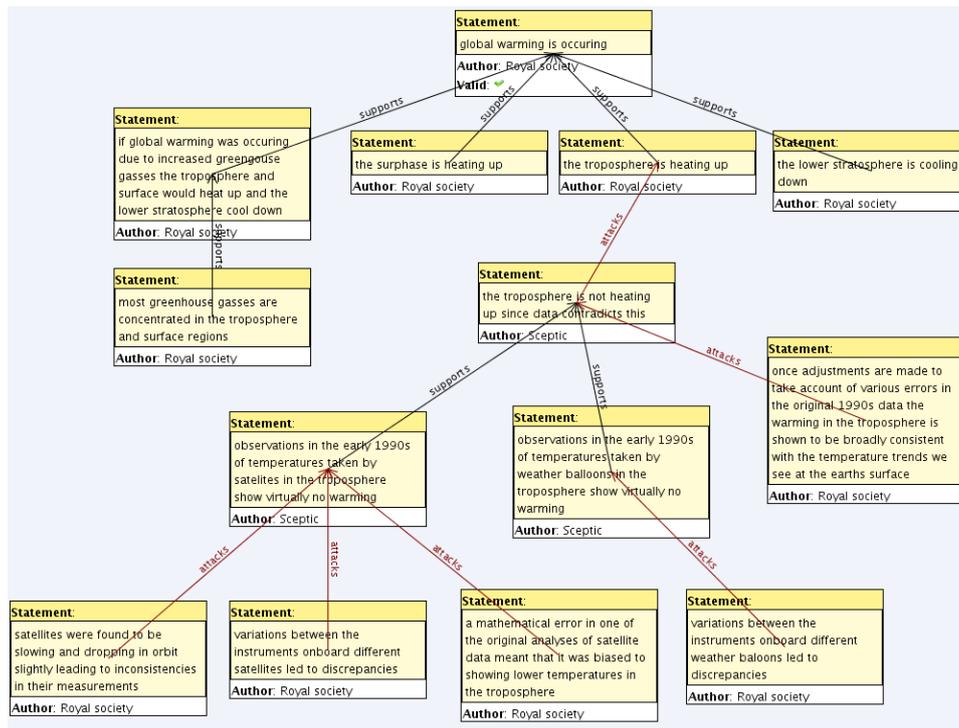


Figure 8.7: The climate change argument visualised

The debate’s root claim (that global warming is occurring) evaluates to true, as seen in figure 8.8. This is consistent with the judgement of a logical reasoner, the attack by the ‘climate skeptic’ has been counter-attacked with a statement which is supported by a number of premises, which have not been counter attacked.



Figure 8.8: The result of evaluating the climate change debate

### 8.2.3 Climate change case study evaluation

Although this debate is somewhat one-sided (we do not show in detail the climate change sceptic’s view), it is a useful example showing how the tool can be used not just to analyse short to-and-fro type dialogues, but also to break down extended arguments, for easier analysis.

The main weakness highlighted through this example is a deficiency in the ability to provide support to statements. It would be useful to show evidence for statements such as “temperature readings in the early 1990s show virtually no warming” with empirical data which could be evaluated. This is currently not possible.

If data on temperature readings was stored in an ontology, a statement such as that below could be evaluated and combined to give a more justified statement of validity for the argument.

“PROPERTY mean.temperature OF INDIVIDUAL atmosphereReading1990 ≤ PROPERTY mean.temperature OF INDIVIDUAL atmosphereReading1985”

Although ArgumentSpace allows OWL based ontologies to be associated with statements, it’s ontological query engine currently does not support properties to be associated with individuals (only classes). Thus, the statement cannot be evaluated. Despite this weakness, ArgumentSpace’s modelling of the debate is effective and it’s evaluation of validity is accurate.

## 8.3 Usability Evaluation

One of the aims of this project was to produce a tool which could extend the use of computational argumentation techniques to a larger audience. As such the tool is designed for computer-competent, but non-technical users. As such, issues of human computer interaction (HCI) are of importance to the project.

To assess the tool in a structured manner, we evaluate the tool against each of the ‘principles of good design’ (section 8.3.1) proposed by Shneiderman in [28], a pivotal researcher in human computer interaction. We describe how well ArgumentSpace obeys each principle, and provide a scoring of ‘poor’ /‘adequate’/‘good’ for comparison. We summarise the findings in section 8.3.2.

### 8.3.1 Principles of Human-Computer Interface Design

1) **Strive for consistency** *Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.*

Associated actions are all accessed from the same areas of the program, helping users ‘know where to look’. For example all action controls are performed through the right click menu. Also, controls for changing view properties are always in a toolbar above the visualisation. As such there is consistency between the ontology viewer, and the argument viewer (as seen in figure 8.9). Help screens, error prompts and labels all use consistent terminology, assured through the use of a language file, rather than holding language directly in the code. This also enables ease of future translation.

Rating: ‘good’

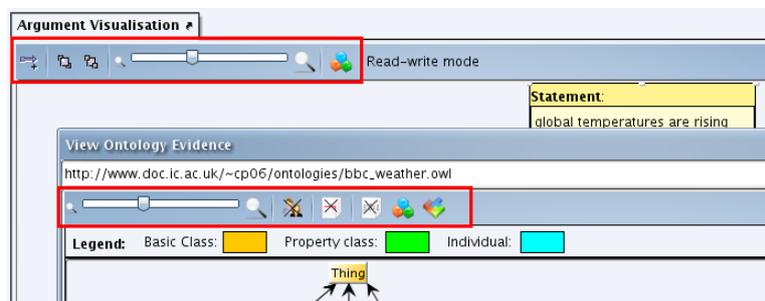


Figure 8.9: Consistency across view toolbars for arguments and ontologies

**2) Enable frequent users to use shortcuts** *As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.*

Although initially some key-combinations were employed, 'delete key' to remove a statement, etc, these were later removed to reduce the complexity of the main argument visualisation codebase. Although fast, efficient interaction should be achievable - currently the main controls (insert/delete/edit statement) are all accessible through a right click menu (seen in figure 8.10) and no shortcut keys now exist. The implementation of short-cut keys would be an improvement for future versions.

Rating: 'adequate'

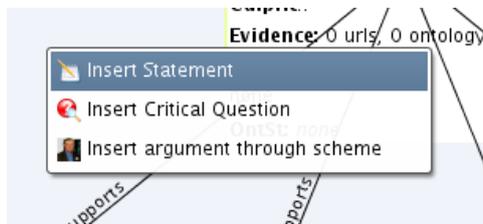


Figure 8.10: Right click action menu

**3) Offer informative feedback** *For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.*

Offering informative feedback is a key feature of any software. We achieved this by providing descriptive but simple dialog messages in the users view wherever needed. Two examples of these are seen below in figures 8.11 and 8.12 respectively.

Rating: 'good'



Figure 8.11: Informing the user that a duplicate exists

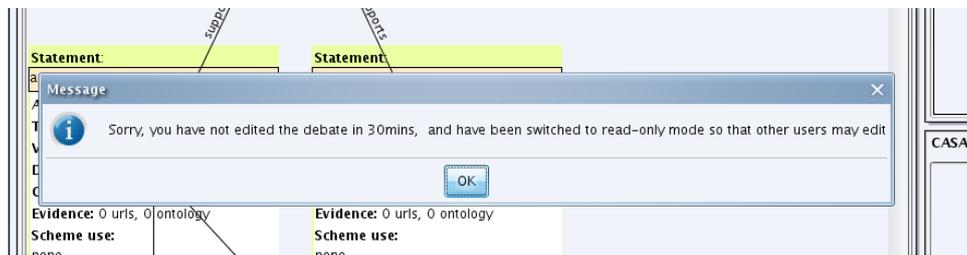


Figure 8.12: Timeout after inactivity

**4) Design dialogs to yield closure** *Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.*

In general, actions performed in ArgumentSpace are atomic, and feedback is instantaneous; providing the user feedback of his actions was ensured by principle (3). One situation in which a sequence of actions is required is the insertion of an argument through a scheme, the sequence for which is seen below:

1. Open ‘insert argument through scheme’ dialog.
2. Select a scheme to use.
3. Enter variables for scheme.
4. Click insert button.
5. Argument appears (dialog closure).

The actions and closure of this dialog can be seen in figures 8.13 and 8.14. Overall, this principle was adhered to in the small number of situations in which it was an issue.

Rating: ‘good’

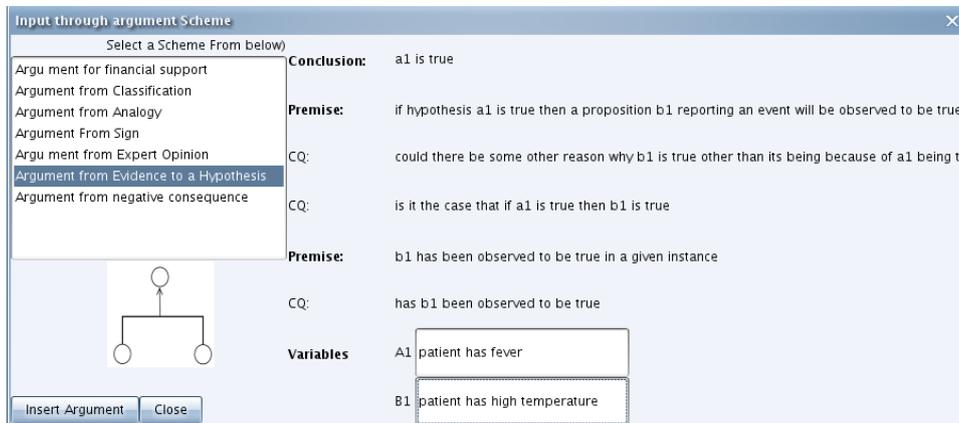


Figure 8.13: Scheme input (step 3)

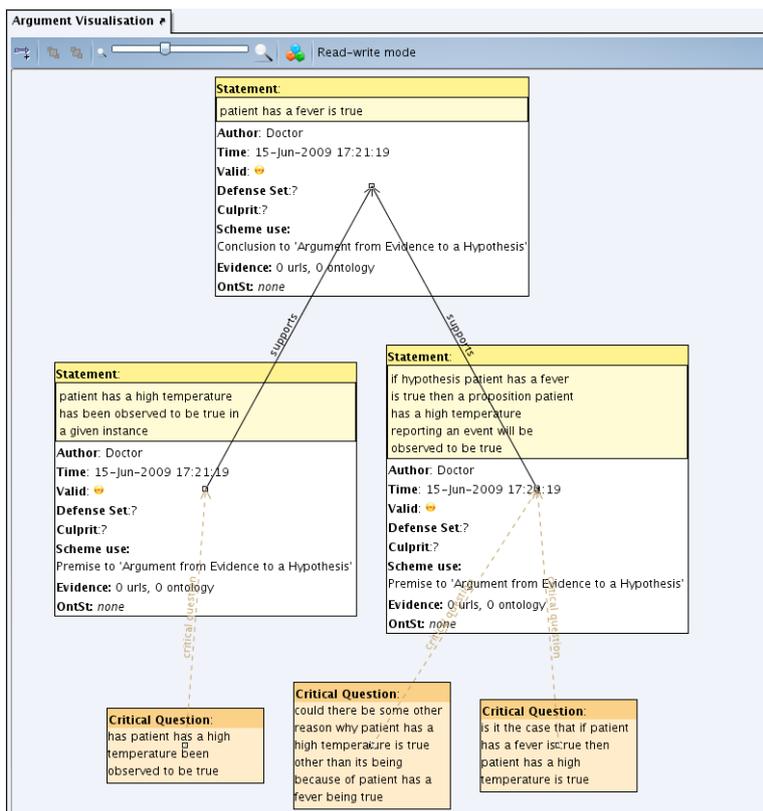


Figure 8.14: Scheme input closure (step 5)

**5) Offer error prevention and simple error handling** *As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.*

The possibility of errors in the system as a result of the user was relatively small. As seen previously in figure 8.11, where a statement is inserted which already exists in the system, the user is informed and he can try again. When a user is logged out due to timeout, he is informed (see figure 8.12). If a user tries to view or validate an invalid ontology URL, he is informed.

However one particular area for error was that of ontological query statements, since the parser and language grammar was invented for this project. Users are informed on entry if they are entering a grammatically invalid statement (figure 8.15), however currently no information is provided as to where in the statement the error has occurred, this is one area of possible improvement.

Rating: ‘adequate’/‘good’

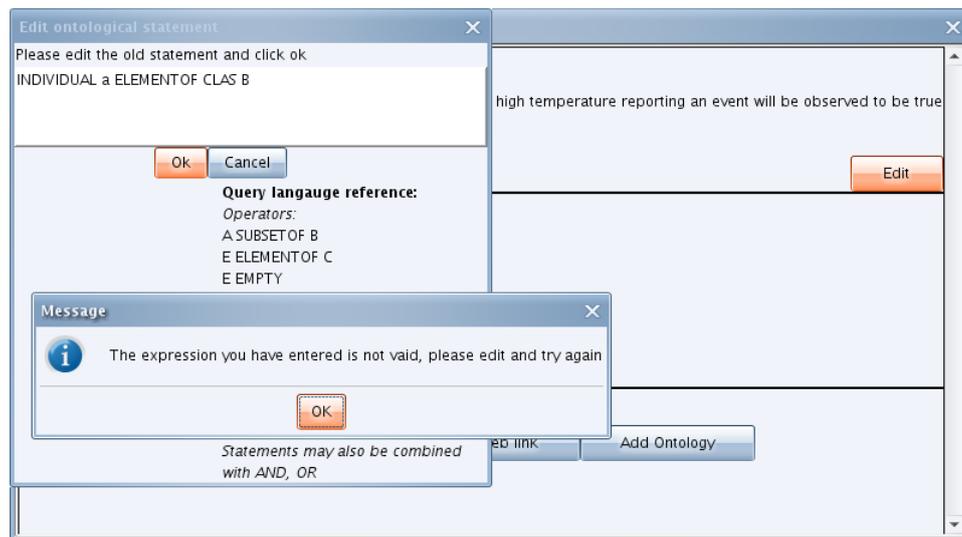


Figure 8.15: Grammar error in an ontological query

**6) Permit easy reversal of actions** *This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.*

As with principle (2), initially undo/redo was implemented in earlier versions, however it was later removed from the program as the complexity of maintaining this functionality increased. Since actions in ArgumentSpace are frequently not atomic (deletion of nodes requires deletion of all associated attack/support relations, insertion through scheme involves adding multiple statements and arcs), implementing undo/redo was not a simple feature.

The usefulness of undo/redo in ArgumentSpace is however acknowledged, and is a feature which could be added in future versions.

Rating: ‘poor’

**7) Support internal locus of control** *Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.*

In ArgumentSpace little action is performed involuntarily of the user. One action which is done without the users control is automatic timeout after inactivity in an argument (a feature which is required to prevent the indefinite holding of an argument’s write-lock, discussed in section 7.3). A further action is the updating of arguments to show new statements made by other distributed user of the system. It is clear that this ‘involuntary action’ is a fundamental part of a synchronised collaborative tool, rather than a deficiency.

Rating: ‘good’

**8) Reduce short-term memory load** *The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.*

This principle has been adhered to throughout, notably when entering an ontological statement, a prompt is provided so that the proprietary syntax does not have to be remembered. Also, all toolbar icons also use hover over tips (see figure 8.16), thus reducing the memory load for remembering precisely what each icon represents.

Rating: ‘good’

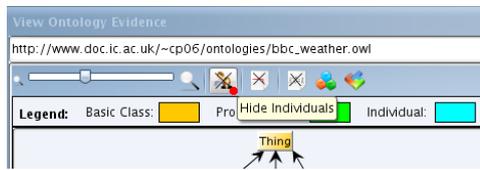


Figure 8.16: Toolbar tips reduce memory load

### 8.3.2 Summary usability analysis

The usability of ArgumentSpace is believed to be strong, this has been supported by comparison to ‘Shneiderman’s Principles of Human-Computer Interface Design’. The results were largely positive, with 4 areas scoring good or good/adequate, 1 scoring adequate and 1 scoring poor.

It is positive that the deficiencies highlighted were with regard to specific usability features missing in the program (shortcut keys and undo/redo respectively) rather than large problems prevalent throughout the program. Also, I am confident that the additional features identified could be implemented without great change. These two features were originally considered, but not implemented due to time constraints and the chosen priorities of the project. Overall, the results of the usability analysis have been positive.

## 8.4 Scalability and Performance Evaluation

In this section we will assess how the performance of ArgumentSpace varies, as it is used to solve increasingly large problems, and used by an increasing number of people. We assess the performance of ArgumentSpace as the size and complexity of debate grows in section 8.4.1, and analyse how the system will cope as the number of users increases in section 8.4.2.

### 8.4.1 Size of debate

To assess how the performance of ArgumentSpace was affected, as the size of debate increased, we performed a series of experiments. For each size of debate, we measured the following:

1. The time to fetch the debate from the ArgumentSpace database server and construct the internal ArgumentSpace debate model.
2. The time to construct the debate visualisation in ArgumentSpace.
3. The time taken to evaluate the debate using CaSAPI, including the time to transform the debate to an ABA representation, and Java-Prolog communication to send the debate and receive the result.

The tests were performed with the ArgumentSpace client and CaSAPI server running on the same 3.6GHz Pentium IV, 1GB RAM lab machine, and the Department of Computing database server storing the Postgre SQL database. All results reported are averages taken over three runs and are expressed in seconds. The times were recorded using the TimerTester API [17], to millisecond precision.

Since the debate structure, as well as size, was likely to effect performance, we conducted tests in two ‘debate structure’ configurations, a linear support-only structure, and a ‘binary tree structure, both of which are explained below.

## Linear debate structure

The first type of debate structure is shown below. Here, a claim is supported by a chain of supports, varying  $k$ , the total number of statements. An example is shown in figure 8.17.

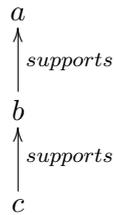


Figure 8.17: Example debate for  $k=3$

Figure 8.18 shows the test results for the three measured variables, and also a comparison for the time to evaluate, to a best-fit exponential trendline.

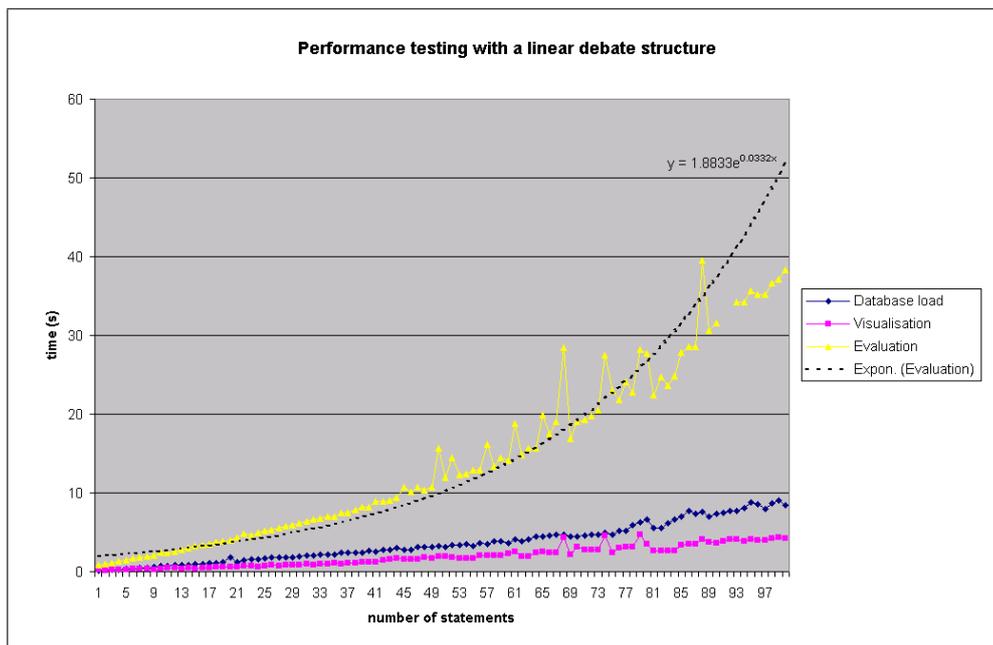


Figure 8.18: Results from the linear debate scalability experiment

From the results we can see that the time to fetch debates from the database

and visualise them in ArgumentSpace is linear for all values tested. However the time to evaluate debates closely fits an exponential curve, indicating poor scalability at large debate sizes.

With more than 50 statements, performance becomes erratic, and a number of tests in the 80-100 statements range failed on evaluation (indicated by gaps in the yellow line). This is possibly due to the high memory demands placed on Prolog (due to the size of the resultant Prolog program), or the limitations of the PrologBeans Java-Prolog communication library in sending large Prolog predicates (which contains the ABA debate framework) to CaSAPI.

We believe the results of this test indicate that ArgumentSpace would be effective in analysing arguments of sizes up to around 50, however, beyond this, both performance and reliability start to deteriorate due to the limits of the current implementation.

### Binary tree debate structure

The second type of debate structure evaluated is that shown in figure 8.19, a binary tree structure of height  $h$  (in which each statement is supported by two more statements) with the addition of an attack, and counter attack, on each leaf statement.

We performed tests for increasing heights ( $h$ ) of the initial tree, and then converted a depth into a number of statements ( $k$ ), for comparison with the linear debate experiment, using the formula<sup>2</sup>:

$$k = 2^{h+2} - 1$$

The graph in figure 8.20 shows the test results from the experiment performed with a binary tree, compared to the previous results performed on a linear debate structure. ‘Db’, ‘Vis and ‘Eval’ indicate the time to load from database, visualise, and evaluate the debate respectively, whilst ‘lin’ and ‘bin’ indicate whether the result corresponds to a linear or binary debate structure. Only a small number of tests were performed (with height varying from 0-5) since an incomplete tree would alter the complexity of evaluation (in some cases the evaluation would result in the debate not being deemed acceptable, as is never the case with a complete tree).

---

<sup>2</sup>based on [30].

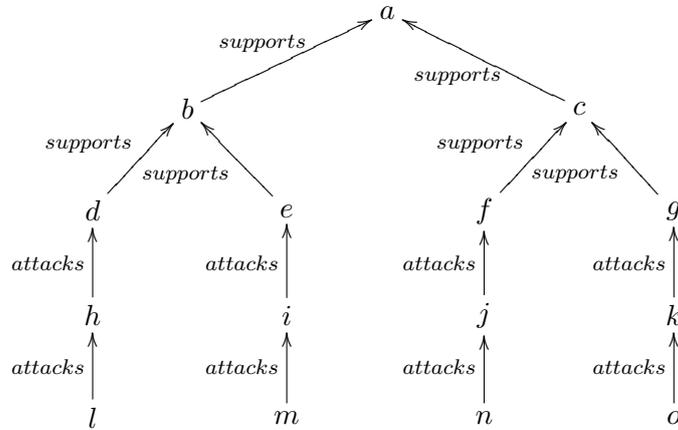


Figure 8.19: Example binary tree debate for  $h=2$

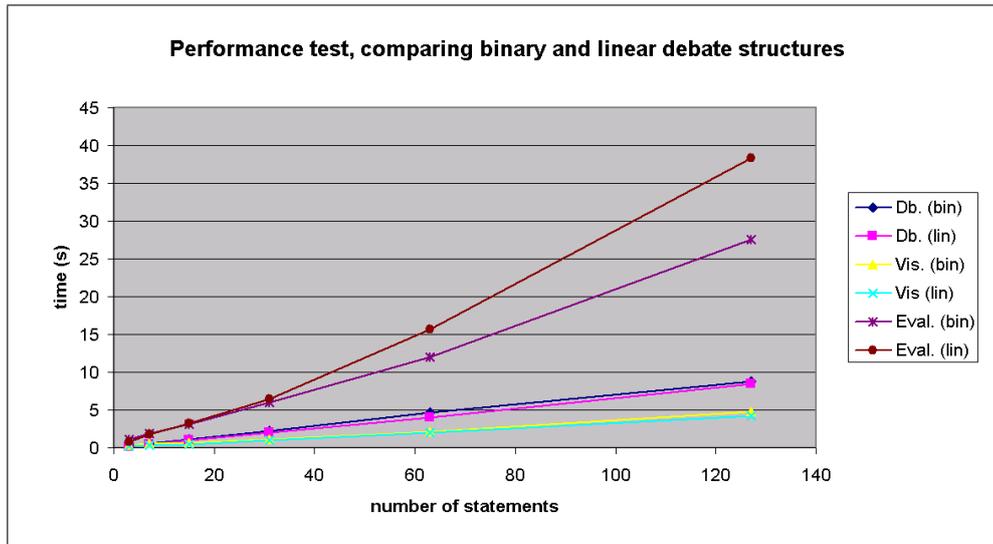


Figure 8.20: Comparison of binary and linear debate structure results

From the results we can see that as with the linear debate structure, for the binary debate structure, the time to load the debate from the database and construct the debate visualisation linearly increases with the size of the debate, and performance is very similar.

However, the results also show that when evaluating debates, performance is significantly slower in a complex binary structure than for a linear structure,

with the same number of statements. This is likely due to the increased complexity of the proof game played out by the Prolog CaSAPI evaluator, requiring switches from proponent to opponent (that did not take place in the linear tree).

### 8.4.2 Number of users

With an increasing number of users, the probability of multiple users making simultaneous evaluation requests to the CaSAPI server also increases. As such, we simulated the effect of multiple ArgumentSpace clients sending evaluation requests to the CaSAPI server in the same moment, measuring the effect this has on response time and reliability.

We performed this experiment from a single lab machine as used in the previous experiment, scheduling multiple evaluation requests to be executed at the same millisecond in time, using the Java Timer API. Since the test was performed from a single machine, the requests were not exactly simultaneous, however the time scale under consideration meant they were close enough (since evaluation takes time in scale of seconds).

The debate evaluated by each ArgumentSpace instance was the same (that shown in figure 8.7). For an increasing number of clients, we recorded the mean delay encountered for all clients, and the number of failed requests. The results are shown in figure 8.21.

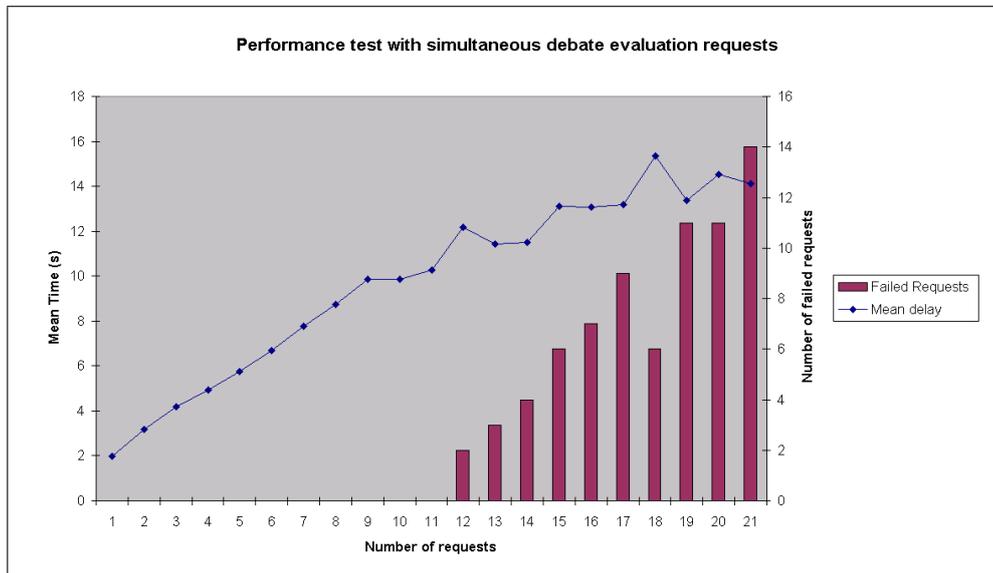


Figure 8.21: Results from the user scalability experiment

The results of this experiment show us that the time to evaluate debates using CaSAPI linearly increases, as a multiple of the number of simultaneous requests, since the Prolog server's time is split between each request. Although it may be noticed that, for a number of requests greater than 10, the increase in delay appears to be less than linear, the additional delay was due to overhead associated with dropping requests, rather than servicing them.

The experiment shows us that the system is unable to deal with more than 10 simultaneous requests. After this threshold, the PrologBeans library cannot cope, and requests start to fail with each additional request, this is due to the monitor (PBMonitor class) in the PrologBeans library having a limit of 10 requests per server.

It is difficult to estimate the probability of more than 10 requests being made simultaneously with a given user base size, since the frequency of evaluation requests will depend on individual user behaviour. However, from this experiment it is clear that should a large group of ArgumentSpace users share a common CaSAPI server in its current form, failed requests are likely to result. This demonstrates the limitations of the current system.

## 8.5 Comparison to Existing Tools

In this section we will compare ArgumentSpace to a number of alternative argumentation tools, namely Compendium (8.5.1), Cohere (8.5.2), Dungein (8.5.3), ArgDF (8.5.4) and SWAFI (8.5.5), before summarising our comparison in section 8.5.6.

Each comparison tool was downloaded and tested in order to understand their features and strengths. For each tool the focus and context of use is described, we then conduct our comparison in terms of the following features considered in this project:

**Visualisation (visual.):** Whether arguments or ideas can be displayed visually within the tool.

**Evaluation (eval.):** Whether the tool has the capability to evaluate the acceptability of a claim/argument/statement.

**Ontologies (ont.):** Whether the tool provides the facility to display or evaluate knowledge held in ontologies.

**Schemes (sch.):** Whether the tool makes use of argument schemes for representation, construction or analysis of arguments.

**Collaboration (collab.):** Whether the tool allows multi-user collaboration on ideas/arguments.

### 8.5.1 Compendium

Compendium [19] is one of the oldest and most widely used open-source sensemaking tools, developed at the Open University’s Knowledge Media Institute. Compendium is based around the broad idea of ‘idea mapping’ rather than argumentation, and was based on the ‘Issue Based Information Systems’ (IBIS) model [27], developed by Horst Rittel and colleagues during the early 1970’s. In IBIS, primitives are of three types: issues (questions that the design or argument is addressing), positions (potential resolutions of an issue) and arguments (statements which support or refute a position).

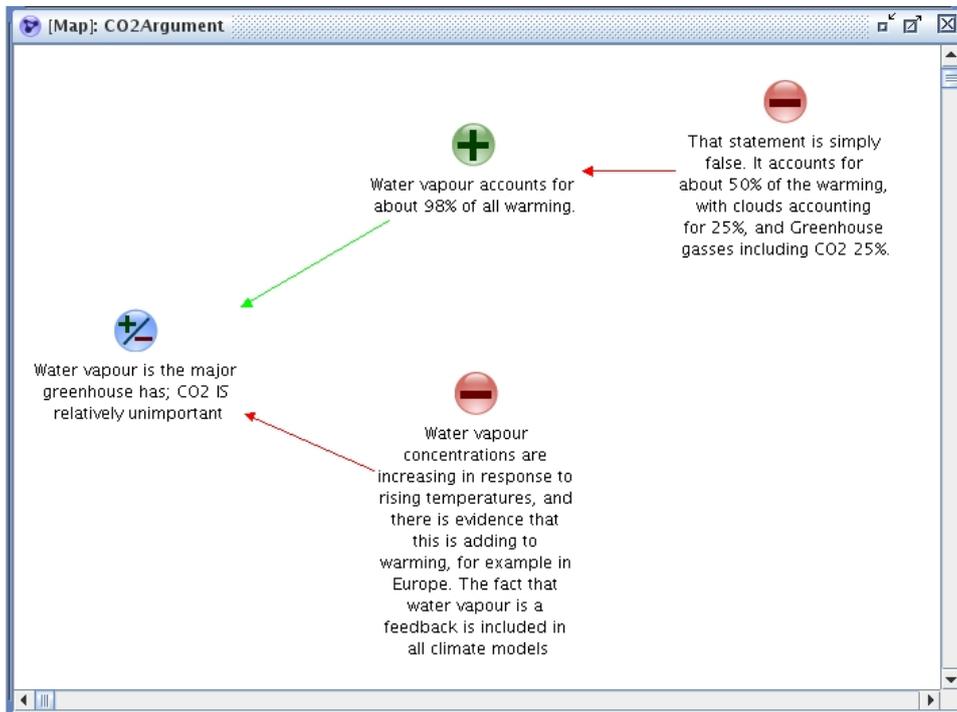


Figure 8.22: Argument Visualisation in Compendium

Compendium was developed to support the organisation of ideas, knowledge and multimedia, concepts larger than the scope of my project or argumentation. However, this broad focus comes at a cost. Compendium provides no means of evaluating the validity of arguments described in the IBIS model, and also makes no use of argument schemes. Compendium provides no collaboration among distributed parties, it also provides no evaluation functionality. It provides the ability to provide ‘evidence’ to statements with web URLs (or an ontology URL). However, there is no ability to evaluate the URL as in ArgumentSpace. Compendium does provide a means of supporting statements with multimedia source of evidence. However, ontologies are not catered for in any specific way. They cannot be viewed or evaluated against.

Another large problem with the program is its lack of collaborative ability, despite its broad idea mapping focus, it provides no means of sharing ideas and knowledge. This lack of consideration in design, lead the Knowledge media institute, designers of Compendium to develop Cohere, described next.

	Visual.	Eval.	Ont.	Sch.	Collab.
<b>Compendium</b>	✓	x	x	x	x

## 8.5.2 Cohere

Cohere [18] is a web-based system also developed at the Open University's Knowledge Media Institute. Cohere is a general idea visualisation tool, similar in nature to Compendium, and also largely based on the IBIS model [27]. Its main advantage over Compendium is its distributed nature, supporting collaboration and debate.

Cohere has two concepts: ideas, and connections. An idea is an arbitrary statement or web resource, and a connection is a relation between two ideas, which must be either of the type positive (such as supports), negative (e.g. is inconsistent with) or neutral (e.g. responds to). New relations can be defined by the user.

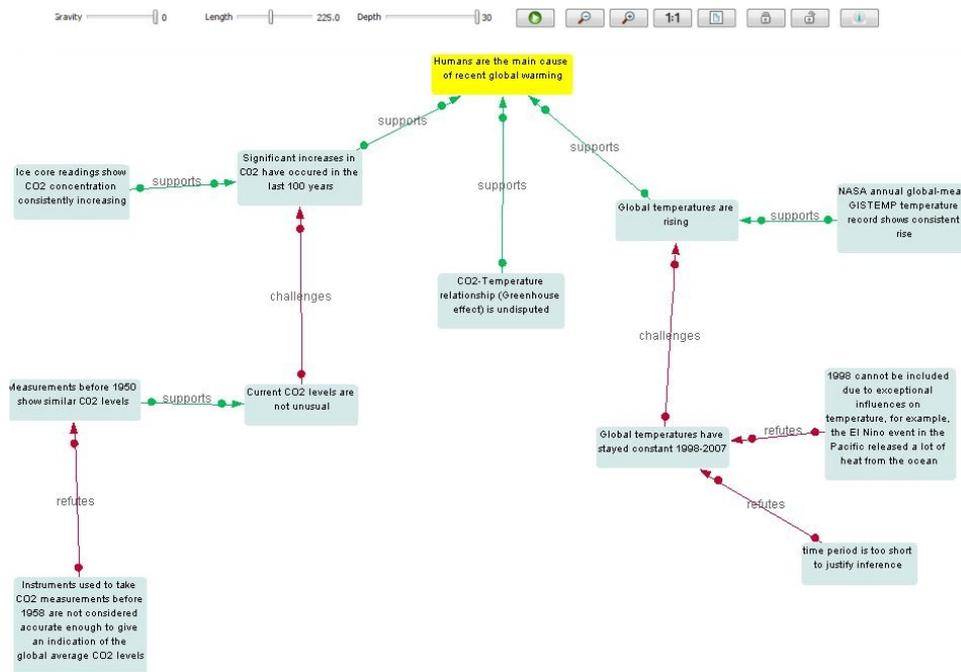


Figure 8.23: Cohere, displaying an example argument on climate change

One of the main disadvantages of Cohere's primary functionality is the fact that the arguments are read only when viewed visually - editing must be done in a text only manner. There also exists no validity checking functionality in Cohere, something which would be difficult to implement with its current unconstrained argument structure.

Cohere provides no support for schemes or ontologies, however, like Com-

pendium, it provides the ability to link statements to URL's - which could be an OWL ontology, but again no functionality is provided to view or evaluate against the ontology.

	<b>Visual.</b>	<b>Eval.</b>	<b>Ont.</b>	<b>Sch.</b>	<b>Collab.</b>
<b>Cohere</b>	✓	x	x	x	✓

### 8.5.3 Dungere

Dungere [22] is a Java library intended for developers which evaluates arguments in the Abstract Argumentation framework (described in section 2.2.1).

The library is intended to be used by applications, to provide evaluative functionality (in a similar way to ArgumentSpace's use of CaSAPI), and so does not directly provide functionality for schemes, ontologies, or collaboration. These functions may be provided by the program using Dungere, therefore we show 'N/A' (not applicable) in the comparison table.

One example integration of Dungere is with the argumentation tool, Araucaria [26], which visualises arguments and allows the use of schemes alongside Dungere evaluation. Dungere is also distributed with a test application (shown in figure 8.24), which allows argument visualisation in a simple graph.

Dungere currently implements a reasoner for grounded and preferred semantics. The reasoner is implemented using argument game proofs, and allows a justification to be shown for each argument's evaluation - something ArgumentSpace currently lacks. Its use of abstract argumentation may provide limitations in integration with existing sense-making tools such as Compendium and Cohere, which do not use an atomic argument structure as in abstract argumentation.

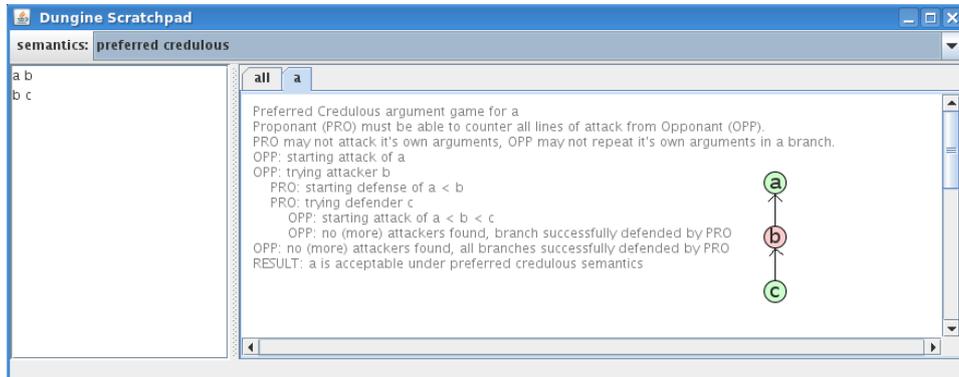


Figure 8.24: Evaluating a simple argument framework with Dungine

	Visual.	Eval.	Ont.	Sch.	Collab.
Dungine	N/A	✓	N/A	N/A	N/A

#### 8.5.4 ArgDF

ArgDF [33], developed by Fouad Zablith at British University in Dubai, is a text based online argumentation system, it was designed as a proof of concept for the use of the argument interchange format (AIF) [described later in section 9.2.4]. The tool provides collaborative functionality through its web-based interface. No login system is used on the website, so anyone can freely edit and contribute to current arguments.

In ArgDF arguments must be designed by conforming to a specified argument scheme, if no scheme exists in the system which matches the argument structure, a new scheme must be designed before defining the argument. I believe this requirement to use schemes to be somewhat restrictive, although the requirement is linked to the definition of AIF, which also places a heavy emphasis on schemes.

ArgDF provides no means of linking statements to ontologies or other evidence, and it also provides no means of evaluating the validity of arguments.



Figure 8.25: Inspecting an argument in ArgDF

ArgDF’s arguments are displayed and constructed in a text only way, which makes large arguments unintuitive to understand, since the argument is structured and broken down into small statements, but the connections between those statements are not easily to see.

	Visual.	Eval.	Ont.	Sch.	Collab.
ArgDF	x	x	x	✓	✓

### 8.5.5 SWAFI

‘Semantic Web Argumentation For Individuals’ (SWAFI) [32, 31] is a hybrid argumentation-ontological tool developed by researchers at the Breast Cancer Research Institute. It is one of only a few tools which, like ArgumentSpace, use argumentative techniques and ontological reasoning together.

SWAFI is a system designed to decide on treatment choices for cancer patients, it uses an argumentation framework developed specifically for the program. Within the system arguments are stored for/against the use of various drugs, with premises based on details of the patient. The arguments are based on the results of clinical trials describing when a particular drug has been effective/ineffective. The framework uses a similar notion of argument-attack and acceptability to ABA, but also resolves conflicts between arguments based on when the clinical trial was conducted (arguments based on newer trials attack those based on older ones). The system uses an ontology to store facts about patients, which may be used as premises to

the arguments described above.

Although the computational techniques used in SWAFI are similar to ArgumentSpace, visually it is very different. The tool is a command-line only, providing a textual output as to the clinical decision (no argument visualisation or use of schemes). Also only limited justification of the process taken to come to this decision is provided.

Its aims are different, it is not designed for evaluation of arbitrary human arguments, but rather specific medical arguments presented in a description logic format, as such it is difficult to compare directly with ArgumentSpace. However it is clearly a useful tool in showing how the idea of combining ontological reasoning and argumentation is gaining attention.

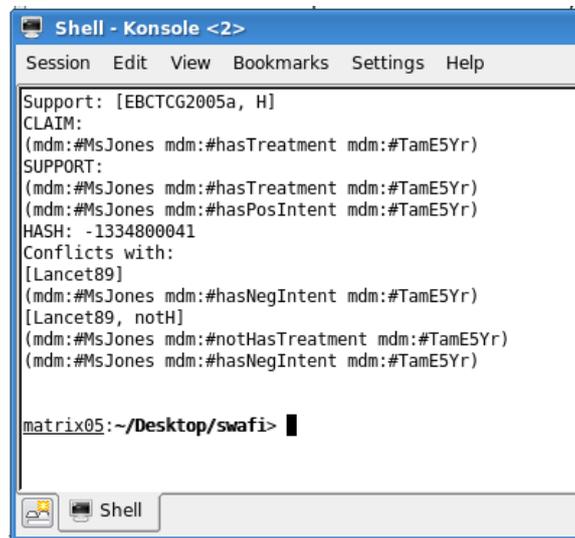


Figure 8.26: SWAFI result

	Visual.	Eval.	Ont.	Sch.	Collab.
<b>SWAFI</b>	x	✓	✓	x	x

### 8.5.6 Summary of comparison

Having assessed other argumentation tools available currently, it is clear that ArgumentSpace is relatively novel in the breadth of features provided. None of the tools we have compared it to has provided all the functionality which ArgumentSpace provides (visualisation, evaluation, use of ontologies, use of schemes and collaboration), as seen in figure 8.27. However in evaluating ArgumentSpace we should not just consider the breadth of functionality

provided, but also how well these features are implemented.

	<b>Visual.</b>	<b>Eval.</b>	<b>Ont.</b>	<b>Sch.</b>	<b>Collab.</b>
<b>Compendium</b>	✓	x	x	x	x
<b>Cohere</b>	✓	x	x	x	✓
<b>Dungine</b>	N/A	✓	N/A	N/A	N/A
<b>ArgDF</b>	x	x	x	✓	✓
<b>SWAFI</b>	x	✓	✓	x	x
<b>ArgumentSpace</b>	✓	✓	✓	✓	✓

Figure 8.27: Features comparison table

In terms of argument visualisation, there is very little difference visually between ArgumentSpace and other tools with this functionality (Compendium is similar, but cohere does not provide visual editing). At times ArgumentSpace can appear more cluttered in its graphic representation than the other tools, however this is only true when metadata about validity is shown on each statement, a feature which can be turned off.

Two tools considered provided a mechanism to evaluate arguments, Dungine and SWAFI. SWAFI uses an argumentation framework designed specifically for the medical application, and comparison is difficult since the arguments are not arbitrary, but prior defined in the program. Dungine provides evaluative functionality, but uses the abstract argumentation framework and so ignores the internal structure of arguments. It does, however, illustrate a feature which ArgumentSpace currently lacks- display of the game proof used to arrive at the decision of acceptability.

Schemes are used in a similar fashion to ArgumentSpace in the ArgDF tool. This program is initially provided with a number of Walton’s schemes, as in ArgumentSpace, and additional schemes can be defined by the user. The program provides an easier means of defining new Schemes than ArgumentSpace, in that, in ArgDF, schemes are defined using a simple web based interface, whereas in ArgumentSpace schemes must be defined in an XML format which is not really feasible for non-technical users.

Cohere and ArgDF both provide similar collaboration functionality, although this is much more limited than ArgumentSpace. ArgDF is a web-based tool in which arguments are displayed in textual form, as such synchronisation issues are not encountered. In Cohere, arguments can be collaborated on, but again only in a textual form. Graphical visualisation is a read-only process. Therefore, ArgumentSpace’s functionality, although not perfect in this respect, is the best of those compared to.

Through comparison to existing tools, a number of features have been identified which could be improved upon in future versions of ArgumentSpace, namely an improved method for defining new schemes, development of the ontological query engine to a level where it could be used in more serious contexts (as with SWAFI) and display of the proof game used to come to a decision on acceptability of a claim. However, the comparison is also encouraging in that it has highlighted the strength of ArgumentSpace in comparison to prior available tools.

## Chapter 9

# Conclusions

In this chapter, we first highlight the achievements of the project in section 9.1, and then identify weaknesses in the implemented system in 9.2.

### 9.1 Achievements and Contributions of the Project

The primary contribution of this project is the design and implementation of ArgumentSpace, a combined argumentation visualisation and evaluation tool, providing a means through which the evaluative functionality of an existing argumentation system (CaSAPI) can be accessed and used by a wider audience than was previously possible. In doing so the project provides a mapping from a support/attack dialog-inspired framework to the assumption based argumentation framework. This approach could be taken up by related tools such as Cohere and Compendium, which share a similar visual structure to ArgumentSpace.

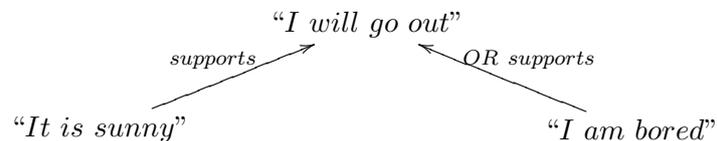
The system builds on work into argumentation schemes to support the development and analysis of arguments built on sound reasoning. In doing so the definition of argumentation schemes has been extended to include ideas from assumption based argumentation and research into the burden of proof.

The application provides one of the first links between the fast expanding area of ontology based knowledge representation and argumentation, allowing users to support statements with evidence from ontologies, and validate those claims automatically against the ontology. To support this I have developed a simple ontological query engine, which uses language semantics and set theory as well as syntax to validate statements.

## 9.2 Current Weaknesses and Further Work

### 9.2.1 Expressibility

One key area in which the system could be developed is the expressibility of the argumentation framework. Under the current implementation, every statement (A) linked by  $\text{support}(A,B)$  becomes a requirement for the acceptability of B. An improvement might be to allow alternative support statement groups (using logical disjunction), for example: ('I will go out') if ('It is sunny')  $\vee$  ('I am bored'). An example of how this might be visualised is seen below.



The reason disjunction, although a primary component of traditional logic was not considered to be of great importance here, is that a defeasible logic-based argumentation systems and natural argumentation do not often involve disjunction.

When arguing for a case, it is rare than one might say "I will go out if it is sunny or I am bored, and I am bored, so I will go out". Since the alternative premise ('it is sunny') is not true, it is usually omitted.

However, there are cases where disjunction would be useful, for example in describing general arguments or rules, and so it would be useful if this feature was supported.

### 9.2.2 Improvements to the ontological query engine

Currently the query engine supports only a small number of operators:  $\{is\ subset\ of,\ is\ member\ of,\ is\ disjoint\ to,\ is\ empty\ set,\ not,\ and,\ or\}$ . This could be extended to support others such as individual properties as well as property classes (for example the query "INDIVIDUAL X *hasproperty* A=B").

Also, the proof system for the query engine could be extended, currently it works on a best effort basis, that is, we attempt a number of disproof methods which are all shown to be sound (see appendix A) using a subset of OWL's language features. However no claim is made as to the completeness

of these methods - in some cases we may return ‘unknown’ when there is in fact some way of proving or disproving a statement. Further work could be conducted into ensuring completeness of the proof methods, exploiting the full range of OWL syntax (such as class union, intersection etc) and semantics. Alternatively, an existing query engine for OWL or RDF could be integrated.

### 9.2.3 Support for alternative notions of validity

Currently ArgumentSpace uses the ABA notion of validity based on admissible belief semantics. ArgumentSpace could be improved to support alternatives such as ideal or grounded semantics (see section 2.3).

Grounded dispute derivations can be computed using the latest version of CaSAPI (v4.5), however ArgumentSpace currently uses an older version (v4.3). Improving ArgumentSpace to use the latest CaSAPI version, in order to support grounded semantics, should be a simple alteration, due to the interface design (see section 7.2).

### 9.2.4 Argument Interchange Format

Various frameworks exist for describing arguments, as seen in section 2.2. Various tools have also been developed to provide sense making and argument visualisation, however different tools often use their own argument frameworks, and even if using the same framework may store the arguments in a different way. A standard was proposed in [8] which was designed as a universal notation for arguments, Argument Interchange Format (AIF).

In AIF, arguments are represented by a set of nodes connected through edges. There are two types of nodes: the information nodes (I-Nodes) which hold pieces of information or data that acts as a claim, premise, data etc, and scheme nodes (S-Nodes) representing the arguments scheme which represent a model of reasoning. In AIF, edges between nodes are not typed, and their meaning is inferred from the scheme of which they are a part.

An example argument can be seen below, where the S-Node has a thicker border than the I-Nodes. Here the S-Node represents Modus Ponens. It graphically represents the argument that if we accept  $p$  and  $p \rightarrow q$ , using the Modus Ponens ‘scheme’, we should accept  $q$ .

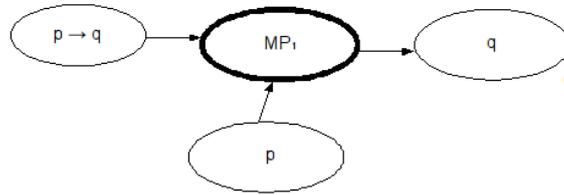


Figure 9.1: A simple argument in AIF

Despite the obvious benefits of a shared format, there has been limited use of the format since its inception, ArgDF (section 8.5.4) being one of the few tools to make use of AIF. Implementation of saving and loading using AIF would be beneficial to the goal of achieving a common format within the argumentation community. It would also be beneficial for users of ArgumentSpace, by allowing the sharing of arguments with ArgDF, and other tools, if the format becomes more widely used.

### 9.2.5 Improved concurrent collaboration

Under the current system, only a single user can be viewing an argument with the ability to write to it at any time. All others must view the argument in read only mode, and receive periodic updates (should any have occurred). This coarse granularity provided a simple means of ensuring synchronisation, however it could be improved.

A better system would allow multiple users to be viewing and editing an argument at any one time, with locks present on specific argument statements rather than the entire debate. This would also require an enhanced synchronisation system, so that updates are made to specific nodes, rather than the entire debate, as in the system at present.

### 9.2.6 Work to increase scalability

As a result of testing into the scalability of ArgumentSpace (section 8.4), a number of potential improvements were identified.

The reliability of debate evaluation currently decreases with increasing debate size, it is believed this is due to a deficiency in the PrologBeans library when dealing with large predicate queries (which are created by ArgumentSpace from large debates). The frequency of this problem could be reduced, by creating more compact Prolog predicates. For example, currently the statement “the weather was very sunny yesterday” would be

turned into “*the\_weather\_was\_very\_sunny\_yesterday*” for evaluation, however the chosen predicate name is merely syntax, and could be replaced with a simple  $p, q$ , resulting in a much smaller Prolog representation of the debate.

The second problem identified was that a single PrologBeans server (as used by ArgumentSpace) can only handle 10 simultaneous requests, after which calls to evaluate debates fail, as might occur if the number of users of ArgumentSpace grew large. One solution to the problem of dealing with a larger number of users might be to have multiple CaSAPI servers, and implement an intermediary program which allocates a particular CaSAPI server depending on the current number of requests each server is dealing with. Alternatively ArgumentSpace could select a random server instance and thus decrease the probability of the threshold being exceeded for any particular server. Work could be conducted to explore solutions to this problem.

# Bibliography

- [1] Gaudenz Alder. Jgraph, a graph drawing library for java, [www.jgraph.com](http://www.jgraph.com).
- [2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [3] T. Bench-Capon and T. F. Gordon. Isomorphism and argumentation. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law (ICAIL 2009)*. ACM Press, 2009.
- [4] Trevor Bench-Capon and Paul Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.
- [5] Hohmann H. Bench-Capon T., Freeman J. and Prakken H. Computational models, argumentation theories and legal practice. *Argumentation Machines; New Frontiers in Argument and Computation*, pages 85–120, 2003.
- [6] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.
- [7] Carlos Iván Chesñevar, Ana Gabriela Maguitman, and Ronald Prescott Loui. Logical models of argument. *ACM Comput. Surv.*, 32(4):337–383, 2000.
- [8] Carlos Iván Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Ricardo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott. Towards an argument interchange format. *Knowledge Eng. Review*, 21(4):293–316, 2006.
- [9] Clark and Parsia LLC. Pellet, an open source reasoner for owl dl in java, [www.clarkparsia.com/pellet](http://www.clarkparsia.com/pellet).

- [10] F. Macagno D. Walton, C. Reed. *Argumentation Schemes*. Cambridge University Press, 2008.
- [11] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–357, 1995.
- [12] Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Artif. Intell.*, 170(2):114–159, 2006.
- [13] Essence, an event organised by the global sensemaking network, [events.kmi.open.ac.uk/essence](http://events.kmi.open.ac.uk/essence), 2009.
- [14] Dorian Gaertner. *Argumentation and Normative Reasoning*. PhD thesis, Imperial College London, 2008.
- [15] Dorian Gaertner and Francesca Toni. Casapi: A system for credulous and sceptical argumentation. In *Proceedings LPNMR-Workshop on Argumentation and Non-Monotonic Reasoning (ArgNMR07)*, 2007.
- [16] Dorian Gaertner and Francesca Toni. Hybrid argumentation and its properties. In *COMMA 08*, pages 183–195, 2008.
- [17] Danilo Gurovich. Testertimer api, a simple java performance test utility, [danilogurovich.wordpress.com](http://danilogurovich.wordpress.com).
- [18] The Open University (Knowledge Media Institute). Cohere, an online visual tool to create, connect and share ideas, [cohere.open.ac.uk](http://cohere.open.ac.uk).
- [19] The Open University (Knowledge Media Institute). Compendium, a software tool for managing ideas and connections between information, [compendium.open.ac.uk/institute](http://compendium.open.ac.uk/institute).
- [20] National institute of standards and technology. Fips 180-2, secure hash standard, federal information processing standard (fips), publication 180-2. Technical report, Department of commerce, August 2002.
- [21] Nicholas R. Jennings, Peyman Faratin, A. R. Lomuscio, Simon Parsons, Michael Wooldridge, and Carles Sierra. Automated negotiation: Prospects methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.
- [22] Gerard Vreeswijk Matthew South and John Fox. Dungine, a java reasoner for evaluating the acceptability of abstract argumentation frameworks, [www.argkit.org](http://www.argkit.org).
- [23] University of Manchester. Owl api, an owl interface library for java, [owlapi.sourceforge.net](http://owlapi.sourceforge.net).

- [24] Sicstus Prolog. Jasper, a prolog emulator for java, [www.sics.se/sicstus/docs/latest4/html/jasper](http://www.sics.se/sicstus/docs/latest4/html/jasper).
- [25] Sicstus Prolog. Prologbeans, a java-prolog communication library, [www.sics.se/sicstus/docs/latest4/html/prologbeans](http://www.sics.se/sicstus/docs/latest4/html/prologbeans).
- [26] Chris Reed and Glenn Rowe. Araucaria, a software tool for analysing arguments, [araucaria.computing.dundee.ac.uk](http://araucaria.computing.dundee.ac.uk).
- [27] Horst W. J. Rittel and Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):155–169, June 1973.
- [28] B. Shneiderman. *Designing the user interface: Strategies for effective human-computer interaction (1st edition)*. Addison-Wesley, 1987.
- [29] D. Walton. *Argumentation Schemes for presumptive reasoning*. Lawrence Erlbaum Associates, 1996.
- [30] Wikipedia. Binary tree entry, [en.wikipedia.org/wiki/binary\\_tree](http://en.wikipedia.org/wiki/binary_tree).
- [31] Matt Williams. *Integrating Ontologies and Argumentation for decision-making in breast cancer*. PhD thesis, University College London, 2008.
- [32] Matt Williams and Anthony Hunter. Harnessing ontologies for argument-based decision-making in breast cancer. In *Proceedings of the International Conference on Tools with AI (ICTAI'07)*, pages 254–261, 2007.
- [33] Fouad Zablith. Argdf, a semantic web-based argumentation system, [www.argdf.org](http://www.argdf.org).

## Appendix A

# Ontological Query Engine Proofs

In chapter 6 we discussed the design of an ontological query engine for evaluating statements about ontologies, and combining the result with arguments. Here we present the proofs of the rules used by the query engine.

### A.1 Individuals

**Proposition (1):**

for arbitrary classes A,B,X and individual x, and where U is the universal set:

if x is declared a member of class X	if $x \in X$
and X is the complement of any super-class of A	$(X = U - B) \wedge (A \subseteq B)$
then x is not a member of the class A	then $x \notin A$

**Proof:**

If  $(x \in X) \wedge (X = U - B)$   
then  $x \notin B$ ;  
since  $(A \subseteq B) \wedge (x \notin B)$   
then  $x \notin A$ .

**Informally:**

If x is a member of X, and X is the complement class of B, then x is not in B (X is everything that is not in B).

If x is not in B, and B is a super-set of A, then x cannot be in A.

## A.2 Classes

### Proposition (2):

For arbitrary classes A, B, and where U is the universal set:

if X is non-empty	if $\exists x : x \in X$
and X is a subclass of A	and $X \subseteq A$
and A is a complement class of B	and $A = U - B$
and C is a subclass of B	and $C \subseteq B$
then X is not a subclass of C	then $X \not\subseteq C$

### Proof:

since  $(x \in X) \wedge (X \subseteq A)$   
 $x \in A$ ;  
since  $(x \in A) \wedge (A = U - B)$   
 $(x \notin B)$ ;  
since  $(C \subseteq B) \wedge (x \notin B)$   
then  $x \notin C$ ;  
since  $(x \in X) \wedge (x \notin C)$   
 $X \not\subseteq C$ .

### Informally:

Since x is in X, and X is a subclass of A, x must be in A.

Since A is the complement class of B, x cannot be in B.

Since C is a subclass of B, if x is not in B, it cannot be in C.

For X to be a subclass of C, all x in X must be in C, since x cannot be in C.

X cannot be a subclass of C

### Proposition (3):

For arbitrary classes A,B

If X is non-empty	if $\exists x : x \in X$
and X is disjoint from A	and $X \cap A = \emptyset$
and A is a superclass of B	and $B \subseteq A$
X cannot be a subclass of B	$X \not\subseteq B$

### Proof:

since  $(X \cap A = \emptyset) \wedge (x \in X)$   
 $x \notin A$ ;  
since  $(B \subseteq A) \wedge (x \notin A)$   
 $x \notin B$ ;  
since  $(x \in X) \wedge (x \notin B)$   
 $X \not\subseteq B$ .

### Informally:

Since  $X$  is disjoint from  $A$  and  $x$  is in  $X$ , then  $x$  cannot be in  $A$ .  
Since  $B$  is a superclass of  $A$ , and  $x$  is not in  $A$ ,  $x$  cannot be in  $B$ .  
For  $X$  to be a subclass of  $B$ , all  $x$  in  $X$  must be in  $B$ , since  $x$  is in  $X$  and not in  $B$ ,  
 $X$  cannot be a subclass of  $B$ .

## Appendix B

# Ontological Query Language

In section 6.4 we presented the query language developed for querying ontologies in ArgumentSpace. Here we formally define the language grammar:

```
or_statement :: and_statement (OR and_statement)*
and_statement :: statement (AND statement)*

statement::  binary_st
            | unary_st

binary_st::  individual (NOT)? ELEMENTOF class
            |class (NOT)? SUBSETOF class
            |class (NOT)? DISJOINTFROM class

unary_st ::  class (NOT)? EMPTY

class::  CLASS name
        |PROPERTY property=value

individual::  INDIVIDUAL name
```

## Appendix C

# XML Scheme: Argument for Financial Support

In section 8.1.4 we presented an argument scheme in relation to a piece of Germany family law, describing the situation in which one family member must support another. The scheme's XML representation is shown here.

```
<SCHEME>
  <NAME>Argument for financial support</NAME>

  <PREMISE>
    <TEXT>A is in direct lineage of B</TEXT>
    <CQ>
      <CQTEXT>How are they in direct lineage?</CQTEXT>
      <CQTYPE>1</CQTYPE>
    </CQ>
  </PREMISE>

  <PREMISE>
    <TEXT>B is needy, he is unable to support himself</TEXT>
    <CQ>
      <CQTEXT>What proof is there that B is unable to support himself?</CQTEXT>
      <CQTYPE>1</CQTYPE>
    </CQ>
  </PREMISE>
```

*Scheme definition continued on next page*

```
<PREMISE>
  <TEXT>A has the capacity to provide support</TEXT>
  <CQ>
    <CQTEXT>
      Are there any factors which show he is unable to provide support?
    </CQTEXT>
    <CQTYPE>0</CQTYPE>
  </CQ>
</PREMISE>

<PREMISE>
  <TEXT>The neediness of B was not caused by immoral behavior</TEXT>
  <CQ>
    <CQTEXT>Was Bs neediness caused by alcoholism?</CQTEXT>
    <CQTYPE>0</CQTYPE>
  </CQ>
</PREMISE>

<CONCLUSION>
  <TEXT>A is obliged to support B</TEXT>
</CONCLUSION>

<VARIABLE>A</VARIABLE>
<VARIABLE>B</VARIABLE>

</SCHEME>
```