Arguments in OWL: A Progress Report

Iyad RAHWAN [a,b,1], Bita BANIHASHEMI [a]
[a] *Faculty of Informatics, The British University in Dubai, Dubai, UAE*
[b] *(Fellow) School of Informatics, University of Edinburgh, UK*

Abstract. In previous work, we presented an RDFS ontology, based on the Argument Interchange Format (AIF), for describing arguments and argument schemes. We also implemented a pilot Web-based system, called ArgDF, for authoring and querying argument structures represented in RDF. In this paper, we discuss some of the limitations of our earlier reification of the AIF. We then present a new ontology which takes advantage of the higher expressive power of OWL. We demonstrate how this enables the use of automated Description Logic reasoning over argument structures. In particular, OWL reasoning enables significantly enhanced querying of arguments through automatic scheme classifications, instance classification, and inference of indirect support in chained argument structures.

Keywords. Argumentation, Argument Interchange Format, Semantic Web, OWL

1. Introduction

A number of Web 2.0 tools now provide explicit support for argumentation, enabling a more explicit structuring of arguments. Such tools include *Truthmapping*,[2] *Debatabase*,[3] *Standpoint*,[4] and *Standpedia*.[5] These systems have a number of limitations. There is limited or no integration between argument repositories. This limits the ability to provide services (e.g. question answering systems) that make use of arguments from multiple repositories, or the ability of users to easily access arguments across tools.

Another, related limitation of existing systems is that argument structure is relatively shallow. Most Web 2.0 applications distinguish only between premises and conclusions, and possibly between pro- and con- arguments. But they do not distinguish between different types of arguments, or subtle types of attack among arguments. Moreover, existing tools do not provide semantically rich links among arguments. For example, in truthmapping, while user-contributed text (i.e. premises, conclusions, critiques and rebuttals) can contain hyperlinks to any Web content including other arguments, which does enable cross-referencing among arguments, these references carry no explicit semantics (e.g. expressing that a link represents a support or an attack). This limits the possibilities for automated search and evaluation of arguments.

[1]Correspondence to: Iyad Rahwan, the British University in Dubai, P.O.Box 502216, Dubai, UAE. Tel.: +971 4 367 1959; Fax: +971 4 366 4698; E-mail: irahwan@acm.org.

[2]`http://www.truthmapping.com`

[3]`http://www.idebate.org/debatabase/`

[4]`http://www.standpoint.com`

[5]`http://www.standpedia.com`

Semantic Web technologies [1] are well placed to facilitate the integration among mass argumentation tools. A unified argument description ontology could act as an interlingua between the different tools. If Web 2.0 mass argumentation tools can provide access to their content through a common ontology, developers could build tools to exchange (e.g. import and export) or integrate arguments between tools. Another benefit of specifying arguments in standard ontology languages is the potential for automated inference over argument structures, such as inference based on Description Logic [2]. In previous work [3], we presented the first (pilot) realisation of a Semantic Web system for argument annotation, based on the argument interchange format (AIF) [4].

In this paper, we discuss some of the limitations of our earlier AIF reifications. We then present a new ontology which takes advantage of the higher expressive power of OWL [5]. We demonstrate how this enables the use of automated Description Logic reasoning over argument structures. In particular, OWL reasoning enables significantly enhanced querying of arguments through automatic scheme classifications, instance classification, and inference of indirect support in chained argument structures.

The paper advances the state of the art in the computational modelling of argumentation in two main ways. Firstly, the new OWL ontology significantly enhances our previous RDF Schema-based implementation [3].[6] In particular, we provide a new reification of the AIF specification and model schemes as classes (as opposed to instances), which enables explicit classification of schemes themselves. Secondly, our new system enables the first explicit use of Description Logic-based OWL reasoning for classifying arguments and schemes in a Web-based system. This provides a seed for further work that combines traditional argument-based reasoning techniques [7] with ontological reasoning in a Semantic Web environment.

2. Background: The Core Argument Interchange Format

The AIF is a core ontology of argument-related concepts, and can be extended to capture a variety of argumentation formalisms and schemes. The AIF core ontology assumes that argument entities can be represented as nodes in a directed graph called an *argument network*. A node can also have a number of internal attributes, denoting things such as author, textual details, certainty degree, acceptability status, etc.

Figure 1 depicts the original AIF ontology reported by Chesñevar et al [4]. The ontology has two disjoint types of nodes: *information nodes* (or I-Nodes) and *scheme nodes* (or S-Nodes). Information nodes are used to represent *passive* information contained in an argument, such as a claim, premise, data, etc. On the other hand, S-nodes capture the application of *schemes* (i.e. patterns of reasoning). Such schemes may be domain-independent patterns of reasoning, which resemble rules of inference in deductive logics but broadened to include non-deductive inference. The schemes themselves belong to a class of schemes and can be classified further into: *rule of inference scheme*, *conflict scheme*, and *preference scheme*, etc.

The AIF classifies S-Nodes further into three (disjoint) types of scheme nodes, namely *rule of inference application nodes* (RA-Node), *preference application nodes* (PA-Node) and *conflict application nodes* (CA-Node). The word 'application' on each of these types was introduced in the AIF as a reminder that these nodes function as in-

[6] To our knowledge, the only other OWL specification was by Bart Verheij [6] and predates the AIF.

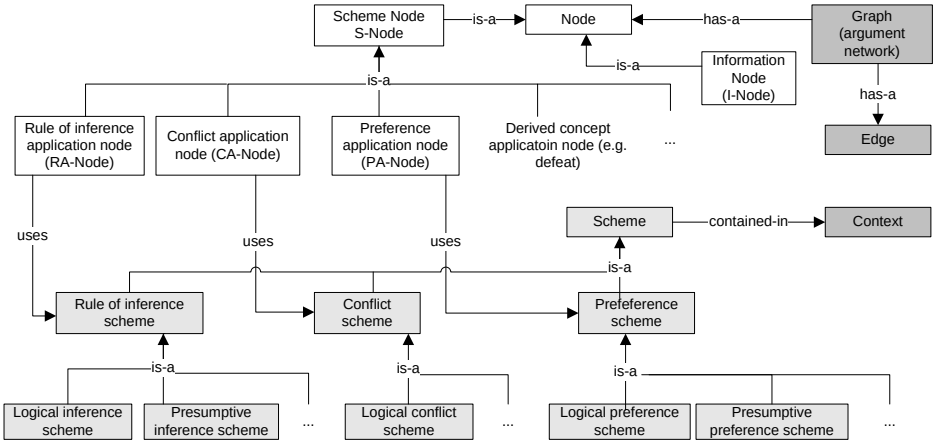

Figure 1. Original AIF Ontology [4]

stances, not classes, of possibly generic inference rules. Intuitively, RA-Nodes capture nodes that represent (possibly non-deductive) rules of inference, CA-Nodes capture applications of criteria (declarative specifications) defining conflict (e.g. among a proposition and its negation, etc.), and PA-Nodes are applications of (possibly abstract) criteria of preference among evaluated nodes. A property named "uses" expresses the fact that an instance of a scheme node *uses* a particular scheme.

The AIF core specification does not type its edges. Edge semantics can be inferred from the types of nodes they connect. The informal semantics of edges are listed in Table 1. One of the restrictions imposed by the AIF is that no outgoing edge from an I-node can be directed directly to another I-node. This ensures that the relationship between two pieces of information must be specified explicitly via an intermediate S-node.

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

Table 1. Informal semantics of untyped edges in core AIF

A simple propositional logic argument network is depicted in Figure 2(a). We distinguish S-nodes from I-nodes graphically by drawing the former with a slightly thicker border. The node marked MP_1 denotes an application of the modus ponens inference rule. An attack or conflict from one information or scheme node to another is captured through a CA-node, which captures the type of conflict. Since edges are directed, symmetric attack would require two sets of edges, one in each direction. Figure 2(b) depicts a symmetric conflict (through propositional negation) between two simple arguments.

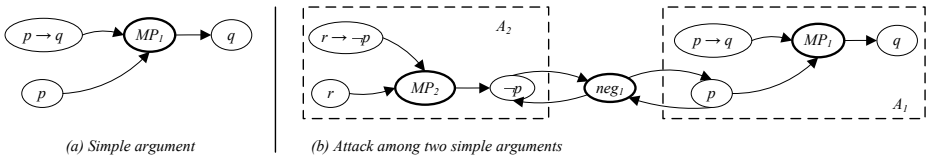

Figure 2. Examples of simple arguments

3. Re-Examining Scheme Reification

3.1. Overview of Argument Schemes

Recently, there has been increasing interest in classifying arguments into different types (or *schemes*) based on the stereotypical inference patterns they instantiate. Many such schemes are referred to as *presumptive* inference patterns, in the sense that if the premises are true, then the conclusion may *presumably* be taken to be true. Structures and taxonomies of schemes have been proposed by many theorists (e.g. Katzav and Reed [8]). But it is Walton's exposition (e.g. recently [9]) that has been most influential in computational work. Each *Walton scheme* has a name, conclusion, set of premises and a set of critical questions. Critical questions enable contenders to identify the weaknesses of an argument based on this scheme, and potentially attack the argument. Here is an example.

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

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

Other schemes include *argument from negative consequence*, and *argument from analogy*, etc. Actual arguments are *instances* of schemes.

Example 2. ***(Instance of Argument from Position to Know)***

- Premise: *The CIA says that Iraq has weapons of mass destruction (WMD).*
- Premise: *The CIA is in a position to know whether there are WMDs in Iraq.*
- Conclusion: *Iraq has WMDs.*

Note that premises may not always be stated, in which case we say that a given premise is *implicit* [9]. One of the benefits of argument classification is that it enables analysts to uncover the hidden premises behind an argument, once the scheme has been identified.

One way to evaluate arguments is through *critical questions*, which serve to inspect arguments based on a particular argument scheme. For example, Walton [9] identified the following critical question for "argument from position to know" (in addition to questioning the, possibly hidden, premises themselves):

Example 3. ***(Critical Questions for Argument from Position to Know)***

1. Trustworthiness: *Is E an honest (trustworthy, reliable) source?*

As discussed by Gordon et al [10], critical questions are not all alike. Some questions may refer to *presumptions* required for the inference to go through, while others may

refer to *exceptions* to the rule, and correspond to Toulmin's *rebuttal* [11]. The contemporary view is that the main difference between presumptions and exceptions lies in the *burden of proof*, but this is beyond the scope of the present paper.

3.2. Schemes in the Original AIF

The initial AIF specification separates the classification of nodes from the classification of schemes (see Figure 1). S-nodes are classified into nodes that capture inference, conflict, etc. Likewise, schemes are classified into similar sub-schemes such as inference schemes, conflict schemes, etc. S-nodes are linked to schemes via a special edge "$uses$."

It should be noted that the original AIF represents an "abstract model," allowing a number of different concrete reifications to be made. The reification of the AIF in the ArgDF ontology defines two classes for representing schemes and nodes [3]. Moreover, ArgDF introduced a new class, *Form node (F-node)*, to capture the generic form of statements (e.g. presumptions, premises) that constitute presumptive arguments (e.g., *PremiseDescriptor* is a sub-class of F-node that captures the generic form of premises).

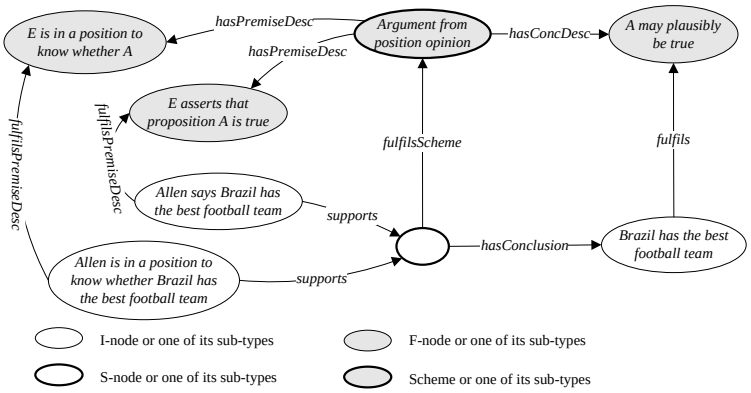

Figure 3. An argument network linking instances of argument and scheme components

In the ArgDF ontology, actual arguments are created by instantiating nodes, while actual schemes are created by instantiating the "scheme" class. Then, argument instances (and their constituent parts) are linked to scheme instances (and their part descriptors) in order to show what scheme the argument follows. Figure 3 shows an argument network for an "argument from position to know" using the ontology of ArgDF. Each node in the argument (unshaded nodes) is explicitly linked, via a special-purpose property, to the form node it instantiates (shaded nodes). These proprieties (e.g. $fulfilsScheme$) are reifications of the "$uses$" relation (between S-nodes and schemes) in the AIF specification.

It is clear that ArgDF's reification of the AIF causes some redundancy. Both arguments and schemes are described with explicit structure at the instance level. Thus, the property "$fulfilsScheme$" does not capture the fact that an S-node represents an *instantiation* of some generic *class of arguments* (i.e. scheme). Having such relationship expressed explicitly can enable reasoning about the classification of schemes (as we shall demonstrate below). The ontology presented in this paper captures this relationship explicitly; presenting a simpler and more natural ontology of arguments. The AIF model is reified by interpreting schemes as classes and S-nodes as instances of those classes; in this case, the semantics of the "$uses$" edge can be interpreted as "$instance - of$".

3.3. Classification of Schemes

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

Example 4. ***(Scheme for Appeal to Expert Opinion)***

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

It is clear that this scheme *specialises* the scheme for argument from position to know. Apart from the fact that both schemes share the conclusion and the assertion premise, the statement "Source E is an expert in domain D" can be seen as a specialisation of the statement that "E is in a position to know (things about A)." Having expertise in a field causes one to be in a position to know things in that field.[7]

Consider also the critical questions associated with the scheme for appeal to expert opinion [9] (again, here we omit Walton's "field" and "opinion" question since it merely questions one of the explicit premises). Notice that the trustworthiness question is repeated, while additional expertise-related questions are added.

Example 5. ***(Critical Questions for Appeal to Expert Opinion)***

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

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

4. A New Argument Ontology in Description Logic

Our formalisation is done using the Web ontology language OWL [5] in Description Logic (DL) notation [2] (see appendix for a short overview of DL). We use a particular dialect of OWL, called OWL DL, which is equivalent to logic $\mathcal{SHOIN}(D)$ [2].

At the highest level, we distinguish between three concepts: *statements* that can be made, *schemes* that represent classes of arguments made up of statements,[8] and *authors* of those statements and arguments. All these concepts are disjoint.

$$\begin{array}{lll} Scheme \sqsubseteq Thing & Author \sqsubseteq Thing & Author \sqsubseteq \neg Statement \\ Statement \sqsubseteq Thing & Statement \sqsubseteq \neg Scheme & Author \sqsubseteq \neg Scheme \end{array}$$

As with the original AIF, we distinguish between rule schemes (which describe the class of arguments), conflict schemes, preference schemes, etc.

[7] Indeed, there may be other reasons to be in a position to know A. For example, if E is taken to refer to society as a whole, then the argument from position to know becomes "argument from popular opinion."

[8] We use use the terms "scheme" and "class of arguments" interchangeably.

$RuleScheme \sqsubseteq Scheme$
$ConflictScheme \sqsubseteq Scheme$
$PreferenceScheme \sqsubseteq Scheme$

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

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

We define a number of properties (or *roles* in DL terminology), which can be used to refer to additional information about instances of the ontology, such as authors of arguments, the creation date of a scheme, and so on. The domains and ranges of these properties are restricted appropriately and described below.[9]

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

To capture the structural relationships between different schemes, we first need to classify their components. We do this by classifying their premises, conclusions, presumptions and exceptions into different *classes of statements*. For example, at the highest level, we may classify statements to declarative, comparative, and imperative, etc.[10]

$DeclarativeStatement \sqsubseteq Statement$
$ImperativeStatement \sqsubseteq Statement$
$ComparativeStatement \sqsubseteq Statement \ldots$

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

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

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

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

[9] The range and domain of property R are described using $\top \sqsubseteq \forall R.C$ and $\top \sqsubseteq \forall R^{-}.C$ (see appendix).

[10] We avoid an ontological discussion of all types of statements. Our interest is in a (humble) demonstration of how a classification of argument *parts* may help automate reasoning about argument *types*. How individual parts get categorised into classes (e.g. using automated or manual tagging) is beyond the scope of this paper.

$PresumptiveArgument \sqsubseteq \forall hasPresumption.Statement$
$PresumptiveArgument \sqsubseteq \forall hasException.Statement$

With this in place, we can further classify the above statement types to cater for a variety of schemes. For example, to capture the scheme for "argument from position to know," we first need to define the following classes of declarative statements. Each class is listed an OWL-DL *annotation property* called `formDescription` which describes the statement's typical form. Annotation properties are used to add meta-data about classes.

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

Now we are ready to fully describe the scheme for "argument from position to know." The following are the necessary as well as the necessary-and-sufficient conditions for an instance to be classified as an argument from position to know.

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

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

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

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

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

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

$AppToExpertOpinion \equiv (PresumptiveArgument \sqcap$
$\exists hasConclusion.KnowledgePositionStmnt \sqcap$
$\exists hasPremise.FieldExpertiseStmnt \sqcap \exists hasPremise.KnowledgeAssertionStmnt)$
$AppToExpertOpinion \sqsubseteq \exists hasException.LackOfReliabilityStmnt$
$AppToExpertOpinion \sqsubseteq \exists hasException.ExpertiseInconsistencyStmnt$

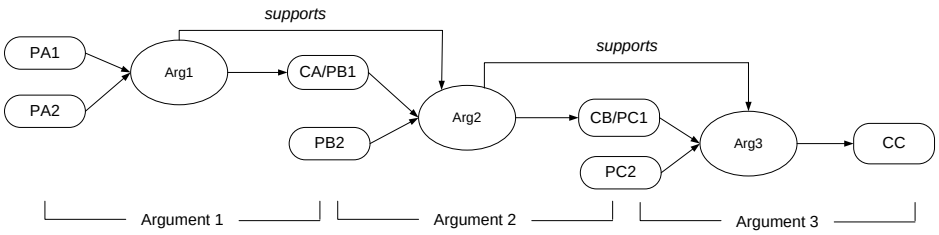

Figure 4. Support among chained arguments

$AppToExpertOpinion \sqsubseteq \exists hasPresumption.CredibilityOfSourceStmnt$
$AppToExpertOpinion \sqsubseteq \exists hasPresumption.ExpertiseBackUpEvidenceStmnt$

It is important to note that a single statement instance might adhere to different types of premises, conclusions or even presumptions and exceptions as the ontology should enable re-using existing statement instances and creating interlinked and dynamic argument networks.

5. OWL Reasoning over Argument Structures

In this section, we discuss ways in which the expressive power of OWL and its support for reasoning can be used to enhance user interaction with arguments. We focus on features that extend our previous work on the RDF Schema-based ArgDF system [3].

5.1. Inference of Indirect Support in Chained Arguments

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

Arguments can support other arguments by supporting their premises. This results in argument *chaining* where a claim acts both as a premise of one argument and as a conclusion of another. This situation is illustrated in Figure 4. In Argument 1, premises PA1 and PA2 have the conclusion CA which is used at the same time as premise PB1 of the argument 2. Premises PB1 and PB2 have the conclusion CB which is used at the same time as premise PC1 of argument 3; PC1 and PC2 have the conclusion CC. Here, we can say that Argument 1 *indirectly* supports Argument 3.

A user may wish to retrieve all arguments that directly or indirectly support conclusion CC. RDF Schema does not provide straightforward support for retrieving this information. We added a transitive property *supports* to the ontology, linking the supporting argument to the supported argument in a chain: $RuleScheme \sqsubseteq \forall supports.RuleScheme$. By using this edge, and the description logic reasoner, small and elegant queries can retrieve the desired information.

5.2. Automatic Classification of Argument Schemes and Instances

As explained above, due to the hierarchy of specialisation among different descriptors of scheme components (i.e. statements) as well as the necessary and sufficient condi-

tions defined on each scheme, it is possible to infer the classification hierarchy among schemes.

Example 6. ***(Inferring scheme relationships)*** *Following from the statement and scheme definitions of "appeal to expert opinion" and "argument from position to know" outlined earlier, the reasoner infers that the former is a sub-class of the latter.*

Similar inferences can be undertaken over other classes. A more elaborate example involves inferring the "fear appeal argument" scheme as sub-class of "argument from negative consequence." Consider the specification of the argument schemes of "argument from negative consequence" and "fear appeal argument." The necessary-and-sufficient part of scheme description of the above arguments are detailed as follows.

$$ArgNegativeConseq \equiv (PresumptiveArgument \sqcap \exists hasConclusion.ForbiddenActionStmnt \sqcap \exists hasPremise.BadConsequenceStmnt)$$

$$FearAppealArg \equiv (PresumptiveArgument \sqcap \exists hasConclusion.ForbiddenActionStmnt \sqcap \exists hasPremise.FearfulSituationStmnt \sqcap \exists hasPremise.FearedBadConsequenceStmnt)$$

The statements are defined as follows. Note that the "Feared Bad Consequence" statement is a specialisation of "Bad Consequence" statement, since it limits the bad consequence to those portrayed in the fearful situation.

$BadConsequenceStmnt \sqsubseteq DeclarativeStatement$
  `formDescription` : "If A is brought about, bad consequences will plausibly occur"
$ForbiddenActionStmnt \sqsubseteq DeclarativeStatement$
  `formDescription` : "A should not be brought about"
$FearfulSituationStmnt \sqsubseteq DeclarativeStatement$
  `formDescription` : "Here is a situation that is fearful to you"
$FearedBadConsequenceStmnt \sqsubseteq BadConsequenceStmnt$
  `formDescription` : "If you carry out A, then the negative consequences portrayed in this fearful situation will happen to you"

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

5.3. Inferring Critical Questions

Since the schemes are classified by the reasoner into a hierarchy, if certain presumptions or exceptions are not explicitly stated for a specific scheme but are defined on any of its super-classes, the system is able to infer and add those presumptions and exceptions to instances of that specific scheme class. Consider the critical questions for "fear appeal argument" and "argument from negative consequence" described below.

Example 7. ***(Critical Questions for Fear Appeal Argument)***

1. *Should the situation represented really be fearful to me, or is it an irrational fear that is appealed to?*

2. *If I don't carry out A, will that stop the negative consequence from happening?*
3. *If I do carry out A, how likely is it that the negative consequence will happen?*

Example 8. ***(Critical Questions for Argument From Negative Consequence)***

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

"Fear appeal argument" is classified as a sub-class of "argument from negative consequence." The critical questions 2 and 3 of "argument from negative consequence" have not been explicitly defined on "fear appeal argument," but can be inferred. Since critical questions provide a way for evaluation of an argument, inferring such additional questions for can enhance the analysis process.

6. Implementation

In this section, we describe our implementation (in-progress) of a Web-based system for creating, manipulating, and querying complex argument structures. The core Website is built on Java. Jena[11] provides the programming environment for manipulating the ontology model. Moreover, ARQ[12] libraries are used to provide the SPARQL[12] query engine. Pellet [13], an open source description logic reasoner for OWL-DL enables inference over the ontology model generated by Jena. The ontology and instances are stored in an SQL Server database. In brief, the new implementation offers the following features:

- Creation of new semantically annotated arguments, using new or existing authored statements. While this feature was implemented in ArgDF, the new system uses subsumption reasoning to infer and add critical questions to each new argument instance.
- Attacking and supporting parts of existing arguments.
- Retrieving supporting or attacking arguments/claims for a given claim. In case of support, both direct and indirect supporting arguments are listed.
- Retrieving scheme details, in order to inspect them, such as the conclusion, premise, presumption or exception descriptors as well as the scheme's inferred super-class(es) and sub-class(es).
- Creation of new schemes through the user interface.
- Search for arguments based on keywords, authors, schemes. When searching for arguments of a specific scheme type, inference is used to return all the arguments that are instances of that specific scheme as well as instances that belong to any of its sub-classes.

Although some of the above features have already appeared in ArgDF, the key feature of the current implementation is its use of OWL inference to enhance the retrieval of arguments (as described in detail in Section 5).

[11] http://jena.sourceforge.net/
[12] http://jena.sourceforge.net/ARQ/

7. Conclusion

We reported on ongoing work that exploits the OWL language for creating, navigating and manipulating complex argument structures. The new ontology enhances our previous RDF Schema-based implementation [3]. In particular, we now model schemes as classes (as opposed to instances), which enables detailed classification of schemes themselves. Secondly, our new system enables the first explicit use of Description Logic-based OWL reasoning for classifying arguments and schemes in a Web-based system. This provides a seed for further work that combines traditional argument-based reasoning techniques [7] with ontological reasoning in a Semantic Web environment. Once the system implementation is complete, we aim to focus on content acquisition. We will explore the integration of arguments from other repositories into our system. We will also work on integrating effective argument visualisation techniques, which can help in acquisition as well as interaction with the argument repository.

Acknowledgement

We are grateful for the detailed comments received from the anonymous reviewers.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 29–37, May 2001.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, Cambridge, UK, 2003.

[3] I. Rahwan, F. Zablith, and C. Reed. Laying the foundations for a world wide argument web. *Artificial Intelligence*, 171(10–15):897–921, 2007.

[4] C. I. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2007.

[5] D. L. McGuinness and F. van Harmelen. OWL web ontology language overview. W3C Recommendation REC-owl-features-20040210/, World Wide Web Consortium (W3C), February 2004.

[6] B. Verheij. An argumentation core ontology as the centerpiece of a myriad of argumentation formats. Technical report, Agentlink Argumentation Interchange Format Technical Forum, 2005.

[7] C. I. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32(4):337–383, 2000.

[8] C. Reed and J. Katzav. On argumentation schemes and the natural classification of arguments. *Argumentation*, 18(2):239–259, 2004.

[9] D. N. Walton. *Fundamentals of Critical Argumentation*. Cambridge University Press, New York, USA, 2006.

[10] T. F. Gordon, H. Prakken, and D. Walton. The carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10–15):875–896, 2007.

[11] S. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.

[12] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Candidate Recommendation CR-rdf-sparql-query-20070614, World Wide Web Consortium (W3C), 2007.

[13] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics*, 5(2):51–53, 2007.

Appendix: Description Logics

Description Logics (DLs) [2] are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain. The idea is to define complex concept hierarchies from basic (atomic) concepts, and to define complex roles (or properties) that define relationships between concepts.

Table 2 shows the syntax and semantics of common concept and role constructors. The letters A, B are used for atomic concepts and C, D for concept descriptions. For roles, the letters R and S are used and non-negative integers (in number restrictions) are denoted by n, m and individuals (i.e. instances) by a, b. An *interpretation* $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

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

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

Table 2. Some Description Logic Role Constructors, Concept Constructors, and Terminological Aximos

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

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

An equality whose left-hand side is an atomic concept is a $definition$. Definitions are used to introduce symbolic names for complex descriptions. For instance, by the axiom $Mother \equiv Woman \sqcap \exists hasChild.Person$, we associate to the description on the right-hand side the name $Mother$. Symbolic names may be used as abbreviations in other descriptions. If, for example, we have defined $Father$ analogously to $Mother$, we can define $Parent$ as $Parent \equiv Mother \sqcup Father$. Table 3 shows a terminology with concepts concerned with family relationships.

The sentence $\top \sqsubseteq \forall hasParent.Person$ expresses that the range of the property $hasParent$ is the class $Person$ (more technically, if the property $hasParent$ holds between any concept and another concept, the latter concept must be of type $Person$).

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

Table 3. A terminology (TBox) with concepts about family relationships