

# **ArgDF: Arguments on the Semantic Web**

Fouad Zablith

Master of Science

Division of Informatics

Knowledge and Data Management Stream

The British University in Dubai Jointly with

The University of Edinburgh

February, 2007

# Abstract

This dissertation explores the incorporation of the Semantic Web technology with one of the latest argument representation formats. First I introduce an extension of the Argument Interchange Format (AIF) ontology; then I put forward AIF-RDF, which is the extended ontology represented in the Resource Description Framework Schema (RDFS) semantic language. Finally, based on the theoretical and ontology foundations described in the thesis, I present the implementation of ArgDF: a Semantic Web-based system built to represent and visualize arguments. ArgDF comprises a set of tools such as the integration of a variety of argumentation schemes with the possibility of adding new schemes from the user interface. Another feature is to attack and support existing expressions in ArgDF, or to use existing statements in new arguments, leading to the possibility of building a set of interlinked arguments networks. ArgDF is the outcome of the fusion of the Semantic Web with the argument representation format theory, resulting in the emergence of a rich application tool for authoring new arguments or manipulating existing ones through structured argumentation schemes, coupled with arguments visualization, navigation and advanced search techniques.

# Acknowledgments

First, getting done with my research could not be possible without the support of my wonderful life partner, Rania, who has always been by my side even during periods when we were far from each other. I am lucky to have Rodolph and Sonia, my great parents, my sister Nadine and my brother Jad. They all had a major impact throughout my life stages. I would like to thank Dr. Iyad Rahwan, my supervisor, for his time and support during all my dissertation phases. Most of the Argument Interchange Format ontology extensions are the result of an extensive cooperation with Dr. Iyad Rahwan and Dr. Chris Reed whom I would like to thank as well. In addition to this thesis document, the theoretical background will appear in a paper in the proceedings of the IJCAI Workshop on Computational Models of Natural Argument (CMNA) 2007 [40]. Many thanks to Simon Wells for maintaining and setting up the University of Dundee ArgDF repository, and to Nikolai Bykov for maintaining the British University in Dubai ArgDF repository. Last, but not least, I am grateful to the Emirates Airlines for the grant that helped me in pursuing my master's degree.

# Table of Contents

<b>List of Figures</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Motivation . . . . .	3
1.3 Goal . . . . .	4
1.4 Scope of the Project . . . . .	5
1.5 Summary of Achievement . . . . .	5
1.6 Structure of the Document . . . . .	6
<b>2 Background</b>	<b>7</b>
2.1 Argumentation Systems . . . . .	7
2.1.1 Systems about Formalisms for Inference of Arguments . .	7
2.1.2 Systems for Argumentation Based Decision Making . . . .	8
2.1.3 Systems for Argumentation-Based Dialogues . . . . .	11

## TABLE OF CONTENTS

2.1.4	Systems for Argumentation in the Legal Domain . . . . .	11
2.1.5	Systems for Visualizing Arguments & Learning Domain .	12
2.1.6	Limitations of Current Argumentation Systems . . . . .	13
2.2	Open Argument Representation on the Web . . . . .	14
<b>3</b>	<b>AIF Core Ontology and Proposed Extensions Applied</b>	<b>16</b>
3.1	Introduction . . . . .	16
3.2	Background: Argument Representation in AIF . . . . .	17
3.3	AIF Extension . . . . .	20
3.3.1	Representing Argumentation Schemes . . . . .	20
3.3.2	Argument Representation . . . . .	26
<b>4</b>	<b>AIF-RDF: Ontology Implementation in RDF</b>	<b>30</b>
4.1	Introduction . . . . .	30
4.2	Background: RDF & RDFS . . . . .	30
4.3	Benefits of Using RDF vs. XML . . . . .	32
4.4	Implementation in RDF & RDFS . . . . .	34
4.4.1	Nodes Instantiation . . . . .	34
4.4.2	Edges Instantiation . . . . .	35
<b>5</b>	<b>ArgDF: A Semantic Web-based Argumentation System</b>	<b>39</b>
5.1	Introduction . . . . .	39

## TABLE OF CONTENTS

5.2	ArgDF Platform Overview . . . . .	40
5.2.1	ArgDF Repository: Sesame RDF Server . . . . .	40
5.2.2	Web Scripting: PHP . . . . .	42
5.2.3	PHP - Sesame Communication: Phesame . . . . .	43
5.2.4	Querying the ArgDF Repository: RQL . . . . .	44
5.2.5	Rendering and Visualization: XSLT & XPath . . . . .	45
5.2.6	URI Automatic Generation: MySQL Database . . . . .	46
5.3	Creating New Arguments . . . . .	47
5.4	Argument Extension . . . . .	50
5.4.1	Support/Attack of Existing Expressions . . . . .	50
5.4.2	Linking Existing Premises to a New Argument . . . . .	51
5.4.3	Attacking Arguments through Implicit Assumptions . . . . .	51
5.5	Advanced Argument Search . . . . .	54
5.6	Creation of New Schemes . . . . .	55
<b>6</b>	<b>Conclusion and Open Issues</b>	<b>56</b>
	<b>Appendices</b>	<b>60</b>
<b>A</b>	<b>ArgDF User Manual</b>	<b>60</b>
A.1	Creating a New Argument . . . . .	61
A.2	Explicitly Attacking an Existing Claim . . . . .	62

## TABLE OF CONTENTS

A.3	Supporting an Existing Claim . . . . .	63
A.4	Inspecting an Existing Claim and Implicit Attacks . . . . .	64
A.5	Using Existing Premises in New Arguments . . . . .	68
A.6	Argumentation Schemes Manipulation . . . . .	68
<b>B</b>	<b>Building ArgDF in Depth</b>	<b>72</b>
B.1	Phesame for PHP and Sesame communication . . . . .	72
B.1.1	Opening the Sesame Connection . . . . .	72
B.1.2	Setting the RDF Repository & Input/Output Parameters .	73
B.2	Writing in the RDF Repository & Uploading RA-Node Example .	74
B.3	RQL and Premises List Query Example . . . . .	75
B.4	Transforming XML Query Output through XSLT . . . . .	76
B.5	Creating a New Argument Full Cycle . . . . .	78
B.5.1	Choosing the Argumentation Scheme . . . . .	78
B.5.2	Creation of the RA-Node . . . . .	79
B.5.3	Creation of the Conclusion & Premises . . . . .	80
B.6	Creating a New Argumentation Scheme . . . . .	84
<b>C</b>	<b>Full AIF-RDF RDFS Ontology Code</b>	<b>87</b>
<b>D</b>	<b>ArgDF Argument RDF Code</b>	<b>92</b>

## List of Figures

3.1	Concepts and relations in the AIF ontology . . . . .	19
3.2	The base for a network representation of the scheme example . .	22
3.3	The extended representation of the scheme example . . . . .	25
3.4	A graphical representation of the scheme including exceptions . .	25
3.5	A full example with an argument & implicit attacks . . . . .	28
3.6	Extensions to the original AIF . . . . .	29
4.1	RDF triples graphical representation . . . . .	31
4.2	RDF triples graphical example . . . . .	32
4.3	Class/ subclass relationship example . . . . .	35
4.4	Statements in conflict example . . . . .	37
4.5	Statements in symmetric conflict example . . . . .	37
5.1	ArgDF System Architecture . . . . .	41
5.2	New Argument Creation Cycle . . . . .	48
5.3	XSLT Table Output of Argument Schemes . . . . .	48



## LIST OF FIGURES

5.4	Argument Creation in ArgDF . . . . .	50
5.5	Listing Existing Claims . . . . .	51
5.6	Interlinked Arguments Network . . . . .	52
5.7	Implicit Attack Through an Exception in ArgDF . . . . .	53
5.8	Implicit Attack Through Undermining a Presumption in ArgDF .	53
5.9	ArgDF Advanced Search Tool . . . . .	54
5.10	ArgDF Advanced Search Result . . . . .	54
5.11	Creating a new Scheme in ArgDF Example . . . . .	55
A.1	ArgDF Home Page . . . . .	60
A.2	Argument Creation Scheme Choice . . . . .	61
A.3	New Argument Notification . . . . .	61
A.4	Argument Parts Creation . . . . .	62
A.5	Entering the Argument's Conclusion . . . . .	62
A.6	Argument Creation Ending . . . . .	62
A.7	Claims List in ArgDF . . . . .	63
A.8	Attacking Existing Claim . . . . .	63
A.9	Supporting an Existing Claim . . . . .	63
A.10	Opening an Existing Argument . . . . .	64
A.11	Existing Argument Details . . . . .	64

## LIST OF FIGURES

A.12 Existing Argument Exceptions Details . . . . .	65
A.13 Attacking Through Exception Confirmation . . . . .	65
A.14 Entering the Exception . . . . .	65
A.15 Argument Exceptions Updated . . . . .	66
A.16 Implicit Argument Attack Through Presumptions - 1 . . . . .	66
A.17 Implicit Argument Attack Through Presumptions - 2 . . . . .	67
A.18 Entering the Undermining Presumption . . . . .	67
A.19 Undermining Presumption Displayed . . . . .	67
A.20 Using an Existing Premise . . . . .	68
A.21 List of Existing Premises . . . . .	68
A.22 Argumentation Scheme Details . . . . .	68
A.23 Entering the New Scheme Name . . . . .	69
A.24 Entering the New Scheme Conclusion . . . . .	69
A.25 Entering the New Scheme Premises . . . . .	70
A.26 Entering the New Scheme Entailed Presumption . . . . .	70
A.27 Entering New Conflict Scheme Name . . . . .	70
A.28 Entering Conflict Scheme Corresponding Premise . . . . .	71
A.29 New Scheme Creation Final Screen . . . . .	71

# Chapter 1

## Introduction

### 1.1 Introduction

Analyzing communication between entities could not be done throughout history without giving argumentation a great deal of study. It started back with the emergence of philosophy and it has always been highly considered in other fields such as psychology [13] and business decision making. Argumentation can be defined as the process of putting together a set of statements aimed to strengthen or weaken a claimed expression. It's considered as a core necessity for reaching a kind of agreement between two or more entities having different understandings about a certain topic.

Today the argumentation field has been extended to the computer science and artificial intelligence domain, especially with areas dealing with multi-agent communication [11, 12, 38]. It has also played a major role in the decision support systems field.

A variety of systems are now available for users to create and represent arguments such as AML Araucaria [42], TruthMapping [48], Reason!Able [22], Compendium [6], and others. But most of them use rigid programming languages such as XML or database structures for representing and storing arguments, making them hard to extend and unable to capture rich ontological concepts. Another drawback of the current systems is the lack of relying on unified argumentation concepts, making it hard to create seamless exchange of data spread across many repositories.

Expressively rich Semantic Web [7] languages offer more flexible and extendable ways for representing arguments. Semantic Web standards have opened the road for new data storage and interchange techniques. The Semantic Web is intended to allow better domain representations and more expressive service or resource descriptions, coupled with an easy exchange and navigation of data.

In addition to relying on rich semantics language, a set of formalized and unified rules for arguments representation facilitate the exchange of arguments between different argumentation systems. A unified representation could as well be serving as a platform in multi-agents environments, where software agents<sup>1</sup> coming from different backgrounds can “argue” using the proposed formalized rules.

Arguments can follow different patterns or scenarios known as argument schemes [51]. An argument scheme is a way to structure an argument following pre-

---

<sup>1</sup>A software agent is an autonomous piece of software, able to operate and act by itself based on the inputs of other software sources. It is also designed to communicate and argue with other software agents to come up with certain decisions.

defined structuring techniques allowing argument builders to follow specific requirements, and helping argument readers to better understand argument representations.

This thesis aims to pass through the different stages, starting from the theoretical layer down to the technical layer, of building an argumentation system with the following properties: Open decentralized Web-based system with highly expressive programming techniques such as the Semantic Web to store, represent and query arguments following specific argument schemes and based on a unified argumentation standard.

## 1.2 Motivation

Different grounds are behind the motivation of this research work. It has been observed in the ASPIC project that:

“Argumentation tools can be seen to be essentially proprietary, introspective, and unable to leverage shared functionality or content.”

[4, page 61]

Firstly, the word proprietary reflects the need for an open argumentation system. Secondly, the word introspective reveals the necessity of a domain independent argumentation system where a variety of argumentation schemes can be used. The final point is pushing towards the need for a system that can share functionalities with other systems, and share its content by being decentralized

and Web enabled.

In addition to the above, the work in this thesis fills a gap in the field of decision support system tools. As presented in Chapter 2 of this document, decision support systems have reached a stagnant point in their development: systems are either highly structured such as decision support systems tools, or highly scalable such as blogs and online forums accessed through websites. There has been no further work on developing support systems that could be at the same time highly structured, and highly scalable.

This has triggered the need for a decentralized open Semantic Web-based argumentation system fulfilled by ArgDF, able to store, create, update and query argument structures, using a unified ontology and representing various argument schemes.

### **1.3 Goal**

The goal of this project is to go through the different stages of creating a domain independent Web-based semantic argumentation system: starting from the theoretical concepts laid by the Argument Interchange Format (AIF) [13] project, then moving to the extension of the AIF core ontology. This extension is implemented in a Semantic Web language, to finally present ArgDF, a semantically rich, open and decentralized Web-based system, enabling users to create new arguments on the Web fulfilling different argumentation schemes as well as manipulating and visualizing existing arguments, that could be later on spread

over many data repositories.

## **1.4 Scope of the Project**

The project's scope is to go through the different stages of building an argumentation Web-based system formed of different components working collaboratively in order to deliver a Semantic Web enabled platform. The system developed does not focus on argument acceptability calculation and evaluation [19], but on the argument representation and authoring. Arguments have a structure but can be represented in many languages such as natural language. Arguments can also belong to different argumentation scheme theories, such as Walton's schemes [51], which are used in this thesis. This tool should be designed in order to fulfill the requirements of an open argumentation system. In brief, this system relies on a unified ontology, Semantic Web and a set of expendable argumentation schemes.

## **1.5 Summary of Achievement**

The work in this thesis involves designing an ontology for describing arguments. The design is based on the recently proposed Argument Interchange Format (AIF), and is extended in order to capture a specific theoretical model of argument based on Walton's schemes set [51]. This is the first extension of AIF capturing Walton's schemes. The design is followed by its implementation

in RDF to form the AIF-RDF ontology. A Web application layer is built on top of this ontology enabling users to author new arguments or manipulate and visualize existing ones as well as keyword or advanced argument search. Resource Description Framework semantic language (RDF) and RDF schema (RDFS) is used. The Sesame [44] RDF repository is relied on, coupled with a PHP scripting interface and MySQL database. Extensible Style Sheet Language (XSLT) and XPath for XML processing and rendering are used for data visualization.

ArgDF is the first generic, or in other words domain independent, open Web-based argumentation support system that uses the Semantic Web technology, relying on unified argumentation standards such as the Argument Interchange Format (AIF). ArgDF is available at this web address: [www.argdf.org](http://www.argdf.org).

## 1.6 Structure of the Document

In the next chapter I discuss the requirements of open arguments' representation and give an overview about the existing argumentation systems. In Chapter 3 I present the actual status of the AIF project and its extension. Chapter 4 is related to the AIF-RDF ontology design of the proposed system. In Chapter 5 I talk about the ArgDF system and its features. Four appendices come after the conclusion: One is a user manual for ArgDF, another is about building the ArgDF in depth, and the last two include the full ontology code and an argument RDF code example.



## Chapter 2

### Background

#### 2.1 Argumentation Systems

Various approaches and computer systems have been previously explored to represent and process arguments. Four main categories are usually targeted by argumentation systems: formalisms for inference of arguments, argumentation-based decision making, argumentation-based dialogues and legal domain, and finally the argumentation and learning category [4].

##### 2.1.1 Systems about Formalisms for Inference of Arguments

Argumentation systems dealing with the formalisms for inference of arguments have an inference engine for drawing argumentation conclusions, which are separated from the construction of conclusions. Their focus is on proving entailment theorems. Pollock's *OSCAR* [35] is an example of a system implemented that can perform inferences in the concept of defeasible reasoning. De-

feasibility is a relation between two statements of arguments allowing one of them to defeat the other at any stage [21]. It has been developed using the LISP programming language. An example of defeasible logic programming (DeLP) rules can be checked over the web.<sup>1</sup>

*IACAS* [50] is another system working on defeasible argumentation. It has a command line based interface, and allows also an evaluation of the status of arguments being: Certain, balanced, undetermined etc.

All the above systems are using simple rule-like based programming languages and are generally used for theoretical concept proofing. They are mainly stand alone systems, except for *OSCAR* which is embedded into other systems.

### 2.1.2 Systems for Argumentation Based Decision Making

Generally speaking, all argumentation systems indirectly help users or software agents in reaching a certain decision or outcome. Argumentation systems considered as decision making helpers are either systems enabling users to represent arguments and leave the assessment for the user, or systems able to evaluate the chain of arguments and suggest the best decision to be taken.

There are systems for decision making under uncertainty such as *REACT* [20] in which explanations are structured as arguments to facilitate decision making. It does not support new arguments structuring or preferences. Arguments are only hard-coded as explanations for specific problems, helping to reach a single

---

<sup>1</sup><http://lidia.cs.uns.edu.ar/DeLP>

decision being the case of the medical domain. *RAGs* [17] is an argumentation system in which the decision reached includes risk assessment. *STAR* [30] is another decision support system based on risk assessment. It is used in commercial applications. *PRETI* [1] is a project aiming to build systems based on a probabilistic approach for decision making. These systems have a weakness in the knowledge expressiveness, where only limited type and domain dependent knowledge can be expressed. Another weakness is in the need for a continuous information update, as old information could become outdated for making decisions under uncertainty. The latter has led to the need of merging and connecting these systems to knowledge sources for providing an up-to-date set of knowledge.

Another group of systems aid in decision making in collaborative environments. This is when there is a group of persons or agents collaboratively looking for a decision about a certain topic or problem. Most applications in this category aim for linking arguments with each other using relationships and claims. Many systems are available such as *graphical Issue Based Information System* (gIBIS) [16] that, developed in 1998, helps linking the Issue Based Information System (IBIS) nodes *Issue*, *Position* and *Argument* with external resources to form an IBIS network. A drawback of gIBIS is that it generates a lot of overhead while using it in decision making scenarios. *QuestMap*<sup>TM</sup> is a more user-friendly version of gIBIS for capturing online discussions among peers in a team [15]. In addition to the features of QuestMap, *Compendium* [14] is a commercial application based on IBIS, which offers enhanced tools such as XML &

RDF support, as well as connection to other data stores. Another application falling in this domain is *SIBYL* [31], which is more expressive than the gIBIS tool and allows qualitative evaluation of arguments. *Zeno* [23], another IBIS based tool, is a Web-based system supporting participants during online discussions. Users are able to structure their arguments and set their preferences among them. Different tools made use of Zeno such as the *GeoMed* system [43] facilitating discussions about city planning decisions. The *Delphi Mediation Online System (DEMOS)* [32] relies on Zeno as a core component. DEMOS is a system used for “online-democracy” facilitating debates among participants. *HERMES* [28] is another decision-support system based on Zeno, including information retrieval supporting specific claims useful in the medical field. HERMES also supports constraint solving and conflict detection features, as well as case-based reasoning allowing the adaptability on the status and specific cases of discussions and the team’s collaborators.

TruthMapping [48] is a Web-based system for capturing conversations in a group. In brief, it helps users store their claims and support them with corresponding links. Participants can agree or disagree over a certain topic that has already been posted. *Parmenides* [5] is another Web-based system for supporting deliberation over actions, and is based on a specific scheme for persuasion over action.

### 2.1.3 Systems for Argumentation-Based Dialogues

Argumentation in the dialogue field is aimed to capture communication between humans or software agents. There are different dialogue types that reflected on the design of the systems: Information-seeking dialogues, inquiry, persuasion, negotiation, and deliberation dialogue [52]. Examples of argumentation-based dialogue systems are *Homey* [8–10] applied in the medical field, and the *PARMA* [25, 26] model. In such systems, an agreement of the semantics should be set between the communicating subjects, in order to be working on unified communication standards to avoid interaction clashes. For example, a software agent set to analyse dialogues in a *Homey* system should be programmed specifically to be able to render *Homey*'s dialogues, and can not be used in other argumentation-based dialogues systems.

### 2.1.4 Systems for Argumentation in the Legal Domain

Most of the systems dealing with argumentation in the legal domain are still under research. Such systems aim to apply the argumentation process in fields where there are conflicting interpretations over the same topic. Another aim is to enable reasoning with precedents like the *HYPO* system [3], which require relying on previous facts to conform and support current similar facts. A third type of such systems is reasoning under disagreement with the presence of conflicting rules or their legal validity. *CABARET* [46] is a system that deals with the tactics for dispute using rules and cases. *ArguMed* [49] and *ProSupport* [37]

are also designed for a legal audience.

### **2.1.5 Systems for Visualizing Arguments & Learning Domain**

Visualizing arguments is dated back to 1913 with Wigmore's "Chart Method" [47]. Now with the presence of computers' visual techniques, it became easier to process and visualize large amount of argumentation data. Visualization of arguments is important in collaborative environments where people are having a certain debate, or in learning environments in which students may be able to understand more about arguments with their visualization features, and have a clearer idea about the basics of argumentation. Today we can see some Web-based systems like TruthMapping using visualization techniques to show the status of the arguments entered by users in the Web system.

Some systems are categorized in the learning field such as: Reason!Able [22], Athena [41] and Araucaria [42]. Reason!Able is a tool through which users can build arguments using a simple straight forward graphical interface. Athena is another easy to use tool offering argument visualization features.

Araucaria AML is an argumentation system through which arguments can be built starting from an argumentation text input file, by specifying the parts of the arguments to be "marked up" using the Argument Mark-up Language (AML). One of the Araucaria's advantages is that it allows the annotation of arguments using specific schemes, and a visualization of the schemes is shown to the users to assist them in the way their arguments should be built.

ClaiMaker [45] is another argumentation system that includes visualization of arguments. Questmap and Compendium also use a set of visualization tools targeting professional areas such as the legal domain, and supporting business firms for decision making.

### **2.1.6 Limitations of Current Argumentation Systems**

Nearly most of the current argumentation systems deal with argumentation without targeting the issue of interconnectivity between different systems, and flexibility of extension. Some few systems, such as Araucaria, use an XML representation with a well specified Document Type Definition (DTD) which could form the basis of such exchange. Relying on databases, as in the case of TruthMapping and Parmenides, could somehow create a kind of rigidity in extending structures. The modification of a database schema usually requires taking care of the tables reference keys and tables structures, requiring the involvement of the database administrators and designers. On the other hand, a Semantic Web schema is somehow more flexible when it comes to building on existing ontologies.

Another drawback is the trade-off between scalability and structure in decision support systems. Systems capturing arguments from a large number of participants such as forums and online blogs are highly unstructured. They are usually of the form of a text box where users can enter any piece of information without abiding by any predefined structures. TruthMapping's arguments are a set of premises, supporting a conclusion, without abiding by specific schemes

or argumentation scenarios. This might not be enough in academic and accurate decision making backups. Users will not be able to apply specific argumentation techniques known as *argument schemes* [51]. On the other side, highly structured systems such as decision support tools are highly structured and are implemented in small scale environments due to their specialized tasks.

Domain dependency is also considered a drawback, as most of current systems depend on specific domains and argumentation theories. For example the Parmenides system is based on a specific theory of persuasion over action. This make them hard to be tested in different domains and scenarios.

## 2.2 Open Argument Representation on the Web

Having a Semantic argumentative structure on the Web can, on one hand, help in achieving a rich representation and navigation of argument structures Web by providing means for tagging arguments and argument parts; and on the other hand, this same structure may also be used in the future by software agents to enable automated processing and exchange of arguments among software agents.

Targeting the niche of having a highly structured argument representation system and at the same time a highly scalable one, has led to have an open argument representation Semantic system with the following features:

- R1 Supporting the storage, creation, update and query of argument structures;



## CHAPTER 2: BACKGROUND

- R2 Having Web-accessibility features and an open data repository;
- R3 Relying on a language based on open standards, thus enabling collaborative development of new tools and features;
- R4 Using a unified argumentation ontology;
- R5 Supporting the representation, annotation and creation of arguments using a variety of argument schemes;

This dissertation works on the theoretical and software basis to come up with a system fulfilling the above requirements.

## **Chapter 3**

# **AIF Core Ontology and Proposed Extensions Applied**

### **3.1 Introduction**

“An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of existence” [27]. Representing knowledge and conceptual pieces of information in a computer system initially requires designing an ontology, which is the backbone of any Semantic Web system, in order to have a well defined set of concepts and relationships. In this thesis’ case, the ontology is based on the AIF concepts, and includes the core ontology of the AIF, extended to capture Walton’s schemes set.

## 3.2 Background: Argument Representation in AIF

The Argument Interchange Format (AIF) is an effort towards standardizing argumentation concepts, in order to have an ontology through which theoretically any scheme can be applied [13].

The AIF project is a work promising to be offering a lot of benefits such as:

- Having a unified platform where the format of exchanging arguments can be cohesive.
- Reaching a kind of understanding and agreement on a core ontology to which different argumentation scenarios can be tested on.
- Having a compatible platform on which different tools and systems can be plugged in order to facilitate the argumentation tests. Visualization tools can also be interrelated and linked using this format.
- Enabling multi-agent systems to have an easy way to exchange arguments in a formalized way whether within the same framework or spread across multiple frameworks.
- Facilitating the analysis of arguments coming from agents through visual representations and arguments spotting.
- Having an explicit machine readable syntax, and (if applicable) enabled machine processing semantics.

The AIF model focuses on:

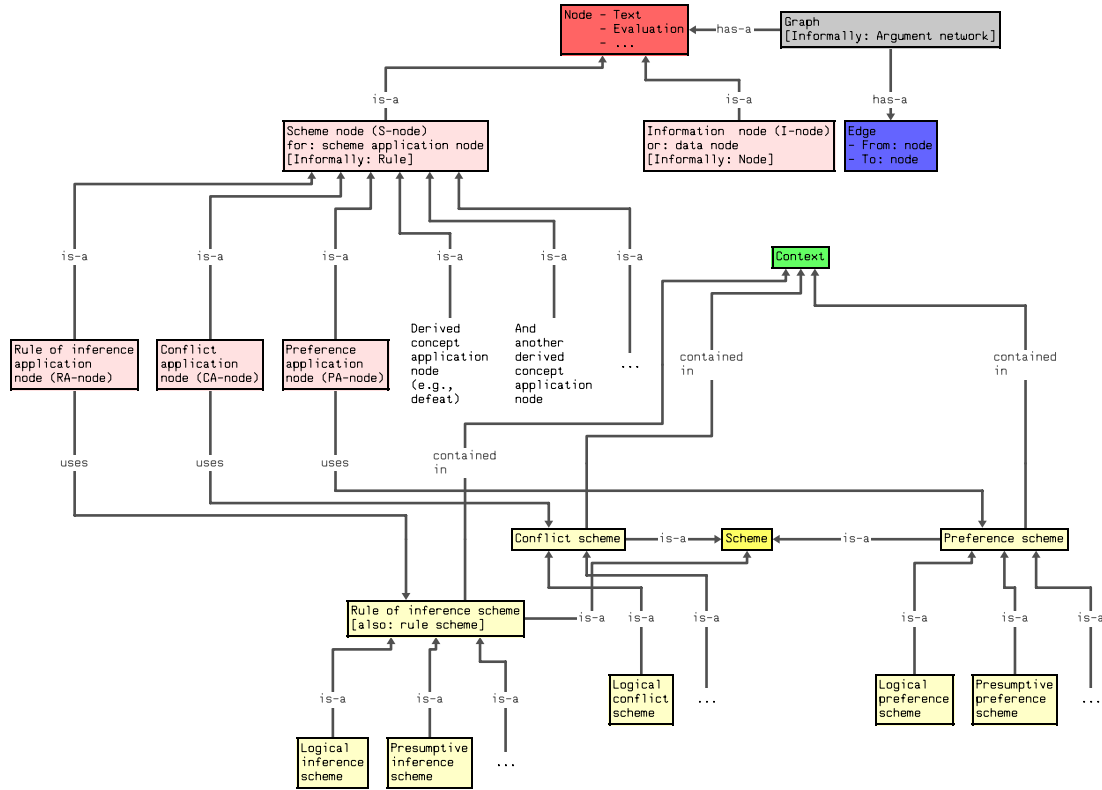
- Having the core concepts that can be enhanced and added-on with time. Then extensions to this core concept can be created, representing more specific domains.
- Having an abstract model in which concepts and relations between them are defined, coupled with concrete syntaxes.

### **Argument Representation: Nodes**

In the AIF ontology, arguments are represented by a set of nodes connected through edges. There are 2 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. Currently there are 3 types of scheme nodes represented in the AIF: the *rule of inference application* (RA-Node) for representing rules such as inference rules, *preference application* (PA-Node) for representing argument preferences and *conflict application* (CA-Node) for modeling conflicts between arguments. The AIF core ontology is represented in Figure 3.1 [13].

### **Argument Representation: Edges**

The edges represent relationships linking argument nodes with each other. In the original AIF specification, edges are not typed. Instead, their semantics is understood implicitly from the types of nodes they connect. There are a set of restrictions imposed by the AIF for managing the directions and allowed con-



**Figure 3.1:** Concepts and relations in the AIF ontology

nections between nodes. For example, an edge coming out from an I-Node can only be directed to an S-Node. Table 3.1 represents the list of implicit semantics of the argument edges in AIF.

As presented in the extended ontology section and in the AIF-RDF chapter, types are explicitly annotated with labels, and more specific types of edges have been added to enable querying and extracting the ontology parts based on the types of edges.

	to <i>I-node</i>	to <i>RA-node</i>	to <i>PA-node</i>	to <i>CA-node</i>
from <i>I-node</i>		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 <i>RA-node</i>	inferring a conclusion in the form of a 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 <i>PA-node</i>	applying a preference over data in I-node	applying a 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 <i>CA-node</i>	applying conflict definition to data in I-node	applying conflict definition to inference application in RA-node	applying conflict definition to preference application in PA-node	showing a conflict holds between a conflict definition and some other piece of information

**Table 3.1:** Informal semantics of untyped edges in the core AIF

### 3.3 AIF Extension

The AIF core ontology has been extended to represent Walton’s scheme sets [51] and represent arguments for delivering a Semantic Web-based system built on the top of this extension.

#### 3.3.1 Representing Argumentation Schemes

The concept of schemes is an idea towards categorization of the way arguments should be built, and offers a common understanding of argument structure. Many theorists such as Toulmin [47] have tackled the issue of argument representation and structures. However, Walton’s schemes were very influential in computational work. As per Walton’s definition [51], “Argumentation schemes are forms of argument representing premise-conclusion and inference structures of common types of arguments.” Each Walton scheme type has a name, conclusion, set of premises and a set of critical questions. Critical questions are

a way to let the user know about the weaknesses of the argument based on a specific scheme, and give a way for others to attack those arguments.

Following is an example of a Walton scheme, for arguments from expert opinion:

*Scheme name:*

- Argument from Expert Opinion

*Premises:*

- E is an expert in domain D
- E asserts that A is known to be true

*Conclusion:*

- A may (plausibly) be taken to be true

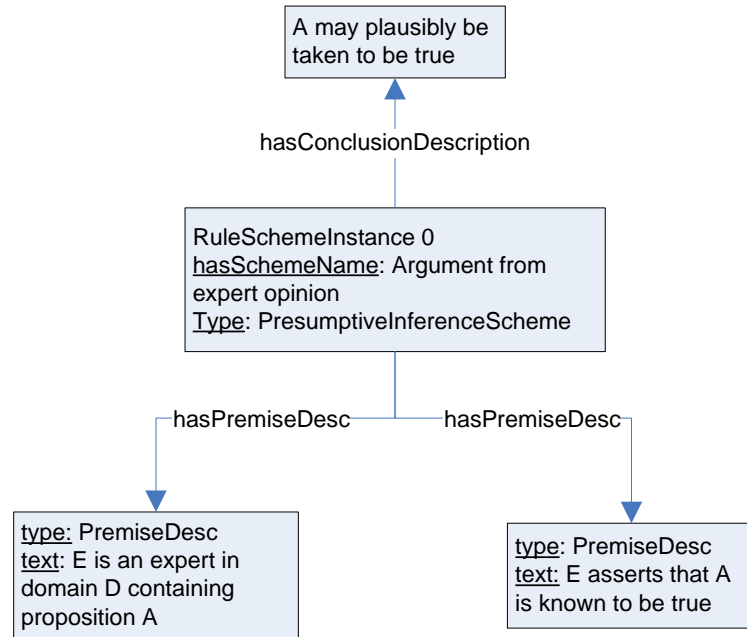
Schemes in our extended ontology are represented as class instances and not as classes. This offers the possibility for the user to add new schemes from the system interface, without having to modify the ontology itself.<sup>1</sup>

The class “SchemeDescription” is the main class handling the main type of the schemes. It has three subclasses: the “ConflictScheme,” “PreferenceScheme” and “RuleScheme.” The “SchemeDescription” general class has 3 attributes:

- *hasSchemeName*: of type “string,” it’s a slot for entering the name of the scheme, having at most one value.

---

<sup>1</sup>This allows for functionality similar to Araucaria’s “schemeset” construction



**Figure 3.2:** The base for a network representation of the scheme example

- *hasPremiseDescription*: used to link the scheme to its corresponding premises.
- *hasConclusionDescription*: used to point to the corresponding scheme’s conclusion.

Walton’s schemes are represented as instances of the class “PresumptiveInferenceScheme.” This scheme design helps in guiding users to specify to which scheme their argument belongs, as in the ontology itself they have scheme instances that act as examples to clarify how to build arguments. An “instance” of a scheme is specified by listing its premises, conclusion and name. Figure 3.2 visualizes the “Argument from expert opinion” scheme graph.

Presumptions and exceptions [24] embedded inside schemes play an important role in argumentation. They open the way for a more fruitful communication among peers who are able to know the presumptions that the argument author is presuming. They are considered as implicit information in a scheme.



Capturing them in the ontology gives a key advantage for the implemented argumentation system.

Presumptions and exceptions are indirectly represented by the critical questions in Walton's schemes as discussed by Prakken, Gordon and Walton [24, 36], and they can be extracted based on them.

For example in the "Argument from Expert Opinion" scheme, there are 6 critical questions. Each one of them either represents a presumption or an exception:

1. How credible is expert *E* as an expert source? *Corresponding Presumption:* *E* is credible as an expert source.
2. Is *E* an expert in the field that the assertion *A*, is in? *Corresponding Presumption:* *E* is an expert in the field that *A* is in
3. Does *E*'s testimony imply *A*? *Corresponding Presumption:* *E*'s testimony does imply *A*.
4. Is *E* reliable? *Corresponding Exception:* *E* is biased.
5. Is *A* consistent with the testimony of other experts? *Corresponding Exception:* Other experts disagree.
6. Is *A* supported by evidence? *Corresponding Presumption:* There is evidence proving *A*.

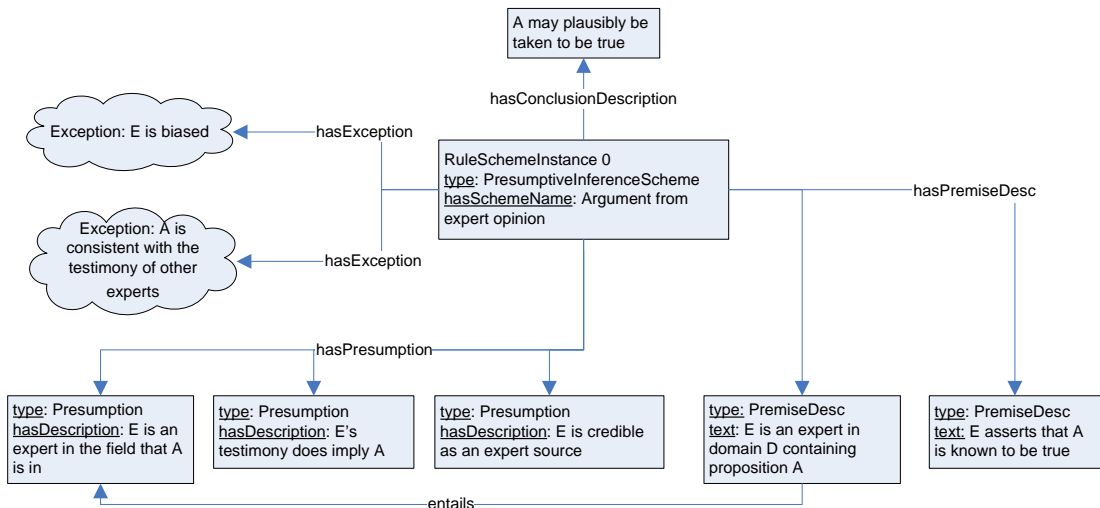
Sometimes presumptions and exceptions can be part of many schemes at the same time. That's why in the extended AIF we represented them as instances

of a class, and not as attributes, in order to allow explicit relationships with one or many schemes. For this reason the scheme parts are integrated in the ontology using the 4 classes: “PremiseDescription,” “ConclusionDescription,” “Presumption” and “Exceptions,” subclasses of class “Form,” which is an additional extended subclass of the “Node” class.

The “presumptiveInferenceScheme” class has additional attributes representing the links to presumptions through the “hasPresumption” and to exceptions through the “hasException” relationship.

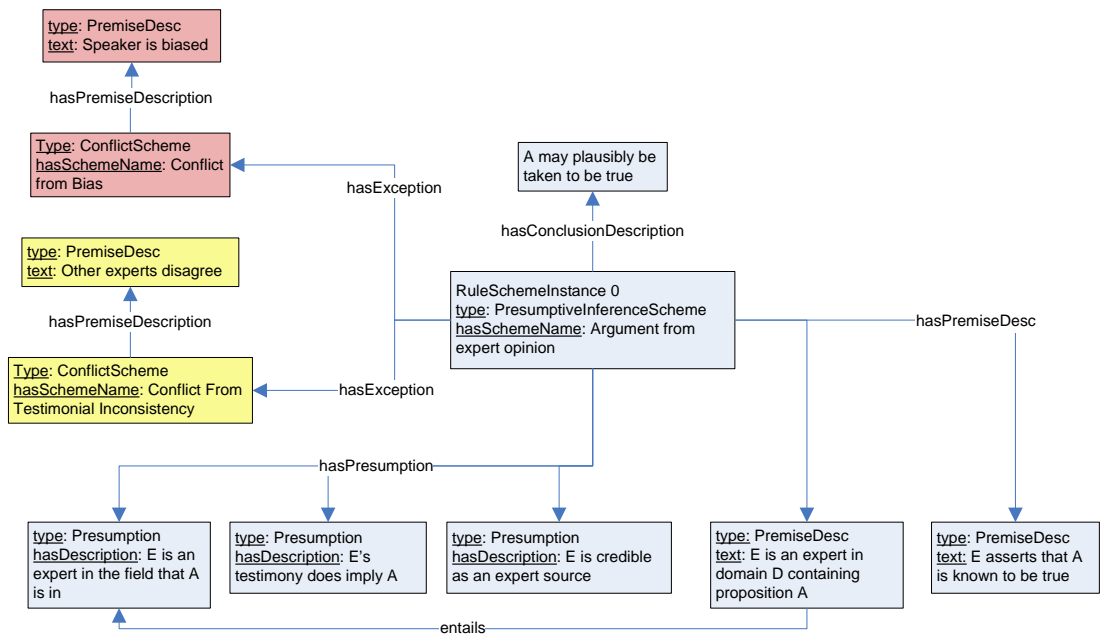
Usually some presumptions in a Walton scheme are linked to a scheme premise, forming a kind of entailment relationship. For example, the scheme premise “E is an expert in domain D containing proposition A” entails the presumption that “E is an expert in the field that A is in.” Figure 3.3 shows an extended graphical representation of the “Argument from expert opinion” scheme. It’s worth to note that with the integration of the implicit presumptions and exceptions, critical questions are indirectly represented. This way there was no need to include them in the ontology.

Exceptions are usually used to represent a certain type of conflict that might occur, if the corresponding exception happens to be true. In the AIF ontology argumentation conflicts are categorized in various types and represented in the form of conflict schemes. For example, the exception “E is biased” is of type “Conflict from bias” scheme. The way this is captured in the ontology is by representing “Conflict from bias” as an instance of the class “conflictScheme,” linked to the exception “E is biased” through the “hasPremiseDescription” rela-



**Figure 3.3:** The extended representation of the scheme example

tionship. A full representation of the “Argument from Expert Opinion” scheme is shown in Figure 3.4.



**Figure 3.4:** A graphical representation of the scheme including exceptions

### 3.3.2 Argument Representation

In the extended AIF, the “RA-Node,” a subclass of the “S-Node” class, has a purpose differing a bit from the one proposed in the AIF standard. It has been used as a rule to connect the different argument parts and to specify to which scheme the argument belongs to. In other words, an argument is an instance of the “RA-Node” class, which has a conclusion, premises and fulfilling a specific scheme. For this purpose the “I-Node” class has been extended by having the “Conclusion” and “Premise” as its subclasses.

The edges connecting the argument parts have been explicitly specified by having an “edge” relationship, under which more specified edge types are created. For example the premise role is to support the conclusion. This has been translated into the ontology by a relationship called “supports,” which is an edge going out from the premise, pointing to the corresponding “RA-Node” which is linked to the conclusion through the “hasConclusion” edge. The inverse of the “Supports” is represented by the “hasPremise” edge, which is a relation from the “RA-Node” to the premise.

The argument is bound to a specific scheme by using a “fulfillsScheme” relationship which is an edge from the “RA-Node” to the scheme used. The proposed ontology also captures more details and allows the link of the argument’s conclusion and premises to the scheme parts (premises and conclusion descriptions). This proved to be very efficient in the implementation as users are supposed to be explicitly building arguments fulfilling the scheme used, not only

by linking on the name level, but also by matching the argument's parts to the scheme parts.

In order to clarify the concepts I presented above, I will take a concrete argument example in the sports domain fulfilling the "Argument from Expert Opinion" scheme:

*Argument's conclusion:*

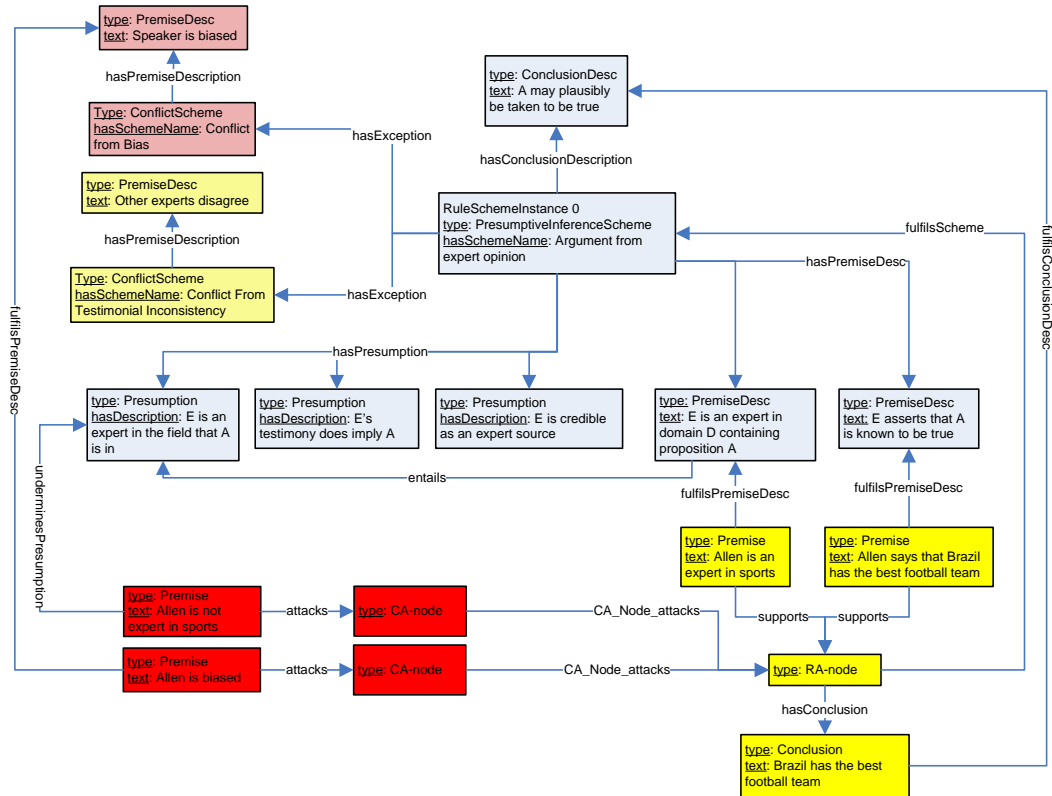
- Brazil has the best football team

*Argument's premises:*

- Allen says that Brazil has the best football team
- Allen is an expert in sports

For the purpose of showing how the implicit argument statements can be used, the above argument is attacked implicitly through one of its exceptions and presumptions. In Figure 3.5, the argument in the bottom right of the figure is fulfilling the scheme as well as the scheme parts. A premise stating that "Allen is not an expert in sport" and *undermining* the presumption that "E is an expert in the field that A is in," attacks the argument through a "CA-Node" (conflict application) going to the argument's "RA-Node." The figure also visualizes how an exception can be used to attack an argument through another "CA-Node."

Finally, after stating the above example, it is worthwhile mention another advantage of the presence of presumptions. In many cases, users sometimes enter



**Figure 3.5:** A full example with an argument & implicit attacks

a missing part of an argument, such as not explicitly specifying that football is a type of sports, which is at the end something obvious. Now with explicitly representing the presumptions in the ontology, it may be automatically inferred that football is *presumed* to be a type of sports.

The full ontology design, including extensions, is displayed in Figure 3.6.

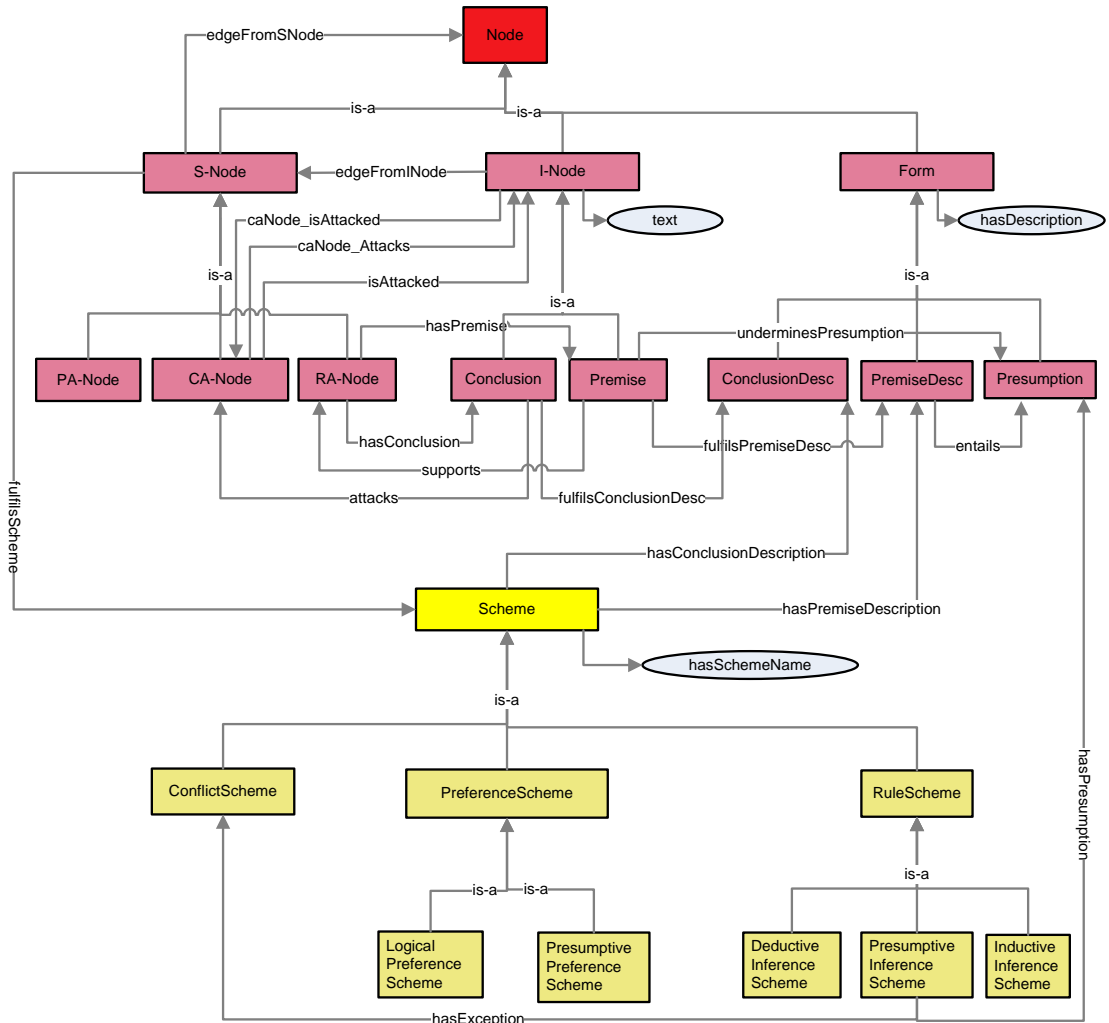


Figure 3.6: Extensions to the original AIF

## **Chapter 4**

# **AIF-RDF: Ontology Implementation in RDF**

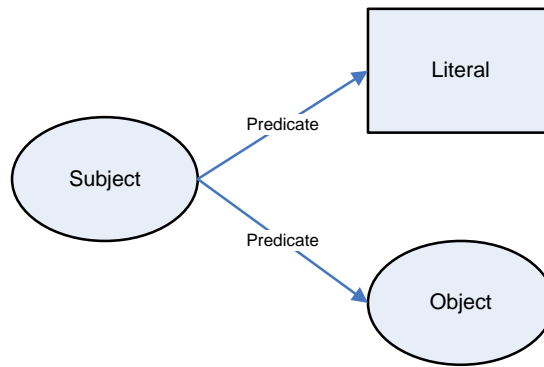
### **4.1 Introduction**

In this chapter I introduce the semantic language used in the project, and how the ontological concepts presented in the previous chapter are encoded in RDF.

### **4.2 Background: RDF & RDFS**

The system implemented in this thesis is based on the Resource Description Framework (RDF). RDF is a knowledge representation language to represent resources in an XML based format. In RDF, each resource has a Universal Resource Identifier (URI), which is its unique identification key. A resource can be considered as a physical entity like an electronic document being for example a picture or a file, or a concept like “person” or the medical domain concepts [18].





**Figure 4.1:** RDF triples graphical representation

In the RDF document, we say that resources are described by statements. RDF statements are usually represented by triples [2] as shown in Figure 4.1. The *subject* in the triple is the entity or resource that is being described in the statement. The relationship between the subject and the object is called the *predicate*. The *object* is mainly the entity that has been pointed to by the predicate in the statement. A *literal* is a well determined value such as a number or a string. RDF statements can be captured in different formats.

For example, capturing the statement “Tweety has a yellow colour” in an RDF statement can be done in the following ways:

- N3 (a simple triples representation): (Tweety, hasColour, Yellow)
- Graphical triples notation as displayed in Figure 4.2.
- RDF/XML format:

```

<rdf:Description rdf:about="Tweety">
  <rdf:hasColour>
    <rdf:Description rdf:about="Yellow"/>
  </rdf:hasColour>
</rdf:Description>
  
```



**Figure 4.2:** RDF triples graphical example

RDF also allows specifying the type of entities such as saying that Tweety is of type bird:

```

<rdf:Description rdf:about="Tweety">
  <rdf:type rdf:resource="bird"/>
</rdf:Description>
  
```

RDF Schema (RDFS) is an XML based language that comes on the top of the RDF documents. Relationships and concepts are defined in the schema. RDFS allows users to have a specific vocabulary related to their RDF statements. It is considered as a bridge between RDF and the ontology conceptual layer. Relationships can be for example a class-subclass relationship. The schema offers many notions that made building Web ontologies possible [2], such as *class*, *domain*, *range*, *label*, *subclassOf* and many others.

### 4.3 Benefits of Using RDF vs. XML

RDF was introduced around one year after the launch of the XML 1.0 standard that is the basis of a number of existing argumentation systems. There are a number of important differences between RDF and XML, which are relevant to this project:

- RDF's data model is based on the creation of statements, while XML is based on the creation of documents. The former lends itself more nat-

urally to representing argumentative statements and their interrelationships.

- RDF uses the graph concept that links different entities of statements with each others, while XML is tree-based. Argument networks follow the graph concepts where nodes are interconnected, making RDF more suitable for representing (and potentially visualising) them.
- RDF clearly presents external relationships between entities. On the other side, the XML document, only presents relationships between different elements in the document implicitly as per their hierarchy in the tree. There are no external relationships that can tie two entities directly.
- The graph concept and the subject-object relationship in RDF makes manipulating network structures (e.g. argument networks) easy. Adding new statements to an RDF statement repository can be done dynamically, without having to worry about the order of the statements inserted –and there are good software tools for doing this. On the other hand, updating an XML document requires taking care of the tree structure and the node's order before being able to update the document.
- The triple model of RDF simplifies the way semantics are represented. In the case of XML, on the other hand, there may be multiple documents that semantically represent the same structure.

## 4.4 Implementation in RDF & RDFS

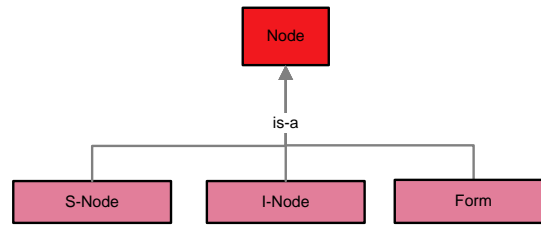
Now I am going to provide an overview of how the extended AIF ontology is translated into RDFS code. The implementation is done using Protégé [39], an ontology builder tool developed at Stanford University to help encoding ontological concepts in Semantic Web languages such as RDFS.

### 4.4.1 Nodes Instantiation

For the extended ontology presented in Figure 3.6, we represented the nodes as classes, and the edges connecting the nodes as class attributes. Class /subclass relationships are built in Protégé by using the classes' hierarchy in the graphical interface. For example, Figure 4.3 that includes the "I-Node," "S-Node" and "Form" as subclasses of the class "Node," has generated the following RDFS code:

```
<rdfs:Class rdf:about="&kb;Node" rdfs:label="Node">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;S-Node" rdfs:label="S-Node">
  <rdfs:subClassOf rdf:resource="&kb;Node"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;I-Node" rdfs:label="I-Node">
  <rdfs:subClassOf rdf:resource="&kb;Node"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Form" rdfs:label="Form">
  <rdfs:subClassOf rdf:resource="&kb;Node"/>
</rdfs:Class>
```

One can clearly see that the top class "Node" is a subclass of a predefined default class "Resource." The "subClassOf" relationship sets the link between the top class and its subclasses.



**Figure 4.3:** Class/ subclass relationship example

Passive information contained in nodes such as text or description are implemented using the string attribute type. For example the “text” attribute that holds the data of premises and conclusions is represented as literal (specific value):

```

<rdf:Description rdf:about="http://protege.stanford.edu/kb#text">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality> 1 </a:maxCardinality>
  <rdfs:label>text</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Description>
  
```

The “maxCardinality” sets the maximum number of values the text can hold. In this case only one value is allowed. The “domain” sets the domain of this attribute such as the I-Node class in the above example. The “range” specifies the allowed values for this attribute which is “literal” in this case.

#### 4.4.2 Edges Instantiation

The edges connecting the different arguments’ parts have been presented as class’ attributes. Restrictions on the edges imposed by the AIF such as an edge coming out from an “I-Node” can only be directed to an “S-Node,” has been implemented using the RDF attributes domain and range. Most of the edge properties have been created as sub-properties of the attribute “edge.” The “edge-

FromINode” relationship represents an edge coming out from an “I-Node” and has the range restricted to the “S-Node” class. The “edgeFromSNode” is another relationship representing the edges going out of an “S-Node” and has as range the complete “Node” class as in the AIF, edges are allowed to go from an “S-Node” to another “S-Node” as well as an “I-Node.” Both edge relationships are required with one or multiple values. This was successfully implemented using Protégé, which for example generated the following RDFS code for the “edgeFromSNode” property:

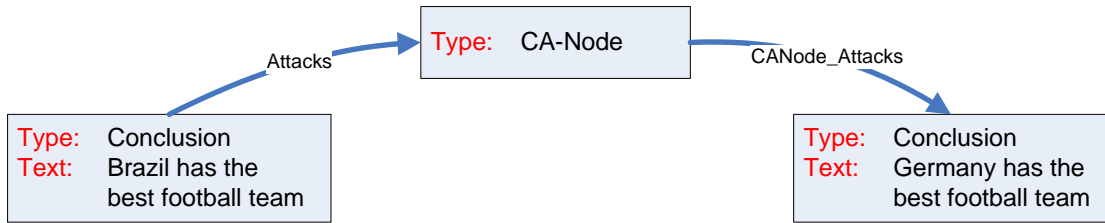
```
<rdf:Description rdf:about="edgeFromSNode">
  <rdf:type rdf:resource="Property"/>
  <a:minCardinality> 1 </a:minCardinality>
  <rdfs:label> edgeFromSNode </rdfs:label>
  <rdfs:range rdf:resource="Node"/>
  <rdfs:domain rdf:resource="S-Node"/>
  <rdfs:subPropertyOf rdf:resource="edge"/>
</rdf:Description>
```

The “supports” edge that comes out of a premise to an “RA-Node” is a sub-property of “edgeFromINode” and is translated into the following RDFS code, noting that “supports” is the inverse property of the “hasPremise” relationship or edge:

```
<rdf:Description rdf:about="http://protege.stanford.edu/kb#supports">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>supports</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromINode"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#hasPremise"/>
</rdf:Description>
```

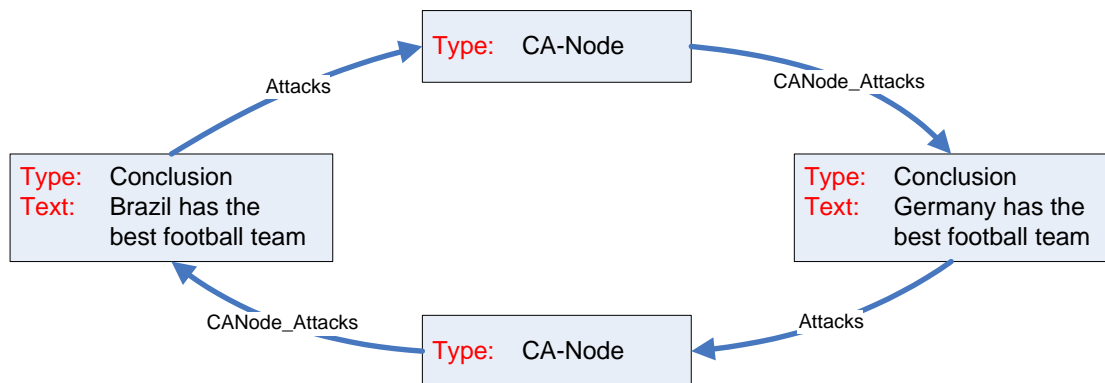
The attack relationship between two conclusions or a conclusion and a premise in AIF-RDF has been implemented following the non-symmetrical conflict approach which is represented by two pairs of edges. It has an advantage to know

from which statement the attack has been issued. An attack issued from a specific I-Node  $i_1$  to another I-Node  $i_2$  is specified by a predicate/edge *attack* going from  $i_1$  to a conflict application node (CA-Node)  $c$  and by another predicate/edge *CANode\_Attacks* from the conflict node  $c$  to the attacked node  $i_2$ . Figure 4.4 visualizes the attack process coming from the statement “Brazil has the best football team” attacking the statement “Germany has the best football team.” The inverse of the attack process has been also implemented to enhance argument querying.



**Figure 4.4:** Statements in conflict example

Note that this attack is *asymmetric*. In order to capture a symmetric attack, two CA-Nodes are needed. A symmetric attack reflects the case when two statements are mutually attacking one another. This is represented in Figure 4.5.



**Figure 4.5:** Statements in symmetric conflict example

Finally, we require that all types of edges and nodes classes are disjoint, as it is forbidden to have a node of type “I-Node” and “S-Node” at the same time.

But unfortunately disjointedness can not be expressed in RDFS, considered one of the limitations of this semantic language. This issue may be handled in the future by using the Web Ontology Language (OWL) [33] that allows the representation of such requirements.

Details of the full encoded AIF-RDF can be found in Appendix C.



## Chapter 5

# ArgDF: A Semantic Web-based Argumentation System

### 5.1 Introduction

ArgDF is a *Semantic Web*-based system built on the top of the AIF-RDF ontology proposed in the previous chapter. ArgDF enables users to create and query semantically annotated arguments on the Web using different argumentation schemes. The system also allows users to manipulate arguments by attacking or supporting parts of existing arguments, and use existing parts of an argument in the creation of new arguments. ArgDF also offers flexible features, such as the ability to create new argumentation schemes from the user interface. As such, ArgDF is an open platform not only for representing arguments, but also for building interlinked and constantly evolving argument networks. In the remainder of this chapter, I describe the system in detail.

## 5.2 ArgDF Platform Overview

Being a Web-based system, ArgDF relies on different components interacting with each other such as Sesame RDF repository, scripting, XSLT and MySQL database. Figure 5.1 visualizes the system architecture, showing the processes of inserting RDF statements and querying the RDF data repository.

The process of querying elements of existing arguments starts by sending a request to query the repository through the PHP [34] classes. The result is returned in XML format, which is then processed through extensible style sheet language transformation (XSLT) and displayed back to the user in a presentable format.

The process of creating new arguments or schemes, and the process of creating new schemes start by checking the URI number stored in a MySQL database. This URI value is extracted and applied to the inserted RDF statement in the repository. The writing process is followed by the increase of the URI counter by one in the MySQL database.

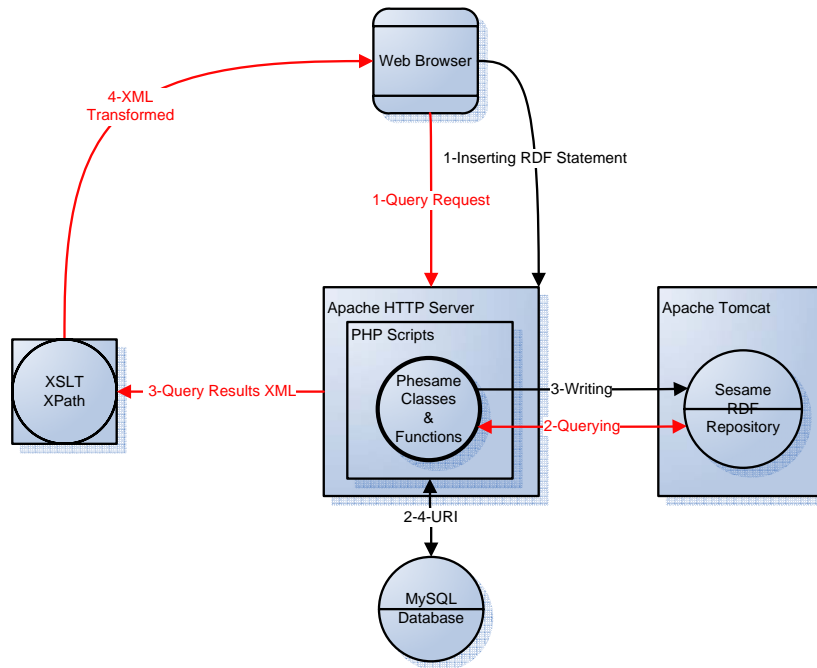
In the coming subsections I briefly introduce each component ArgDF relies on.

### 5.2.1 ArgDF Repository: Sesame RDF Server

Sesame is an RDF repository for storing RDF statements. It works on the Apache Tomcat<sup>1</sup> server as it has Java servlets containers, a necessary requirement for handling Sesame's operations. Sesame has a built in interface offering many

---

<sup>1</sup><http://tomcat.apache.org/>



**Figure 5.1:** ArgDF System Architecture

features such as:

- Uploading RDF and RDFS single statements or complete files.
- Querying the repository using different query languages such as RQL [29].
- Offering different RDF query results output formats: XML, HTML or RDF statements. The XML output is in the form of tuples and literals. The HTML output allows the user to navigate through the different RDF graph nodes in a table format. As for the RDF statements output, it exports the querying results into an RDF file.

Other functionalities are also available, such as clearing the repository, extracting statements from the repository as well as deleting them. The user has to provide a valid administrative username and password to be able to process

repository operations on Sesame. The use of Sesame is expanding in the Semantic Web development domain. It has proved to be efficient even in handling huge RDF repositories tests.

### 5.2.2 Web Scripting: PHP

PHP [34] is a server side scripting language using embedded HTML. PHP stands for “PHP: Hypertext Preprocessor.” I chose PHP as a scripting language for ArgDF Web development for many reasons:

- Previous knowledge in PHP scripting.
- PHP is open source.
- Availability of descent documentation on its official website [34].
- Presence of PHP functions created to communicate with Sesame.
- Ability to process XSLT over XML document through PHP.
- Storing the returned values of URIs extracted from the RQL queries into PHP global variables for passing the values from a page to another.
- PHP is available to download and use for free.
- Possibility to develop in a Microsoft Windows OS environment, and running on Unix platforms.
- Runs efficiently on different servers’ platforms such as Apache and IIS.

- PHP runs smoothly with MySQL database server which has also been used in the development of ArgDF.

### 5.2.3 PHP - Sesame Communication: Phesame

For developing the ArgDF Web interface, communicating with Sesame through PHP pages was of a primary importance. Phesame<sup>2</sup> is a tool built for this specific reason. It is a PHP class containing a series of functions for communicating with Sesame. Phesame has been initially developed for the HyperJournal Project,<sup>3</sup> a rapid application development environment, with features that proved to fulfil the needed functionalities through the ArgDF interface:

- Connecting and logging into the RDF repository through HTTP by using the function *login()*.
- Selecting the Sesame repository to use through the function *setSelectedRepository()*.
- Specifying the upload format of the RDF statements such as the RDFXML format used in ArgDF. The function is *setUploadFormat()*.
- Choosing the format of the query results. In ArgDF I use the XML format. This can be set by using the function *setResultFormat()*.
- Selecting the query language to use, such as RQL, is specified by using the function: *setQueryLanguage()*.

---

<sup>2</sup><http://www.hjournal.org/phesame>

<sup>3</sup><http://www.hjournal.org>

- Passing RQL queries and binding them to a PHP variable. The function is: *simpleQuery()*.
- Uploading RDF statements by using the *uploadData()* function.

The source code of the Phesame class has been made available for usage, and has documentation notes and directions embedded with the source code. Phesame proved to be a kind of bridge between the PHP scripting language, and the Sesame RDF engine.

#### 5.2.4 Querying the ArgDF Repository: RQL

Queries in ArgDF are written using the RDF Query Language (RQL), which is supported by Sesame. RQL queries are similar to databased queries and take the form “Select-From-Where.” They operate based on matching the query constraints with the values of the RDF graph in the repository. For example, querying the ArgDF repository to extract the name of the schemes can be done through the following RQL query:

```
select Scheme, PresumptiveInferenceScheme-hasSchemeName
from Scheme : kb:PresumptiveInferenceScheme kb:hasSchemeName
PresumptiveInferenceScheme-hasSchemeName
using namespace
rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns# ,
rdfs = http://www.w3.org/2000/01/rdf-schema# ,
kb = http://protege.stanford.edu/kb#
```

Noting that the “Scheme” and “PresumptiveInferenceScheme-hasSchemeName” are only variable names and thus can be simply replaced by any variable name.

I used such annotations to make the code more readable.

RQL has many features and there's a tutorial that can be found at the Sesame's website [44]. Queries are passed to the Sesame server using Phesame, and the results are returned in XML format.

### 5.2.5 Rendering and Visualization: XSLT & XPath

XSLT is used to transform an XML document into a readable and presentable format. XPath is the XML path language that helps in traversing XML documents to pick the needed information embraced between the tags. ArgDF scripts internally return the results of the RDF queries into an XML document. This XML document is subject to the application of XSLT and XPath to return for the users the results on the web page, integrated into PHP. For example, if we have the result of a query stored in a variable *Result*, here is the list of steps to render the output for the end user:

1. Creation of the XSLT file containing the HTML formatting as well as XPath expression defined to pick the needed results.
2. Defining a new PHP XSLT processor using the function *xsltProcessor()*.
3. Importing the XSLT file using the PHP function *importStyleSheet()*.
4. Transforming the result variable to a readable XML document by PHP scripts using the function *transformToXML()* and binding it to the created XSLT processor.

5. Finally printing the XSLT processor using the *print()* function of PHP, and the rendered XML result is displayed in the designed format.

XSLT and XPath have also been used to enable *hyperlinking* the options in the interface, and picking up the URI values to be used globally in the system.

### 5.2.6 URI Automatic Generation: MySQL Database

Every instance in ArgDF is required to have a unique identifier value. So for example, if the URI of an instance has a value of *ArgOnt\_Instance\_X*, the next URI value is *ArgOnt\_Instance\_X+1*. I tackled this issue by relying on the MySQL database server, in which the latest number of the instance is stored. MySQL is an open source database system that is in continuous development. Choosing MySQL database is due to my familiarity with MySQL development, and to MySQL's ability to run on different platforms and its easy and efficient management through PHP. MySQL currently forms the backbone of many heavily used Web-systems.

Whenever a new statement is to be uploaded to the repository, the value of the counter in the MySQL table is extracted, and then concatenated to the prefix *ArgOnt\_Instance\_*, which is used as the identifier of the new resource created, and finally increment and store the value of the counter by one.

In brief, here's the algorithm behind the URI generation:

```

If      New RDF resource upload
Then   connect to MySQL database
        And extract the counter value = CountX

```



And Concatenate the Counter value to the prefix ArgOnt\_Instance\_  
 And new RDF resource URI = ArgOnt\_Instance\_CountX  
 Update MySQL database with the new CountX = CountX + 1

### 5.3 Creating New Arguments

In ArgDF a user can create new arguments based on argumentation schemes. At the beginning, the system presents the list of schemes, and the user is allowed to choose the scheme to which the argument belongs. Details of the argumentation scheme selected by the user are then retrieved from the repository, and the way the argument should be built is displayed to users through sample conclusions and premises. Then users will build the argument by pressing on a button next to the corresponding sample, being a conclusion or a premise.

I now explain the background processes performed by the system while creating a new argument. The creation of a new argument involves many processes running in parallel, ranging from the upload of RDF statements, to querying the repository and displaying information to the end user. Figure 5.2 visualizes the steps to give a clearer idea about the complete cycle.

#### *Choosing the Argumentation Scheme*

Whenever there is a screen in ArgDF in which there is a list of options from which the user can choose from, 2 queries are applied to the repository: one to extract the text and details of the resources, and another query to extract the labels and URIs.

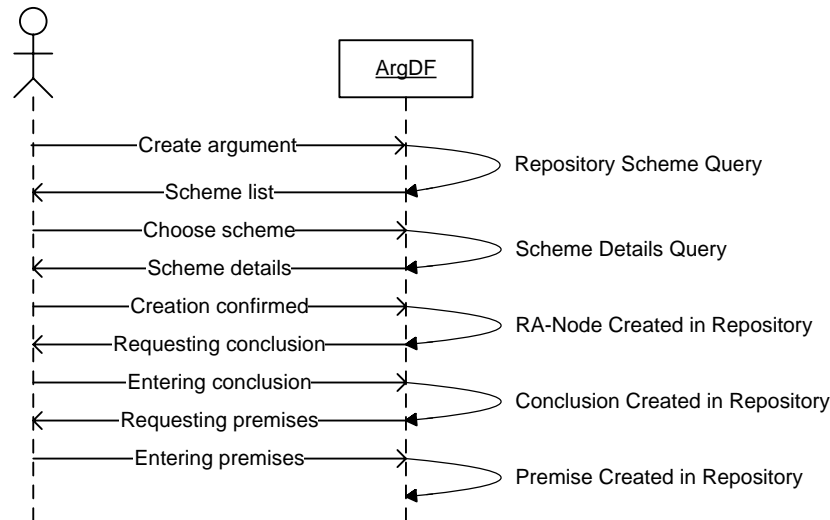


Figure 5.2: New Argument Creation Cycle

This query is passed to the Sesame server using Phesame, and the returned result, in XML format, is then processed by 2 XSLT files including XPath formulas, which prepare them to be displayed properly. The first XSLT manipulates the hyperlink to enable certain subsequent argument navigation tasks by the user. The second XSLT displays the name of the schemes in a table.

For example, the result of the scheme name RQL query presented in the previous section can be passed in XSLT to produce the HTML output shown in Figure 5.3.

Argumentation Scheme	Click to Create Argument
Argument from Example	<a href="#">Create</a>
Argument from Verbal Classification	<a href="#">Create</a>
Argument from expert opinion	<a href="#">Create</a>
Argument from Sign	<a href="#">Create</a>

Figure 5.3: XSLT Table Output of Argument Schemes

### *Displaying the Argumentation Scheme Details*

After choosing the scheme, the URI of the instance scheme is to be passed to the next page, and then again 2 queries are performed: one extracts the conclusion's

text of the scheme instance matching the URI of the one chosen by the user, and the other extracts its premises' text. The scheme details are then rendered using 2 XSLT files applied during all the argument creation process.

#### *Creation of the RA-Node*

The first thing ArgDF uploads to the repository during the creation of a new argument is the RA-Node: the scheme node that holds the various argument pieces together. This process happens automatically before creating the conclusion and the premises. A unique URI, extracted from the MySQL database, is applied to the RA-Node instance, and is linked to the URI of the previously chosen scheme using the *fulfilsScheme* relationship. This links the newly created argument to the scheme chosen by the user. The RDF code uploaded to Sesame for the creation of the RA-Node looks as the following:

```
<rdf:RDF>
  <kb:RA-Node rdf:about="&kb;MySQL_URI_Generation"
    rdfs:label="MySQL_URI_Generation">
    <kb:fulfilsScheme rdf:resource="&kb;Selected_Scheme"/>
  </kb:RA-Node>
</rdf:RDF>
```

#### *Creation of the Conclusion and Premises*

After uploading the RA-Node RDF statement, the user is redirected to enter the conclusion and the premises of the argument. The system guides the user during this process based on the selected scheme structure. Each argument conclusion and premise entered by the user must fulfil the conclusion and premise description of the scheme as shown in Figure 5.4. Also the conclusion and premises instances get a unique URI, and have the *edgeFromINode* property

linked to the previously created RA-Node, and the RA-Node is updated to include the *hasConclusion* and *hasPremise* attributes. In addition to the *edgeFromINode* property, the conclusion is assigned a *fulfilsConclusionDesc* linked to the URI of the scheme conclusion description, and the premise has *supports* linked to the RA-Node and *fulfilsPremiseDesc* linked to the scheme premise description.

Scheme Conclusion Description	-	Current Argument's Conclusion
A may plausibly be taken to be true		Brazil has the best football team

Scheme Premise Description	-	Current Argument's Premises
E asserts that A is known to be true	Create Premise	Allen says that brazil has the best football team
E is an expert in domain D containing proposition A	Create Premise	Allen is an expert in sports including football

Figure 5.4: Argument Creation in ArgDF

## 5.4 Argument Extension

In this section, I describe the process of *extending* an existing argument, either by supporting it, attacking it, or using an existing premise to be the premise of a newly created argument.

### 5.4.1 Support/Attack of Existing Expressions

ArgDF allows users to support/attack existing expressions. The list of existing expressions in the repository can be displayed as shown in Figure 5.5. The user can choose the statement s/he wants to support or attack. The support and attack of expressions can be done on premises as well as on conclusions. When a user chooses to support an existing premise, this premise is both a premise in one argument, and a conclusion in another one. Thus, the system allows for the

*chaining* of arguments.

To *support* existing expressions, the user can create supporting premises after choosing a scheme to be used in the support. If the user chooses to *attack* a statement, on the other hand, s/he is redirected to choose an appropriate scheme for attack, and create a whole new argument with a conclusion (statement) attacking the existing one, and backed by new premises.

Brazil has the best football team	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>
Allen says that brazil has the best football team	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>
Allen is an expert in sports including football	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>

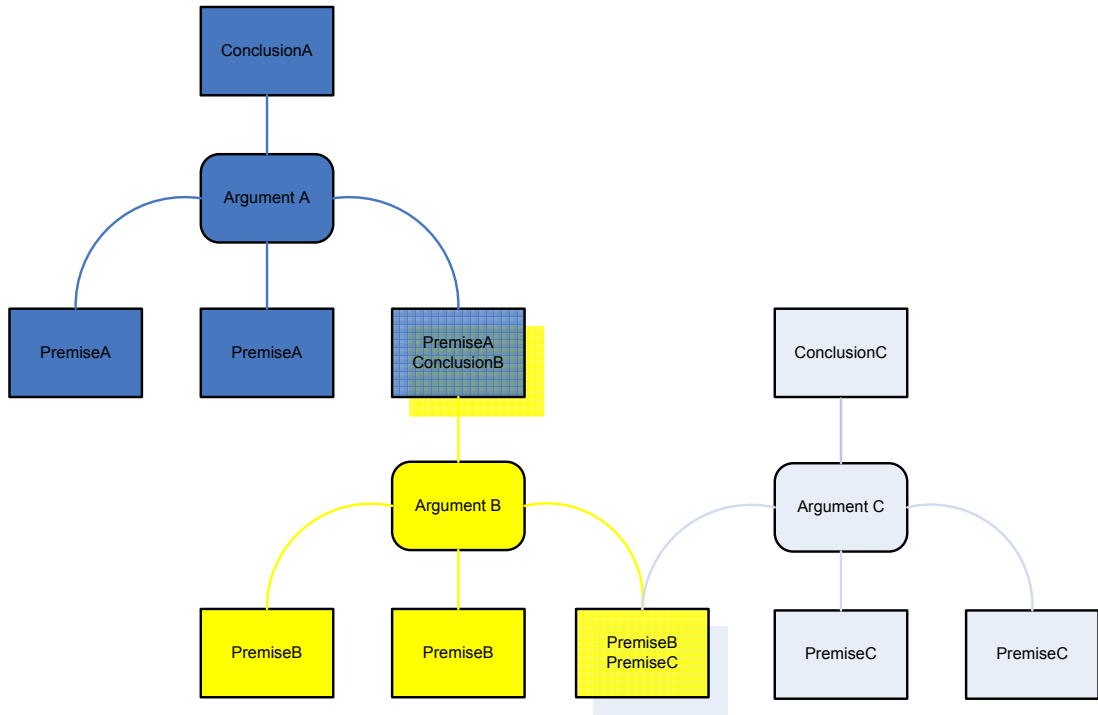
Figure 5.5: Listing Existing Claims

### 5.4.2 Linking Existing Premises to a New Argument

While creating premises supporting a given conclusion through a new argument, the user can *reuse* existing premises from the system. This functionality can be useful, for example, in Web-based applications that can allow users to use existing Web content (e.g. a news article, a legal document) to support new or existing claims. This way a premise can be used for 2 or more different arguments. The resulting network structure is exemplified in Figure 5.6, in which a single claim constitutes a premise for 2 arguments.

### 5.4.3 Attacking Arguments through Implicit Assumptions

As mentioned in Chapter 3, the implementation of the premises with presumptions and exceptions made the *presuming* mechanism programmatically possi-



**Figure 5.6:** Interlinked Arguments Network

ble. For example if a user creating a new argument skips the creation of a certain premise fulfilling a certain presumption, the implicit statements (extracted from the corresponding scheme) aid in delivering the complete argument structure.

ArgDF allows the user to *inspect* existing claims by displaying all the arguments in which this claim is involved: being a conclusion or a premise supporting a conclusion. After opening an argument, exceptions and presumptions can be visualized leading the way for an implicit attack of the argument either through an exception (as in Figure 5.7), or through undermining a presumption (as in Figure 5.8).



Figure 5.7: Implicit Attack Through an Exception in ArgDF



Figure 5.8: Implicit Attack Through Undermining a Presumption in ArgDF

## 5.5 Advanced Argument Search

Querying the Semantic Web presents powerful data extraction techniques. An advanced search tool is implemented in ArgDF allowing the users to extract arguments based on specific questions and following a specific argument scheme such as: What are the premises supporting the statement “Brazil has the best football team” and following the “Argument from Expert Opinion” scheme? Figure 5.9 shows how this question can be formalized, and the answer to the advanced search is visualized in figure 5.10.

*Advanced ArgDF Search*

Step 1: Enter a string found in

a- Premise:

(Leave blank for all premises)

☒ For  
☐ Against

b- Conclusion:

(Leave blank for all conclusions)

Figure 5.9: ArgDF Advanced Search Tool

*Advanced ArgDF Search*

Search Results Matching

*Supported* conclusions containing *Brazil has the best football team* string, and supported by premises containing *ALL (\*)* string.

Allen says that Brazil has the best football team	--Supporting-->	Brazil has the best football team	<a href="#">Open Argument</a>
Allen is an expert is sports	--Supporting-->	Brazil has the best football team	<a href="#">Open Argument</a>

Figure 5.10: ArgDF Advanced Search Result



## 5.6 Creation of New Schemes

The user can also create new argumentation schemes through the interface of ArgDF without having to modify the ontology itself.<sup>4</sup> Figure 5.11 shows a screen shot of the creation “Argument from Example” scheme in ArgDF.

The screenshot shows a web browser window with the address `http://localhost/argont/frame_index.php`. The page title is "ArgDF: Arguing using RDF". On the left is a blue sidebar with yellow buttons: "Home", "New Argument", "Claims List", and "Argument Schemes". The main content area is titled "Scheme Sample" and contains several sections:

- Scheme Conclusion Description**: A text box containing "Generally, if x has property F then x also has property G".
- Scheme Premise Description**: A table with two columns. The first column contains two lines: "In this case, the individual a has property F and also property G" and "a is typical of things that have F and may or may not have G". The second column contains two links: "Entails Presumption" and "Entails Presumption".
- Scheme Presumptions**: Two text boxes containing "a has F and G" and "a an instance of the generalisation".
- Scheme Exceptions**: Three text boxes containing "a is not a case that the generalisation ranges over", "Generalisation is not strongly, widely applicable", and "Circumstance impairing a\'s generalisability".

Below these sections, there is a text input field labeled "Enter the scheme premises:" and a button labeled "Create Premise & Add Another". At the bottom, there are two links: "Create Generic Scheme Presumptions" and "Create Scheme Exceptions".

**Figure 5.11:** Creating a new Scheme in ArgDF Example

Coding and programming details for building ArgDF are explored in depth in appendix B.

<sup>4</sup>Recall that actual schemes are instances of the “Scheme” class.

## Chapter 6

### Conclusion and Open Issues

The work in this thesis comes out with a semantically rich system for authoring and visualizing arguments. This system, ArgDF, meets the requirements for an open argument representation on the Web that I presented in Chapter 2:

- R1 *Supporting the storing, creating, updating and querying of information structures.* ArgDF is a Web-based system that supports the storage, creation, update and querying of argument data structures based on Walton's argument schemes. Though the prototype implementation employs a centralized server, the model is able to support large-scale distribution.
- R2 *Having Web-accessibility features and an open data repository.* The arguments are uploaded on a Sesame RDF repository which can be access and queried openly through the Web and using a variety of RDF standard query languages.
- R3 *Relying on a language based on open standards, thus enabling collaborative development of new tools and features.* Arguments in the ArgDF system are ex-

pressed in the RDF Semantic Web annotation language, and based on the RDF Schema ontology language, which are an open standard endorsed by the W3C. A variety of software development tools can be used for taking advantage of this.

R4 *The semantics must employ a unified argumentation ontology.* The AIF-RDF ontology captures the main concepts in the AIF ontology [13], which is the best current model for such an ontology.

R5 *Supporting the representation, annotation and creation of arguments using a variety of argumentation schemes.* AIF-RDF preserves the AIF's strong emphasis on scheme-based reasoning patterns, conflict patterns and preference patterns, and is designed specifically to allow extended and modified scheme sets to be accommodated.

ArgDF combines the features of a highly scalable argumentation system, and at the same time a structured argument representation using different argumentation patterns. The latter has tackled the limitation of many argumentation tools such as Parmenides where only one argumentation theory is applied, and TruthMapping where arguments do not follow any specific argumentation pattern. On the contrary to the database limitations, for example used in Parmenides, TruthMapping and others, the ArgDF's ontology can be easily extended by manipulating the RDF statements stored in the open Sesame RDF repository.

The end of this thesis left many open issues for ArgDF's future enhancements such as utilizing user preferences in the presentation, and possibly evaluation of arguments. This helps in having better decision outcomes in a collaborating team. Another enhancement that can be worked on is making use of description logic reasoning for inferring new things about the argument structure such as the equivalence of two arguments etc. Investigating the issue of ArgDF serving as the platform for multi-agent systems (MAS) communication can also be of an added value to the project. Targeting a wider end-user audience: users interested in structured arguments for academic and analysis purposes, and blogging users. The first one can be tackled by populating ArgDF with Araucaria's set of schemes, making the structuring ways more diverse and serving the academic audience in a better way. As for the latter, related to blogging users, may be supported by offering the feature to have the normal blogs text box, and having the text parsed by automated text processing, and extracted arguments are inserted in ArgDF in a structured way as per the AIF ontology requirements. Adding more querying features in ArgDF could offer users various ways of arguments' extraction. Another possible area of work is the incorporation of large-scale argument evaluation on corpora of arguments on the Web, which is now doable with the ability to store arguments over many repositories. Finally plugging ArgDF to RDF visualization tools may improve the navigation of complex argument structures.

The fusion of the AIF ontology, generic argumentation schemes based on Walton's theory, and Semantic Web gave the birth of ArgDF, and opened the way

## CHAPTER 6: CONCLUSION AND OPEN ISSUES

of new argument authoring and representation techniques.

## Appendix A

### ArgDF User Manual

In this appendix I present a reference manual for using ArgDF. The design of the pages is still in its primitive stage. The home page of ArgDF is visualized in Figure A.1. The main menu on the left includes the option for new arguments creation, listing existing claims, manipulating argumentation schemes and an advanced search utility. In order to have a concrete hands-on the features, I will be using in this manual the football sports argument presented earlier.



Figure A.1: ArgDF Home Page

## A.1 Creating a New Argument

When the user chooses to create a new argument, s/he is redirected to choose the scheme the argument is fulfilling. Figure A.2 includes only one scheme. By the end of the appendix I show how to populate the repository with additional schemes.

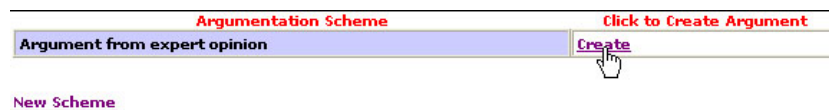


Figure A.2: Argument Creation Scheme Choice

After choosing the scheme by pressing the “create button,” the user is notified that s/he is in process of creating a new argument as in Figure A.3, with the details of the scheme selected.

**Conclusion Description:**

- A may plausibly be taken to be true

**Premises Description:**

- E asserts that A is known to be true
- E is an expert in domain D containing proposition A

*You are in the process of creating a new argument*



Figure A.3: New Argument Notification

After the screen in Figure A.3, until the end of the process, the details of the scheme are always displayed on the screen. In order to create the argument’s conclusion and premises, the user should press next to the corresponding scheme part to be fulfilled as shown in Figure A.4.

Figure A.5 displays how a user enters the conclusion, and at the same time s/he

Scheme Conclusion Description	
A may plausibly be taken to be true	<a href="#">Create Conclusion</a>

Scheme Premise Description	
E asserts that A is known to be true	<a href="#">Create Premise</a>
E is an expert in domain D containing proposition A	<a href="#">Create Premise</a>

Figure A.4: Argument Parts Creation

is able to see which statement it's fulfilling.

Enter your Statement (Conclusion):	<input type="text" value="Brazil has the best football team"/>
Fulfilling the Scheme Conclusion Description:	A may plausibly be taken to be true

[Create](#)

Figure A.5: Entering the Argument's Conclusion

Similarly to Figures A.4 and A.5 the premises are created to finally come up with a full argument as in Figure A.6.

Scheme Conclusion Description	-	Current Argument's Conclusion
A may plausibly be taken to be true		Brazil has the best football team

Scheme Premise Description	-	Current Argument's Premises
E asserts that A is known to be true	<a href="#">Create Premise</a>	Allen says that Brazil has the best football team
E is an expert in domain D containing proposition A	<a href="#">Create Premise</a>	Allen is an expert in sports including football

Figure A.6: Argument Creation Ending

Claims list (conclusions and premises) can be listed in ArgDF, allowing users to support, attack and inspect them. Figure A.7 visualizes this feature.

## A.2 Explicitly Attacking an Existing Claim

After listing the claims, users can explicitly attack existing claims. This is done by pressing the attack button next to the claim to be attacked as in Figure A.7. Then users are redirected to choose the scheme of the attacking argument. In



Brazil has the best football team	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>
Allen says that Brazil has the best football team	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>
Allen is an expert in sports including football	<a href="#">Support</a>	<a href="#">Attack</a>	<a href="#">Inspect</a>

Figure A.7: Claims List in ArgDF

this example the *argument from expert opinion* scheme is also used, and finally the screen in Figure A.8 shows up, and the conclusion and premises of the attacking argument are created as in the previous section.

**Attacking:**

Brazil has the best football team

**Scheme Conclusion Description**

A may plausibly be taken to be true [Create Conclusion](#)

**Scheme Premise Description**

E asserts that A is known to be true [Create Premise](#)

E is an expert in domain D containing proposition A [Create Premise](#)

Figure A.8: Attacking Existing Claim

### A.3 Supporting an Existing Claim

When a user chooses to support an existing claim, the selected claim appears on the top of the screen (Figure A.9) and premises are directly entered.

**Current Argument's Conclusion**

Brazil has the best football team

**Scheme Premise Description**

E asserts that A is known to be true [Create Premise](#)

E is an expert in domain D containing proposition A [Create Premise](#)

Figure A.9: Supporting an Existing Claim

## A.4 Inspecting an Existing Claim and Implicit Attacks

The claim inspection visualizes to the user in which argument this claim is appearing, being a conclusion or a premise. For example inspecting *Allen is an expert in sports* shows that it supports the conclusion *Brazil has the best football team*, and users can open the argument (Figure A.10).

The selected claim supports the following conclusions:



Figure A.10: Opening an Existing Argument

The opened argument is displayed as well as the scheme it fulfils (Figure A.11), with the possibility to show the exceptions and presumptions.



Figure A.11: Existing Argument Details

Exceptions expanded in Figure A.12 can be used for attack and get a notification as in Figure A.13.

In the screen of Figure A.14 the user enters the exception.

Argumentation Scheme

Argument from expert opinion

Current Argument's Conclusion

Brazil has the best football team

Current Argument's Premises

Allen says that Brazil has the best football team

Allen is an expert in sports including football

Scheme Exceptions

Other experts disagree

Use to Attack

Speaker is biased

Use to Attack

Current Argument Exceptions

[Hide implicit details](#)

Figure A.12: Existing Argument Exceptions Details

You are in the process of internally attacking the argument through an exception

Continue

Figure A.13: Attacking Through Exception Confirmation

Enter your premise:

Allen is biased

Fulfilling the Scheme Exception:

Speaker is biased

Create

Figure A.14: Entering the Exception

Figure A.15 shows the updated argument with its implicit attack through exception.

<b>Argumentation Scheme</b>	
Argument from expert opinion	
<b>Current Argument's Conclusion</b>	
Brazil has the best football team	
<b>Current Argument's Premises</b>	
Allen says that Brazil has the best football team	
Allen is an expert in sports including football	
<b>Scheme Exceptions</b>	
Other experts disagree	<a href="#">Use to Attack</a>
Speaker is biased	<a href="#">Use to Attack</a>
<b>Current Argument Exceptions</b>	
Allen is biased	
<a href="#">Hide implicit details</a>	

Figure A.15: Argument Exceptions Updated

Undermining an argument's presumption process, which is also an implicit attack, is visualized in Figures A.16, A.17 and A.18.

<b>Argumentation Scheme</b>	
Argument from expert opinion	
<b>Current Argument's Conclusion</b>	
Brazil has the best football team	
<b>Current Argument's Premises</b>	
Allen says that Brazil has the best football team	
Allen is an expert in sports including football	
<b>Scheme Presumptions</b>	
E's testimony does imply A	<a href="#">Undermine</a>
E is an expert in the field that A is in	<a href="#">Undermine</a>
E is credible as an expert source	<a href="#">Undermine</a>
<b>Current Argument Undermining Presumptions</b>	
<a href="#">Hide implicit details</a>	

Figure A.16: Implicit Argument Attack Through Presumptions - 1

Figure A.19 displays the argument with its undermining presumptions.

*You are in the process of internally attacking the argument by undermining its presumptions*



Figure A.17: Implicit Argument Attack Through Presumptions - 2

Enter your premise:  
Undermining the Scheme Presumption:

Allen is not expert in sports

E is an expert in the field that A is in

Create

Figure A.18: Entering the Undermining Presumption

Argumentation Scheme

Argument from expert opinion

Current Argument's Conclusion

Brazil has the best football team

Current Argument's Premises

Allen says that Brazil has the best football team

Allen is an expert in sports including football

Scheme Presumptions

E's testimony does imply A

Undermine

E is an expert in the field that A is in

Undermine

E is credible as an expert source

Undermine

Current Argument Undermining Presumptions

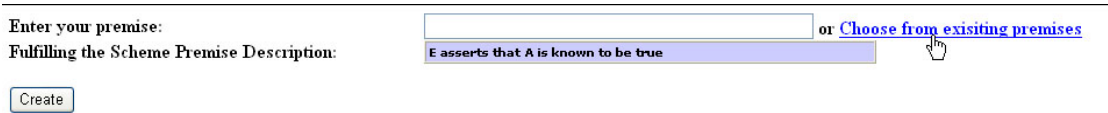
Allen is not expert in sports

[Hide implicit details](#)

Figure A.19: Undermining Presumption Displayed

## A.5 Using Existing Premises in New Arguments

Users can, while creating new arguments, pick an existing premise from the repository and link it to their new argument as shown in Figure A.20. The list displays only the available premises (Figure A.21).



Enter your premise:  or [Choose from existing premises](#)

Fulfilling the Scheme Premise Description: 

E asserts that A is known to be true

Figure A.20: Using an Existing Premise

Allen says that Brazil has the best football team	<a href="#">Use</a>
Allen is an expert in sports including football	<a href="#">Use</a>
Bob says that Germany has the best football team	<a href="#">Use</a>
Bob is an expert in sports including football	<a href="#">Use</a>
Allen is biased	<a href="#">Use</a>
Allen is not expert in sports	<a href="#">Use</a>

Figure A.21: List of Existing Premises

## A.6 Argumentation Schemes Manipulation

Argument schemes details can be checked by pressing on the *Argument Schemes* button in the main menu, then pressing on the *details* button as in Figure A.22. Here I display the creation of the *Argument from Example Walton* scheme.



**Presumptive Inference Schemes**

Argument from expert opinion	<a href="#">Details</a>
<a href="#">New Presumptive Inference Scheme</a>	

**Conflict Schemes**

Conflict from bias
Conflict from testimonial inconsistency

Figure A.22: Argumentation Scheme Details

To create a new presumptive inference scheme, the link *New Presumptive Inference Scheme* should be pressed and users are redirected to enter the scheme name (Figure A.23).

*You are in the process of creating a new argumentation scheme*

Scheme name:



**Figure A.23:** Entering the New Scheme Name

Then the user is automatically redirected to enter the conclusion description as in Figure A.24.

*Enter the scheme conclusion sample:*



**Figure A.24:** Entering the New Scheme Conclusion

Finally the screen of Figure A.25 is displayed, enabling the creation of the new scheme premises, entailed or generic presumptions, and exceptions.

Clicking on the *Entails Presumption* next to the corresponding premise in Figure A.25, takes the user to enter the specific presumption as in Figure A.26.

The *Create Generic Scheme Presumptions* link creates presumptions that are not entailed by any premise. Exceptions are created by clicking on the *Create Scheme Exceptions* link and the user is redirected to create the conflict scheme by which the new exception abides (Figure A.27), and its premises description (exceptions) as in Figure A.28.

Finally the way the newly created *Argument from Example* scheme looks is visu-

Scheme Sample

Scheme Conclusion Description

Generally, if x has property F then x also has property G

Scheme Premise Description

In this case, the individual a has property F and also property G

a is typical of things that have F and may or may not have G

Entails Presumption

Entails Presumption

Scheme Presumptions

Scheme Exceptions

Enter the scheme premises:

Create Premise & Add Another

Create Generic Scheme Presumptions

Create Scheme Exceptions

Figure A.25: Entering the New Scheme Premises

Enter your presumption:

Is it actually the case that a has F and G

Entailed by the Scheme Premise Description:

In this case, the individual a has property F and also property G

Create

Figure A.26: Entering the New Scheme Entailed Presumption

alized in Figure A.29.

You are in the process of creating a new conflict scheme

Conflict scheme name: exception to generalisation

Create

Figure A.27: Entering New Conflict Scheme Name



*Enter the scheme conflict premise:*

Create

[Go back to the presumptive scheme creation](#)

Figure A.28: Entering Conflict Scheme Corresponding Premise

Scheme Sample

Scheme Conclusion Description

Generally, if x has property F then x also has property G

Scheme Premise Description

In this case, the individual a has property F and also property G

[Entails Presumption](#)

a is typical of things that have F and may or may not have G

[Entails Presumption](#)

Scheme Presumptions

Is it actually the case that a has F and G

a an instance of the generalisation

Scheme Exceptions

a is not a case that the generalisation ranges over

Generalisation is not strongly, widely applicable

Circumstance impairing a\'s generalisability

*Enter the scheme premises:*

Create Premise & Add Another

[Create Generic Scheme Presumptions](#)

[Create Scheme Exceptions](#)

Figure A.29: New Scheme Creation Final Screen

## **Appendix B**

### **Building ArgDF in Depth**

In this appendix, I present technical details about ArgDF implementation. Details of design and coding issues are explored, ranging from the data entry level, to the querying and output results level.

#### **B.1 Phesame for PHP and Sesame communication**

The purpose of using Phesame in ArgDF was to have a set of functions facilitating the interaction between PHP and Sesame. Part of the required Phesame functions have been used and proved to be a benefit to the ArgDF project.

##### **B.1.1 Opening the Sesame Connection**

In order to process any Sesame command, like uploading, creating or querying RDF statements, a connection should be opened. Opening a new connection is done using the following function, accepting the value of the path to the Sesame servlet as a variable:

```
$phesame = new Phesame($sesame_url)
```

where *\$sesame\_url* is the PHP variable and should contain the path to the Sesame servlet repository on the Apache Tomcat Server. If the Tomcat server is installed locally, it should be for example:

```
$sesame_url = "localhost:8080/sesame/servlets".
```

After opening the connection, a valid user name and password having administrative privileges to manipulate the repository should be provided. The *login* function accepting 2 variables will do the job:

```
$login = "testuser"  
$passwd = "opensesame"  
$phesame->login($login,$passwd)
```

### B.1.2 Setting the RDF Repository & Input/Output Parameters

After opening the connection with Sesame, the repository to work on should be specified through a function called *setSelectedRepository* that accepts one value being the repository name. Our code as an example:

```
$phesame->setSelectedRepository('mem-rdfs-db')
```

Where 'mem-rdfs-db' is the repository that ArgDF uses.

Phesame allows setting the format of the uploading RDF statements format. In ArgDF, the upload is done using the RDF/XML format. This is set in the function *setUploadFormat*, accepting one parameter being format value:

```
$phesame->setUploadFormat('rdfxml')
```

Also the RDF queries' results can be set through the function *setResultFormat* that accepts the value of the format. In ArgDF the output used is XML, as the result has been later on processed in XSLT to display results to the final user in the required design:

```
$phesame->setResultFormat('xml')
```

The querying language used in the project is RQL. This can also be set through Phesame by using the *setQueryLanguage* function:

```
$phesame->setQueryLanguage('RQL')
```

In the following sections, I present more Phesame functions on a case by case basis.

## B.2 Writing in the RDF Repository & Uploading RA-Node Example

After opening the connection with Sesame and setting the upload format, RDF statements to be uploaded through the PHP interface are stored inside a PHP variable called for example *\$data*. Then these RDF statements are uploaded to the repository using the *uploadData* function. As an example, let's say that we need to create a new RA-Node having a URI as ArgOnt\_Instance\_X. As our uploading selected language is RDF/XML, our uploading data should follow the RDF/XML syntax. The variable *\$data* includes all the RDF details representing the code of the RA-Node to be uploaded. In order to do that, the below *\$data* PHP variable is bound to the RDF statement in the following way:

```

$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:kb="&kb;" xmlns:rdfs="&rdfs;">
  <kb:RA-Node rdf:about="&kb;ArgOnt_Instance_X" rdfs:label=
    \"ArgOnt_Instance_X\">
  </kb:RA-Node>
</rdf:RDF>

```

At the end, the variable *\$data* is uploaded in the following manner:

```

$phesame->uploadData($data,$verifydata=true,$showwarnings=false,
  $shownotices=false)

```

Where the parameter *\$verifydata* is set to true in order to check the data before uploading.

In the Sesame interface menu, there's a function to extract data from the repository. After the upload of the previous RA-Node, the extracted data from the Sesame RDF repository shows the presence of the following RDF statement:

```

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_X">
<rdf:type rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
<rdfs:label>ArgOnt_Instance_X</rdfs:label>
</rdf:Description>

```

### B.3 RQL and Premises List Query Example

Querying the ArgDF repository has been done using RQL. Passing queries to Sesame is also handled through the Phesame functions. For example if we need to extract the list of all the premises' text available in the repository. The query is written in a PHP variable, then passed to Sesame using the *simpleQuery()* function:

```
$serql =  
select INode, INode-text  
from {INode : kb:Premise} kb:text {INode-text}  
using namespace  
  rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#,  
  rdfs = http://www.w3.org/2000/01/rdf-schema#,  
  kb = http://protege.stanford.edu/kb#
```

Then the query is passed to Phesame in the following way, and the result is stored in a variable: `$result = $phesame->simpleQuery($serql)`

The `$result` variable is filled with XML syntax code. For example, the returned premises query result can be:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<tableQueryResult>  
  <header>  
    <columnName>INode-text</columnName>  
  </header>  
  <tuple>  
    <literal>Premise A Text</literal>  
  </tuple>  
  <tuple>  
    <literal>Premise B Text</literal>  
  </tuple>  
  <tuple>  
    <literal>Premise C Text</literal>  
  </tuple>  
</tableQueryResult>
```

The next step is to visualize the result and present it in a user friendly way through the user interface.

### B.4 Transforming XML Query Output through XSLT

In this section I present how the results are visualized in the browser interface to present an output similar to the one in Figure 5.3. Previously we saw how to store the result of a query in a PHP variable called `$result` containing XML

syntax code. Visualization is done after processing the result into an XSLT file and rendering it on the HTML page.

The XSLT as well as XPath expression code for creating a table filled with the literal values is:

```
<xsl:for-each select="tableQueryResult/tuple/literal">
<table width="499" border="1" cellpadding="0" cellspacing="2">
  <tr>
    <th width="385" scope="col"><div align="left" class="style2">
      <xsl:value-of select="."/></div></th>
    </tr>
  </table>
</xsl:for-each>
```

HTML design coding has been added to create a better interface. In order to process the XML document using XSLT, PHP offers this functionality by creating a new XSLT processor:

```
$xslt = new xsltProcessor
```

Then specifying the XSLT file to load into the new XSLT processor.

```
$xslt->importStyleSheet(DomDocument::load('file_name.xml'))
```

The query result contained in the \$result variable is processed using the *transformToXML()* PHP function:

```
$xslt->transformToXML(DomDocument::loadXML($result))
```

Finally the processed XML is visualized on the interface by the *print* PHP function:

```
print $xslt
```

## B.5 Creating a New Argument Full Cycle

In this section I go through the technical details of a full argument creation cycle. Figure 5.2 shows the argument creation cycle. The user starts by choosing the scheme s/he that the argument fulfils. Then the scheme details are displayed, and the RA-Node, connecting the different parts of the argument is created. At the end, the user enters the conclusion and premises fulfilling the scheme parts. While entering the premises and conclusion, the argument details are instantly updated.

### B.5.1 Choosing the Argumentation Scheme

The first step of building a new argument is to choose an argumentation scheme. There is a dual repository query to extract the scheme names and scheme labels (instance number).

*Query 1: Picking up the scheme names*

```
select Scheme, PresumptiveInferenceScheme-hasSchemeName
from {Scheme : kb:PresumptiveInferenceScheme} kb:hasSchemeName
  {PresumptiveInferenceScheme-hasSchemeName}
using namespace
  rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#,
  rdfs = http://www.w3.org/2000/01/rdf-schema#,
  kb = http://protege.stanford.edu/kb#
```

*Query 2: Picking up the scheme labels*

```
select Scheme, PresumptiveInferenceScheme-label
from {Scheme : kb:PresumptiveInferenceScheme} rdfs:label
  {PresumptiveInferenceScheme-label}
using namespace
  rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns# ,
  rdfs = http://www.w3.org/2000/01/rdf-schema#,
  kb = http://protege.stanford.edu/kb#
```



After the scheme querying, 2 XSLT files are applied to the results of the queries. One XSLT displays the name of the scheme, the other XSLT is used to pick the URI value, and insert it into the hyperlink URL variable, so that it can be used in the next step while creating the RA-Node.

### B.5.2 Creation of the RA-Node

After choosing the scheme, the repository is queried again to show the details of the scheme selected and a new RA-Node is uploaded to the RDF repository. Showing the scheme details requires doing a dual query on the repository; One to get the conclusion, and another one to get the list of premises details.

Extracting the conclusion and premises descriptions of the scheme necessitates a nested RQL query, as we need to get the description details that are inside the conclusion or premises which are inside the scheme matching the scheme number that is now available in the URL using the `$_GET` PHP function.

*Getting the conclusion of the scheme selected query*

```
select ConclusionDesc-hasDescription
from {ConclusionDesc} kb:hasDescription {ConclusionDesc-hasDescription}
where ConclusionDesc in
  select PresumptiveInferenceScheme-hasConclusionDescription
  from {PresumptiveInferenceScheme} kb:hasSchemeName
    {PresumptiveInferenceScheme-hasSchemeName},
    {PresumptiveInferenceScheme} rdfs:label {PresumptiveInferenceScheme-label},
    {PresumptiveInferenceScheme} kb:hasConclusionDescription
    {PresumptiveInferenceScheme-hasConclusionDescription}
where PresumptiveInferenceScheme-label like \"{$_GET['scheme']}\"/>
using namespace
  rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#,
  rdfs = http://www.w3.org/2000/01/rdf-schema#,
  kb = http://protege.stanford.edu/kb#
```

A similar query is used to extract the premises' description of the schemes.

After the query, 2 XSLT files process the results and display them to the user so that s/he is able to see how to structure the new argument created.

The RA-Node to be created should have the following attributes:

- *fulfilsScheme*: Which has the value of the previously chosen scheme
- *A new unique label (URI)*: The corresponding URI is bound to the value of the counter extracted from the MySQL database. In the background of this process, the value of the counter in MySQL is incremented by one.

The RA-Node RDF code to be uploaded:

```
$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:kb="&kb;" xmlns:rdfs="&rdfs;">
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%RA-Node Creation RDF Statement
  <kb:RA-Node rdf:about="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}'
    rdfs:label="ArgOnt_Instance_{$_row_ArgCount['count_A']}">
    <kb:fulfilsScheme rdf:resource="&kb;{$_GET['scheme']}" />
  </kb:RA-Node>
</rdf:RDF>
```

In the above RDF statement, ArgOnt\_Instance\_\$\_row\_ArgCount['count\_A'] is the instance label and URI value. The \$\_row\_ArgCount['count\_A'] is the value of the counter extracted from the queried MySQL database.

### B.5.3 Creation of the Conclusion & Premises

Now that a new RA-Node is uploaded to the repository, the user is redirected to enter the conclusion & premises that are all linked to the RA-Node of the argu-

ment. As in the previous step, the scheme details is also queried and displayed to the user.

*Entering the conclusion requires the following:*

1. Upload of the conclusion's RDF statement with the following attributes:

- *edgeFromINode*: having the value of the previously created RA-Node's URI.
- *fulfilsConclusionDesc*: having the value of the selected scheme's conclusion description to be fulfilled.
- *label and URI* with a unique value.

2. Update of the previously created RA-Node with the following attributes:

- *hasConclusion*: having the value of the current conclusion's URI.
- *edgeFromSNode*: having the value of the current conclusion's URI.

The RDF/ XML code of the conclusion to upload:

```
$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:kb="&kb;" xmlns:rdfs="&rdfs;">
%%Conclusion RDF Statement
  <kb:Conclusion
    rdf:about="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}"
    kb:text="\${$_POST['conclusion']}"
    rdfs:label="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}">
  <kb:edgeFromINode rdf:resource=
    "&kb;{$_SESSION['MM_Current_RA']}" />
  <kb:fulfilsConclusionDesc rdf:resource=
    "&kb;{$_GET['conclusion_fulfils']}" />
  </kb:Conclusion>
%%Conclusion RDF Statement
```

```
%%RA-Node Update RDF Statement
<kb:RA-Node rdf:about="\&kb;{$_SESSION['MM_Current_RA']}\ ">
  <kb:hasConclusion rdf:resource=
    "\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}\ ">
  <kb:edgeFromSNode rdf:resource=
    "\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}\ ">
</kb:RA-Node>
</rdf:RDF>
```

The PHP variable `$_POST['conclusion']` is used to extract the value from the interface textbox (text entered by the user that includes the content of the conclusion). The variable `$_SESSION['MM_Current_RA']` is a global session variable, used to store the value of the previously created RA-Node. Session variables in PHP maintain the values while navigating from one page to another.

`$_GET['conclusion_fulfil']` extracts the URI value of the selected scheme conclusion to fulfil from the URL.

*Entering the premises requires the following:*

Premises are created following the similar steps above of the conclusion creation. They differ in the attributes' type and that a user can create as many premises as s/he wants fulfilling a specific scheme premise description.

The premises created is linked to the RA-Node, leading to the following repository update:

1. Creation and upload of the premises' RDF statements having the below attributes:

- *edgeFromINode*: with the value of the previously created RA-Node's URI.

- *Supports*: also having the value of the previously created RA-Node's URI.
- *fulfilsPremiseDesc*: with the selected scheme's premise description URI to be fulfilled as value.
- *label and URI* of the premise with a unique value.

2. Accordingly the created RA-Node should be updated in the system:

- *hasPremise*: having the current premise's URI as value.
- *edgeFromSNode*: having the current premise's URI as value.

This step includes the upload of the following RDF/ XML statements code:

```
$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:kb="&kb;" xmlns:rdfs="&rdfs;">
%%Premise RDF Statement
  <kb:Premise rdf:about="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}"
    kb:text="{$_POST['premise']}"
    rdfs:label="ArgOnt_Instance_{$_row_ArgCount['count_A']}">
  <kb:edgeFromINode rdf:resource=
    "&kb;{$_SESSION['MM_Current_RA']}" />
  <kb:supports rdf:resource=
    "&kb;{$_SESSION['MM_Current_RA']}" />
  <kb:fulfilsPremiseDesc rdf:resource=
    "&kb;{$_SESSION['MM_Prem_Fulfils']}" />
</kb:Premise>
%%RA-Node Update RDF Statement
  <kb:RA-Node rdf:about="&kb;{$_SESSION['MM_Current_RA']}">
  <kb:hasPremise rdf:resource=
    "&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}" />
  <kb:edgeFromSNode rdf:resource=
    "&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}" />
</kb:RA-Node>
</rdf:RDF>
```

## B.6 Creating a New Argumentation Scheme

The necessary steps for scheme creation in ArgDF are:

1. Creation of the presumptive inference scheme with the following attributes:

- *hasSchemeName*: Which is a literal of type string taking the value of the scheme name text box entered by the user.
- *Unique label and URI*: With an automatically generated value.

The corresponding RDF/ XML code of the scheme creation:

```
$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY a 'http://protege.stanford.edu/system#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:kb="&kb;" xmlns:rdfs="&rdfs;">
  <kb:PresumptiveInferenceScheme
    rdf:about="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}"
    rdfs:label="&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}"
    kb:hasSchemeName="&kb;{$_POST['sname']}"
  </kb:PresumptiveInferenceScheme>
</rdf:RDF>
```

2. Creation of the scheme's conclusion and premises descriptions. This will result in the repository change by updating the previously created inference scheme and adding the attributes *hasConclusionDescription* and *hasPremiseDescription* respectively. Conclusion and premises descriptions are also created, linked to the scheme, and having the attribute *hasDescription* representing the information inside. I am not going in the details of this process as it's similar to the one in the new argument creation.

3. Creation of schemes also involves entering the according presumptions and exceptions. Presumptions that are entailed from certain premises are created by uploading the following RDF statement:

```
$data =
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY kb 'http://protege.stanford.edu/kb#'>
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>
<rdf:RDF xmlns:rdf="\&rdf;" xmlns:kb="\&kb;" xmlns:rdfs="\&rdfs;">
%%Presumption RDF Statement
  <kb:Presumption
    rdf:about="\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}"
    kb:hasDescription="\${$_POST['presumption']}"
    rdfs:label="\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}"
  </kb:Presumption>
%%Premise Description RDF Statement Update for entailment
  <kb:PremiseDesc rdf:about="\&kb;{$_GET['premisedesc_entails']}"
    <kb:entails
      rdf:resource="\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}"
    </kb:PremiseDesc>
%%Presumptive inference scheme RDF Statement update
  <kb:PresumptiveInferenceScheme
    rdf:about="\&kb;{$_SESSION['MM_Current_Scheme']}"
    <kb:hasPresumption
      rdf:resource="\&kb;ArgOnt_Instance_{$row_ArgCount['count_A']}"
    </kb:PresumptiveInferenceScheme>
</rdf:RDF>
```

The generic scheme's presumptions are the one that are not entailed by premises. The same RDF code above is used, but without the *entails* relationship.

The exceptions creation involves 2 steps. The first is for creating the conflict scheme with the *hasSchemeName* attribute to capture the conflict scheme name, followed by the exception (premises supporting the conflict scheme) description creation. Finally the presumptive scheme is updated to be linked to the newly created conflict scheme. The second is for the creation

of the exception as premise of the conflict scheme newly created.

The corresponding RDF code of the process:

```
$data =
<!DOCTYPE rdf:RDF [" .
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>" .
<!ENTITY kb 'http://protege.stanford.edu/kb#'>" .
<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>]>" .
<rdf:RDF xmlns:rdf="\&rdf;" xmlns:kb="\&kb;" xmlns:rdfs="\&rdfs;">" .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Exception of type PremiseDesc is created
  <kb:PremiseDesc
    rdf:about="\&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}"
    kb:hasDescription="\${$_POST['conclusion']}"
    rdfs:label="\&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}">" .
  </kb:PremiseDesc>" .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Conflict scheme linked to the exception
  <kb:ConflictScheme rdf:about="\&kb;{$_GET['conflict_scheme']}">" .
  <kb:hasPremiseDescription
    rdf:resource="\&kb;ArgOnt_Instance_{$_row_ArgCount['count_A']}">" .
  </kb:ConflictScheme>" .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Presumptive inference scheme RDF Statement update with hasException
  <kb:PresumptiveInferenceScheme
    rdf:about="\&kb;{$_SESSION['MM_Current_Scheme']}">" .
    <kb:hasException rdf:resource="\&kb;{$_GET['conflict_scheme']}">" .
  </kb:PresumptiveInferenceScheme>" .
</rdf:RDF>;
```

I will not go into coding details of other processes, as they are similar to the ones I already presented.



# Appendix C

## Full AIF-RDF RDFS Ontology Code

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:kb="http://protege.stanford.edu/kb#"
  xmlns:a="http://protege.stanford.edu/system#">

  <rdf:Description rdf:about="http://protege.stanford.edu/kb#edgeFromINode">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:label>edgeFromINode</rdfs:label>
    <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
    <rdfs:range rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
    <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edge"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://protege.stanford.edu/kb#ConclusionDesc">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:label>ConclusionDesc</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Form"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://protege.stanford.edu/kb#hasPremise">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <a:minCardinality>1</a:minCardinality>
    <rdfs:label>hasPremise</rdfs:label>
    <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Premise"/>
    <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
    <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromSNode"/>
    <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#supports"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://protege.stanford.edu/kb#RA-Node">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:label>RA-Node</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://protege.stanford.edu/kb#hasPremiseDescription">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <a:minCardinality>1</a:minCardinality>
    <rdfs:label>hasPremiseDescription</rdfs:label>
    <rdfs:range rdf:resource="http://protege.stanford.edu/kb#PremiseDesc"/>
    <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
  </rdf:Description>
```

## APPENDIX C: FULL AIF-RDF RDFS ONTOLOGY CODE

```
<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasException">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>hasException</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#ConflictScheme"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#PresumptiveInferenceScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Form">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Form</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#isAttacked">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>isAttacked</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#CA-Node"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Conclusion"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#attacks"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromINode"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasDescription">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <a:minCardinality>1</a:minCardinality>
  <rdfs:label>hasDescription</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Form"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasConclusionDescription">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>hasConclusionDescription</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#ConclusionDesc"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#RuleScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>RuleScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#PA-Node">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>PA-Node</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#S-Node">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>S-Node</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#PreferenceScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>PreferenceScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#PremiseDesc">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>PremiseDesc</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Form"/>
</rdf:Description>
```

## APPENDIX C: FULL AIF-RDF RDFS ONTOLOGY CODE

```
<rdf:Description rdf:about="http://protege.stanford.edu/kb#InductiveInference">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>InductiveInference</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#RuleScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ConflictScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>ConflictScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Conclusion">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Conclusion</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#I-Node">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>I-Node</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasSchemeName">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>hasSchemeName</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Scheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Scheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#attacks">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>attacks</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#CA-Node"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromINode"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#isAttacked"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Premise">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Premise</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#LogicalPreferenceScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>LogicalPreferenceScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#PreferenceScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#edge">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>edge</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#DeductiveInference">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>DeductiveInference</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#RuleScheme"/>
</rdf:Description>
```

## APPENDIX C: FULL AIF-RDF RDFS ONTOLOGY CODE

```
<rdf:Description rdf:about="http://protege.stanford.edu/kb#fulfilsConclusionDesc">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>fulfilsConclusionDesc</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Conclusion"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#ConclusionDesc"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50396">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <kb:text>test exception</kb:text>
  <rdfs:label>ArgOnt_Instance_50396</rdfs:label>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50394"/>
  <kb:attacks rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50394"/>
  <kb:fulfilsPremiseDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_12"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#entails">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>entails</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#PremiseDesc"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Presumption"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#fulfilsScheme">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>fulfilsScheme</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Scheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#caNode_isAttacked">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>caNode_isAttacked</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#CA-Node"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#caNode_Attacks"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromSNode"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#caNode_Attacks">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>caNode_Attacks</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#CA-Node"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#caNode_isAttacked"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromSNode"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#PresumptivePreferenceScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>PresumptivePreferenceScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#PreferenceScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#PresumptiveInferenceScheme">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>PresumptiveInferenceScheme</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#RuleScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasConclusion">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <a:minCardinality>1</a:minCardinality>
  <rdfs:label>hasConclusion</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Conclusion"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
</rdf:Description>
```

## APPENDIX C: FULL AIF-RDF RDFS ONTOLOGY CODE

```
<rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromSNode"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#hasPresumption">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>hasPresumption</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Presumption"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#PresumptiveInferenceScheme"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#supports">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:label>supports</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edgeFromINode"/>
  <a:inverseProperty rdf:resource="http://protege.stanford.edu/kb#hasPremise"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#CA-Node">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>CA-Node</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#edgeFromSNode">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:minCardinality>1</a:minCardinality>
  <rdfs:label>edgeFromSNode</rdfs:label>
  <rdfs:range rdf:resource="http://protege.stanford.edu/kb#Node"/>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#S-Node"/>
  <rdfs:subPropertyOf rdf:resource="http://protege.stanford.edu/kb#edge"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#text">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <a:maxCardinality>1</a:maxCardinality>
  <rdfs:label>text</rdfs:label>
  <rdfs:domain rdf:resource="http://protege.stanford.edu/kb#I-Node"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Presumption">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Presumption</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://protege.stanford.edu/kb#Form"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#Node">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label>Node</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>
```

## Appendix D

### ArgDF Argument RDF Code

The below code, extracted from the Sesame RDF server, represents 2 arguments under attack created in ArgDF. The purpose of this appendix is to show in full how the resources are interconnected in RDF. Resources have unique identifications, with a certain type like “premise” and specific attributes which can either be literals such as “text,” or relationships heading to other resources such as the “supports” relationship.

The code flows by representing the first argument’s premises, conclusion and RA-Node. Then the CA-Node, linking the arguments in conflict is presented, followed by the second argument’s RA-Node, attacking the former one, as well as its premises and conclusion.

```
<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_16">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <kb:text>Allen says that Brazil has the best football team</kb:text>
  <rdfs:label>ArgOnt_Instance_16</rdfs:label>
  <kb:supports rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_13"/>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_13"/>
  <kb:fulfillsPremiseDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_6"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_15">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <kb:text>Allen is an expert is sports</kb:text>
  <rdfs:label>ArgOnt_Instance_15</rdfs:label>
```

## APPENDIX D: ARGDF ARGUMENT RDF CODE

```
<kb:supports rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_13"/>
<kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_13"/>
<kb:fulfilsPremiseDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_7"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_14">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Conclusion"/>
  <kb:text>Brazil has the best football team</kb:text>
  <rdfs:label>ArgOnt_Instance_14</rdfs:label>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_13"/>
  <kb:fulfilsConclusionDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_5"/>
  <kb:CANode_isAttacked rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50486"/>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50486"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_13">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
  <rdfs:label>ArgOnt_Instance_13</rdfs:label>
  <kb:hasConclusion rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_14"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_14"/>
  <kb:hasPremise rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_15"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_15"/>
  <kb:hasPremise rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_16"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_16"/>
  <kb:fulfilsScheme rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_4"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50486">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#CA-Node"/>
  <rdfs:label>ArgOnt_Instance_50486</rdfs:label>
  <kb:CANode_Attacks rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_14"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_14"/>
  <kb:isAttacked rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50487"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50485">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#RA-Node"/>
  <rdfs:label>ArgOnt_Instance_50485</rdfs:label>
  <kb:fulfilsScheme rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_4"/>
  <kb:hasConclusion rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50487"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50487"/>
  <kb:hasPremise rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50488"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50488"/>
  <kb:hasPremise rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50489"/>
  <kb:edgeFromSNode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50489"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50487">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Conclusion"/>
  <kb:text>Germany has the best football team</kb:text>
  <rdfs:label>ArgOnt_Instance_50487</rdfs:label>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50485"/>
  <kb:attacks rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50486"/>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50486"/>
  <kb:fulfilsConclusionDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_5"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50489">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <kb:text>Jim is an expert in sports including football</kb:text>
  <rdfs:label>ArgOnt_Instance_50489</rdfs:label>
  <kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50485"/>
  <kb:supports rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50485"/>
  <kb:fulfilsPremiseDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_7"/>
</rdf:Description>

<rdf:Description rdf:about="http://protege.stanford.edu/kb#ArgOnt_Instance_50488">
  <rdf:type rdf:resource="http://protege.stanford.edu/kb#Premise"/>
  <kb:text>Jim says that Germany has the best football team</kb:text>
  <rdfs:label>ArgOnt_Instance_50488</rdfs:label>
```

## APPENDIX D: ARGDF ARGUMENT RDF CODE

```
<kb:edgeFromINode rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50485"/>
<kb:supports rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_50485"/>
<kb:fulfilsPremiseDesc rdf:resource="http://protege.stanford.edu/kb#ArgOnt_Instance_6"/>
</rdf:Description>
```



## References

- [1] L. Amgoud, M. Serrut, and H. Prade. Flexible querying with argued answers. In *EUROFUSE Workshop on Data and Knowledge Engineering*, Poland, Sept. 2004.
- [2] G. Antoniou and F. van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. MIT Press, Cambridge MA, USA, 2004.
- [3] K. Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals*. MIT Press, Cambridge, 1990.
- [4] ASPIC. Argumentation service platform with integrated components, a European Commission-funded research project (no. ist-fp6-002307), 2004.
- [5] K. Atkinson, T. Bench-Capon, and P. McBurney. PARMENIDES: facilitating deliberation in democracies. *Artificial Intelligence and Law – T. van Engers and A. Macintosh (editors), Special Issue on eDemocracy*, page (to appear), 2006.
- [6] M. Bachler, S. B. Shum, D. D. Roure, D. Michaelides, and K. Page. Ontological mediation of meeting structure: Argumentation, annotation, and

## REFERENCES

- navigation. In *Proceedings of the 1st International Workshop on Hypermedia and the Semantic Web (HTSW2003)*, Nottingham, UK, 2003.
- [7] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, 2001.
- [8] M. Beveridge and D. Milward. Combining task descriptions and ontological knowledge for adaptive dialogue. In *In Proc. TSD 2003, International Conference on Text Speech and Dialogue*, Ceske Budejovice, Czech Republic, 2003.
- [9] M. Beveridge and D. Milward. Definition of the high-level task specification language. technical report, deliverable d11, eu homey project, ist-2001-3243, 2003.
- [10] M. Beveridge and D. Milward. Ontology-based dialogue systems. In *In Proc. IJCAI 3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Acapulco, Mexico, aug 2003.
- [11] D. Carbogim, D. Robertson, and J. Lee. Argument-based applications to knowledge engineering. *Knowledge Engineering Review*, 15(2):119–149, 2000.
- [12] C. I. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32(4):337–383, 2000.
- [13] C. I. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari,

## REFERENCES

- M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 2007.
- [14] Compendium. The Compendium Institute, 2005.  
<http://www.compendiuminstitute.org>.
- [15] J. Conklin. Designing organizational memory: Preserving intellectual assets in a knowledge economy, 2001. Gognexus White Paper.
- [16] J. Conklin and M. L. Begeman. gIBIS: a hypertext tool for exploratory policy discussion. *ACM transactions on office information systems*, 6(4):303–331, 1988.
- [17] A. S. COULSON, D. W. GLASSPOOL, J. FOX, and J. EMERY. Rags : A novel approach to computerized genetic risk assessment and decision support from pedigrees. *Methods of information in medicine (Methods inf. med.)*, 40(4):315–322, 2001.
- [18] M. Dacota, L. Obrst, and K. Smith. *The Semantic Web – A Guide to the Future of XML, Web Services, and Knowledge Management*. Wiley, 2003.
- [19] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
- [20] D.W.Glasspool, J.Fox, F.D.Castillo, and V.Monaghan. Interactive decision support for medical planing. In *9th Conference on Artificial Intelligence in Medicine in Europe (AIME 2003)*. Springer, 2003.

## REFERENCES

- [21] A. Garcia and G. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 2002.
- [22] T. V. Gelder. A reasonable approach to critical thinking. *Principal Matters: The Journal for Australasian Secondary School Leaders*, 34(6), 2002.
- [23] T. F. Gordon and N. Karacapilidis. The Zeno argumentation framework. In *Proceedings of the Sixth International Conference on AI and Law*, pages 10–18, New York, NY, USA, 1997. ACM Press.
- [24] T. F. Gordon and D. Walton. The Carneades argumentation framework. In P. Dunne and T. Bench-Capon, editors, *Proceedings of the 1st International Conference on Computational Models of Argument (COMMA)*, pages 195–207, Amsterdam, The Netherlands, 2006. IOS Press.
- [25] K. Greenwood, T. Bench-Capon, and P. McBurney. Structuring dialogue between the people and their representatives. In R. Traunmuller, editor, *Electronic Government: Proceedings of the Second International Conference (EGOV03), Prague, Czech Republic*, volume 2739 of *Lecture Notes in Computer Science*, pages 55–62, Berlin, Germany, 2003. Springer.
- [26] K. Greenwood, T. Bench-Capon, and P. McBurney. Towards a computational account of persuasion in law. In G. Sartor, editor, *Proceedings of the Ninth International Conference on AI and Law (ICAIL-03)*, pages 22–31, New York, NY, USA, 2003. ACM Press.
- [27] T. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

## REFERENCES

- [28] N. Karacapilidis and D. Papadias. Computer supported argumentation and collaborative decision making: the HERMES system. *Information Systems*, 26:259–277, 2001.
- [29] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the semantic web with rql. *Computer Networks* 42, 42(5):617–640, 2003.
- [30] P. Krause, P. Judson, and M. Patel. Qualitative risk assessment fulfills a need. In A. Hunter and S. Parsons, editors, *Applications of Uncertainty Formalisms*. Springer-Verlag, Berlin, Germany, 1998.
- [31] J. Lee. Sibyl: a tool for managing group decision rationale. In *Proceedings of the conference on computer-supported cooperative work*, pages 79–92, New York, USA, 1990. ACM Press.
- [32] R. Lüehrs, T. Malsch, and K. Voss. Internet, discourses and democracy. In T. Terano, T. Nishida, A. Namatame, S. Tsumoto, Y. Ohsawa, and T. Washio, editors, *New Frontiers in Artificial Intelligence*, volume 2253 of *Lecture Notes in Computer Science*, pages 67–74. Springer-Verlag, Heidelberg, Germany, 2001.
- [33] D. L. McGuinness and F. van Harmelen. Web ontology language (OWL): Overview. Technical report, W3C Working Draft, 31 March 2003.
- [34] php. <http://www.php.net>, 2006.

## REFERENCES

- [35] J. L. Pollock. Rational cognition in oscar. In *Intelligent Agents V (ATAL-99)*, London, UK, 1999. Springer-Verlag.
- [36] H. Prakken, C. Reed, and D. N. Walton. Dialogues about the burden of proof. In *Proceedings of the 10th international conference on artificial intelligence and law (ICAIL)*, pages 115–124, New York NY, USA, 2005. ACM Press.
- [37] H. Prakken and G. Vreeswijk. Encoding schemes for a discourse support system for legal argument. In *Working Notes of the ECAI 2002 workshop on Computational Model of Natural Argument (CMNA), Lyon, France, 22 July*, pages 31–39, 2002.
- [38] H. Prakken and G. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 4, pages 219–318. Kluwer Academic Publishers, Dordrecht, Netherlands, second edition, 2002.
- [39] Protégé. <http://protege.stanford.edu>, 2006.
- [40] I. Rahwan, C. Reed, and F. Zablith. On building argumentation schemes using the argument interchange format. In *Proceedings of the IJCAI Workshop on Computational Models of Natural Argument (CMNA)*, Hyderabad, India, Jan 2007.
- [41] B. Rolf and C. Magnusson. Developing the art of argumentation: A software approach. In F. H. van Eemeren, C. A. Willard, and A. F. S. Henkema, editors, *Proceedings of the 5th Conference of the International Society*

## REFERENCES

- for the Study of Argumentation (ISSA), University of Amsterdam, June 25-28, pages 919–925, Amsterdam, The Netherlands, 2002. Sic Sat.*
- [42] G. W. A. Rowe, C. A. Reed, and J. Katzav. Araucaria: Marking up argument. In *European Conference on Computing and Philosophy*, 2003.
- [43] B. Schmidt-Belz, C. Rinner, and T. F. Gordon. Geomed for urban planning - first user experiences. In *GIS '98: Proceedings of the 6th ACM international symposium on Advances in geographic information systems*, pages 82–87, New York, NY, USA, 1998. ACM Press.
- [44] Sesame. <http://www.openrdf.com>, 2006.
- [45] S. B. Shum, V. Uren, G. Li, B. Sereno, and C. Mancini. Modelling naturalistic argumentation in research literatures: Representation and interaction design issues. *International Journal of Intelligent Systems, Special Issue on Computational Modelling of Naturalistic Argumentation (to appear)*, 22(1), 2007.
- [46] D. Skalak and E. Rissland. Arguments and cases. an inevitable intertwining. *Artificial Intelligence and Law*, 1:3–44, 1992.
- [47] S. Toulmin. *The Uses of Argument*. Cambridge University Press, UK, 1964.
- [48] truthmapping. <http://www.truthmapping.com>, 2006.
- [49] B. Verheij. Artificial argument assistants for defeasible argumentation. *Artificial Intelligence*, 150(1–2):291–324, 2003.

## REFERENCES

- [50] G. Vreeswijk. Iacas: an implementation of chisholm's principles of knowledge.
- [51] D. N. Walton. *Argumentation Schemes for Presumptive Reasoning*. Erlbaum, Mahwah NJ, USA, 1996.
- [52] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, New York, 1995.