Defeasible Reasoning based Argumentative Web-IDSS for Virtual Teams (VTs)

Naeem Khalid Janjua  
Digital Ecosystems and Business Intelligence Institute  
Curtin University of Technology  
Perth, WA  
Email: naeemkhalid.janjua@curtin.edu.au

Farookh Khadeer Hussain  
Digital Ecosystems and Business Intelligence Institute  
Curtin University of Technology  
Perth, WA  
Email: farookh.hussain@cbs.curtin.edu.au

Abstract—The Web-based intelligent decision support system (Web-IDSS) is pivotal for a Virtual Team (VT) to successfully execute business-related tasks. The current generation of Web-IDSS built on top of semantic web technologies for VTs lacks the capability to provide decision support when underlying information is incomplete and/or contradictory. In this article, we address this limitation of current Web-IDSS through defeasible logic based argumentation formalism. The proposed Web-IDSS uses a hybrid reasoning approach: forward chaining (data-driven) for the construction of arguments over incomplete information, and backward chaining (goal-driven) for conflict identification and resolution with explanation. The proposed Web-IDSS adheres to web standards and publishes the outcome of argumentative reasoning in Argument Interchange Format (AIF).

I. INTRODUCTION AND RELATED WORK

Today, business environments are becoming more complex, competitive and dynamic, demanding that organizations be more flexible and responsive to environmental changes. As a result, organizations are establishing Information Technology as a primary enabler in order to adapt to an ever changing competitive environment. One of the outcomes of such efforts is the establishment of Virtual Team (VT) to accomplish one or more organizational tasks [1]. Technological support for these virtual teams for collaboration in distributed environments is now a topic of great interest.

The VT deals with two types of knowledge. The first is static knowledge composed of facts and ontologies which remain static over a period of time; this requires monotonic reasoning based upon open world assumptions to guarantee the correct propagation of truth. The second category, which needs special attention, is dynamic knowledge such as business policies, business contracts etc. that often change according to business needs and strategies, leading to conflicts among business policies. Such dynamic knowledge that is potentially incomplete and inconsistent needs non-monotonic or defeasible reasoning [2]. The term ‘defeasible reasoning’ was coined to mean ‘convincing’ although not rigorous reasoning as a concept introduced in the philosophy of law. Defeasible reasoning is a simple rule-based approach to perform reasoning about uncertain information where a rule supporting a conclusion may be negated or invalidated with the emergence of new information.

Business policies or contracts are also rule-based statements that are used by organizations to run their business activities. The importance of explicit, declarative and automatically executable business rules has been identified by several researchers [3, 4]. Attempts have been made to apply defeasible logic based approaches to represent and reason over dynamic knowledge that potentially is incomplete and conflicting as depicted in Table I. Although these attempts are very promising, they define explicit priorities among business rules at compile time, in order to resolve conflicts among business rules in advance. Whereas, VT is a source of defeasible knowledge as it is open by nature and subject to inconsistencies deriving from multiple sources; therefore, it is not possible to define priorities in advance among conflicting rules. Additionally, in current implementations, the use of these priorities is usually embedded in the derivation mechanism and competing rules are compared individually during the derivation process. In such formalisms, the derivation notion is bound to one single comparison criterion. In such a scenario, the explanation of the results is based on a single criterion only and fails to take into account the multiple factors important for decision-making in VT.
In contrast, if we look at Artificial Intelligence research, the challenge of incomplete and conflicting knowledge representation and reasoning over it in software agents has been addressed using logic-based argumentation formalism, i.e. defeasible logic programming (DeLP) [5]. Argumentation formalisms are defeasible reasoning systems which work by considering the reasons that lead to a given conclusion (claim) through a piece of reasoning (the supporting arguments) and potential challenges (counter arguments) for accepting the conclusion [6]. Argumentation plays a pivotal role in identifying and organizing what can be justifiably concluded, and presenting it systematically to human users or merging it with the justified conclusions of other machines in the absence of complete or accurate information.  

In this article, we identify the shortcomings of DeLP in the context of providing decision support to a VT and extend it with data-driven reasoning and make it interoperable with semantic web standards. We propose defeasible logic based argumentative Web-IDSS for a VT. The novelty of the system is its hybrid reasoning approach: forward chaining (data-driven) for the construction of arguments, and backward chaining (goal-driven) for the evaluation of conclusions. The proposed system provides graphical explanation to the members of a VT for better understanding and traceability of results. From the Web-IDSS perspective, such powerful support systems would be able to carry out reasoning on data across members of a VT. This is the potential area of growth and research in Intelligent Web-DSS as depicted in Figure 1.

### II. Defeasible Logic Programming

Defeasible logic programming (DeLP)[5] is a general-purpose defeasible argumentation formalism based on logic programming, intended to model inconsistent and potentially contradictory knowledge. A defeasible logic program has the form \( \Pi \leftrightarrow (\Pi, \Delta) \), where \( \Pi \) and \( \Delta \) stand for strict knowledge and defeasible knowledge, respectively. The set \( \Pi \) involves strict rules of the form \( P \leftarrow Q_1 \ldots Q_n \) and facts (strict rules with empty body), and it is assumed to be non-contradictory (i.e., no complementary literals \( P \) and \( \sim P \) can be inferred, where \( \sim P \) denotes the contrary of \( P \)). The set \( \Delta \) involves defeasible rules of the form \( P \leftarrow Q_1 \ldots Q_n \) which stand for \( Q_1 \ldots Q_n \) provide a tentative reason to believe \( P \). Rules in DeLP are defined in terms of literals. A literal is an atom \( A \) or the strict negation (\( \sim A \)) of an atom. Default negation (denoted not \( A \)) is also allowed in the body of defeasible rules. For more details about syntax and semantics of DeLP interested readers are referred to [5].

DeLP uses the argumentation formalism for the treatment of contradictory information by identifying conflicting information in the knowledge base and applying the dialectical process to decide which information prevails leading to conclusion. Therefore in DeLP, in order to decide between competing conclusions, the arguments that support the conclusions are compared. Thus, the comparison criterion is independent of the derivation process, and could be replaced in a modular way. Although DeLP addresses the challenge of incomplete and conflicting knowledge representation and reasoning, it cannot be used for the development of Web-IDSS for VT because of following limitations:

1. DeLP uses backward chaining or goal-driven reasoning only, whereas most of the reasoning in Web-IDSS for VTs is primarily data-driven requiring forward chain reasoning.
2. There is no tool available for VTs to translate business rules defined in RuleML\(^1\) format into DeLP rule format.
3. There is no tool that provides proof explanation by graphical representation of reasoning steps and conclusion generated by an argumentative reasoner for traversal by VT members.

### III. Proposed System Architecture

In this section, we discuss the proposed system architecture in detail as depicted in Figure 1. Let us assume that a virtual team for the Olympic Games is comprised of the Olympic International Committee (OIC), Organising Committees for the Olympic Games (OCOG) and host city. The objective of the virtual team is to make important decisions about sports activities. These three organizations have their own particular goals and expectations which impact on the overall organisation of the sports events. Let us further assume that the current task of VT members is to decide "whether or not a scheduled match will be played in rainy conditions". To accomplish this task, each member of a VT provides his/her views in form of rules about the stated task, and with the help of our proposed system, they reach to a justifiable conclusion.

\(^1\)http://ruleml.org/
The key components of the proposed system architecture are as follows:

A. **View points against a business task**

RuleML is an important step to provide a uniform format for the representation of business rules for a virtual team. RuleML supports different business rule types via the pre-defined `implies` element and allows them to be named using the pre-defined `oid` element. RuleML syntax has been extended to express defeasible rules [10]. All members of a virtual team define their business rules in RuleML format and submit them through a graphical user interface of our proposed system as their point of view against a certain business task. The important thing to note here is the assumption that all members of a VT use the same domain ontology concepts to specify their business rules.

B. **Knowledge base**

A collection of facts is known as working memory and collection of rules is known as rule base. A working memory and a rule base are collectively known as knowledge base. In the proposed framework, the rule base comprises of strict rules and defeasible rules. Figure 3 depicts a knowledge base for a VT. In rule base a business rule takes the following form \([\text{rule identifier}] \ [\text{rule body}] \ [\text{type of rule}] \ [\text{head}], \) where:

- **rule identifier**: is an identifier or name of the business rule;
- **rule body**: is a tuple of predicates. Each predicate is called as premise;
- **head**: a predicate whose instances could be intuitively considered to be added to the working memory when the rule is activated during argument construction (defined later on);
- **type of rule**: system supports two types of rules: strict rules and defeasible rules. The strict rule is used to represent non-negotiable information represented by a solid arrow such as \(\rightarrow\). Whereas, a defeasible rule is used to represent tentative, negotiable information and is represented by dotted arrow such as \(\rightarrow\).

C. **Translation of viewpoints and domain ontology**

The proposed system parses and translates the business rules defined in RuleML notation to DeLP rule format and saves them in a knowledge base. Table II represents the ontology schema translation rules which also reside in the knowledge base to bring semantic interoperability among the business rules. Let us consider that ‘ground’ and ‘stadium’ are two ontological concepts from the domain ontology such that ground is subclass of stadium. The working memory contains ‘ground(perth)’. On execution of Rule 2 depicted in Table II i.e., `ground(X) \(\rightarrow\) stadium(X)` results in addition of new fact i.e. `stadium(perth)`, in the working memory.

D. **Construction of arguments from view points**

To construct arguments from a set of rules and working memory, forward chain reasoning is carried out which involves a rule engine matching the conditions of the rules in the rule base against the facts in working memory. For each match, a rule instance is created and put into the Active rule set. Once the matching phase is completed, the instances of all the rules in Active rule are fired. Firing the rule instance will result in the addition of a new fact to the working memory and the addition of an instance of rule known as argument in the Argument Set.

- A label is used to identify the argument.
- A conclusion is known as its claim.
- Ground predicate is known as the premise, supporting the claim.

A key issue to be noted here is that the new inferred facts may conflict with the existing knowledge base. The purpose is to retain conflicting information instead of eliminating it, in order to obtain a better insight when deciding on business strategies. Figure 4 represents the updated working memory and argument set.

E. **Conflict detection, resolution and justifiable explanation**

Once the argument construction process is complete, the process of conflict detection, resolution with explanation...
phase is initiated. For this purpose, we used a built-in mechanism of DeLP. We first identify the argument having a counter-argument. We define counter-argument as:

- Given an argument set, an argument ‘r’ counter-argues argument ‘s’ if and only if claim of argument ‘r’ is inconsistent with claim of ‘s’ or claim of argument ‘r’ is inconsistent with the premises of argument ‘s’.

In Figure 4, an argument ‘hc1’ is counter-argument to argument ‘ocog1’. The system forwards the claim of the argument to the DeLP engine to construct its dialectical trees or justification trees as depicted in Figure 5 (Left) where graphically an argument is represented as \((hc1, \sim groundReady(perth))\) where hc1 is argument identifier and \(\sim groundReady(perth)\) is claim of argument. Once the dialectical tree construction is complete, the nodes in dialectical trees are marked as either defeated or un-defeated as shown in Figure 5 (Right) and the DeLP engine return the results to the system. If the marked dialectical tree is defeated, then the counter-argument has priority over the argument and vice versa. Similarly, the system resolves the remaining conflicts between arguments and their counter-arguments by passing the argument’s claim to DeLP engine to acquire its support from the knowledge base. In case, the marked dialectical tree of both, the argument and its counter-argument are undefeated, then those arguments are considered as blocking arguments and the system needs human intervention to resolve the conflict between them.

F. Conclusion

The last step of an argumentation process is the construction of a conclusion in the form of a reasoning chains. We define a supporting argument or sub-argument as:

- Given an argument set, an argument ‘r’ is a sub-argument of argument ‘s’ if and only if claim of argument ‘r’ belongs to premise of argument ‘s’.

In Figure 4 an argument ‘oic1’ is supporting argument to argument ‘ocog2’.

During this process, all sub-arguments with undefeated dialectical trees are linked together as a reasoning chain. This process will continue until all possible arguments are linked up into a reasoning chain. The top argument i.e. conclusion, of the reasoning chain is called the ‘result’ of the reasoning chain, and the chain of arguments supporting the top argument are called to support the conclusion as depicted in Figure 7. The reasoning chain is always consistent (i.e., there is no contradiction in the result and support for the result). Therefore, for example, \(groundReady(perth)\) and \(\sim groundReady(perth)\) will not belong to one reasoning chain, but each one of them can belong to different reasoning chains and those reasoning chains represent alternative paths or choices.

G. Publish conclusion

The Argument Interchange Format (AIF) [11] is an international effort to develop a representational mechanism for exchanging argument resources between research groups, tools, and domains using a semantically rich language. The system make use of Argument Interchange Format (AIF) to export the output of the argumentation process i.e. reasoning chains, over the web to ensure its interoperability with other argumentative systems. For annotation of a reasoning chain, we developed ‘reasoning chain ontology’ on top of the ArgDF ontology\(^2\) and serialized the AIF compliant reasoning chain in RDF/XML format.

\(^2\)http://www.argdf.org/source/ArgDF Protege Ontology.zip
H. Query the knowledge base

The members of a VT can query the knowledge base. The query consists of a predicate, and can be executed on the argument set to check the support for the predicate in the argument set. There are four possible answers to a query. If the answer is ‘yes’, then the result will be an undefeated dialectical tree. If the answer is ‘no’, then the result will be a defeated dialectical tree. If the answer is ‘undecided’, then the result will be a blocked dialectical tree. If unknown, then the predicate in the query is not in the language of the program.

IV. Prototype development and conclusion

The prototype development of the system is carried on a machine having Apache Web server version 2.2.11, PHP version 5.3.0, PHP Tree Graph Ext library\(^3\), MYSQL database version 5.1.36 and SWI-Prolog installed on it. We extended the PHP Tree Graph Ext library with certain extensions to differentiate between fact and claim of a rule, strict and defeasible inference etc. The proposed system provides an interface to members of a VT to submit their rules against a business task defined in RuleML format. After translation of rules, the proposed system performs argumentative reasoning as discussed in Section III.

Figure 7 shows graphical representation of a reasoning chain. Figure 7 depicts results of argumentative reasoning in the form of a graphical reasoning chain and a small pop window which shows justification for an argument $\text{hc1}, \sim \text{groundReady(berth)}$). We conclude here, by saying that, applications built on top of the proposed work, will provide more practical, understandable results to members of a VT for intelligent decision making.

REFERENCES


\(^3\)http://download.getabest.com/new/php-tree-graph-ext-222943.html