Argumentation For All

Nikolaos Spanoudakis
nikos@amcl.tuc.gr
School of PEM,
Technical University of Crete
Chania, Greece

Konstantinos Kostis
kkostis@isc.tuc.gr
School of ECE,
Technical University of Crete
Chania, Greece

Katerina Mania
k.mania@ced.tuc.gr
School of ECE,
Technical University of Crete
Chania, Greece

ABSTRACT

This paper proposes the use of a web-based system for the development of applications of argumentation. It focuses on bringing the capability to develop decision policies based on argumentation to people that have little or no knowledge of logic programming or of an argumentation framework. To achieve this, it presents an implementation of the table formalism that has recently been put forward by previous work. Our system was evaluated using the think aloud protocol from the early stages of development.

CCS CONCEPTS

• **Human-centered computing → Web-based interaction**; *User centered design*; • **Computing methodologies → Knowledge representation and reasoning**.

KEYWORDS

computational argumentation, web application, knowledge engineering, hierarchical argumentation frameworks

ACM Reference Format:
Nikolaos Spanoudakis, Konstantinos Kostis, and Katerina Mania. 2020. Argumentation For All. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3341105.3374122

1 INTRODUCTION

Argumentation is a relatively-new, fast-paced technology that, following the AI trend, has started producing real-world applications. Argumentation has been addressed as a way to deal with contentious information and draw conclusions about it. The main focus of its applications are for making context-related decisions [6].

There is a number of software libraries [1] for developing applications of argumentation, e.g. Gorgias [5], CaSAPI [3], DeLP [4], *ASPIC*$^+$ (TOAST system) [11] and SPINdle [7], however, these require a substantial logic programming effort by experts.

Recently, the Gorgias-B [12] Java-based tool offered a higher level development environment aiding the user to develop a decision policy. Gorgias-B is built on top of the Gorgias framework and on, one hand, aids in the elicitation of the expert/user knowledge in the form of scenario-based preferences among the available options, and, on the other hand, automatically generates the corresponding executable Gorgias code. Moreover, the Gorgias-B tool supports scenario execution that helps the user to put to test the generated argumentation theory. However, Gorgias-B still needs the user to follow an argumentation domain specific method and have knowledge of Prolog style logic programming application development. Moreover, its use requires a complex installation process including the installation of Java and SWI-Prolog. A web-based system, Spindle [7], allows for web-based development and testing of defeasible logic applications. Its environment, however, just includes a text editor for writing logic programming rules.

In this paper, we propose a *web application*, named ***WebGorgiasB***, using as a reference the Gorgias-B [12] tool, emphasizing on the ease of use. Our aim is to eliminate the need for logic programming knowledge for application developers, thus, allowing even naive users to define decision policies. Moreover, we introduce, for the first time, the use of the table formalism, that has been recently proposed in the literature [6], for naive users to define their scenarios and select the available options in each scenario.

During system development we used the *think aloud protocol* [9] for evaluating the user interface. Based on that, features helping the accomplishment of each application task were added and the size, colors and layout of original controls were determined. All applications stored are available in the cloud, so that they can be ready to be edited, demonstrated or executed at the user's convenience.

2 MOTIVATION AND BACKGROUND

The Gorgias-B tool has been based on a systematic methodology for developing hierarchical argumentation frameworks applications. Hierarchical Argumentation Frameworks (HAF) allow developers to not only define preference among arguments, but also to define preference on preferences, thus, allowing to have default preferences but also context based preferences. The following example will help the reader familiarize with the terms *option*, *fact*, *belief*, *preference* and *argument rule*, concepts.

Working with Gorgias-B, a decision problem is defined as the process of choosing the best option o_i, $i \in 1, .., n$ among the set O of n available options. We will use an example throughout the paper to help explain our work. In this example, a user, Ralf, defines a policy for his personal assistant to accept or deny an incoming call. His set of options is outlined in (1).

$$O = \{o_1 = allow(call), o_2 = deny(call)\} \quad (1)$$

In the paper, we will use the same notation with the one used in a relevant paper that set the theoretical foundation of the table-based argumentation theory generation [6]. We will also use the abbreviated symbols of predicates and ground atoms in order to save space and not clutter the equations, e.g. we will use o_1 instead of

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03.
https://doi.org/10.1145/3341105.3374122

allow(*call*). To choose among the options we define scenario-based preferences, using the syntax $SP^{level}_{scenario} = \langle S^{level}_{scenario}; O^{level}_{scenario} \rangle$:

$$SP^1 = \langle S^1 = \{true\}; O^1 = \{o_1, o_2\} \rangle \quad (2)$$

where SP^1 is the scenario-based preference of level 1, where the scenario 1 holds (S^1) and both options are acceptable. Note that we have omitted the *scenario* subscript as the scenario doesn't have any conditions (always *true*). In the Gorgias structured argumentation framework, argument rules link a set of premises with their *position*. An *argument* is a set of one or more such *argument rules*, denoted by $Label = Conditions \rhd Position$. Such an argument rule links a set of *Conditions* with a *Position*. The SP in (2) implies the following two object level arguments:

$$arg^{SP^1}_{o_1} = \{true\} \rhd o_1 \quad (3)$$

$$arg^{SP^1}_{o_2} = \{true\} \rhd o_2 \quad (4)$$

where $arg^{SP^1}_{o_2}$ denotes the object level argument for the scenario preference of level 1, see equation (2) and option o_1.

We can also use the ">" operator between two argument rules' labels to denote that the one on the left hand side is preferred over the one on the right hand side. This operator assigns preference over other rules. When the labels are of object-level rules (at the first level) then we have a preference at the second level. When the labels are of n^{th} level rules then we have a preference at level $(n+1)$. The following example shows how to connect a Scenario Preference (SP) to arguments generation:

$$SP^2_{f,f} = \langle S^2_{f,f} = S^1 \cup C^2_{f,f} = \{true\} \cup \{family\_time, family\_call\}; O^2_{f,f} = \{o_1\} \rangle \quad (5)$$

$$SP^2_{f,bu} = \langle S^2_{f,bu} = S^1 \cup C^2_{f,bu} = \{true\} \cup \{family\_time, business\_call\}; O^2_{f,bu} = \{o_2\} \rangle \quad (6)$$

$$arg^{SP^2_{f,f}}_{o_1\_over\_o_2} = \{family\_time, family\_call\} \rhd arg^{SP^1}_{o_1} > arg^{SP^1}_{o_2} \quad (7)$$

$$arg^{SP^2_{f,bu}}_{o_2\_over\_o_1} = \{family\_time, business\_call\} \rhd arg^{SP^1}_{o_2} > arg^{SP^1}_{o_1} \quad (8)$$

Argument rule (7) is implied by the scenario preference (5), as Ralf answers a call from family members when spending time with his family. Rule (8) is implied by the scenario preference (6), as Ralf doesn't accept business calls in the *family time* context.

Object level rules can take along priority rules to build stronger arguments. In Gorgias [5], which is based on Dung's abstract argumentation framework [2], we have a set of arguments *Arg* and the *Att* attack binary relation between them. An argument attacks another if they draw complementary conclusions (options). An argument that attacks back all its attackers is an *admissible* argument.

Thus, when Ralf spends family time and there is an incoming call from a business associate, a number of arguments can be constructed, however the $\{arg^{SP^1}_{o_2}, arg^{SP^2_{f,bu}}_{o_2\_over\_o_1}\}$ is the only admissible (i.e. no other company of argument rules can fight it back). For the detailed semantics the interested reader can refer to Kakas et al. [6].

Gorgias-B [12] guides the user in defining object-level arguments and then allows users to define priorities among them in the second level. If there are contrary priorities then they are resolved in a next level and this process iterates until there are no conflicts.

Recently, the method presented by Kakas et al. [6] defined a table formalism for capturing requirements and a basic theoretical algorithm for generating code for refined scenarios (i.e. scenarios continuously advancing in levels by adding contextual information).

Our work expands Gorgias-B and implements a similar functionality in a web application. For the first time, it proposes an implementation for the table view presented by Kakas et al. [6]. The main key feature introduced is the Argue Table, where users can review their scenario preferences in a more responsive and clear way. This feature is expected to benefit users in creation of arguments and definition of option properties. Also, from the table view, users are able to expand and refine their already created scenario preferences by adding new facts and beliefs that they would like to include into their new scenarios.

Installing the Gorgias-B original application requires the installation of a number of tools (Java, SWI-Prolog) and the editing of a configuration file, restricting its use to experienced users. Based on this work, we aim to make the decision policy development capability available to naive users, i.e. users without technical knowledge on logic programming.

3 THE WEB APPLICATION

The overall application was designed to take advantage of the principles and benefits of the Model-View-Controller [8] (MVC) design pattern. This means that distinct modules are created to control the presentation of the data, filtering it according to the user's criteria and managing it in a data model. We used modern technologies such as Angular (https://angular.io/) for the client side and REST services [10] and the Spring-Boot programming framework (https://spring.io/projects/spring-boot) on the server side.

A number of **REST services** were developed. The most important of them are outlined in what follows. The **PrologService** contains all the functions needed to successfully compile and simulate the execution of each project. Code generation is conducted using the method followed in Gorgias-B. To establish a connection between Prolog and Spring Boot we use the JPL library provided by SWI-Prolog (https://www.swi-prolog.org/).

The **CoreNLPService** was implemented to achieve the main functional requirements of creating a user-friendly GUI easy to use by non-expert users, difficult and complex Prolog elements, such as Predicates should be visualised in another way. The main idea was to guide the user to write a free form text, which after Natural Language Processing (NLP), would be transformed into Prolog's argument's structure predicate form. A verb as a predicate identifies a relation between entities denoted by the subjects and complements. So, utilizing the Stanford's coreNLP tool (https://nlp.stanford.edu/software/) entities are transformed into word functions. The abstract form of representation is $verb(subject, object, nouns)$ with minor changes per input. This transformation is presented to the user, who can approve it or adjust it.

The **ScenarioService** includes the functions needed to group all the scenarios created, by their name, in a Table View. Table 1 shows the scenarios presented in section 2. The graphical user interface (GUI) of the ***WebGorgiasB*** application includes the original Argue Table custom view (see how Table 1 looks in the application in Figure 1). There is a function to create a scenario preference based

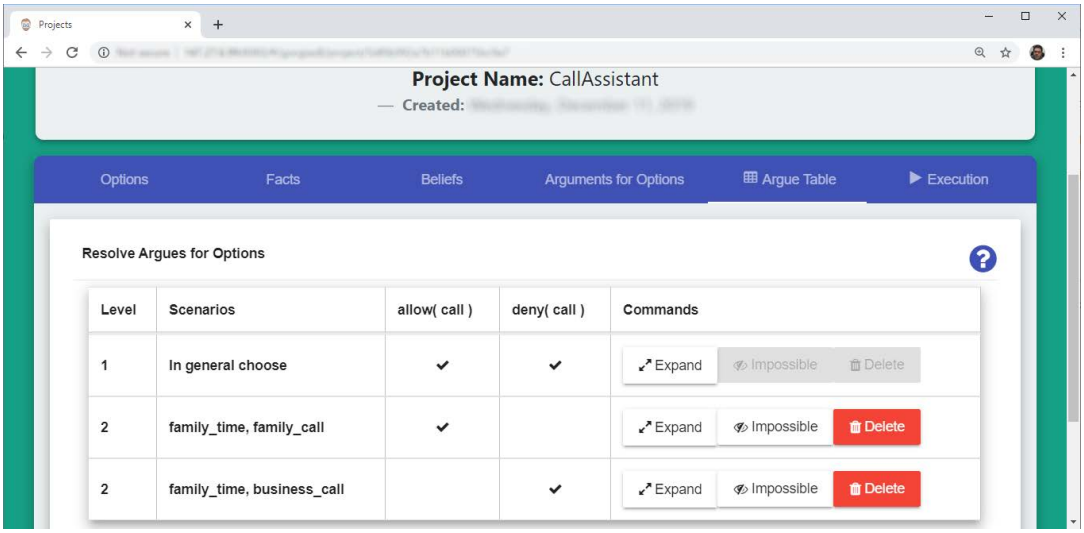

Figure 1: Argue Table View. In the screenshot we see the *Basic View* of the requirements presented in Table 1.

on selected beliefs and facts, accompanied with the appropriate option(s) whenever the user adds a line to the table. For each selected option o, the algorithm accesses all arguments in that scenario that have as preferred option that selected option and also the arguments that have the selected option as a non-preferred option and then checks if the preferred options of these arguments are complements. If they are, then the algorithm generates the preference arguments at a higher level (see [6] for more details).

Thorough evaluation, both informal and structured, was conducted so that the system's usability was assessed during the system's development. The **think aloud protocol** was selected as the main evaluation methodology because of the complexity of the system and the need to allow for free-form conversation between the user and the researcher guiding the evaluation [9]. Various user comments were integrated in the user interface design throughout its implementation. Help messages were added, size of buttons was affected as well as colours, etc. The clarity of labels was enhanced.

4 CONCLUSION

This paper reports on a web-based application for decision policy definition and a simulation application for the Gorgias argumentation framework. Its main goal was to create a web interface accessible by the general public, i.e. people without knowledge of logic programming. The functionality of the system was evaluated using the think aloud protocol.

	Scenarios	***Options***	
		o_1	o_2
1	$S^1 = \{true\}$	x	x
2	$S^2_{f,f} = \{family\_time, family\_call\}$	x	
3	$S^2_{f,bu} = \{family\_time, business\_call\}$		x

Table 1: Argue table for Call Assistant example.

We present, for the first time, an argumentation-based implementation of the table-based requirements gathering formalism that was proposed recently in the literature [6]. Additionally, we proposed a new feature that uses NLP for forming the scenario predicates from natural language expressions.

Future work is focused on allowing the user to define options in a scenario that are not present in previously selected scenarios. In order to allow for large-scale application development, we will explore ways to have different tables for all diverging contexts so that the user can focus only in the branch of the scenario currently refined.

REFERENCES

[1] Cerutti, F., Gaggl, S. A., Thimm, M., and Wallner, J. P. Foundations of implementations for formal argumentation. *The IfCoLog Journal of Logics and their Applications; Special Issue Formal Argumentation 4*, 8 (September 2017).

[2] Dung, P. M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence 77*, 2 (1995), 321–357.

[3] Gaertner, D., and Francesca. Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems 22*, 6 (2007), 24–33.

[4] García, A. J., and Simari, G. R. Defeasible logic programming: An argumentative approach. *TPLP 4*, 1-2 (2004), 95–138.

[5] Kakas, A. C., and Moraitis, P. Argumentation based decision making for autonomous agents. In *AAMAS 2003* (Melbourne, Victoria, Australia, 14-18 July 2003), ACM, pp. 883–890.

[6] Kakas, A. C., Moraitis, P., and Spanoudakis, N. I. *GORGIAS*: Applying argumentation. *Argument & Computation 10*, 1 (2019), 55–81.

[7] Lam, H.-P., and Governatori, G. The Making of SPINdle. In *RuleML 2009* (Las Vegas, Nevada, USA, 5-7 Nov. 2009), Springer-Verlag, pp. 315–322.

[8] Leff, A., and Rayfield, J. T. Web-Application Development Using the Model/View/Controller Design Pattern. In *EDOC 2001* (Seattle, WA, USA, 4-7 Sept. 2001), IEEE, pp. 118–127.

[9] McDonald, S., and Petrie, H. The effect of global instructions on think-aloud testing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), ACM, pp. 2941–2944.

[10] Pautasso, C., Zimmermann, O., and Leymann, F. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 805–814.

[11] Snaith, M., and Reed, C. TOAST: online aspic$^+$ implementation. In *COMMA 2012* (Vienna, Austria, 10-12 Sept. 2012), pp. 509–510.

[12] Spanoudakis, N. I., Kakas, A. C., and Moraitis, P. Gorgias-B: Argumentation in Practice. In *COMMA 2016* (Potsdam, Germany, 12-16 Sept. 2016), pp. 477–478.