Web-Gorgias-B: Argumentation For All

Nikolaos I. Spanoudakis[1][a], Konstantinos Kostis[2] and Katerina Mania[2]
[1]*Applied Mathematics and Computers Laboratory (AMCL),*
School of Production Engineering and Management, Technical University of Crete, Chania, Greece
[2]*SURREAL Team, Distributed Multimedia Information Systems and Applications Laboratory (MUSIC),*
School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece
nikos@amcl.tuc.gr, kkostis@isc.tuc.gr, k.mania@ced.tuc.gr

Keywords: Computational Argumentation, Knowledge Representation and Reasoning, Knowledge Engineering, Hierarchical Argumentation Frameworks, Web-based Interaction, Intelligent User Interfaces.

Abstract: This paper proposes the use of a web-based authoring tool for the development of applications of argumentation. It focuses on aiding people that have little, or no knowledge of logic programming, or of an argumentation framework, to develop argumentation-based decision policies. To achieve this, it proposes an implementation of the table formalism that has recently been proposed in the literature. The proposed implementation contains original features that were evaluated by experts in web-application development, students and experts in argumentation. The main feature of the proposed system is the ability to define a default preferred option in a given scenario, thus, allowing for other options to be used in further refinements of the scenario. We followed a user-centered development process using the think aloud protocol. We evaluated the usability of the system with the System Usability Scale, validating our hypothesis that even naive users can employ it to define their decision policies.

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 (Van Eemeren et al., 2004). The main focus of its applications is for making context-related decisions.

Medica, for example, is an argumentation system that allows for deciding if a specific person can have access to sensitive medical files, based on a) who is the requester, e.g. the owner, a medical doctor, etc, b) what is the reason for requesting access (research, treatment, etc), and, c) what additional support is available, e.g. order from the medical association, written consent from the owner and other similar requests. Argumentation allows such decisions to be *explainable* to humans (Spanoudakis et al., 2017).

Modern argumentation-based *cognitive assistants* serve users by learning from them their habits and preferences. They are capable of common sense reasoning and sense the users' environment in order to gather as much information as possible before choosing a course of action (Kakas and Loizos, 2016; Costa et al., 2017).

There is a number of software libraries (Cerutti et al., 2017) for developing applications of argumentation, e.g., Gorgias (Kakas and Moraitis, 2003), CaSAPI (Gaertner and Toni, 2007), DeLP (García and Simari, 2004), *ASPIC*$^+$ (TOAST system) (Snaith and Reed, 2012) and SPINdle (Lam and Governatori, 2009), however, these require a substantial logic programming effort by experts.

Recently, the Gorgias-B (Spanoudakis et al., 2016) 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

[a] https://orcid.org/0000-0002-4957-9194

complex installation process including the installation of Java and SWI-Prolog.

A web-based system, SPINdle (Lam and Governatori, 2009), 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 ***Web-Gorgias-B***. 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 produce, for the first time, an implementation of the table formalism, that has been recently proposed in the literature (Kakas et al., 2019), enabling naive users to define their scenarios and select the available options in each scenario.

During system development, we evaluated our proposed system using the *think aloud protocol* (McDonald and Petrie, 2013) involving naive and expert users in order to evaluate its performance in a qualitative manner. Following that, useful features such as tips that help the accomplishment of each application task, or the size, colors and layout of original controls were determined. All developed policies are stored in the cloud, so that they can be edited, demonstrated or executed at the user's convenience. Lastly, we employed a formal evaluation method based on the System Evaluation Scale (SUS) (Bangor et al., 2009) to quantify the system's usability and identify its strengths and weaknesses.

This paper extends a short paper published recently (Spanoudakis et al., 2020). The original content herein is the presentation of the two new features (not referred to in the short version), a) the *impossible scenario* feature and b) the ability to define options in scenarios as *default*. It also presents the algorithms and the evaluation using SUS.

In the following sections we first discuss background, define standard argumentation concepts and what is already possible with the Gorgias-B tool. Then, we discuss the goals we set in this work to advance the state of the art, before outlining the system's architecture. Subsequently, we focus on the technical contribution. Finally, we discuss the thorough system evaluation conducted.

2 Background and Related Work

We will present the Gorgias-B tool as our background, along with the SPINdle tool, the only other tool for developing defeasible logic-based decision theories, as related work.

2.1 SPINdle

SPINdle is a logic reasoner that can be used to compute the consequence of defeasible logic theories in an efficient manner (Lam and Governatori, 2009). This implementation covers both the basic defeasible logic and modal defeasible logic. SPINdle can also be used as a standalone theory prover or as an embedded reasoning engine. SPINdle's user interface can be characterized as too poor with only one textarea, where users define their arguments and defeasible facts. In order to define these, the user has to be knowledgeable of defeasible logic and its syntax. It is prohibitive to naive users who cannot efficiently use this tool.

2.2 Gorgias-B

The Gorgias-B tool is based on the Gorgias Argumentation framework, which is written in Prolog (Kakas and Moraitis, 2003). It employs a Graphical User Interface (GUI) written in Java and encapsulates the essential features of the Gorgias framework, hiding from users the underlying technology. Therefore, users do not have to be experts in argumentation. However, they need to be familiar with the Prolog language and its syntax.

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 (Modgil, 2006). The following example will help the reader familiarize with the terms *option*, *fact*, *belief*, *preference* and *argument rule*, concepts that are important for further reading.

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. For the better understanding of our work, we will illustrate a working example that will be used throughout the paper. An interested reader can compare the process here with the one followed by Gorgias-B for a quite similar example [1]. In that example, Ralf, a professional, defines the decision policy for his phone's cognitive assistant. The available options for our example are:

$$o_1 = allow(call) \quad (1)$$

$$o_2 = deny(call, without\_explanation) \quad (2)$$

$$o_3 = deny(call, with\_explanation) \quad (3)$$

[1] http://gorgiasb.tuc.gr/Tutorial3.html#ca

In the paper, we will use the same notation with the one used by the authors of the paper that set the theoretical foundation of the table-based argumentation theory generation (Kakas et al., 2019). We will also use 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 $allow(call)$. To choose among the options we define scenario-based preferences, using the syntax $SP^{level}_{scenario} = \langle S^{level}_{scenario}; O^{level}_{scenario} \rangle$:

$$SP^1 = \langle S^1 = \{true\}; O^1 = \{o_1, o_2, o_3\} \rangle \tag{4}$$

where SP^1 is the scenario-based preference of level one, where scenario S^1 holds and all three available options are acceptable. Note that we have omitted the *scenario* subscript as the scenario doesn't have any conditions (i.e. is *true*). In the Gorgias hierarchical 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 (4) implies the following object level arguments:

$$arg^{SP^1}_{o_1} = \{true\} \rhd o_1 \tag{5}$$

$$arg^{SP^1}_{o_2} = \{true\} \rhd o_2 \tag{6}$$

$$arg^{SP^1}_{o_3} = \{true\} \rhd o_3 \tag{7}$$

where $arg^{SP^1}_{o_2}$ is a label for the *object level* argument for the scenario preference of option one at level one, see expression (4). An object level argument links conditions (or premises) to its position. The premises are those that unlock the supported position. More context may be added later as we will see. In this case, the semantics behind the object level arguments of our example is that when there is a new incoming call, then Ralf's assistant can select any one of the three options (credulously).

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:

$$SP^2_{f,f} = \langle S^2_{f,f} = S^1 \cup C^2_{f,f} = \{true\} \cup \{family\_time, family\_call\}; O^2_{f,f} = \{o_1\} \rangle \tag{8}$$

$$SP^2_{f,bu} = \langle S^2_{f,bu} = S^1 \cup C^2_{f,bu} = \{true\} \cup \{family\_time, business\_call\}; O^2_{f,bu} = \{o_1, o_2\} \rangle \tag{9}$$

$$SP^3_{f,bu,bo} = \langle S^3_{f,bu,bo} = \{family\_time, business\_call, call\_from\_boss\}; O^3_{f,bu,bo} = \{o_1\} \rangle \tag{10}$$

$$arg^{SP^2_{f,f}}_{o_1\_over\_o_2} = \{family\_time, family\_call\} \rhd arg^{SP^1}_{o_1} > arg^{SP^1}_{o_2} \tag{11}$$

$$arg^{SP^2_{f,f}}_{o_1\_over\_o_3} = \{family\_time, family\_call\} \rhd arg^{SP^1}_{o_1} > arg^{SP^1}_{o_3} \tag{12}$$

$$arg^{SP^2_{f,bu}}_{o_1\_over\_o_2} = \{family\_time, business\_call\} \rhd arg^{SP^1}_{o_1} > arg^{SP^1}_{o_2} \tag{13}$$

$$arg^{SP^2_{f,bu}}_{o_1\_over\_o_3} = \{family\_time, business\_call\} \rhd arg^{SP^1}_{o_1} > arg^{SP^1}_{o_3} \tag{14}$$

$$arg^{SP^2_{f,bu}}_{o_2\_over\_o_1} = \{family\_time, business\_call\} \rhd arg^{SP^1}_{o_2} > arg^{SP^1}_{o_1} \tag{15}$$

$$arg^{SP^2_{f,bu}}_{o_2\_over\_o_3} = \{family\_time, business\_call\} \rhd arg^{SP^1}_{o_2} > arg^{SP^1}_{o_3} \tag{16}$$

$$arg^{SP^3_{f,bu,bo}}_{o_1\_over\_o_2} = \{call\_from\_boss\} \rhd arg^{SP^2_{f,bu}}_{o_1\_over\_o_2} > arg^{SP^2_{f,bu}}_{o_2\_over\_o_1} \tag{17}$$

Argument rules (11) and (12) are implied by the scenario preference (8), where only o_1 is acceptable, i.e. Ralf wants his phone to ring, if there is an incoming call from his family members when spending time with his family. Note that the premises (or conditions) of argument rule (11) are two facts, i.e. *family_time* and *family_call*. Similarly, argument rule (12) is supported by the *family_time* and *family_call* facts.

Rules (13)-(16) are implied by the scenario preference (9), as only options o_1 and o_2 are allowed in the scenario. Both are preferred to the other arguments of level one, indicating that when Ralf spends some time with his family he never explains in a text message his situation to callers related to his work (he either replies or denies the call without explaining the reason for doing so). In the third level scenario (10), however, only o_1 is allowed, i.e. when it is his boss that is calling he will answer the phone, therefore the preference in formula (17) is added.

Object level rules can take along priority rules to build stronger arguments. In Gorgias (Kakas and

Moraitis, 2003), which is based on Dung's abstract argumentation framework (Dung, 1995), 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 his boss a number of arguments can be constructed, however the $\{arg^{SP^1}_{o_1}, arg^{SP^2_{f,bu}}_{o_1\_over\_o_2}, arg^{SP^3_{f,bu,bo}}_{o_1\_over\_o_2}\}$ is the only admissible (i.e. no other company of argument rules can fight it back). An interested reader can find the detailed semantics and formal definition of the Gorgias framework in the work of Kakas and Moraitis (Kakas and Moraitis, 2003).

Gorgias-B (Spanoudakis et al., 2016) 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, researchers defined a table formalism for capturing requirements and a basic theoretical algorithm for generating code for refined scenarios (Kakas et al., 2019). Refined scenarios are continuously advancing in levels by adding more specific contextual information.

3 Problem statement - Motivation

Our work proposes an implementation for the table view recently presented by researchers in a theoretical paper (Kakas et al., 2019). 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. The implementation posed specific challenges and was not straight forward as the algorithm presented by the authors that introduced this formalism only works for refining a single scenario (Kakas et al., 2019). We found out that there were quite some challenges in generating code when there are default options in a given scenario (an original feature proposed in this paper, see next paragraph) and then new information extends that given scenario, or when scenarios can be combined. We present the algorithms that we developed in Section 4.2.

Furthermore, in the Argue Table we introduce two new features, a) the *impossible scenario* feature and b) the ability to define options in scenarios as *default*. The second is the one with more technical interest for the Knowledge Representation and Reasoning community. Users can define options as default at each particular scenario. For example, Ralf might want to define that even though it is possible for a business call to be answered while he is spending time with his family, his default response is to deny the call. Thus, only if it is the case of the boss calling, whose call is also a business call, and, thus, a more specific scenario related to the previous one, is a reply possible. This functionality is impossible with the existing form of scenario-based preferences.

If we removed o_1 from $SP^2_{f,bu}$, then formulas (13) and (14) would not have been generated, and, thus, in the next refinement of the scenario, they would not be there for the call from boss context to give them priority. Thus, we would have to define $arg^{SP^3_{f,bu,bo}}_{o_1\_over\_o_2}$ as a new scenario in level two $arg^{SP^2_{f,bu,bo}}_{o_1\_over\_o_2}$. In that case, as his boss is also a business associate, then both $arg^{SP^2_{f,bu,bo}}_{o_1\_over\_o_2}$ and $arg^{SP^2_{f,bu}}_{o_2\_over\_o_1}$ would be admissible for an incoming call from his boss and his personal assistant might select to deny the call. Of course there could be a true scenario in the next level clarifying the situation, i.e. $arg^{SP^3_{f,bu,bo}}_{o_1\_over\_o_2}$, however, this approach would clatter the table formalism as the same scenario would have to appear to a next level, thus effectively confusing the naive user.

This example, which motivates our work is not just a possible theoretical case but a real-world requirement. It comes from consulting users that want to define their policies to create applications of argumentation. For example, Pison (Pison, 2017), a highly qualified ophthalmologist doctor of a public hospital in Paris (France), proposed the development of an eye-clinic support system based on the set of known ocular diseases (there are more than 80). The different diseases are the available options in the decision policy. The use of tables provided a compact representation of the expert knowledge and one she could use as a naive user. The manual translation of the table (provided in spreadsheet format) to a Gorgias decision theory was a tedious task that has taken many months, done by an expert who used the Gorgias-B tool. A prototype web application is under development using hundreds of scenarios with the prospect of being deployed in the collaborating eye clinic as a commercial product. The automatic translation of the scenarios to an argumentation theory would save many work hours and prevent human errors.

Another problem is that to install the Gorgias-B

application is difficult as it requires the installation of a number of tools (Java, SWI-Prolog) and the editing of a configuration file, restricting its use to experienced users. We want to make the decision policy development capability available to naive users, i.e. users without technical knowledge of logic programming, or of managing complex configuration files requiring paths to installed software libraries .

Therefore, we set forward with the hypothesis that if we undertook these tasks we would develop a system useful even for non-experts to define their decision policy using argumentation.

4 System Architecture

We provide an overview of the system architecture in Subsection 4.1. This Subsection is quite technical and of more interest to colleagues with experience in the software engineering and/or web interface development areas. Then, in Subsection 4.2, we focus on the functionality of the Scenario service, which offers most of the innovative features of the system and which is the most interesting for the Knowledge Representation and Reasoning community.

4.1 Overview

The overall application was designed to take advantage of the principles and benefits of the Model-View-Controller (Leff and Rayfield, 2001) (MVC) design pattern (see Figure 1). 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. CRUD (Create, Read, Update, Delete) services provide access to the database.

Figure 2 shows the generated data that are transmitted through a REST service (Pautasso et al., 2008) at system Model level, where they are stored for use in queries that will result in their execution in Prolog environment and the return of the result.

4.1.1 Client Side

The client-side application employs technologies that can run on any standard browser without the need for any additional software (such as a runtime environment). More specifically, we employed HTML5 (https://www.w3.org/TR/html/), CSS3 (https://www.w3.org/TR/CSS) and Angular (https://angular.io/).

4.1.2 Server Side

The **ScenarioService** is implemented in the Spring Boot framework (https://spring.io/projects/

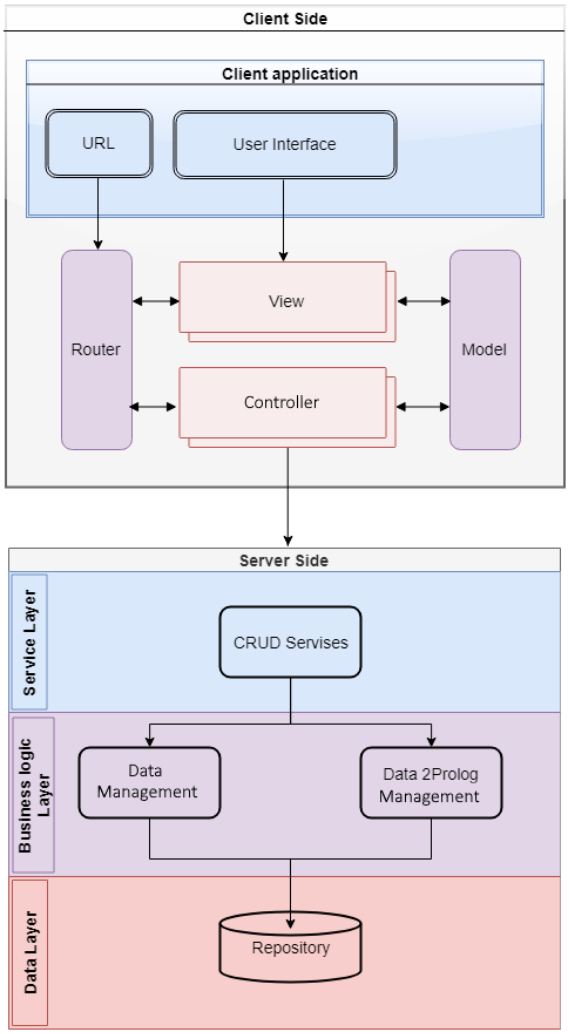

Figure 1: Application's main design model

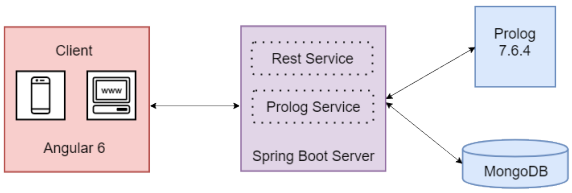

Figure 2: Application's distinct modules design

spring-boot). The functionality of the developed application that refers to scenarios is implemented in this service. This service is described in detail in Section 4.2.

The **Prolog Service** is implemented by the Spring-Boot programming framework. The user's decision model is defined using the concepts and data structures presented in Section 2.2. The *Eclipse EMF* (http://eclipse.org) technology, together with its *xtext* extension (Eysholdt and Behrens, 2010), provides automated Prolog code generation for the Gorgias the-

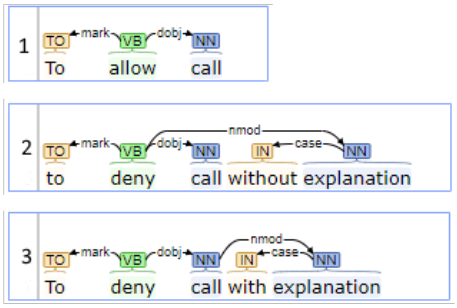

Figure 3: Core NLP text transform. Each of the three numbered cells are translated to the corresponding option of our example.

ory. To establish a connection between Prolog and Spring Boot we use the Java-Prolog Interface (JPL) library (Ali et al., 2016). The Prolog Service contains all the functions needed to instantiate and execute each project in Prolog.

All the other functionalities are implemented as **RESTful services** (Pautasso et al., 2008). Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating web services. RESTful services are bound by the principle of statelessness, which means that each request from the client to server must include all the details to understand the request. This improves visbility, reliability and scalability for requests.

The **CoreNLPService** was also implemented using the Spring-Boot framework. In order 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 is to let the user write in a free form text, and then, after applying an appropriate Natural Language Processing (NLP) method, automatically transform it into Prolog's predicate form. Or, in other words, the goal is to extract a predicate from free text in a way that ensures almost complete acquisition of predicate-argument structures from text. We consider extraction of predicate-arguments, structures from a single text with a substantial narrative part.

Verbs play a fundamental role in NLP, so verb information in lexicons is essential. A verb as a predicate identifies a relation between entities denoted by the subjects and complements. So, the *CoreNLPService*, utilizing the power of Stanford's coreNLP tool (https://nlp.stanford.edu/software/), processes the given sentence and analyzes the entity relation of the sentence by verb. When the syntactical analysis completes, entities are transformed into word functions as presented in Figure 3. The abstract form of representation is $verb(subject, object, nouns)$ with

Table 1: Argue table for Call Assistant example. Lower case "x" denotes that the option in the column is valid within the scenario and upper case "X" denotes a default option.

	Scenarios	***Options***		
		o_1	o_2	o_3
1	$S^1 = \{true\}$	x	x	x
2	$S^2_{f,f} = \{family\_time, family\_call\}$	x		
3	$S^2_{f,bu} = \{family\_time, business\_call\}$	x	**X**	
4	$S^3_{f,bu,bo} = S^2_{f,bu} \cup \{from\_boss\_call\}$	x		

minor changes per input. This transformation is presented to user to approve it or to adjust it.

The **Database Service** was implemented with a NoSQL family database system. The open-source MongoDB database was selected (https://www.mongodb.com/. MongoDB is used to store eight types of Collections:

- **User Data**, the user access credentials
- **Project**, a user can have zero or more projects that have:
  - **Options, Complements** (i.e. incompatible options), **Facts, Beliefs**[2]), **Argument Rules**, and, **Impossible Scenarios** in real-life applications

4.2 ScenarioService

The *ScenarioService* includes the functions needed to group all the created scenarios, by their name, in a table view. Table 1 shows the scenarios presented in Section 2. It includes a notation (bold, uppercase lettering) for allowing to define default options at each Scenario. The graphical user interface (GUI) of the ***Web-Gorgias-B*** application includes the Argue Table view. The reader will have the chance to see how the theoretical view presented in Table 1 looks in the developed system in Section 5 and in Figure 5.

There is a function to create a scenario preference based on selected beliefs and facts, accompanied with the appropriate option(s). Whenever the user adds a line to the table, the *expandScenario* function is called. This function implements Algorithm 1. So, for each selected option o, the algorithm searches the previous level for arguments for and against that option. Then it creates new arguments at the current

[2]Sometimes the premises of arguments are themselves defeasible. We call them **beliefs**, and they can a) be argued for or against, i.e. the position of an argument can also be a literal on a belief, or b) be assumed. In the latter case, they are called *abducibles*. The only restriction posed by our framework for beliefs is that when they are the position of an argument rule, the premises supporting it cannot hold option predicates. An interested reader can refer to Kakas and Moraitis (Kakas and Moraitis, 2003) for more details

level preferring the arguments for that option to the arguments against that option.

Function `expandScenario`$(SP_x^{lvl}, defaults)$**:**

1 **for** ***each*** $o \in O_x^{lvl}$ **do**
2 **for** ***each*** $arg^{SP_{x'}^{lvl-1}}_{o\_over\_o'}, x' \subset x$ **do**
3 **for** ***each*** $arg^{SP_{x'}^{lvl-1}}_{o'\_over\_o}$ **do**
4 **if** $complements(o, o')$ **then**
5 **create** $arg^{SP_x^{lvl}}_{o\_over\_o'} = (S_x^{lvl} - S_{x'}^{lvl-1}) \triangleright arg^{SP_{x'}^{lvl-1}}_{o\_over\_o'} > arg^{SP_{x'}^{lvl-1}}_{o'\_over\_o}$
end
end
end
end
6 call $insertPreference(SP_x^{lvl}, defaults)$
7 call $autoCorrectArgs(SP_x^{lvl})$

Algorithm 1: Central algorithm for refining a scenario.

As previously stated, one of the innovations that this implementation offers is the function for users to give to an option higher priority over all others at a scenario. Algorithm 2 is called for defining default options. The algorithm searches for argument rules of a specific scenario S, whose non-preferred option is the one that the user wants to define as default. Then, for each of these arguments, the algorithm creates a counter-argument at a higher level, by setting higher priority for the default option and the context to true.

Function `insertPreference`$(SP_x^{lvl}, defaults)$**:**

1 **for** ***each*** $o \in defaults$, **do**
2 **for** ***each*** $o', arg^{SP_x^{lvl}}_{o'\_over\_o}$ **do**
3 **if** $\nexists arg^{SP_x^{lvl+1}}_{o\_over\_o'}$ **then**
4 **create** $arg^{SP_x^{lvl+1}}_{o\_over\_o'} = \{true\} \triangleright arg^{SP_x^{lvl}}_{o\_over\_o'} > arg^{SP_x^{lvl}}_{o'\_over\_o}$
end
end
end

Algorithm 2: Central algorithm for inserting a default preference for an option.

Due to the possibility of the existence of default options in the scenarios examined, to avoid conflicting preferences, a correction algorithm is executed to correct possible mistakes about hierarchies (see Algorithm 3).

Given the example shown in Table 1 and having created the object level rules for each option, when a user tries to expand the newly created scenario to produce the second line (2) of the table, the

Function `autoCorrectArgs`(SP_x^{lvl})**:**

1 **for** *each* $x', SP_{x'}^{lvl}, x' \subset x$ **do**
2 **for** ***each*** $arg^{SP_{x'}^{lvl}}_{o'\_over\_o}, conditions(arg^{SP_{x'}^{lvl}}_{o'\_over\_o}) = \{true\}$ **do**
3 **for** ***each*** $arg^{SP_x^{lvl}}_{o\_over\_o'}, conditions(arg^{SP_x^{lvl}}_{o\_over\_o'}) \neq \{true\}$ **do**
4 **create** $arg^{SP_x^{lvl+1}}_{o\_over\_o'} = \{true\} \triangleright arg^{SP_x^{lvl}}_{o\_over\_o'} > arg^{SP_{x'}^{lvl}}_{o'\_over\_o}$
end
end
end

Algorithm 3: Central algorithm for correcting arguments after scenario refinement

expandScenario function (shown in Algorithm 1) is called:

$$expandScenario(SP^2_{f,f}, \varnothing)$$

and the arguments that it generates are the ones in formulas (11)-(12). There are no default arguments, and no default priorities for previous defaults at the second level. Then, as soon as the user completes the third line (3) of the table, the *expandScenario* function is called again:

$$expandScenario(SP^2_{f,bu}, \{o_2\})$$

and the arguments that will be generated are the ones in formulas (13)-(16). After the arguments are generated the *insertPreference* function will be invoked in line 6 of Algorithm 1. This time there is a default option, o_2, and the following argument rule will be created:

$$arg^{SP^3_{f,bu}}_{o_2\_over\_o_1} = \{true\} \triangleright arg^{SP^2_{f,bu}}_{o_2\_over\_o_1} > arg^{SP^2_{f,bu}}_{o_1\_over\_o_2} \quad (18)$$

Finally, the user defines line 4 of Table 1. The *expandScenario* function is called again:

$$expandScenario(SP^3_{f,bu,bo}, \varnothing)$$

and the argument that will be generated is the one in formula (17). This time there is no default option, however, the invocation of the *autoCorrectArgs* function in line 7 of Algorithm 1 will produce an argument rule, as there was a default preference at the previous level and formula (18):

$$arg^{SP^4_{f,bu,bo}}_{o_1\_over\_o_2} = \{true\} \triangleright arg^{SP^3_{f,bu,bo}}_{o_1\_over\_o_2} > arg^{SP^3_{f,bu}}_{o_2\_over\_o_1} \quad (19)$$

4.2.1 Other features of the ScenarioService

Multiple scenarios may have common elements and can also be combined, which results in possible conflicts between the available options of each one. These conflicts are automatically presented to the user in Argue Table, in order to make a decision for these. Thus, normally, as soon as the user enters line 3 of Table 1 then a line not shown in the table will be automatically proposed to the user to select the available options combining the lines 2 and 3 contexts, i.e. $\{family\_time, family\_call, business\_call\}$. ***Web-Gorgias-B*** allows the user to mark this scenario as impossible and from then on it is no longer proposed.

Furthermore, when a user selects a scenario to expand, e.g. S_x^{lvl}, to define the scenario preferences of $S_{x'}^{lvl+1}$ the user must select the conditions that apply (the user is not allowed to select conditions already in x) and, thus, add context to x and define x'. We say that x' is a refinement of x. Then the user is asked to select the valid options among those in O_x^{lvl}.

Web-Gorgias-B allows the user to validate her decision policy using the *Execution* view. In that view, the user selects the appropriate context for the scenario she wants to instantiate by choosing among the facts that she has defined. Then, the user can either test the applicability of all the defined options in that scenario, or just test a specific option. The ***Web-Gorgias-B*** tool transforms the user policy model to a Gorgias theory and executes the relevant Prolog query. Then it shows the results to the user. See the execution of one scenario for our example in Figure 6.

5 Evaluation

A thorough multi-level evaluation was conducted. Evaluation consisted of two stages; initially employing think-aloud evaluation protocol (McDonald and Petrie, 2013) and, lastly, evaluation of the system based on a standard usability scale, in our case, the System Usability Scale (SUS) (Bangor et al., 2009). In both stages, most users were non-experts in logic programming.

In this section we will also include screenshots and material from our working example development process using ***Web-Gorgias-B***, so that the reader can also evaluate this work.

5.1 Think aloud evaluation

Thorough evaluation, both informal and structured, was conducted so that the system's usability was assessed during the system's development. Various user comments were integrated in the user interface design throughout its implementation. 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 (McDonald and Petrie, 2013).

The think aloud protocol involves performing certain sets of actions, while at the same time users express aloud any thoughts in relation to their experience interacting with a system or a user interface. Users are instructed to express aloud potential usability difficulties and, in general, state how they feel while using a system. Observers record the users' interactions by taking notes or capturing videos after relevant permissions have been acquired. System developers use the video to transcribe how users reacted to what they were asked to do, combined with detailed and time-stamped user logs. This way, a complete picture is formed in relation to the functionalities to be developed and improved, as well as on the design and organization of user interface elements.

At a **primary stage** a detailed think-aloud protocol was applied to three users; two expert designers of web sites and a technically competent computer engineering student, all with minimal experience of logic programming and argumentation frameworks. The two expert web designers focused on suggesting simplifications of the user interface, but also complementing it with additional elements when needed (Petrie and Power, 2012). They advised how to enhance the ease of navigating the application and prompted a change of position of the side column which includes the derived results. The designer noticed the lack of a help section and proposed to add at each page a help button containing tips accompanied with simple examples, in order to guide the users when using the application. They influenced the way that the NLP functionality would aid the predicates definition process (see Figure 4).

Lastly, the information and the tabs included in the web application were reported as potentially too many and complex. The solution to this problem was to create two custom views. One for naive users and one for expert users (see the slider that switches between the views in Figure 5 just below the question mark).

When the final system was completed, a thorough final evaluation was also conducted by asking users to browse the web pages in succession. The first evaluation was conducted by a user interface developer with extensive experience in web design and development. His comments moved around the aesthetics of the pages, their colors and functionality.

The second evaluation was conducted by a tech-

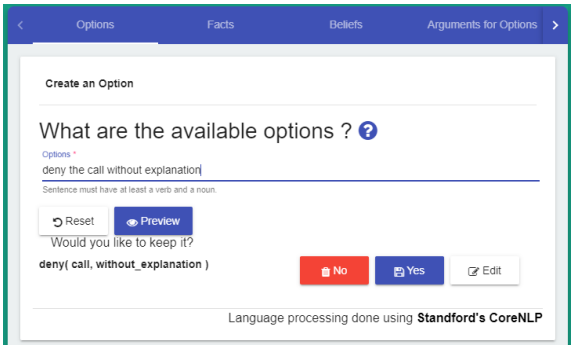

Figure 4: Automatically proposing the predicate structure based on natural language input. The user writes the option in natural language, in this case "deny the call without explanation". Then, by pressing *enter* in the keyboard or the *Preview* button, the system proposes a structure for the predicate, in this case "deny(call, without_explanation)". The user can accept it by pressing the *Yes* button, clear it by pressing *No*, or edit it by pressing *Edit*.

nically proficient undergraduate student who focused on the application when it is run on a mobile phone. He suggested changes in the layout of the pages, so that page content fits the screen size of mobile phone screens. Furthermore, he indicated improvements in relation to the text input forms as shown on a mobile phone's screen, so that it is easier to input text.

The third evaluation was conducted by an expert user in argumentation and the Gorgias framework. His suggestions enhanced the simplicity of the implementation and the effectiveness of its functionality. He observed that in certain cases, the automatically generated scenarios based on the combination of other scenarios that have conflicting options, may have in their context, facts or beliefs that are objectively impossible to happen. To overtake this situation, he proposed to add a button to define such scenarios as impossible and then hide it from the Argue Table View (this view for our working example is presented in Figure 5). Furthermore, it was noted that certain help messages and labels of form inputs were misleading and needed clarity. He also proposed to add floating notifications to all pages, informing users about the progress of their requests.

User interface improvements suggested by all reviewers were applied. Evaluators were invited to reassess the improved usability of the system based on the think-aloud protocol. Evaluators were generally satisfied by the overall usability of the system.

For the interested reader we provide the Gorgias rules that were automatically generated for the example shown in Figure 5 [3]:

```
rule(r1(call), allow(call), []).
rule(r2(call, with_explanation),
  deny(call, with_explanation), []).
rule(r3(call, without_explanation),
  deny(call, without_explanation), []).
rule(p1(call), prefer(
  r1(call),r2(call, with_explanation)), [])
    :-family_time, family_call.
rule(p2(call), prefer(
  r1(call), r3(call, without_explanation)), [])
    :-family_time, family_call.
rule(p3(call), prefer(
  r1(call), r2(call, with_explanation)), [])
    :-family_time, business_call.
rule(p4(call), prefer(
  r1(call), r3(call, without_explanation)), [])
    :-family_time, business_call.
rule(p5(call, without_explanation), prefer(
  r3(call, without_explanation), r1(call)), [])
    :-family_time, business_call.
rule(p6(call, without_explanation), prefer(
  r3(call, without_explanation),
  r2(call, with_explanation)),[])
    :-family_time, business_call.
rule(c1(call, without_explanation), prefer(
  p5(call, without_explanation), p4(call)),[]).
rule(c2(call), prefer(
  p4(call), p5(call, without_explanation)), [])
    :-from_boss_call.
rule(c3(call), prefer(
  c2(call), c1(call, without_explanation)),[]).
```

5.2 Usability evaluation based on the System Usability Scale (SUS)

In order to introduce the innovative features of ***Web-Gorgias-B*** and formally evaluate its interface, we also conducted a usability test to explore usability of the system and potential issues. We employed the **System Usability Scale (SUS)** widely employed in literature for interface evaluation (Brooke, 1996). SUS has become an industry standard with more than 500 systems tested with it (Klug, 2017). When SUS is used, participants are asked to score the following 10 items, selecting one of five responses in a Likert scale that ranges from Strongly Agree to Strongly disagree:

- Q1 I think that I would like to use this system frequently.
- Q2 I found the system unnecessarily complex.
- Q3 I thought the system was easy to use.
- Q4 I think that I would need the support of a technical person to be able to use this system.
- Q5 I found the various functions in this system were well integrated.

[3] Explaining the Gorgias rules syntax is out of the scope of this paper, the interested reader can consult the Gorgias (http://www.cs.ucy.ac.cy/~nkd/gorgias/) or Gorgias-B (http://gorgiasb.tuc.gr/) sites.

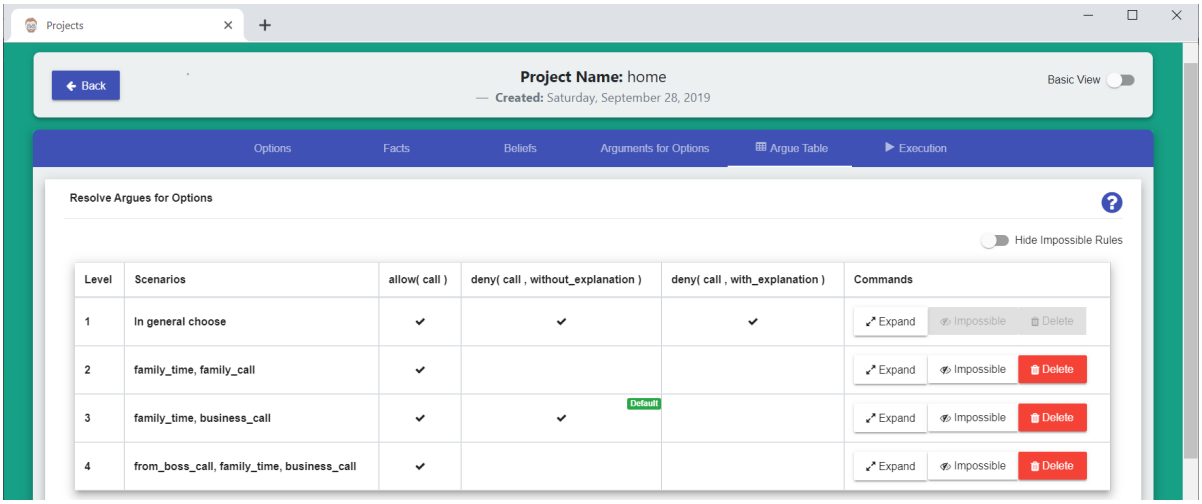

Figure 5: Argue Table View. The check sign indicates valid options and the *Default* indication on the upper right corner of the cell defines the option as default. In the screenshot we see the *Basic View* of the requirements presented in Table 1.

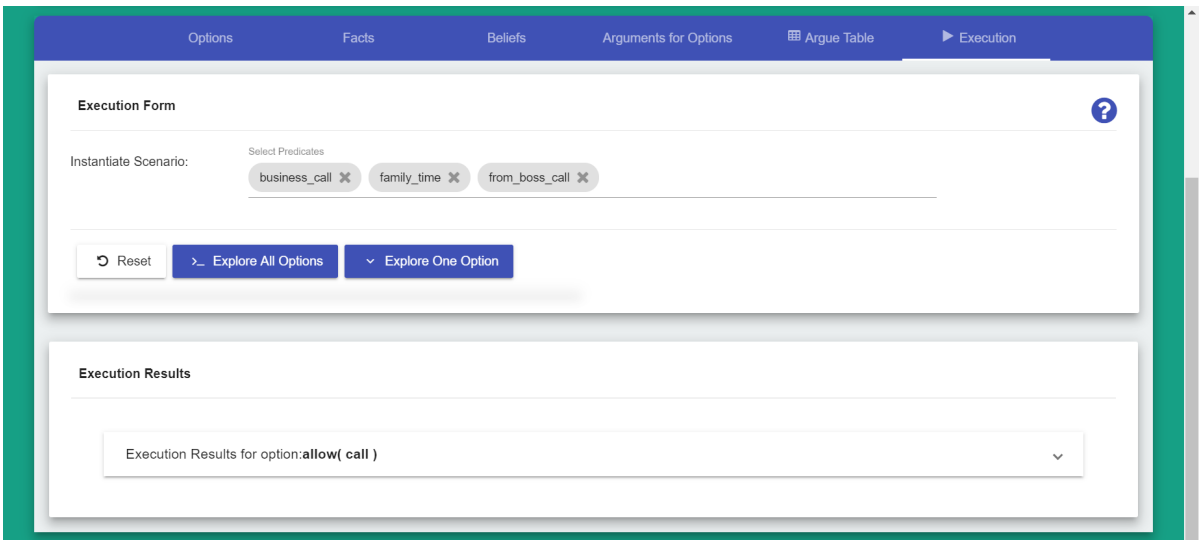

Figure 6: Execution View. The user has added to the simulated scenario three conditions that apply, i.e. *family_time, business_call, from_boss_call* and after clicking the "Explore All Options" button, the only applicable option is to *allow(call)*.

Q6 I thought there was too much inconsistency in this system.

Q7 I would imagine that most people would learn to use this system very quickly.

Q8 I found the system very cumbersome to use.

Q9 I felt very confident using the system.

Q10 I needed to learn a lot of things before I could get going with this system.

We organized the experiment in our laboratory and we invited users associated with our research laboratories at the Technical University of Crete, who were, however, naive in relation to the system's goals. Seven members of our teaching staff, five post-graduate students and three undergraduate students volunteered and participated in this formal usability study. Two of the research staff were AI professors who hadn't used the system but understood the basic principle of argumentation. The remaining participants were not experts in AI, nor was AI their research field. All the participants followed a 10-minute tutorial on the use of ***Web-Gorgias-B*** [4]. Afterwards, they were able to ask any questions they had. Then, we asked them to author another decision policy, of similar difficulty to the one created in the tutorial, us-

[4]The interested reader can follow a video tutorial showing the functionality of the developed system for authoring a simple decision policy: https://youtu.be/T9nBk1h20Xs.

ing ***Web-Gorgias-B*** all by themselves. All the participants successfully authored their policy. The requested decision policy may seem simple but it is a situation occurring in the real world and adopted by other researchers for showing cases of argumentation-based decision making (Kakas and Moraitis, 2006). Moreover, this example includes the usage of all the innovative features of ***Web-Gorgias-B***.

Subsequently, they responded to the above questions using a Likert scale ranging from 1 to 5 with 1 signifying "strongly agree" and 5 "strongly disagree". The relevant statistics are presented in Figure 7. For odd questions the favorable results are towards "1" and in even questions towards "5". It is worth noticing that the boxplot for question 7, regarding how quickly someone can learn the system, has limited negative responses as three quarters of users agree that the system was very easy to learn. Other favourable responses indicated that the system was not unnecessarily complex (Question 2) or cumbersome (Question 8) and overall, proved to be quite consistent (Question 6).

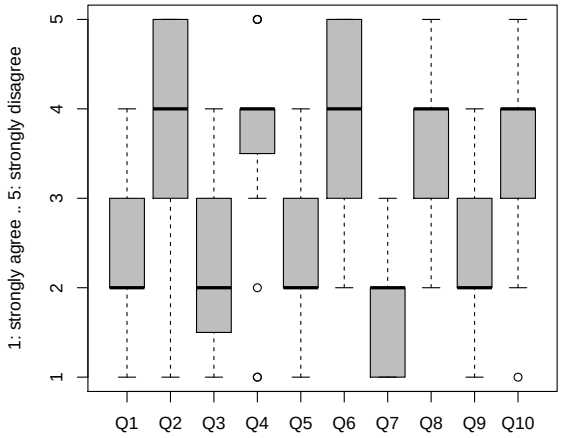

Figure 7: Responses boxplots for the 10 questions of our survey. The boxplots were created using R.

Finally, the results were summed up and the SUS score was calculated as an average of their scores (Brooke, 1996). The statistics for the SUS score are given in the boxplot in Figure 8. The average SUS score from all past studies is 68. A SUS score above a 68 would be considered above average (Klug, 2017), thus, we can support the hypothesis that our ***Web-Gorgias-B*** *system usability is above average* as the mean value of our sample is 69.33 (median: 70, standard deviation: 16.73).

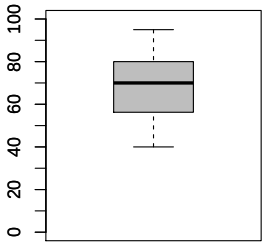

Figure 8: The boxplot of the SUS score of our survey.

6 Conclusion

The objective of this paper was to implement a web-based application for decision policy definition and a simulation application for the Gorgias argumentation framework. The main goal of creating a web interface was accomplished, easily accessible by the general public. The software was designed to incorporate cutting-edge technologies and programming frameworks, handle a large volume of transactions and be compatible with both desktop and mobile devices. The architectural components were described and the functionality of the system was evaluated using the think aloud protocol.

Moreover, we offer, for the first time, an argumentation-based implementation of the table-based requirements gathering formalism that was proposed recently in the literature (Kakas et al., 2019). Additionally, we proposed new features that allow for better expressing the requirements of a decision maker, mainly the ability to define a default option in a scenario. Evaluation showcased that users with mostly no argumentation experience learnt the system quite quickly and mostly thought that it was simple and consistent.

Future work is focused on allowing the user to define options in a scenario that are not present in previously selected scenarios. Moreover, and 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 on the branch of the scenario that she currently refines. This way, Table 5 would, instead, be converted to two tables; one with lines 1 and 2 and one with lines 1, 3 and 4.

REFERENCES

Ali, T., Najem, Z., and Sapiyan, M. (2016). Jpl : Implementation of a prolog system supporting incremental tabulation. In *Sixth International conference on Computer Science and Information Technology (CCSIT 2016), Zurich, Switzerland, January 02-03, 2016*, pages 323–338.

Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

Brooke, J. (1996). SUS-a quick and dirty usability scale. In Jordan, P. W., Thomas, B., Weerdmeester, B., and McClelland, I. L., editors, *Usability Evaluation In Industry*, chapter 21, pages 189–194. CRC Press.

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

Costa, Â., Heras, S., Palanca, J., Jordán, J., Novais, P., and Julian, V. (2017). Using argumentation schemes for a persuasive cognitive assistant system. In Criado Pacheco, N., Carrascosa, C., Osman, N., and Julián Inglada, V., editors, *Multi-Agent Systems and Agreement Technologies*, pages 538–546, Cham. Springer International Publishing.

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

Eysholdt, M. and Behrens, H. (2010). Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309.

Gaertner, D. and Toni, F. (2007). Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems*, 22(6):24–33.

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

Kakas, A. and Moraitis, P. (2006). Adaptive agent negotiation via argumentation. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, page 384–391, New York, NY, USA. Association for Computing Machinery.

Kakas, A. C. and Loizos, M. (2016). Cognitive systems: Argument and cognition. *IEEE Intelligent Informatics Bulletin*, 17(1):14–20.

Kakas, A. C. and Moraitis, P. (2003). Argumentation based decision making for autonomous agents. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 883–890. ACM.

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

Klug, B. (2017). An overview of the system usability scale in library website and system usability testing. *Weave: Journal of Library User Experience*, 1(6).

Lam, H.-P. and Governatori, G. (2009). The Making of SPINdle. In Paschke, A., Governatori, G., and Hall, J., editors, *Proceedings of the 2009 International Symposium on Rule Interchange and Applications (RuleML 2009)*, pages 315–322, Las Vegas, Nevada, USA. Springer-Verlag.

Leff, A. and Rayfield, J. T. (2001). Web-Application Development Using the Model/View/Controller Design Pattern. In *5th International Enterprise Distributed Object Computing Conference (EDOC 2001), 4-7 September 2001, Seattle, WA, USA, Proceedings*, pages 118–127. IEEE.

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

Modgil, S. (2006). Hierarchical argumentation. In *European Workshop on Logics in Artificial Intelligence*, pages 319–332. Springer.

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

Petrie, H. and Power, C. (2012). What do users really care about?: a comparison of usability problems found by users and experts on highly interactive websites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2107–2116. ACM.

Pison, A. (2017). Développement d'un outil d'aide au triage des patients aux urgences ophtalmologiques par l'infirmière d'accueil à l'aide du système d'argumentation Gorgias-B. Technical report, LIPADE, Paris Descartes University.

Snaith, M. and Reed, C. (2012). TOAST: online aspic$^+$ implementation. In *Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012*, pages 509–510.

Spanoudakis, N., Kostis, K., and Mania, K. (2020). Argumentation for all. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20)*, pages 980–982, New York, NY, USA. Association for Computing Machinery.

Spanoudakis, N. I., Constantinou, E., Koumi, A., and Kakas, A. C. (2017). Modeling data access legislation with gorgias. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 317–327. Springer.

Spanoudakis, N. I., Kakas, A. C., and Moraitis, P. (2016). Gorgias-B: Argumentation in Practice. In *Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016.*, pages 477–478.

Van Eemeren, F. H., Grootendorst, R., and Eemeren, F. H. (2004). *A systematic theory of argumentation: The pragma-dialectical approach*, volume 14. Cambridge University Press.