Towards Multipolicy Argumentation

Nick Bassiliades
Department of Informatics, Aristotle University of Thessaloniki, Greece
nbassili@csd.auth.gr

Nikolaos I. Spanoudakis
School of Production Engineering and Management, Technical University of Crete, Greece
nispanoudakis@isc.tuc.gr

Antonis C. Kakas
Department of Computer Science, University of Cyprus, Cyprus
antonis@cs.ucy.ac.cy

ABSTRACT

In this paper, we develop a novel computational argumentation framework for resolving conflicts that arise in a community of multiple stakeholders where each one of them bears a private policy/strategy for shared and inter-related decisions. Decisions taken individually by stakeholders can be contradicting, so there is a need for an arbitration service that will resolve the conflict and conclude on a single decision. Centralized mediation approaches gather all relevant context information and decide on the prevailing decision option as suggested individually by multiple stakeholders. There is high complexity on resolving all possible competing option conflicts among all competing stakeholders, thus, usually centralized solutions do not scale. Our approach avoids this complexity because it is based on defining an arbitration meta-policy for deciding on the priorities among stakeholders, which are few, and not among competing decisions of stakeholders. Then, this meta-policy is automatically rewritten into a full meta-policy about conflicting options, but without user intervention. Thus, human arbitrators can seamlessly define their arbitration meta-policies without a heavy cognitive load.

CCS CONCEPTS

• **Computing methodologies** → Nonmonotonic, default reasoning and belief revision • **Information systems** → Decision support systems

KEYWORDS

Computational argumentation, conflict resolution, multi-stakeholder decision making

SETN '18, July 9–15, 2018, Rio Patras, Greece

ACM ISBN 978-1-4503-6433-1/18/07...$15.00
https://doi.org/10.1145/3200947.3201032

ACM Reference format:

N. Bassiliades, N.I. Spanoudakis, A.C. Kakas. 2018. In *Proceedings of 10th Hellenic Conference on Artificial Intelligence, Rio Patras, Greece, July 2018 (SETN'18)*, 10 pages. https://doi.org/10.1145/3200947.3201032

1 INTRODUCTION

Computational argumentation has been recognized as a solid theoretical framework to model human decision making [1], [5]. Existing approaches deal with various aspects of decision making, such as uncertainty or multi-criteria, but the problem of modeling shared decisions among individual stakeholders with possibly conflicting options for the decision outcome has not been given enough attention. Moreover, defeasible reasoning has been applied in smart contracts [4], which can be captured as logic statements. A smart contract is a piece of software that captures the contents of a contractual agreement and allows for implementing it at run time. In [4], the authors explain the added value of using logic-based smart contracts as their nature makes them suitable for negotiation, enforcement and monitoring but also, importantly, for dispute resolution between different parts of the contract that may come from different stakeholders.

In this paper, we consider cases where argumentation is used at different levels of decision making, however, these must be considered together, either because there is a conflict of interest of stakeholders and they want to argue, or because there are diverse opinions within an organization and a higher (executive) authority has the final say on the decision policy.

One such example would be when people request access to personal or medical data. There are diverse organizations (e.g. hospitals, insurance companies, national health care systems) that can give access to data and each of them has its own business model that might conflict to each other. For example, in the case of a hospital, access to a patient's personal information is only allowed for treating medical doctors, and, even in that case, under strict circumstances. However, a fireman might request access to such data as he is at an accident scene and the victim is unconscious. How can these conflicts in decision making among conflicting stakeholders be resolved? In these cases, the stakeholders want to argue their case. And when this happens, they all reference relevant legislation. The latter can be used as a meta-policy over these business models to regulate them.

Another example could be an energy management scenario in a smart building, where various stakeholders would like to manage the energy from their own perspective. An individual user

would strive for comfort, turning on/off the cooler/heater beyond a temperature threshold. The energy manager of the building would strive for energy savings, by turning off devices like coolers, when there is no apparent need for using them, e.g. nobody is the room, it's too late, etc. Finally, the safety officer of the building would strive for safety of people and equipment by turning off devices when there is an emergency in a room, such as fire. In this scenario, each of three types of stakeholder defines their own policy, independently from the rest. These stakeholders might not even know each other – so they could not coordinate their policies development.

In this paper, we present a novel argumentation-based framework for handling such situations. Our framework proposes the use of a meta-policy for defining priorities among stakeholders. These priorities arise naturally or legally in a society of self-interested agents. Priorities may be strict, or they may depend on the situation context. Our multipolicy framework can capture all such cases, by extending the "Gorgias" argumentation framework [6], [14] for representing stakeholders' priorities in an arbitrator's metapolicy.

Our approach can be considered as a centralized mediation approach, without having, though, the scaling problems of existing similar approaches, which have a high complexity on resolving all possible competing option conflicts among all competing stakeholders. Our approach avoids this complexity because it is based on defining an arbitration meta-policy for deciding on the priorities among stakeholders, which are few, and not among the competing decisions of stakeholders. Then, this meta-policy is automatically rewritten into a full meta-policy about conflicting options, but without user intervention.

Thus, human arbitrators can seamlessly define their arbitration meta-policies without a heavy cognitive load. Another benefit is that individual stakeholder policies can be kept and maintained privately, without needing to re-organize the arbitrator's meta-policy. In doing so, we may give up the possibility to define fine-grained contextual priorities among specific competing options of specific stakeholders. However, in practice, such fine-grained priorities are rarely a need in cases that independent stakeholders define their decision-making privately and do not have access to the decision-making mechanism of each other, relying on an arbitrator to resolve the conflict based on norms and laws about stakeholders' authority. Nevertheless, an advanced arbitrator user can still manually edit the final translated meta-policy for fine-grained refinements.

Finally, our implementation in the Gorgias prolog-based argumentation system presents full explanations about the argumentation process if needed (both for private and global decision-making levels).

In the rest of the paper, we briefly review related work on resolving conflicts using argumentation through an arbitrator in section 2; we present background information on our argumentation framework and methodology in section 3; we describe in detail the conceptual framework and the methodology of the proposed approach in sections 4 and 5 and we present two case studies in sections 6 and 7. Finally, we conclude the paper in section 8, setting our future research goals.

2 Related work

Oguego et al. [11] use weighted contexts to compute the weight of an argument in order to resolve conflicts between policies. This approach, however, seems quite limited, as in their example they only put weight in one issue (i.e. television program types). It is not clear how this issue would be weighed against another, more or less serious one. Likewise, it would be difficult to explain a combined situation to the user.

Howlett [3] proposed a multi-level nested model of policy instrument choice and policy design. He argues on that policy design passes through different levels where each one considers arguments of a different nature. Such a situation calls for arguing at different levels for the application of a policy.

Karafili and Lupu [8] used argumentation for representing the rules of data sharing agreements and for analyzing them to identify conflicts. For example, it is possible that a nation-wide agreement is applied, but when the international dimension comes in, national issues are superseded by higher level authority levels, considering different contexts (international agreements) and roles of the person it is about (tourist or victim).

Our framework can be seen as an instance of the abstract Hierarchical Argumentation Framework proposed by Modgil [10], abstracting from earlier hierarchical structured frameworks ([6], [12]), where arguments in a level n argumentation framework resolve conflicts between arguments of a framework at level $n-1$, a process called meta-argumentation. Our implemented framework brings this abstract framework to practice, extending it also, since we can have several argumentation frameworks at level *n-1*.

This work focuses on arbitration, i.e. a settlement process where a third party gathers the arguments of the disputing parties and renders a decision. It should not be confused with mediation or multi-party facilitation, where a third-party assists disputing parties to reach an agreement, possibly through a negotiation process [18], [12]. Works in Group, or Collaborative Decision Support Systems are also not sufficiently covering this domain as typically all participants can evaluate the arguments of their counterparts and assign priorities between them and their arguments [7]. This paper is mainly concerned with cases where the conflict resolution process needs to be resolved automatically as in the cases of Data Sharing Agreements or automation decisions in IoT environments.

3 Background

The background of this work is the Gorgias argumentation framework [6] and the SoDA methodology [14].

3.1 The Gorgias Argumentation Framework

Argumentation technology caters for situations where systems need to support decision making under complex preference policies that consider diverse factors [2]. We can abstractly define argumentation as the mechanism that allows the interaction of different, possibly conflicting, arguments and provides the semantics for resolving conflicts. The conclusion supported by the

arguments can be an action to take, the validity of a proposition, e.g. to validate or not a data access request.

A hierarchical argumentation framework uses object level arguments for representing the decision policies and then priority arguments for expressing preferences on the object level arguments to resolve possible conflicts. Additional priority arguments can be employed for resolving conflicts between priority arguments of a previous level. One such framework is Gorgias [6], which has been successfully applied in different applications, e.g. recently, for developing a theory capturing the Cyprus law for medical data access [16], for defining data access policies [8] and for conflict resolution [15]. We briefly provide some formal definitions from [6]:

Definition 1. A **theory** is a pair (T, P) whose sentences are formulae in the **background monotonic logic** $(L, \vdash)$ of the form $L \leftarrow L_1,\ldots,L_n$, where L, L_1, ..., L_n are positive or negative ground literals. For rules in P the head L refers to an (irreflexive) higher priority relation, i.e. L has the general form $L = h\_p(rule1, rule2)$. The derivability relation, $\vdash$, of the background logic is given by the simple inference rule of modus ponens.

An **argument** for a literal L in a theory (T, P) is any subset, Arg, of this theory that derives L, $Arg \vdash L$, under the background logic. A part of the theory $T_0 \subset T$, is the **background theory** that contains facts and non-defeasible rules that always apply. An argument attacks another when they derive a contradictory conclusion. These two arguments are considered as **conflicting arguments**. A conflicting argument (from T) is admissible if it counter-attacks all the arguments that attack it. It counter-attacks an argument if it can use priority arguments (from P) and make itself at least as strong as a conflicting argument.

Definition 2. An agent's **argumentative policy theory** is a theory $APT = ((T, T_0), P_R, P_C)$ where T contains the **argument** rules in the form of definite Horn logic rules, P_R contains **priority** rules which are also definite Horn rules with head $h\_p(r_1, r_2)$ s.t. $r_1, r_2 \in T$ and all rules in P_C are also priority rules with head $h\_p(R_1, R_2)$ s.t. $R_1, R_2 \in P_R \cup P_C$. T_0 contains auxiliary rules of the agent's background knowledge.

All in all, we specify rules in three different levels for defining a decision-making theory. The first level (or object-level) rules (T) refer directly to the subject domain and reflect the background knowledge needed for reaching the different goals. The second level rules define priorities over the first level rules, resolving possible conflicts. These situations usually reflect the needs of a role that the decision maker assumes or a context in which he finds himself, usually also including a default context. The third level (and also higher level) rules define priorities over the rules of the previous level but also over the rules of this level to define specific contexts, which can be specializations of the previous level contexts or their combinations.

3.2 The SoDA methodology

The SoDA methodology [14] has been developed to allow for hierarchical argumentation framework-based software development. SoDA defines the steps that the modeler should follow to define the decision policy leading from options definition to code generation. SoDA defines the following tasks:

T1. Defining the different available options.
T2. Identifying the knowledge needed to describe the application environment.
T3. Knowledge is distinguished in information that always exists for all instances of the problem and information that is circumstantial, i.e. which may be present in all instances of the problem.
T4. Sorts circumstantial information from more general to more specific contexts, starting from level one (more general contexts). Independent contexts (i.e. when the one is not a refinement of the other) can appear at the same level.
T5. Defines for each option, O_i, the different problem environments, i.e. the sets of preconditions, C_i, in terms of non-circumstantial predicates, where the option is possible.
T6. Iteratively defines sequences of increasingly more specific scenarios of the world and considers how options might win over others. This starts with the information from T5 to precondition the world and iterates, getting each time the next level of circumstantial information. At each level of iteration, it defines which option is stronger over another under the more specific contextual information. In the final iteration, the winning options (if they exist) for each partial model are defined without extra information.

Since its conception in 2016 [14], SoDA has been applied for developing argumentation theories in diverse application domains, i.e., for capturing legislation [16], for defining a decision policy [8] and for conflict resolution [15].

4 Conceptual Description of Framework

Our framework assumes the following stakeholders:

- Multiple peer stakeholders S_i that decide among k competing options O_j and suggest their decisions to an arbitrator A, according to their own private policies APT_i.
- An arbitrator (or administrator) A that decides among k competing options O_j, suggested independently by n competing peer stakeholders S_i with different levels of authority or preference. The level of authority for each stakeholder is defined by law or by the administrator preferences and is realized through the administrator's meta-policy MP.

Formally, our framework is defined by the tuple $<A, S, O>$, where A is the arbitrator, $S = \{S_i \mid i \in [1..n]\}$ is the set of peer stakeholders and $O = \{O_j \mid j \in [1..k]\}$ is the set of competing options. Each peer stakeholder S_i is defined by the tuple $<O, APT_i>$, where APT_i is a private argumentative policy theory, as in Definition 2, and the object-level argument rules refer to O_j in their conclusions, i.e. $\forall r \in T(APT_i)$, $r \equiv (\text{cond}(r) \rightarrow O_j)$. Each stakeholder decides independently which among the k different possible options O_j to choose. We assume that options are common to all peer stakeholders (or they can be made compatible through rewriting rules).

The arbitrator A is given by the tuple $<S, MP>$, where $S_i \in S$ are considered as mutually conflicting meta-options for each of the peer stakeholders. The MP (called *meta-policy*) is an argu-

mentative policy theory where the object-level argument rules refer to S_i in their conclusions, i.e. $\forall\ r \in \mathrm{T}(MP),\ r \equiv \mathrm{cond}(r) \longrightarrow S_i$. Note that it is conceptually easier to define authoritative priorities on stakeholders alone, rather than on multiple options that each stakeholder might have. The private policies may have default priorities for the administrator, e.g. the energy saving policy is always preferable over the individual user's room comfort policy, or priorities may be conditional, e.g. the previous default priority is inversed when the user is the IT manager defining the server's room energy policy. The administrator is not interested at the conditions under which private policies become valid, assuming that they all compete for the right to decide on the competing options among the competing peer stakeholders.

Since the ultimate goal of the arbitrator is to decide on the action (option) to be performed, the meta-policy *MP* on competing policies can be automatically transformed, transparently to the end user, into an argumentation theory *MP'* on the $n \times k$ competing action options O_{ij} suggested by the peer stakeholders S_i,. This is achieved by: a) transforming the k O_j options of the n private policies into $n \times k$ O_{ij} options by having an extra parameter indicating the stakeholder, b) having one object-level rule for each of the competing options, and c) combining all preference rules of the meta-policy theory *MP* with all competing options O_j.

Formally, the initial arbitrator tuple is converted into the tuple $<O', MP'>$, where $O' = \{\ O_{ji} \mid j \in [1..k],\ i \in [1..n]\ \}$ are $n \times k$ conflicting options, i.e. $\forall\ l \neq m\ \forall\ w \neq z\ O_{wl} \wedge O_{zm} \vdash \bot$, and *MP'* is an argumentative policy theory where the argument rules refer to O_{ji} in their conclusions. *MP'* is constructed as follows:

- Each O_{ji} option suggested by stakeholder S_i is renamed to O^{local}_{ji}.
- The object-level rules of the argumentative policy theory T(*MP'*) comprise of k mapping rules $O^{local}_{ji} \longrightarrow O_{ji}$ that map stakeholders' decisions to arbitrator's decision.
- The first-level preference rules of the *MP'* theory $P_R(MP')$ are constructed by combining all first-level preference rules of the *MP* theory with all competing options O_j of the framework, as follows:

$$\forall\, h\_p(r_i, r_k) \in P_R(MP) \wedge \forall\ O_j, O_{j'} \in O \wedge O_j \neq O_{j'} \longrightarrow \exists\ h\_p(r_{jik}, r_{j'ik}) \in P_R(MP')$$

  where r_i, r_k are argument rules in the *MP* (r_i, $r_k \in \mathrm{T}(MP)$) for stakeholders S_i, S_k, respectively.

- The second-level preference rules of the *MP'* theory $P_C(MP')$ are constructed by combining all second-level preference rules of the *MP* theory with the corresponding but competing first-level rules of the *MP'* theory (i.e. first-level preference rules that give contradicting preferences to object-level rules), as follows:

$$\forall\ h\_p(h\_p(r_i, r_k), h\_p(r_k, r_i)) \in P_C(MP) \wedge \forall\ h\_p(r_{jik}, r_{j'ik}) \in P_R(MP') \longrightarrow \exists\ h\_p(h\_p(r_{jik}, r_{j'ik}), h\_p(r_{jki}, r_{j'ki})) \in P_C(MP')$$

Thus, for each policy preference rule of the meta-policy theory *MP* a number of preference rules (all 2-permutations of the k conflicting options) will be constructed at the transformed theory *MP'*. Therefore, if the initial *MP* meta-policy has R preference rules, the transformed *MP'* meta-policy will have $R \cdot P(k,2) = R \cdot \frac{k!}{(k-2)!}$ rules. So, even in the simplest case where k=2, the transformed *MP'* meta-policy will have double the size of the preference rules of the original meta-policy *MP*.

5 Methodological Approach

Each stakeholder defines their own private policy, using the SoDA methodology [14] and the Gorgias argumentation framework [6], independently and privately from the other stakeholders (including the arbitrator), using as input knowledge any belief about the environment or background knowledge they have. This is also valid for the arbitrator who is omniscient and has access to all data – although he may be unaware about the details of a sub-policy. The only thing that the stakeholders must agree upon is the naming of the competing options and the exchange language for the options, using one of the following methodologies. The difference between them lies in who defines the competing options; the arbitrator (top-down) or the stakeholders (bottom-up).

5.1 Top-down methodology

The administrator *A* uses the SoDA methodology in order to define the meta-policy *MP* for deciding among competing peer stakeholders S_i. Thus, in task T1 of the SoDA methodology the administrator decides on the competing options / stakeholders S_i. The input knowledge for the meta-policy (task T2) are: a) the stakeholders themselves and b) the union of the knowledge of all the private stakeholders' policies. As stated above, the administrator is omniscient and, therefore, has access to all the knowledge that the peer stakeholders hold. The rest of the steps in the SoDA methodology are followed as usual.

The actual decision to be taken involves not just the winning stakeholder, but also the action (option) to be performed. Therefore, as stated above, the *MP* meta-policy is converted into the *MP'* meta-policy which dictates which option O_j suggested by a stakeholder S_i will be selected. Here, we remind that the options are supposed to be common / compatible among all stakeholders, including the administrator. In the top-down methodology, the administrator defines the options O_j and informs the peer stakeholders, in an enhanced version of the task T1 of the SoDA methodology. The competing options O_j are augmented with an extra argument that indicates the stakeholder S_i that suggests an option O_{ji}. The meta-policy uses the "local" decisions, suggested by each stakeholder S_i to the administrator as input knowledge. Actually, in order not to confuse the local stakeholder option predicate with the global administrator option predicate, the local options are named as O^{local}_{ji} and there exist mapping rules as explained in Section 4.

The administrator then dictates to each stakeholder S_i the alternative options O^{local}_{ji} for which the stakeholder needs to decide. Then, each stakeholder S_i defines its own policy APT_i for deciding on the competing options O^{local}_{ji} using the SoDA methodology. To ease policy authoring, each peer stakeholder defines their private policy APT_i using the options O_j, as if he/she was

the only stakeholder to decide, and then our multi-policy framework automatically transforms the APT_i theory into one whose options are expressed as O^{local}_{ji}, where i is a constant for each stakeholder S_i.

5.2 Bottom-up methodology

Each peer stakeholder S_i defines its own private policy APT_i for deciding on the j competing options O_j using the SoDA methodology. Notice that there must be a global agreement between the peer stakeholders on the naming of the competing options which must be common to all, according to the current implementation of our multi-policy framework. However, this is not an intrinsic feature of our framework, since mapping rules at the administrator's meta-policy *MP'* (see above) can take care of converting all local options to common global ones.

Subsequently, peer stakeholders report to arbitrator A their competing options O_j, which are automatically transformed into O^{local}_{ji}, adding one extra argument for the name of the stakeholder. The administrator uses the SoDA methodology to define the meta-policy *MP'* for deciding among the competing options O^{local}_{ji} suggested by the peer stakeholders. Actually, in order to ease policy authoring, the administrator defines the *MP* meta-policy for deciding among competing peer stakeholders S_i, using the level of authority defined by the law or its own preferences, including special circumstance exceptions. During task T1 of the SoDA methodology the administrator decides on the competing options / stakeholders S_i. The input knowledge for the meta-policy (task T2) is: a) the stakeholders S_i, and b) the union of the background knowledge (T_0) of all the private policies. As stated above, the administrator is omniscient and, therefore, has access to all the knowledge that the peer stakeholders hold. The rest of SoDA methodology is followed as usual.

As it is for the case of the top-down methodology, the *MP* meta-policy is automatically converted into the *MP'* meta-policy which dictates which option O_{ji} suggested from a stakeholder S_i will be selected. As explained in Section 4, there exist mapping rules between O^{local}_{ji} and O_{ji}.

6 Energy Management Case Study

In this section, we apply the top-down methodology of our approach to an energy management case study in a Smart Building, adopted from the work of Stavropoulos et al. [17]. We assume that various peer stakeholders S_i would like to manage the consumed energy from their own perspective. The decision is to control (switch on or off) a certain device (e.g. a cooler) in a specific room. In our specific example, $S = \{S_{prs}, S_{mgm}, S_{emr}\}$:

- S_{prs}: An individual user would strive for their own comfort, turning on/off the cooler/heater beyond a temperature threshold.
- S_{mgm}: An energy manager would strive for energy savings. Their policy is to turn off devices like coolers, when there is no apparent need using them, e.g. nobody is the room, it's too late, etc.
- S_{emr}: A safety officer would strive for safety of people and equipment. Their policy is to turn off devices when there is an emergency in a room, such as fire.

Furthermore, there is a higher-level authority / administrator of the building, i.e. the building owner or the general manager of the organization, that receives possibly competing peer stakeholders' decisions and must decide which decision will be applied to control various devices on various rooms, based on a meta-policy dictated by the level of authority of each stakeholder and / or based on contextual information.

Meta-policy

According to the top-down methodology, the administrator defines two competing options O_j, namely `switchON(Dev,Room)`, and `switchOFF(Dev,Room)`, with the obvious meaning. Then, the administrator should define the *MP* meta-policy on the competing stakeholders S_i. In our specific example, we have chosen to give the following default strict ordering: S_{emr} stakeholder is superior to the S_{mgm} stakeholder, which is superior to the S_{prs} stakeholder. The only exception is when the room to be controlled is the Server room and the device to be controlled is the cooling device (e.g. you do not ever turn off the server room's cooler for energy saving reasons, except when there is an emergency). In this case, the S_{prs} stakeholder (*personal* meaning the person who oversees the server room, e.g. the IT manager of the organization) is superior to the S_{mgm} stakeholder, but still inferior to the S_{emr} stakeholder. The default superiority relationships among stakeholders are captured by the following meta-policy object-level rules r1-3 and first-level priority rules p1-p3:

```
r1(prs): stakeholder(prs) ←
r2(mgm): stakeholder(mgm) ←
r3(emr): stakeholder(emr) ←
p1(emr): prefer(r3(emr), r1(prs)) ←
p2(emr): prefer(r3(emr), r2(mgm)) ←
p3(mgm): prefer(r2(mgm), r1(prs)) ←
```

Object-level rules are just defeasible facts (body-less rules) that conclude every potential winning stakeholder. The first-level priority rules (P_R) give the default priority between the stakeholders, e.g. p1 states that the emergency stakeholders is preferable (superior) to the personal stakeholders, and so on so forth. The exception of the default priority is given by the following two rules:

```
p4(prs): prefer(r1(prs), r2(mgm)) ←
                device(cooler, srvroom).
c1(prs): prefer(p4(prs), p3(mgm)) ←
```

Priority rule p4 is a first-level priority rule, that gives preference of the personal stakeholder over the management stakeholder when the device is a cooler in the server room. Priority rules p3 and p4 are in conflict now because they give the exactly opposite preferences, therefore there is a need to resolve this conflict at a second-level (P_C), with priority rule c1 which gives preference to priority rule p4 over p3, thus to personal over management stakeholder, when the special condition is met.

Then, the *MP* meta-policy is converted into the *MP'* meta-policy which dictates which option O_j suggested from a stakeholder policy APT_i will be selected. According to our methodolo-

gy, the above options are augmented by an extra argument that indicates the stakeholder S_i (represented by the variable parameter S) that suggests a certain decision: `switchON(Dev, Rm, S)`, `switchOFF(Dev, Rm, S)`. These decisions are input knowledge (beliefs) to the meta-policy so they are renamed as `switchON_local(Dev, Rm, S)`, `switchOFF_local(Dev, Rm, S)` to avoid confusion with the "global" decisions. Furthermore, there exist 2 mapping rules between the suggested "local" decisions from the peer stakeholders and the meta-policy "global" options, which play the role of the object-level rules for the converted meta-policy *MP'*:

```
r1(Rm, Dev, S): switchOFF(Rm, Dev, S) ←
                switchOFF_local(Rm, Dev, S).
r2(Rm, Dev, S): switchON(Rm, Dev, S) ←
                switchON_local(Rm, Dev, S).
```

Notice that the competing options $O_j(S_i)$ can be between different stakeholders, therefore the competing options facts use different variables for the competing options:

```
complement(switchOFF(Rm, Dev, S2),
                switchON(Rm, Dev, S1)).
complement(switchON(Rm, Dev, S1),
                switchOFF(Rm, Dev, S2)).
```

Consequently, all first-level preference rules of the *MP* meta-policy are combined with all competing options O_j. For example, rule `p2` of *MP* should be combined with both competing options `switchON/3` and `switchOFF/3`, so that the emergency policy options will be preferred over the competing management policy options, i.e. `switchON(Dev, Rm, emr)` is preferred over `switchOFF(Dev, Rm, mgm)` and `switchOFF(Dev, Rm, emr)` is preferred over `switchON(Dev,Rm, mgm)`. This is achieved by the following pair of first-level preference rules that give preferences over the object-level rules `r1` and `r2`:

```
p211(emr, Rm, Dev, mgm):
       prefer(r1(Rm,Dev,emr),r2(Rm,Dev,mgm)) ←
p221(emr, Rm, Dev, mgm):
       prefer(r2(Rm,Dev,emr),r1(Rm,Dev,mgm)) ←
```

Similar rules are generated for every first-level preference rule of the *MP* meta-policy. If there are more options, then more preference rules should exist, for all pairwise combinations of preferences between all object-level rules, both ways.

Finally, for each second-level preference rule of the meta-policy theory *MP*, similar multiple second-level preferences rules are generated in the *MP'* meta-policy, resolving the conflicts between all pairwise combinations of first-level preference rules, both ways. For example, rule `c1` in the MP meta-policy resolves the conflict between the default preference of management - personal stakeholders (rule `p3`) and the server room exception (rule `p4`), giving priority to the exception. This means that rule `c1` will be converted to two second-level preference rules in the *MP'* policy, since there are two options at the object-level:

```
c111(prs, Rm, Dev, mgm):
       prefer(p411(prs,Rm,Dev,mgm),
              p321(mgm, Rm, Dev, prs)) ←
c121(prs, Rm, Dev, mgm):
       prefer(p421(prs, Rm, Dev, mgm),
              p311(mgm, Rm, Dev, prs)) ←
```

Notice that in this case the first-level preference rules in conflict (e.g. `p421` and `p311`) give priorities to conflicting options (`p411` gives priority to `r2` over `r1`, when conflicting stakeholders are personal and management respectively, whereas `p311` gives opposite priority to `r1` over `r2`, for the same pair of stakeholders.

```
p311(mgm, Rm, Dev, prs):
       prefer(r1(Rm,Dev,mgm), r2(Rm,Dev,prs)) ←
p321(mgm, Rm, Dev, prs):
       prefer(r2(Rm,Dev,mgm), r1(Rm,Dev,prs)) ←
p411(prs, Rm, Dev, mgm):
       prefer(r1(Rm,Dev,prs), r2(Rm,Dev,mgm)) ←
              device(Dev, Rm),
              Dev=cooler, Rm=srvroom.
p421(prs, Rm, Dev, mgm):
       prefer(r2(Rm,Dev,prs), r1(Rm,Dev,mgm)) ←
              device(Dev, Rm),
              Dev=cooler, Rm=srvroom.
```

When more options exist at the object-level, this leads to more preference rules at the first-level and at the second-level, as well, since all conflicting first-level preference rules should be combined pairwise to resolve their conflict at the second-level.

Then, the administrator informs the three peer stakeholders the alternative options `switchON(Rm, Dev)`, `switchOFF(Rm, Dev)`, for which each peer stakeholder needs to decide. Notice that the administrator needs as input the above decisions from each stakeholder, but in order not to be confused, an extra argument indicating the stakeholder that suggests the decision augments the above predicates: `switchON(Rm, Dev, S)`, `switchOFF(Rm, Dev, S)`. For each stakeholder, the `S` parameter is a constant. For example, the options for the S_{mgm} stakeholder are: `switchON(Rm, Dev, mgm)`, and `switchOFF(Rm, Dev, mgm)`. However, peer stakeholders do not need to define that extra argument, because our framework adds it automatically, by converting the initial private policies. Each stakeholder defines their own policy for deciding on the above competing options using the SoDA methodology.

Personal Policy

In the personal policy, the stakeholder wants to switch on the cooler device in the room when the temperature is high. According to their preferences, the temperature is high when it is more than 28 degrees. Otherwise, they want the cooler turned off.

The definition of the policy in Gorgias notation follows:

```
r1(Rm): neg(tempHigh(Rm)) ← tmprtr(Rm, T), 28>=T.
r2(Rm): tempHigh(Rm)      ← tmprtr(Rm, T), 28<T.
r3(Rm, Dev): switchON(Rm, Dev) ←
              tempHigh(Rm),
              device(Dev, Rm), Dev=cooler.
r4(Rm, Dev): switchOFF(Rm, Dev) ←
              neg(tempHigh(Rm)),
              device(Dev, Rm), Dev=cooler.
complement(switchON(Rm,Dev), switchOFF(Rm,Dev)).
complement(switchOFF(Rm,Dev), switchON(Rm,Dev)).
```

Notice that in the above policy definition the extra "Stakeholder" argument is not used. Thus, in our framework the stakeholders define their own private policy using simplified predicates and rules. However, when the individual policies are merged our framework performs the following source code transformations automatically:

- Adds a bound "Stakeholder" argument to the options of a policy, to the rules that infer the options, and to the preference rules that resolve conflicts among options up to all levels.

- Renames the rules and the options of the policies in order not to be confused when loaded into a single Prolog KB.

An example of the transformations performed is:

```
prs_r1(Rm): neg(tempHigh(Rm)) ←
              tmprtr(Rm, T), 28>=T.
prs_r2(Rm): tempHigh(Rm) ← tmprtr(Rm, T), 28<T.
prs_r3(Rm,Dev,prs): switchON_local(Rm,Dev,prs) ←
              tempHigh(Rm),
              device(Dev, Rm), Dev=cooler.
prs_r4(Rm,Dev,prs): switchOFF_local(Rm,Dev,prs) ←
              neg(tempHigh(Rm)),
              device(Dev, Rm), Dev=cooler.
complement(switchON_local(Rm, Dev, prs),
              switchOFF_local(Rm, Dev, prs)).
complement(switchOFF_local(Rm, Dev, prs),
              switchON_local(Rm, Dev, prs)).
```

Management Policy

In the energy manager's policy, the manager wants the cooler device turned off when the room is inferred to be at an energy saving mode or when there is nobody in the room. The room is in saving mode when it is late in the night (after 10 o'clock) or when the consumption exceeds 2KWatts/hr. However, when there is somebody in the room or when there is a PC operating in the room it seems that the cooler is still needed so there is no need to switch it off. Since there are conflicting options in this policy there are preference rules that dictate that energy saving is more important than an operating PC, but when a user is inside the room then this is more important than energy saving. However, even if the room is not in a saving mode, when there is no motion in the room, energy saving is preferred even in the presence of an operating PC. The manager's policy is defined as:

```
r1(Rm): savingMode(Rm) ←
              consumption(Rm,C), 2000<C.
r2(Rm, Dev): switchOFF(Rm,Dev) ← neg(motion(Rm)),
              device(Dev,Rm), Dev=cooler.
r3(Rm, Dev): switchOFF(Rm,Dev) ← savingMode(Rm),
              device(Dev,Rm), Dev=cooler.
r4(Rm, Dev): neg(switchOFF(Rm,Dev)) ← motion(Rm),
              device(Dev,Rm), Dev=cooler.
r5(Rm, Dev): neg(switchOFF(Rm, Dev)) ←
              device(Dev,Rm), Dev=cooler,
              device(Dev1,Rm), Dev1=pc.
r6(Rm): savingMode(Rm) ←
              mytime(T), 2200<T, room(Rm).
p1(Rm, Dev): prefer(r2(Rm, Dev), r5(Rm, Dev)) ←
p2(Rm, Dev): prefer(r4(Rm, Dev), r3(Rm, Dev)) ←
p3(Rm, Dev): prefer(r3(Rm, Dev), r5(Rm, Dev)) ←
abducible(neg(motion(Rm)), []).
abducible(motion(Rm), []).
```

Notice that the `motion` fact is an abducible one, meaning that when there is no evidence if there is motion or not, the system can still hypothesize on both facts and come to alternative decisions based on those. Of course, if there is hard evidence on motion in the room, based on motion detectors that are part of Internet-of-Things, then this can be turned into a hard fact.

Emergency Policy

In the emergency policy, when there is smoke, or high levels of CO_2 in a room, then there is an alert that triggers an alarm and dictates to switch off the cooler device.

```
r1(Rm): alert(Rm) ← smoke(Rm).
r2(Rm): alert(Rm) ← highCO2(Rm).
r3(Rm, Dev): switchON(Rm, Dev) ← alert(Rm),
              device(Dev,Rm), Dev=alarm.
r4(Rm, Dev):switchOFF(Rm, Dev) ← alert(Rm),
              device(Dev,Rm), Dev=cooler.
```

6.1 Verification

In this subsection, we verify our multipolicy framework for different scenarios regarding the decision-making context (facts / beliefs about the environment) for the energy management case.

Scenario 1: Hot room

In this scenario, we assume that it is a hot day and the tenant of room `a6` is the stakeholder that has defined the "personal" policy. The knowledge base (background file) contains the fact:

```
tmprtr(a6,30).
```

The decisions about whether to switch on or off the cooler at the local / private policy level are as follows:

```
?- prove([switchON_local(a6,cooler,prs)],A).
A = [prs_r2(a6),prs_r3(a6,cooler,prs)]
?- prove([switchOFF_local(a6,cooler,mgm)],A).
A = [ass(neg(motion(a6))),mgm_r2(a6,cooler,mgm)]
```

This happens, because motion is an abducible fact, so both options are possible: the personal policy decides to switch on the cooler because it is hot, whereas the management policy assumes that there is none in the room and decides to switch off the cooler. On the meta-policy level, still both options are deducible even though the management stakeholder is supposed to be preferable over the personal stakeholder, because the abducible motion fact is assumed to be true, so this defeats the management stakeholders by undercut (Listing 1). This can be better demonstrated by visualizing the argumentation tree of the first goal, where it is shown that the switch on decision of the personal policy is attacked by the switch off decision of the (superior) management policy, which, however, is based on the abducible (thus defeasible) not-motion fact, which is counterattacked by the motion assumption.

If the motion fact is fixed using the defeasible fact:

```
myfact: motion(a6) ←
```

then there will be only one of the two decisions, both at the local and the global level, because there is only one option coming from the personal policy:

```
?- prove([switchON_local(a6,cooler,prs)],A).
A = [prs_r2(a6),prs_r3(a6,cooler,prs)] .
?- prove([switchON(a6,cooler,P)],A).
P = prs,
A = [myfact,prs_r2(a6),prs_r3(a6,cooler,prs),
              r2(a6,cooler,prs)]
```

In the case where the negation of motion is true:

```
myfact1: neg(motion(a6)) ←
```

the decision of management prevails over the personal policy:

```
?- prove([switchON_local(a6,cooler,prs)],A).
A = [prs_r2(a6),prs_r3(a6,cooler,prs)]
?- prove([switchOFF_local(a6,cooler,mgm)],A).
A = [myfact1,mgm_r2(a6,cooler,mgm)]
?- prove([switchOFF(a6,cooler,P)],A).
P = mgm,
A = [myfact1,mgm_r2(a6,cooler,mgm),
              r1(a6,cooler,mgm)]
```

Scenario 2: Hot room, late at night, someone is inside

In this scenario, we assume that it is a hot night and the tenant of room a6 is still inside the room, although it is very late. The knowledge base (background file) contains the following facts:

```
tmprtr(a6,30).        mytime(2300).
myfact: motion(a6) ←
```

In the management policy, the `neg(switchOFF)` option prevails over the `switchOFF` option, because someone is in the room. Thus, personal policy can suggest switching on the cooler both at local and global levels (Listing 2). The argumentation tree shows that the management policy attacked personal (being superior) by assuming the abducible negative motion fact, but this was counterattacked by the defeasible positive motion fact. Furthermore, the management policy activated saving mode, since it is late night (23:00), which led to the decision to switch off the cooler, attacking the personal policy decision, but the motion fact led to counterattacking this inside the management policy.

Scenario 3: Hot, late at night, someone is inside, there is smoke

In this scenario, we assume that it is a hot night and the tenant of room a6 is still inside the room, although it is very late. However, the smoke detector is showing that there is smoke in the room. The knowledge base contains the following facts:

```
tmprtr(a6,30). mytime(2300).  smoke(a6).
myfact: motion(a6) ←
```

Now, the emergency policy prevails over the personal policy due to smoke in the room (Listing 3). Preference rule `p111` gives preference to the emergency policy over personal, whereas `p311` gives preference to management over personal.

Scenario 4: Hot server room, no one is inside

In this scenario, we assume that it is hot in the server room and none is inside. The knowledge base contains the following facts:

```
tmprtr(srvroom,35).
myfact2: neg(motion(srvroom)) ←
```

Although none is inside the server room and the management policy suggest switching off the cooler, the meta-policy exception suggests that the personal policy defeats the management policy in this case, because it is the server room (Listing 4). Preference rule `p421` gives preference to the personal policy over the management policy for the server room and the cooler device, whereas `c121` gives preference to `p421` over `p311` that gives the opposite preference for the general case.

Scenario 5: Hot server room, no one is inside, there is fire

In this scenario, compared to scenario 4, the knowledge base contains one more fact about smoke in the server room:

```
tmprtr(srvroom,35).   smoke(srvroom).
myfact2: neg(motion(srvroom)) ←
```

The management and emergency policies suggest that the cooler should be switched off. Although in this case the personal policy still prevails over the management policy (see scenario 4), the emergency policy is stronger than the personal policy, so due to fire in the room suggests switching off the cooler (Listing 5).

```
?- prove([switchON(a6,cooler,P)],A).
P = prs, A = [ass(motion(a6)),prs_r2(a6), prs_r3(a6,cooler,prs),r2(a6,cooler,prs)]
?- prove([switchOFF(a6,cooler,P)],A).
P = mgm, A = [ass(neg(motion(a6))),mgm_r2(a6,cooler,mgm),r1(a6,cooler,mgm)]
?- visual_prove([switchON(a6,cooler,P)],A).
[prs_r2(a6),prs_r3(a6,cooler,prs),r2(a6,cooler,prs)]
|___[r1(a6,cooler,mgm),ass(neg(motion(a6))), mgm_r2(a6,cooler,mgm),p311(mgm,a6,cooler,prs)]
    |___[ass(motion(a6))]
P = prs, A = [ass(motion(a6)),prs_r2(a6),prs_r3(a6,cooler,prs),r2(a6,cooler,prs)]
```

Listing 1. Proofs and argumentation tree for scenario 1

```
?- prove([switchON_local(a6,cooler,prs)],A).
A = [prs_r2(a6),prs_r3(a6,cooler,prs)]
?- prove([switchOFF_local(a6,cooler,mgm)],A).
false.
?- visual_prove([switchON(a6,cooler,P)],A).
[prs_r2(a6),prs_r3(a6,cooler,prs),r2(a6,cooler,prs)]
|___[r1(a6,cooler,mgm),ass(neg(motion(a6))),mgm_r2(a6,cooler,mgm),p311(mgm,a6,cooler,prs)]
|   |___[myfact]
|___[r1(a6,cooler,mgm),mgm_r6(a6),mgm_r3(a6,cooler,mgm),p311(mgm,a6,cooler,prs)]
    |___[mgm_r4(a6,cooler,mgm),myfact,mgm_p2(a6,cooler,mgm)]
P = prs, A = [mgm_r4(a6,cooler,mgm),mgm_p2(a6,cooler,mgm),myfact,prs_r2(a6),prs_r3(a6,cooler,prs),r2(a6,cooler,prs)]
```

Listing 2. Proofs and argumentation tree for scenario 2

```
?- prove([switchOFF(a6,cooler,P)],A).
P = emr, A = [emr_r1(a6),emr_r4(a6,cooler,emr),r1(a6,cooler,emr)]
?- visual_prove([switchON(a6,cooler,P)],A).
[prs_r2(a6), prs_r3(a6,cooler,prs), r2(a6,cooler,prs)]
|___[r1(a6,cooler,emr), emr_r1(a6), emr_r4(a6,cooler,emr), p111(emr,a6,cooler,prs)]
|   |___{NO DEFENSE}
|___[r1(a6,cooler,mgm), ass(neg(motion(a6))), mgm_r2(a6,cooler,mgm), p311(mgm,a6,cooler,prs)]
|   |___[myfact]
|___[r1(a6,cooler,mgm), mgm_r6(a6), mgm_r3(a6,cooler,mgm), p311(mgm,a6,cooler,prs)]
    |___[mgm_r4(a6,cooler,mgm), myfact, mgm_p2(a6,cooler,mgm)]
P = prs, A = 'FAIL' .
```

Listing 3. Proofs and argumentation tree for scenario 3

```
?- visual_prove([switchON(srvroom,cooler,P)],A).
[prs_r2(srvroom),prs_r3(srvroom,cooler,prs),r2(srvroom,cooler,prs)]
|___[r1(srvroom,cooler,mgm),myfact2,mgm_r2(srvroom,cooler,mgm),p311(mgm,srvroom,cooler,prs)]
    |___[p421(prs,srvroom,cooler,mgm),c121(prs,srvroom,cooler,mgm)]
P = prs,
A = [p421(prs,srvroom,cooler,mgm),c121(prs,srvroom,cooler,mgm),prs_r2(srvroom),
      prs_r3(srvroom,cooler,prs),r2(srvroom,cooler,prs)]
```

Listing 4. Proofs and argumentation tree for scenario 4

```
?- visual_prove([switchON(srvroom,cooler,P)],A,[failed(true)]).
[prs_r2(srvroom),prs_r3(srvroom,cooler,prs),r2(srvroom,cooler,prs)]
|___[r1(srvroom,cooler,emr),emr_r1(srvroom),emr_r4(srvroom,cooler,emr),p111(emr,srvroom,cooler,prs)]
|   |___{NO DEFENSE}
|___[r1(srvroom,cooler,mgm),myfact2,mgm_r2(srvroom,cooler,mgm),p311(mgm,srvroom,cooler,prs)]
    |___[p421(prs,srvroom,cooler,mgm), c121(prs,srvroom,cooler,mgm)]
P = prs, A = 'FAIL'
?- visual_prove([switchOFF(srvroom,cooler,P)],A).
[myfact2, mgm_r2(srvroom,cooler,mgm),r1(srvroom,cooler,mgm)]
|___[r2(srvroom,cooler,prs),prs_r2(srvroom),prs_r3(srvroom,cooler,prs),p421(prs,srvroom,cooler,mgm)]
    |___[r1(srvroom,cooler,emr),emr_r1(srvroom),emr_r4(srvroom,cooler,emr),p111(emr,srvroom,cooler,prs)]
P = mgm,
A = [r1(srvroom,cooler,emr),emr_r1(srvroom),emr_r4(srvroom,cooler,emr),p111(emr,srvroom,cooler,prs),
      myfact2,mgm_r2(srvroom,cooler,mgm),r1(srvroom,cooler,mgm)] ;
[emr_r1(srvroom), emr_r4(srvroom,cooler,emr), r1(srvroom,cooler,emr)]
P = emr, A = [emr_r1(srvroom),emr_r4(srvroom,cooler,emr),r1(srvroom,cooler,emr)]
```

Listing 5. Proofs and argumentation tree for scenario 5

```
?- visual_prove([allowAccess(john,bob,address,X)],Delta).
[myfact, firem_r2(john,bob,address,firem), r1(john,bob,address,firem)]
|___[r2(john,bob,address,perso),perso_r1(john,bob,address,perso),p421(perso,john,bob,address,firem)]
    |___[p911(firem,john,bob,address,perso), myfact, c211(firem,john,bob,address,perso)]
X = firem,
Delta = [p911(firem,john,bob,address,perso),c211(firem,john,bob,address,perso),myfact,
         firem_r2(john,bob,address,firem),r1(john,bob,address,firem)]
```

Listing 6. Argumentation tree for the victim scenario of the data sharing case study

7 Data Sharing Case Study

In this section, we apply the bottom-up methodology of our approach to a data sharing case study we have created by merging the scenarios from the works of Karafili and Lupu [8] and Martinelli et al. [9]. We have identified five stakeholders competing to allow or deny access to a medical, or personal, data file:

- The individual *owner* of the data file. The owner can define his policy for giving access, making it strict or free
- Health institutions and *hospitals*. They have their business models and rules for data access
- Security and *emergency response* organizations. They assess incidents and want access to information, either to notify relatives, or to understanding what happened, or to give first aid. All these organizations have their own business models and processes (e.g. *firemen* and *red cross*).
- Finally, there is the *legislation* that limits access to data files

All these stakeholders are independent entities, with their own business models and goals. There are many cases where their policies conflict, and in that case an arbitrator is needed to resolve the conflicts. Their business models can of course be diverse and they can have specific bilateral data sharing agreements – which can themselves be encoded to rules. The ones we use for demonstrating our approach originate from the works of Karafili and Lupu [8] and Martinelli et al. [9], described in the sequel. Due to space limitations we present only the owner's and firemen's policies, and the meta-policy.

Owner Policy

The owner typically wants to access her own data but also protect them from being accessed by others. It is possible that she agrees that medical personnel can access these data without her consent only if her life is in danger. This policy is a combination of the victim policy in Martinelli et al. [9] and the owner policy in Karafili and Lupu [8] (`P` stands for person, `O` for owner and `D` for data):

```
r1(P,O,D): denyAccess(P,O,D) ←
r2(P,O,D): allowAccess(P,O,D) ← owner(O, D), P=O.
r3(P,O,D): allowAccess(P,O,D) ←
               medicalPersonnel(P).
p1(P,O,D): prefer(r3(P,O,D), r1(P,O,D)) ←
               lifeInDanger(O).
p2(P,O,D): prefer(r1(P,O,D), r3(P,O,D)) ←
c1(P,O,D): prefer(p1(P,O,D), p2(P,O,D)) ←
```

Firemen Policy

The firemen have full data access for a victim in a crime scene (Martinelli et al. [9] propose a more complete business model).

```
r1(P,O,D): denyAccess(P,O,D) ←
r2(P,O,D): allowAccess(P,O,D) ←
```

```
                victim(O), fireman(P), owner(O,D).
p1(P,O,D): prefer(r2(P,O,D), r1(P,O,D)) ←
```

Meta-policy

The meta-policy states that the priorities are (in the rules only the five first characters of the policies have been used):

- The personal stakeholder is preferred over all others.
- The legislation is preferred over all others (except personal).
- When the owner is hospitalized (is a patient) then the hospital stakeholder is preferred over personal.
- When the owner is a victim in an, e.g. accident, scene then firemen and Red Cross stakeholders are preferred over personal.

```
r1(perso): policy(perso) ←
…
r5(firem): policy(firem) ←
…
p4(perso): prefer(r1(perso),r5(firem)) ←
…
p6(firem): prefer(r5(firem),r1(perso)) ←
                victim(O).
…
c2(firem): prefer(p6(firem),p4(perso)) ←
```

7.1 Verification

Due to space limitations, we verify our multipolicy framework for the data sharing case, using a single scenario, where *Bob* is a victim found by a fireman, *John*, and John asks for access to Bob's address. The knowledge base contains the following facts:

```
fireman(john).
owner(bob,address).
myfact: victim(bob) ←
```

Access to John for Bob's address is granted, as the fireman's policy takes precedence over the personal one (Listing 6).

8 Conclusions

In this paper, we have proposed a novel computational argumentation framework for resolving conflicts that arise in a community of multiple stakeholders where each one of them bears a private policy/strategy for shared and inter-related decisions. This is also the case of smart contracts that foresee the possibility of a trusted third party, a notary or an arbitrator, to certify the contracts, as they can be disputed too [4]. Our work is a step towards that direction allowing not only the certification of a smart contract but also for conflict resolution if the contract conflicts with law or other contractual obligation.

Hence, in our framework there is a mediator that resolves the conflict and concludes on a single decision, using a meta-policy that defines preferences over competing stakeholders' decisions. Our approach avoids the high complexity of resolving all possible competing option conflicts among all competing stakeholders, because it is based on defining an arbitration meta-policy for deciding on the priority among stakeholders, instead, which are few. This meta-policy is automatically rewritten into a full meta-policy about conflicting options, but without user intervention. Thus, human arbitrators can seamlessly define their arbitration meta-policies without a heavy cognitive load. One of the advantages of our approach is that when changes are made in an individual policy, the meta-policy does not have to change.

Our framework has been tested with two scenarios, for energy saving and data sharing, indicating its generality. For the future, we would opt to extend Gorgias-B so that the meta-policy can run in combination with the simple policies from within the Gorgias-B GUI, since currently we execute the framework manually from the textual Prolog environment. Furthermore, we are working on providing a more comprehensible explanation for the decisions based on the argumentation trees.

REFERENCES

[1] Leila Amgoud, Henri Prade. 2009. Using arguments for making and explaining decisions. *Artif. Intell.* 173, 3–4, 413-436.

[2] Trevor J.M. Bench-Capon, Paul E. Dunne. 2007. Argumentation in artificial intelligence. *Artif. Intell.* 171, 10-15, 619-641.

[3] Michael Howlett. 2009. Governance modes, policy regimes and operational plans: A multi-level nested model of policy instrument choice and policy design. *Policy Sciences* 42, 1, 73-89.

[4] Florian Idelberger, Guido Governatori, Régis Riveret, Giovanni Sartor. 2016 Evaluation of Logic-Based Smart Contracts for Blockchain Systems. In *Rule Technologies. Research, Tools, and Applications (RuleML 2016)*, LNCS 9718. Springer, Cham, 167-183.

[5] Antonis Kakas, Michael Loizos. 2016. Cognitive Systems: Argument and Cognition, *IEEE Intelligent Informatics Bulletin* 17, 1 (December 2016), 14-20.

[6] Antonis Kakas, Pavlos Moraitis. 2003. Argumentation based decision making for autonomous agents. In *Proceedings of the 2nd International joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*. 883-890.

[7] Nikos Karacapilidis, Dimitris Papadias. 2001. Computer supported argumentation and collaborative decision making: the HERMES system. *Information Systems* 26, 4, 259-277.

[8] Erisa Karafili, Emil C. Lupu. 2017. Enabling Data Sharing in Contextual Environments: Policy Representation and Analysis. In *Proc. of the 22nd ACM Symposium on Access Control Models and Technologies (SACMAT '17)*. 231-238.

[9] Fabio Martinelli, Ilaria Matteucci, Marinella Petrocchi, Luca Wiegand. 2012. A Formal Support for Collaborative Data Sharing. In *Multidisciplinary Research and Practice for Information Systems (CD-ARES 2012)*. LNCS 7465. Springer, Berlin, Heidelberg, 547-561

[10] Sanjay Modgil. 2006. Hierarchical Argumentation. In Logics in Artificial Intelligence (JELIA 2006). LNCS, vol 4160. Springer, Berlin, Heidelberg, 319-332.

[11] Chimezie L. Oguego, Juan C. Augusto, Andrés Muñoz, Mark Springett. 2018. Using argumentation to manage users' preferences. *Future Generation Computer Systems* 81, 235-243.

[12] Henry Prakken, Giovanni Sartor. 1997. Argument-Based Extended Logic Programming with Defeasible Priorities. *Journal of Applied Non-Classical Logics* 7, 1, 25-75.

[13] Carles Sierra, Ramon L. de Mantaras, Simeon Simoff. 2016. The argumentative mediator. In *Multi-Agent Systems and Agreement Technologies*. LNCS 10207. Springer, Cham, 439-454.

[14] Nikolaos I. Spanoudakis, Antonis C. Kakas, Pavlos Moraitis. Applications of Argumentation: The SoDA Methodology. In *Proc. of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, The Hague, Holland, 1722-1723

[15] Nikolaos I. Spanoudakis, Antonis C. Kakas, Pavlos Moraitis. Conflicts Resolution with the SoDA Methodology. In *Conflict Resolution in Decision Making (COREDEMA 2016)*, LNAI 10238, Springer, 82-99.

[16] Nikolaos I. Spanoudakis, Elena Constantinou, Adamos Koumi, Antonis C. Kakas. 2017. Modeling Data Access Legislation with Gorgias. In *Proceedings of the 30th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2017)*, Arras, France, 317-327

[17] Thanos G. Stavropoulos, Efstratios Kontopoulos, Nick Bassiliades, John Argyriou, Antonis Bikakis, Dimitris Vrakas, Ioannis Vlahavas. 2015. Rule-based Approaches for Energy Savings in an Ambient Intelligence Environment. *Pervasive and Mobile Computing* 19, 1-23.

[18] Gregg B. Walker, Steven E. Daniels. 1995. Argument and alternative dispute resolution systems. *Argumentation* 9, 5, 693-704.