Bipolar Argumentation for Supporting Decisions in Software Design

Gianluca Amato*
gianluca.amato@unich.it
University of Chieti–Pescara
Pescara, Italy

Fabio Fioravanti*
fabio.fioravanti@unich.it
University of Chieti–Pescara
Pescara, Italy

Maria Chiara Meo*
mariachiara.meo@unich.it
University of Chieti–Pescara
Pescara, Italy

Francesca Scozzari†*
francesca.scozzari@unich.it
University of Chieti–Pescara
Pescara, Italy

Abstract

We present ongoing work on the definition of an argumentation framework to support technology selection in software architecture using bipolar argumentation. Our approach models standard design options, such as the choice of programming languages, libraries, and their configurations, as arguments, and captures their relations via attack and support among arguments. The framework enables reasoning under conflicting technical constraints and stakeholder goals, such as energy efficiency, security, and open source compliance. We formalize different kinds of argument interaction, including incompatibilities, goal-driven conflicts, and dependencies. By leveraging argumentation semantics, we derive rational and explainable design decisions based on user-defined criteria.

CCS Concepts

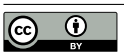
• **Software and its engineering** → *Software design tradeoffs*; Software architectures; • **Information systems** → **Decision support systems**; • **Computing methodologies** → **Knowledge representation and reasoning**.

Keywords

Bipolar Argumentation Framework, technology selection, multi-criteria decision making

ACM Reference Format:
Gianluca Amato, Fabio Fioravanti, Maria Chiara Meo, and Francesca Scozzari. 2026. Bipolar Argumentation for Supporting Decisions in Software Design. In *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26), March 23–27, 2026, Thessaloniki, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3748522.3779886

*All authors contributed equally to this research.
†Corresponding author.



SAC '26, Thessaloniki, Greece

ACM ISBN 979-8-4007-2294-3/2026/03
https://doi.org/10.1145/3748522.3779886

1 Introduction

In the software architecture field, choosing the right components, such as programming languages, libraries and configuration options, may be a difficult task. The decisions should take into account not only the availability but must often balance different constraints and goals from several stakeholders, for instance performance, energy efficiency, and commitment to open source. In order to support this decision-making process, we propose a formal framework based on bipolar argumentation [10]. A Bipolar Argumentation Framework extends the classical argumentation theory [11] by introducing the notion of support relation between arguments. This richer expressivity enables us to represent not only conflicts among arguments (attacks), but also positive dependencies, so that a specific configuration may contribute to achieve a given goal, or a module can depend on another component.

In this paper, we describe ongoing work on the definition of an argumentation framework for modeling software architectural decisions where goals and options are represented as arguments. Their relations, such as software dependencies or incompatibilities between different options and objectives, are encoded by means of attack and support relations. The result is a semantics where we can identify coherent sets of design options that satisfy a given goal and where configuration conflicts are resolved through support relations. We also show some possible extension of the Bipolar Argumentation Framework, whose semantics, striving for maximality, often include unused arguments, still supporting the specific goal.

The paper is organized as follows. In Section 2 we provide the background on the Bipolar Argumentation Frameworks. In Section 3 we present our modeling approach, and in Section 4 we apply it to a practical case study. In Section 5 we present our conclusions and discuss potential future work.

2 Background

In this section, we recall some fundamental notions concerning Computational Argumentation. The first definition we need is that of Abstract Argumentation Framework (AF), proposed by Dung [11] to define an abstract framework that disregards the underlying logic of arguments and focuses solely on the relationship between them.

Definition 1 (AFs). *An Abstract Argumentation Framework is a pair $AF = \langle Arg, Att \rangle$ where Arg is a set of arguments and Att is a binary relation on Arg (representing attacks among arguments).*

In the basic Abstract Argumentation Framework, negative interactions are explicitly represented by the attack relation, whereas positive interactions are only implicitly modeled through the derived notion of defence (an attack to an attacker). This makes support and attack inherently interdependent.

Within an AF, the goal is to identify subsets of arguments by employing specific criteria known as argumentation semantics [6, 11], e.g., admissible, complete, stable, semi-stable, preferred, and grounded (denoted as *adm*, *com*, *stb*, *sst*, *prf* and *gde*, respectively).

To formalize more realistic argumentative scenarios, it is necessary to extend beyond Dung's Abstract Argumentation Framework, which only considers attacks. In real-world contexts, indeed, positive (support) and negative (attack) interactions often coexist and operate independently of each other. To account for this distinction, Cayrol et al. [10] proposed the Abstract Bipolar Argumentation Framework, which explicitly models support and attack as two separate and independent types of relations within the argumentation structure.

Definition 2 (BAFs). *A Bipolar Argumentation Framework is a triple $AF = \langle Arg, Att, Supp \rangle$ where $\langle Arg, Att \rangle$ is an AF and Supp is another binary relation on Arg representing supports among arguments.*

For example, in a Bipolar Argumentation Framework, the set *Arg* may include arguments representing different software settings and system goals. The support relation can capture how certain configurations contribute to achieving specific objectives, while the attack relation models conflicts between incompatible settings or between settings and goals that cannot be jointly satisfied.

Given a Bipolar Argumentation Framework, the goal is to determine which arguments are considered acceptable based on a specific semantics, that is, a particular selection criterion. Arguments that do not meet this criterion are regarded as rejected.

Just like in classical Argumentation Frameworks, bipolar frameworks also allow for the evaluation of argument acceptability using various extension-based semantics (e.g., [8, 9]). These semantics aim to identify "good" subsets of arguments, each reflecting different desirable properties. However, in the bipolar setting, attack and support relations are treated as separate and independent components.

In order to define acceptability in Bipolar Argumentation Frameworks, [10] generalizes the key concept of defeat[1] between two arguments, by combining a sequence of supports with a direct defeat:

Definition 3 (Supported defeat). *[10] A supported defeat for an argument B from an argument A is a sequence $A_1\ R_1\ A_2 \cdots R_{n-1} A_n$, $n \geq 3$, with $A = A_1$, $A_n = B$, such that for each $i \in [1, \ldots, n-2]$, $R_i = Supp$ and $R_{n-1} = Att$. By extension, a sequence reduced to two arguments (A Att B) will be also called a supported defeat for B from A.*

Definition 4 (Indirect defeat). *[10] An indirect defeat for an argument B from an argument A is a sequence $A_1\ R_1\ A_2 \cdots R_{n-1} A_n$, $n \geq 3$, with $A = A_1$, $A_n = B$, such that for each $i \in [2, \ldots, n-1]$, $R_i = Supp$ and $R_1 = Att$.*

Taking into account sequences of supports and defeats, the article by Cayrol and Lagasquie-Schiex [10] introduces the following definitions concerning sets of arguments:

Definition 5 (Defeat/support by a set of arguments). *[10] Let $S \subseteq Arg$ and $A \in Arg$.*

- *S set-defeats A iff there exists a supported defeat or an indirect defeat for A from an element of S.*
- *S set-supports A iff there exists a a sequence of the form $A_1\ R_1\ A_2 \cdots R_{n-1}\ A_n$, $n \geq 2$, with $A_1 \in S$, $A_n = A$, such that for each $i \in [1, \ldots, n-1]$, $R_i = Supp$.*

The notions of set-defeat and set-support extend the defeat and support relations to sets of arguments. Based on the concept of set-defeat, the following definition of collective defence is introduced in [10]:

Definition 6 (Defence by a set of arguments). *Let $S \subseteq Arg$ and $A \in Arg$. S defends collectively A iff for each $B \in Arg$, if $\{B\}$ set-defeats A then there exists $C \in S$ such that $\{C\}$ set-defeats B.*

[10] identifies key properties that a set of arguments must satisfy in Bipolar Argumentation Frameworks, including coherence, the ability to win disputes, and often maximality. It also introduces new semantics for argument acceptability based on these criteria.

Definition 7 (Conflict-free set). *[10] Let $S \subseteq Arg$. S is conflict-free iff there do not exist $A, B \in S$ such that $\{A\}$ set-defeats B.*

External coherence is taken into account by the following

Definition 8 (Safe and closed set). *[10] Let $S \subseteq Arg$.*

- *S is safe iff there does not exist $B \in Arg$ such that S set-defeats B and either S set-supports B or $B \in S$.*
- *S is closed for Supp iff for every pair of arguments $A, B \in Arg$, if $A \in S$ and A Supp B then $B \in S$.*

Among the various semantics proposed in abstract argumentation theory, preferred semantics are particularly well-suited for selecting appropriate technologies to implement a software system that meets specific requirements. In such systems, reasoning mechanisms often need to handle conflicting information, (such as performance versus security or cost versus scalability), incomplete data, and dynamic environments.

Preferred semantics help identify sets of arguments that defend the most coherent and justifiable choices, resolving the maximum number of conflicts through counterarguments. Unlike stable semantics — which may fail to yield any extension in certain frameworks — preferred semantics always ensure the existence of at least one extension. This guarantees that the reasoning process remains functional even in the presence of inconsistency or uncertainty.

By resolving the maximum number of conflicts, preferred extensions offer a reliable foundation for selecting suitable components. This ensures that the system's reasoning capabilities remain operational even under challenging or contradictory scenarios.

[1] In accordance with the terminology adopted in [10], where the term defeat is used to denote negative interactions among arguments, we will use defeat and attack interchangeably throughout this paper.

Additionally, preferred semantics identify maximal admissible sets, capturing the largest collections of defensible arguments without internal conflict. This property supports the development of systems that aim to be inclusive and comprehensive in their decision-making logic.

In [10] the notion of admissibility is introduced as the foundation for defining preferred extensions as maximal admissible sets. It presents three definitions of admissibility, ordered by generality.

The first is d-admissibility, defined as follows:

Definition 9 (d-admissible set). *[10] Let $S \subseteq Arg$. S is d-admissible iff S is conflict-free and defends all its elements.*

Taking into account external coherence leads to the definition of s-admissibility.

Definition 10 (s-admissible set). *[10] Let $S \subseteq Arg$. S is s-admissible iff S is safe and defends all its elements.*

Finally, [10] strengthens external coherence by requiring that an admissible set be closed for *Supp*, leading to the notion of c-admissibility.

Definition 11 (c-admissible set). *[10] Let $S \subseteq Arg$. S is c-admissible iff S is conflict free, closed for Supp and defends all its elements.*

Building on the different notions of admissibility, preferred extensions are defined as those admissible sets that are maximal with respect to set inclusion. Depending on the underlying admissibility criterion (d-, s-, or c-admissibility) different types of preferred extensions can be identified, as formalized below:

Definition 12 (Preferred extension). *[10] A $S \subseteq Arg$ is a d-preferred (resp. s-preferred, c-preferred) extension iff S is S is maximal (for set-inclusion) among the d-admissible (resp. s-admissible, c-admissible) subsets of Arg.*

To compute preferred extensions on BAFs we used Aspartix [12] (https://www.dbai.tuwien.ac.at/proj/argumentation/systempage/)

3 Argumentation-based framework for technology selection

In this section, we illustrate how bipolar argumentation can be used to model a realistic design scenario. We consider the problem of selecting appropriate technologies (e.g., programming languages, libraries, configurations) for implementing a software system [14]. We define a Bipolar Argumentation Framework $AF = \langle Arg, Att, Supp \rangle$, in which application and configuration relations, as well as goals, are mapped to arguments. Positive and negative dependencies among them are captured through support and attack relations, respectively. This formal approach allows:

- modeling technical incompatibilities and preferences
- reasoning under conflicting goals
- justifying and explaining the final design decision.

In the following, we will use the single arrow $\rightarrow$ and the double arrow $\Rightarrow$ to denote the attack and support relations, respectively.

First, we define the arguments *Arg* used in our modeling framework:

- Arguments of the form

  `use(Product, Context)`

  are used to indicate a relationship between two elements, where `Product` represents a technology or library and `Context` represents the category or context in which it is used. For example, contexts can be used to model the DBMS or the programming language, and products can refer to the specific choices we have for each context.
- Arguments of the form

  `conf(Product, Context, Configuration, Value)`

  are used to indicate a setup or adjustment process for a specific `Product` (a technology or a library) in a given `Context`, to achieve a particular `configuration` (or setting). For example, a database library can use secure or unsecure connections, or a DBMS can be configured to use different storage engines.
- Arguments of the form

  `goal(Goal)`

  are used to model the objectives or criteria being considered in the configuration process. For example, in a software design project we may consider non-functional requirements such as performance, efficiency, security, portability.

Moreover we define supports *Supp* between arguments as follows

- *Goal-based support*: Supports of the form

  $\texttt{use(\_,\_)} \Rightarrow \texttt{goal(\_)}$

  indicate that the given technology or method supports the achievement of a specific objective or criterion. Within the considered set of arguments, this support relation implies that the element on the left contributes to promoting the satisfaction of the element on the right.
- *Requirement-based supports*: Supports of the form

  $\texttt{use(\_,\_)} \Rightarrow \texttt{use(\_,\_)}$

  indicate a dependency constraint between two `use(_,_)` relations.
- *Enabling supports*: Supports of the form

  $\texttt{use(L,M)} \Rightarrow \texttt{conf(L,M,\_,\_)}.$

  In these cases, support reflects a practical dependency between the use of a technology and the actions required to make it functional.
- *Goal-level supports*: The framework can also account for the fact that certain goals may positively influence or enable others, and such relationships are modeled as support between goals.

  $\texttt{goal(G1)} \Rightarrow \texttt{goal(G2)}.$

In our framework, we consider four main categories of attacks between arguments as follows:

- Incompatibility-based attacks: Attacks of the form

  $\texttt{use(\_,\_)} \rightarrow \texttt{use(\_,\_)}$

  or

  $\texttt{conf(\_,\_,\_,\_)} \rightarrow \texttt{conf(\_,\_,\_,\_)}.$

This type of attack represents a logical incompatibility or mutual exclusivity between two project choices. If a technology or tool is tied to a specific platform or environment, then its use contradicts the use of an incompatible platform.
- Dependency based attacks: In certain cases, a design choice may inherently entail another, thereby implicitly attacking all alternative options to the latter choice.
- Goal-based attacks: This type of attack models a situation where choosing a particular technology contradicts a high-level objective of the system, such as openness, sustainability, or security.
- Goal-level attacks: The framework can also take into account conflicts between goals, which are modelled as attacks between goals.

Given an argument that expresses a specific objective or criterion relevant to the configuration process, it is possible to compute the preferred extensions that include this argument. These extensions represent the most comprehensive (i.e., maximal) sets of compatible arguments that support design choices aimed at achieving the given objective, while resolving conflicts through counterarguments and reinforcing coherence through support relations.

Therefore, we define the extensions that satisfy a design choice or a specific objective as follows:

Definition 13 (Goal-based Preferred extension). *Let* `goal(G)` $\in$ *Arg be an argument representing a specific objective or evaluation criterion. A set* $S \subseteq$ *Arg is a G-based d-preferred (resp. s-preferred, c-preferred) extension with respect to* `G` *if and only if* S *is a d-preferred (resp. s-preferred, c-preferred) extension and* `goal(G)` $\in S$.

4 Case Study: Applying the Modeling Framework

In this section, we present a toy instance of the modeling framework, demonstrating how it can be used to evaluate alternative technology choices based on goal satisfaction.

4.1 Arguments

In our framework, arguments represent concrete design decisions that shape the system's architecture and behavior. These decisions may involve selecting a specific library, choosing a programming language, or defining configuration options. The following arguments outline common design patterns within the framework.

The design choice to use a library for database access is modeled by using the `dblib` context. Product values that can be used within the `dblib` context are `pdo`, `mysqli`, `jdbc` that correspond, respectively, to the choice of the PHP Data Object (PDO), MySQLi, JDBC library, and `mysql_any`, `pgsql_any` corresponding to the choice of any library that is compatible with MySQL or PostgreSQL, respectively. Thus the set of arguments for the `dblib` context are the following:

- `use(pdo,dblib)`
- `use(mysqli,dblib)`
- `use(jdbc,dblib)`
- `use(mysql_any,dblib)`
- `use(pgsql_any,dblib)`

The choice of which DBMS to use for a database is modeled by using the `db` context, with `mysql`, `oracle` and `pgsql` being the products corresponding to MySQL, Oracle and PostgreSQL, respectively, thereby obtaining the following arguments:

- `use(mysql,db)`
- `use(oracle,db)`
- `use(pgsql,db)`

Potentially, we could model a system with multiple databases, using contexts of the form `db(X)` and `dblib(X)`, where `X` denotes a specific database instance. This is shown in the full source code of the example which is available on-line, but not pursued here to avoid too much cumbersome notation.

Programming language choices are also modeled as arguments within the `pl` context:

- `use(php,pl)`: Choosing PHP as the programming language
- `use(python,pl)`: Choosing Python
- `use(java,pl)`: Choosing Java

Arguments corresponding to configuration options further refine some of the choices indicated above:

- `conf(pdo,dblib,secure_conn,yes)`: Enabling secure connection in PDO
- `conf(pdo,dblib,secure_conn,no)`: Allowing unsecure connections in PDO
- `conf(mysqli,dblib,secure_conn,yes)`: Enabling secure connection in MySQLi
- `conf(mysqli,dblib,secure_conn,no)`: Allowing unsecure connections in MySQLi.

The apparent redundancy of the second argument of `conf` (i.e., `dblib`), is useful for configuring the same library in different contexts, such as, for accessing different databases as outlined above.

Finally, high-level project goals that influence design decisions are explicitly modeled within the framework. In particular, we consider the following arguments corresponding to the key goals of environmental sustainability (`green`), cybersecurity (`security`) and open source software licensing (`opensource`).

- `goal(green)`
- `goal(security)`
- `goal(opensource)`
- `goal(privacy)`

4.2 Attack relations

Attacks represent conflicts between argument, which may arise due to incompatibility, mutual exclusivity, or trade-offs with respect to goals like energy efficiency or security.

Incompatibility-based attacks. When two design choices cannot coexist due to technical limitations or mutual exclusivity, we introduce incompatibility-based attacks between them. In the following, we will provide some examples of these incompatibility constraints related to the choice of DBMS or programming languages.

- If the DB library `L1` cannot be used to access the DBMS `L2` then we introduce an attack between `L1` and `L2`:

  `use(L1,dblib)` $\rightarrow$ `use(L2,db)`.

For example, mysqli is a DB library that is specifically designed to access the MySQL DBMS and cannot be used with other DBMS's. Thus, we have:

`use(mysqli,dblib)` ⟶ `use(pgsql,db)`.

- Similarly, we add an attack between a DB library `L1` and a programming language `L2` when `L1` cannot be used with `L2`:

  `use(L1,dblib)` ⟶ `use(L2,pl)`.

  For example, since the PDO DB library is a PHP-only library, it cannot be used with languages other than PHP. Thus, for expressing the incompatibility between PDO and Python we add the following attack:

  `use(pdo,dblib)` ⟶ `use(python,pl)`.

- If we stipulate that in a given context `C`, we cannot adopt multiple solutions at the same time, we may introduce attacks for modeling mutual exclusion as follows:

  `use(L1,C)` ⟶ `use(L2,C)` when `L1` ≠ `L2`.

  So, for example, in order to model the forced choice between the two programming languages PHP and Python, we add the following attacks:

  `use(php,pl)` ⟶ `use(python,pl)`.

- Additionally, a mutual exclusion constraint holds between different values `V1` and `V2` of a configuration option `O` for the same product `P` in context `C`. We can model such constraint by adding the following attack:

  `conf(P,C,O,V1)` ⟶ `conf(P,C,O,V2)` when `V1` ≠ `V2`.

  For instance, secure connections for the DB library mysqli are either enabled or disabled. Thus, we have

  `conf(mysqli,dblib,secure_conn,yes)` ⟶ `conf(mysqli,dblib,secure_conn,no)`
  `conf(mysqli,dblib,secure_conn,no)` ⟶ `conf(mysqli,dblib,secure_conn,yes)`.

Goal-based attacks. Other attacks can be introduce to model inadequacy of the use of some solutions with respect to some high-level project goals, such as security or software licensing.

For example, the decision to use Oracle as the database management system conflicts with the goal of adopting open source technologies.

`use(oracle,db)` ⟶ `goal(opensource)`.

Since Python is generally less efficient[2] than PHP, which is available as an alternative choice, we might add

`use(python,pl)` ⟶ `goal(green)`.

4.3 Support relations

Support relations model positive interactions, such as enabling choices or promoting goal satisfaction. In this example, we consider goal-based support only.

- Configuring a library to use secure connections supports the goal of enhancing the security of the system.

  `conf(L,dblib,secure_conn,yes)` ⇒ `goal(security)`

- Likewise, disabling secure connections reduce power consumption and hence enables the green goal.

  `conf(L,dblib,secure_conn,no)` ⇒ `goal(green)`

- Moreover, we may use a goal level support for expressing that the security goal supports the privacy goal (intended as compliance to privacy laws).

  `goal(security)` ⇒ `goal(privacy)`

4.4 Results

We have implemented the example discussed above using the Aspartix [12] tool. The implementation is available online[3] and we plan to apply static analysis techniques [2, 3] to formally verify the correctness of the code. Aspartix implements different semantics of Dung's argumentative framework and its extensions, such as Bipolar Argumentation Frameworks. It is essentially a collection of ASP (Answer Set Programming) programs for either Clingo [13] or DLV [1], although the implementation of the bipolar framework only works on the latter.

When we compute the *c*-preferred extensions, we get two extensions that are reported below. Both extensions correspond to the choice of PHP as the programming language, MySQL and the mysqli library for accessing the DBMS. However, since they differ in the configuration of secure connections, they support different goals and represent two alternative choices for the software designer.

(1) This solution does not use secure connections and, thus, supports the green goal but not the security goal.

```
{ goal(open_source), goal(green),
use(php,pl),
use(mysql,dbms), use(mysqli,dblib),
conf(pdo,dblib,secure_conn,no),
conf(mysqli,dblib,secure_conn,no) }
```

(2) This alternative solution, on the contrary, uses secure connections and supports the security (and privacy) goal but not the green one.

```
{ goal(open_source), goal(security),
goal(privacy), use(php,pl),
use(mysql,dbms), use(mysqli,dblib),
conf(pdo,dblib,secure_conn,yes),
conf(mysqli,dblib,secure_conn,yes) }
```

In both cases the opensource goal is supported. Note that the PDO DB library is configured even if not used. We will discuss this issue in the conclusions.

5 Conclusions and future work

We have applied the Bipolar Argumentation Framework to model a problem of technology selection in software architecture. The aim of this paper is twofold: to show the expressive power of BAF in representing attack and support relations and to highlight some possible extensions of the framework. Our approach is to encode all the design choices (technical options such as the use of specific libraries or configuration settings and project requirements such as security, energy efficiency, or open source) as arguments and to analyze them with a standard argumentation semantics. In particular,

[2] See: https://akcoding.com/1-billion-nested-loop-iterations-in-different-languages

[3] https://github.com/CLAI-UdA/Argumentation-Decision-Framework

our case study shows that BAF supports multi-criteria decisional analysis in a realistic scenario, capturing the relations between the components and offering valuable insights to software architects.

A previous work with aims similar to ours is [5]. In contrast, we model both library choices and overall objectives within a single bipolar argumentation framework, while that work relies on a standard Dung-style framework [11] and handles objectives via an external structure mapping design choices to objectives.

While the BAF provides a powerful formalism for modeling both negative (attacks) and positive (supports) interactions between the decision choices, our previous example shows some difficulties.

A key issue arises when computing the c-preferred extension in the BAF. Since the extensions are defined as maximal sets of arguments (which are conflict-free, defensible and support-closed), they may include some configuration arguments, such as `conf(pdo,secure_conn)` even if the corresponding usage argument, such as `use(pdo,dblib)`, is not present in the extension. The result is a set of design choices which may appear incoherent since it includes a configurated component which is not part of the system. From a software architect perspective, this violates a widely adopted consistency principle: configuring a component that is not used has no practical effect and then should be avoided. This happens since BAF can only model the problem as an attack/support graph, but it cannot express the notion of operational coherence to configure only those components that are actually used. This could align with stakeholder which may prefer a minimal design, with the smallest number of components, which produces a simpler configuration.

There are multiple research directions that could further improve and broaden the applicability of the proposed framework. For example, it is important to identify the conditions under which all goals—or at least the maximal number of goals—are guaranteed to be included in an extension. Providing such guarantees would enhance the reliability and predictability of the decision-making process.

In realistic scenarios it is often impossible to satisfy all goals simultaneously, so it would be useful to integrate multi-criteria optimization techniques that could help prioritize goals based on stakeholder preferences.

Another direction for future work could be to perform a comprehensive experimental evaluation to assess the scalability of the approach and its performance in real-world settings. Analysis of the results would provide valuable information on computational feasibility and help identify potential bottlenecks.

Finally, a key limitation observed in our case study is the lack of operational coherence: extensions may include configuration arguments without the corresponding usage arguments, leading to incoherent designs. To overcome this issue, we could consider adopting richer frameworks such as for instance [8] which exhibits a notion of coherence or framework with preferences [4, 7, 15]. This could help in designing a semantics more aligned with the goal.

Acknowledgments

We acknowledge the support of the PNRR MIUR project FAIR - Future AI Research (PE00000013), Spoke 9 - Green-aware AI and the GNCS group of INdAM.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Mario Alviano, Wolfgang Faber, Nicola Leone, Simona Perri, Gerald Pfeifer, and Giorgio Terracina. 2011. The disjunctive datalog system DLV. In *Datalog Reloaded*, 282–301. ISBN: 978-3-642-24205-2. DOI: 10.1007/978-3-642-24206-9_17.

[2] Gianluca Amato, Maria Chiara Meo, and Francesca Scozzari. 2020. On collecting semantics for program analysis. *Theoretical Computer Science*, 823, 1–25. DOI: 10.1016/j.tcs.2020.02.021.

[3] Gianluca Amato and Francesca Scozzari. 2023. The ScalaFix equation solver. In *Formal Methods, 25th International Symposium, FM 2023, Lübeck, Germany, Proceedings* (LNCS). Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, (Eds.) Vol. 14000, 142–159. DOI: 10.1007/978-3-031-27481-7_10.

[4] Leila Amgoud and Claudette Cayrol. 2002. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34, 1, 197–215. DOI: 10.1023/A:1014490210693.

[5] Ebrahim Bagheri and Faezeh Ensan. 2011. Consolidating multiple requirement specifications through argumentation. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, 659–666.

[6] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. 2011. An introduction to argumentation semantics. *Knowl. Eng. Rev.*, 26, 4, 365–410.

[7] Trevor J. M. Bench-Capon. 2003. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13, 3, 429–448. DOI: 10.1093/LOGCOM/13.3.429.

[8] Guido Boella, Dov M. Gabbay, Leendert W. N. van der Torre, and Serena Villata. 2010. Support in abstract argumentation. In *Computational Models of Argument: Proceedings of COMMA 2010* (Frontiers in Artificial Intelligence and Applications). Vol. 216. IOS Press, 111–122.

[9] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. 2010. Coalitions of arguments: A tool for handling bipolar argumentation frameworks. *Int. J. Intell. Syst.*, 25, 1, 83–109.

[10] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. 2005. On the acceptability of arguments in bipolar argumentation frameworks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Proceedings* (Lecture Notes in Computer Science). Vol. 3571. Springer, 378–389.

[11] Phan Minh Dung. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77, 2, 321–358.

[12] Wolfgang Dvořák, Sarah A Gaggl, Anna Rapberger, Johannes P Wallner, and Stefan Woltran. 2020. The aspartix system suite. In *Computational Models of Argument*. IOS Press, 461–462.

[13] Martin Gebser and Roland Kaminski. 2019. Multi-shot ASP solving with clingo. *TPLP*, 19, 1, 27–82.

[14] Viktor Pekar, Michael Felderer, and Ruth Breu. 2014. Improvement methods for software requirement specifications: a mapping study. In *2014 9th International Conference on the Quality of Information and Communications Technology*. IEEE, 242–245.

[15] Francesca Toni. 2007. Assumption-based argumentation for selection and composition of services. In *International Workshop on Computational Logic in Multi-Agent Systems*. Springer, 231–247.