Multi-Agent Cooperative Argumentation in Arg2P

Multi-Agent Cooperative Argumentation in Arg2P

Giuseppe Pisano[1][0000−0003−0230−8212], Roberta Calegari[2][0000−0003−3794−2942], and Andrea Omicini[2][0000−0002−6655−3869]

[1] Alma AI – Alma Mater Research Institute for Human-Centered Artificial Intelligence, Alma Mater Studiorum – Università di Bologna, Italy
g.pisano@unibo.it http://giuseppepisano.apice.unibo.it
[2] Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum – Università di Bologna, Italy
roberta.calegari@unibo.it http://robertacalegari.apice.unibo.it
andrea.omicini@unibo.it http://andreaomicini.apice.unibo.it

Abstract. This work focuses on cooperative argumentation and conversation in multi-agent systems by introducing an extension of the Arg2P technology that enables parallelisation and distribution of the argumentation process. The computational model and the implementation underpinning the Arg2P technology are presented and discussed.

Keywords: argumentation · Arg2P · cooperative argumentation · multi-agent systems · cooperative reasoning

1 Introduction

Human-centred intelligent systems are densely populated by agents (either software or human) capable of understanding, arguing about, and reporting, via factual assertions and arguments, what is happening and what they could make happen [19]. A multi-agent system (MAS) based on argumentation, dialogue, and conversation can then work as the basis for designing human-centred intelligent systems: through argumentation, dialogue, and adherence to social justice, the behaviour of the intelligent system can be reached, shaped, and controlled [1, 25], and conflict can be resolved by adopting a *cooperative argumentation* approach [10].

There, the purpose of multi-agent argumentative dialogues is to let agents reach an agreement on *(i)* the evaluation of goals and corresponding actions (or plans), and *(ii)* the adoption of a decentralised strategy for reaching a goal, by allowing agents to refine or revise other agents' goals and defend one's proposals. In this scenario, intelligent behaviours are likely to become associated with the capability of arguing about situations as well as the current state and circumstances, by reaching a consensus on what is happening around and what is needed, and by triggering and orchestrating proper decentralised semantic conversations so as to determine how to collectively act in order to reach a future desirable state [8]. Thus, argumentation [14] and related technologies become

a fundamental building block for the design of these systems, thanks to their potential to be an effective communication medium for heterogeneous intelligent agents while enabling a natural form of interaction between users and computational systems, towards explainability features.

However, for argumentation tools to be able to meet the aforementioned expectations, a huge effort is required from a software engineering perspective. The last decades' continuous improvement in the design and development of technologies for human-centred intelligent systems has not been matched by an analogous improvement of argumentation technologies, where the technological landscape is nowadays populated by very few systems—and most of them are mere prototypes [6]. A key problem in existing argumentation technology is that a widely-acknowledged well-founded computational model for argumentation is currently missing: this makes it difficult to investigate convergence and scalability of argumentation techniques in highly-distributed environments [18, 10]. At the same time, the field has seen a constant flow of theoretical contributions [17, 20].

Arg2P [9] is a logic-based technology, offering a thorough instantiation of the ASPIC$^+$ framework [21] for structured argumentation. The purpose of this paper is to effectively distribute the argumentation process (evaluation of arguments) so as to enable the exploitation of Arg2P in the context of cooperative argumentation, according to the aforementioned perspective. Accordingly, the work is structured as follows. Section 2 contains a brief introduction to structured argumentation. Section 3 presents the core contribution of this work, i.e., the distribution of the argumentation process and its implementation. Finally, Section 4 concludes the work.

2 Background notion: structured argumentation

Let us start by defining a generic structured argumentation framework. This introduction has two purposes: *(i)* to give the reader with no specific knowledge in the formal argumentation field an idea of its main concepts and notions, (ii) to serve as a basis for the analysis contained in subsequent sections. For a more complete introduction we invite the reader to consult the vast amount of available literature on the topic [3, 4].

We first introduce the notion of *argumentation language*. In the argumentation language, a literal is either an atomic proposition or its negation.

Notation 1 *For any literal ϕ, its complement is denoted as $\overline{\phi}$. That is, if ϕ is a proposition p, then $\overline{\phi} = \neg p$, while if ϕ is $\neg p$, then $\overline{\phi}$ is p.*

Literals are brought into relation through rules.

Definition 1 (Rules). *A **defeasible rule** r has the form: $\rho : \phi_1, ..., \phi_n \Rightarrow \psi$ with $0 \leq n$, and where*

- *ρ is the unique identifier for r;*
- *each $\phi_1, \ldots \phi_n, \psi$ is a literal;*
- *the set $\{\phi_1, \ldots \phi_n\}$ is denoted by Antecedent(r) and ψ by Consequent(r).*

Defeasible rules – denoted by DefRules – are rules that can be defeated by contrary evidence. Pragmatically, a defeasible rule is used to represent defeasible knowledge, i.e., tentative information that may be used if nothing could be posed against it. For the sake of simplicity, we define non-axiom premises via defeasible rules with empty *Antecedent*. A theory consists of a set of rules.

Definition 2 (Theory). *A **defeasible theory** is a set* $\mathsf{Rules} \subseteq \mathsf{DefRules}$.

Arguments are built from defeasible rules. Given a defeasible theory, arguments can be constructed by chaining rules from the theory, as specified in the definition below—*cf.* [21].

Definition 3 (Argument). *An **argument** A constructed from a defeasible theory $\langle \mathsf{Rules} \rangle$ is a finite construct of the form:*

$$A : A_1, \ldots A_n \Rightarrow_r \phi$$

with $0 \leq n$, where

- *r is the top rule of A, denoted by $\mathsf{TopRule}(A)$;*
- *A is the argument's unique identifier;*
- *$\mathsf{Sub}(A)$ denotes the entire set of subarguments of A, i.e., $\mathsf{Sub}(A) = \mathsf{Sub}(A_1) \cup \ldots \cup \mathsf{Sub}(A_n) \cup \{A\}$;*
- *ϕ is the* conclusion *of the argument, denoted by $\mathsf{Conc}(A)$;*

Arguments can be in conflict, accordingly to two kinds of attack: rebuts and undercutting, here defined as in [21].

Definition 4 (Attack). *An argument A **attacks** an argument B (i.e., A is an **attacker** of B) at $B' \in \mathsf{Sub}(B)$ iff A undercuts or rebuts B (at B'), where:*

- *A undercuts B (at B') iff $\mathsf{Conc}(A) = \overline{\mathsf{TopRule}(B')}$*
- *A rebuts B (at B') iff $\mathsf{Conc}(A) = \overline{\phi}$ and $\mathsf{Conc}(B') = \phi$*

Then, an abstract argumentation framework can be defined by exploiting arguments and attacks.

Definition 5 (Argumentation Framework). *An **argumentation framework** constructed from a defeasible theory T is a tuple $\langle \mathcal{A}, \rightsquigarrow \rangle$, where $\mathcal{A}$ is the set of all arguments constructed from T, and $\rightsquigarrow$ is the* attack *relation over $\mathcal{A}$. The corresponding **argumentation graph** is a directed graph whose arcs are attacks and nodes are arguments.*

Notation 2 *Given an argumentation framework $G = \langle \mathcal{A}, \rightsquigarrow \rangle$, we write $\mathcal{A}_G$ and $\rightsquigarrow_G$ to denote the framework's arguments and attacks, respectively.*

Given an argumentation framework, we leverage on labelling semantics [14, 2] to compute the sets of arguments that are accepted or rejected. Accordingly, each argument is associated with one label which is either IN, OUT, or UND—respectively meaning that the argument is either accepted, rejected, or undecided. Given a labelling for a framework, a IN, OUT, UND labelling for the statements claimed by the arguments in the graph can be also derived.

3 Distributed argumentation in Arg2P

Arg2P is a logic-based technology, an easily-deployable argumentation tool built to meet the requirements of intelligent software systems.[3] It is built upon 2P-Kt—a reboot of the tuProlog [11, 13] project offering a general, extensible, and interoperable ecosystem for logic programming and symbolic AI. Whereas a complete overview of the features of this specific implementation is out of the scope of this paper, we refer the reader to [9, 7, 24] for more details. In this section we focus on how to effectively distribute its argumentation process (evaluation of arguments).

A first version of a message-based distributed argumentation algorithm is here discussed as the basic pillar of a computational model for cooperative argumentation in MAS. We ignore issues such as agent autonomy and MAS coordination artefacts [22, 23], and focus instead on the distribution issues of cooperative argumentation, which enables agent dialogue and defeasible reasoning in MAS.

The first issue when facing computational issues of cooperative argumentation is the parallelisation of the argumentation process. Parallelisation needs to be tackled under two distinct perspectives: *(i)* the algorithmic perspective and *(ii)* the data perspective. Under the algorithmic perspective, we divide the argument evaluation (w.r.t. a given semantics) into smaller sub-tasks to be executed in parallel. Under the data perspective, instead, we split the data used by the algorithm—i.e., the argumentation defeasible theory. Action here is therefore at the data level, looking for possible data partitioning on which the argumentation process can be run in parallel. As a premise, we first introduce the algorithm that served as a starting point in the parallelisation of the argumentation process.

Among the available libraries, Arg2P includes a *query-based* mode, which allows for single-query evaluation according to the selected semantics.[4] The feature is accessible in the default instance of the Arg2P framework through the predicate

```
answerQuery(+Goal, -Yes, -No, -Und)
```

which requests the evaluation of the given *Goal*, and gets the set of facts matching the goal distributed in the three sets IN, OUT, and UND as a result.

The algorithm used to evaluate a single claim (or query) according to grounded semantics is inspired by the DeLP dialectical trees evaluation [15]. Listing 1.1 shows the pseudo-code – `AnswerQuery(Goal)` – for the `answerQuery/4` predicate: given a claim (`Goal`) as input, the function first builds all the arguments sustaining that claim (`buildSustainingArguments(Goal)`), and then requires their evaluation via the `Evaluate(A, Chain)` function. In order to assess the $A_1, ..., A_n$ status (acceptability or rejection), three conditions are evaluated:

(Cond1) if a conflicting argument labelled as IN exists, then A_1 is OUT;
(Cond2) if a cycle in the route from the root to the leaves (`Chain`) exists, then A_1 argument is UND;

[3] http://arg2p.apice.unibo.it
[4] At the time of writing, only grounded semantics is fully implemented

Listing 1.1. Structured argumentation, Arg2P answer query algorithm for grounded semantic (pseudo-code).

```
AnswerQuery(Goal):
  A_1, ..., A_n = buildSustainingArguments(Goal)
  Res = ∅
  for A in A_1, ..., A_n:
    Res = Res ∪ Evaluate(A, ∅)
  return Res.

Evaluate(A, Chain):
  if(∃ B ∈ Attacker(A): Evaluate(B, A ∪ Chain) = IN)
    return OUT
  if(∃ B ∈ Attacker(A): B ∈ Chain)
    return UND
  if(∃ B ∈ Attacker(A): Evaluate(B, A ∪ Chain) = UND)
    return UND
  return IN.
```

(Cond3) if a conflicting argument labelled as UND exists, then also the A_1 argument is UND.

If none of the above conditions is met, then the argument can be accepted.

Example 1. Let us consider the following theory and the corresponding arguments (depicted in Fig. 1).

r1 : $\Rightarrow a$	A0 : $\Rightarrow_{r1} a$
r2 : a $\Rightarrow b$	A1 : A0 $\Rightarrow_{r2} b$
r3 : $\Rightarrow \neg b$	A2 : $\Rightarrow_{r3} \neg b$
r4 : b $\Rightarrow c$	A3 : A1 $\Rightarrow_{r4} c$

According to grounded semantic $A0$ is IN – there are no arguments contending its claim or undercutting its inferences – whereas $A1$, $A2$ and $A3$ are UND—$A1$ and $A2$ have opposite conclusions and thus attack each other; the conflict is then propagated to the derived argument $A3$.

Let us suppose we require the evaluation of claim b via the `AnswerQuery(Goal)` function in Listing 1.1. First, the arguments sustaining b are created, in this case only $A1$. Then the evaluation conditions on $A1$ attackers – only $A2$ in this case – are assessed. However, $A2$ admissibility depends, in turn, on $A1$—as you can see in Fig. 1 also $A1$ attacks $A2$. There is a cycle in the graph (Cond2), and no other attackers matching (Cond1). As a consequence, $A2$ is UND and thus $A1$ (Cond3). Accordingly, claim b is labelled UND as expected.

Let us now consider the algorithm in Listing 1.1 to analyse the requirements and implications of its parallelisation. The algorithm structure is simple: the argument evaluation leverages the evaluation obtained from its attackers—i.e., the attackers are recursively evaluated using the same algorithm and the result is exploited to determine the state of the target argument. Intuitively, a first point

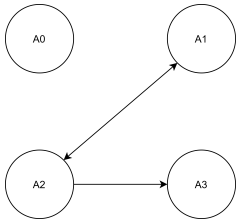

Fig. 1. Argumentation graph for arguments from Example 1, in which nodes are arguments and edges are attacks between arguments.

of parallelisation can be found in the search and evaluation of the `Attackers`. Indeed, every condition exploited by the algorithm – (Cond1), (Cond2), and (Cond3) – to evaluate an argument requires one and only one attacker to match the constraint. Those conditions directly suggest a parallelisation in the search and evaluation of the attackers. We could evaluate the arguments simultaneously under different branches, and the success in one of the branches would lead to the success of the entire search.

However, the algorithm exposes another point of parallalisation. The order in the evaluation of the conditions is essential for the soundness of the algorithm—as illustrated by the following example.

Example 2. Let us consider argument A and its two attackers B and C. Let it be the case in which we know B and C's labelling, IN for the former and UND for the latter. If we do not respect the order dictated by the algorithm, A's labelling is either UND (Cond3) or OUT (Cond1). Of course, the first result would be in contrast with the original grounded semantic requirements for which every argument having an IN attacker should be definitively OUT. Conversely, if we respect the evaluation order, A's labelling would be OUT in every scenario.

Although the evaluation order is strict, we can evaluate all the conditions simultaneously and consider the ordering only while providing the labelling for the target argument. In other words, the three conditions are evaluated in parallel, but the result is given accordingly to the defined priorities. If (Cond1) is met, the argument is labelled as OUT. Conversely, even if (Cond2) or (Cond3) are met, one should first verify that (Cond1) does not hold. Only then the argument can be labelled as UND.

Listing 1.2 contains the version of the algorithm taking into account both points of parallelisation. The three conditions – (Cond1), (Cond2) and (Cond3) – are evaluated at the same time. Then the results of the three sub-tasks are combined to provide the final solution according to the conditions' priority. Of course, if we consider a scenario where only the first condition (Cond1) is required to determine the status of the argument in input, the parallel evaluation of all three conditions would lead to a waste of computational resources. However, this problem is easily mitigated by evaluating the sub-task results as soon as they are individually available—i.e. in the case we receive a positive result from a

Listing 1.2. `Evaluate` predicate with both parallel conditions evaluation and parallel attackers

```
Evaluate(A, Chain):
  PARALLEL {
    Cond1 = PARALLEL { ∃ B ∈ Attacker(A):
        Evaluate(B, A ∪ Chain) = IN }
    Cond2 = PARALLEL { ∃ B ∈ Attacker(A): B ∈ Chain }
    Cond3 = PARALLEL { ∃ B ∈ Attacker(A):
        Evaluate(B, A ∪ Chain) = UND }
  }
  if(Cond1) return OUT
  if(Cond2 AND NOT Cond1) return UND
  if(Cond3 AND NOT Cond1) return UND
  if(NOT Cond1 AND NOT Cond2 AND NOT Cond3) return IN
```

single sub-task, and it is enough to compute the argument status, we can cut the superfluous computational branches and return the final solution.

In the first part of our analysis we focused on the parallelisation problem from a pure computational perspective, by discussing whether the evaluation task could be split into a group of sub-task to be executed simultaneously. However, there is another perspective to take into account when parallelising: the one concerning the data.

Example 3. For instance, let us consider a job computing the sum and the product of a set of numbers. Using the sub-task approach, we could have two subroutines running in parallel, one computing the sum and the other computing the product of the numbers. However, leveraging the associative property of addition and multiplication, we can split the problem into a series of tasks computing both sum and product on a subset of the original data. Then the final result would be the sum and the multiplication of the tasks' results.

Let us suppose to apply the same principle to the argumentation task. We build arguments from a base theory according to the relations illustrated in Section 2. The logic theory is, for all intents, the input data of our algorithm (argumentation task). Now, the question is whether we can effectively split the data into sub-portions to be evaluated in parallel without affecting the global soundness of the original algorithm. Let us consider a splitting principle based on rules dependency – i.e., if two rules can be chained, they must stay together –, and the algorithm in Listing 1.2. According to the algorithm, the search and evaluation of the attackers are performed in a distinct subtask (concurrent evaluation). Then, we can split the knowledge concerning attacked and attackers into separate sets, since the subtasks evaluating an attacker require only the knowledge to infer such an attacker—i.e., the *Dependency* principle must be respected. Indeed, there is no task that needs to know how to build both an argument and its attackers, since the search is delegated to another process. In

other words, a single subprocess in charge of evaluating an argument needs only the portion of the theory needed to infer the argument itself—i.e., the chainable rules concluding the target claim.

3.1 Computational model: the master-slave actor model

We can now provide a complete and sound mechanism for the admissibility task in a fully-concurrent way, exploiting the insights from Section 3 and applying them to an actor-based model [16].

In short, the actor model is based on a set of computational entities – the actors – communicating with each other through messages. The interaction between actors is the key to computation. Actors are pure reactive entities that, only in response to a message, can:

- create new actors;
- send messages to other actors;
- change their internal state through a predefined behaviour.

Actors work in a fully-concurrent way – asynchronous communication and message passing are fundamental to this end – making the actor model suited to concurrent applications and scenarios. We choose this model for its simplicity: it presents very few abstractions making it easy to study both how to model a concurrent system and its properties. The final goal is to provide a sound model for agents' cooperative argumentation in MAS, enabling concurrent evaluation of the argumentation algorithms (focusing on distribution). The actor paradigm is a straightforward choice for an analysis of this sort.

Since the actor model focuses on actors and their communication, the following design will review the structure and behaviour of the actors involved. Although a fully-distributed version of the model is possible, we choose to adopt a master-slave approach in order to simplify the functioning of the system as much as possible. Accordingly, two main sorts of actors are conceived in the system: *master* and *worker*. Master actors coordinate the knowledge-base distribution phase, while the workers hold a portion of the theory, concurring with the evaluation of a claim through their interaction.

Let us start with the knowledge distribution. Since actors are reactive entities, in order to completely adhere to the actor model the master knowledge base can be changed from outside the actor system. If the master receives the order to add a new element to the theory, three possible scenarios can be configured:

1. none of the workers contains a compatible knowledge base (kb) – i.e., it is not possible to chain the new rule to the knowledge base – and consequently, the master creates a new worker containing the portion of the theory;
2. one or more workers have a compatible knowledge base, and they add the element to their kb;
3. a set of workers possess overlapping knowledge bases – i.e. the union set of workers' knowledge bases can be used to create a unique inference chain –, and, as a consequence, we merge their knowledge bases and destroy the extra workers;

Iterating this procedure for all the elements of an input knowledge base, as a result, we should obtain a set of workers each of them containing a portion of the theory in accordance with the *dependency* splitting principle.

Once the knowledge has been correctly split between workers, we can proceed with the actor-based evaluation of an argument. Each actor is responsible for evaluating those arguments that can be built using his portion of the theory. When the actor receives an evaluation request, it first checks if attackers exist, w.r.t. its portion of the knowledge base. Then, the actor can: *(i)* register the impossibility to evaluate the argument – only if a cycle through the evaluation chain is detected –, *(ii)* require the attacker arguments evaluation to all the other actors. In the latter case, the actor shall answer the original evaluation request only after receiving a response from others actors. The conditions to match while evaluating an argument are the same as the original algorithm in Listing 1.1:

- if one counterargument is admissible, we evaluate the argument as `OUT`;
- if any number of actors decide for the argument undecidability with none advancing its rejection, we mark the argument as `UND`;
- if all the actors agree that no counterarguments can be provided as acceptable, we evaluate the argument as `IN`;

Actors provide their suggestions on the state of the requested argument according to all the labels of their counterarguments.

We can describe the interactions between the system's actors as a sequence diagram (Fig. 2) of messages exchanged between masters and workers, where:

- `Add`, sent from the master to a worker, through which the master sends the new theory member to be stored in the workers' kb; the decision on which is the right worker to send the data to is the responsibility of the master that knows the entire state of the system and how data has been divided;
- `RequireEvaluation`, sent from outside the system to the master to require the evaluation of a claim;
- `Eval`, sent from the master to all workers to require the evaluation of a claim
- `FindAttacker`, sent from a worker to master to require the broadcasting of a request for counterarguments to all the available workers;
- `ExpectedResponses`, sent from master to a worker to communicate the number of expected responses to a request for counterarguments;
- `AttackerResponse`, sent from a worker to a worker in response to a request for counterarguments; the message contains the state of the counterargument obtained through a new `FindAttacker` evaluation;
- `EvalResponse`, sent from workers to the master to communicate their decision on a claim; the decision is taken after all the `AttackerResponse` containing the state of possible counterarguments have been received;
- `EvaluationResponse`, message sent from master containing the system decision on the state of a claim.

Note that the `Add` and `RequireEvaluation` messages come from outside the actor system and start the distribution and evaluation process. This interaction

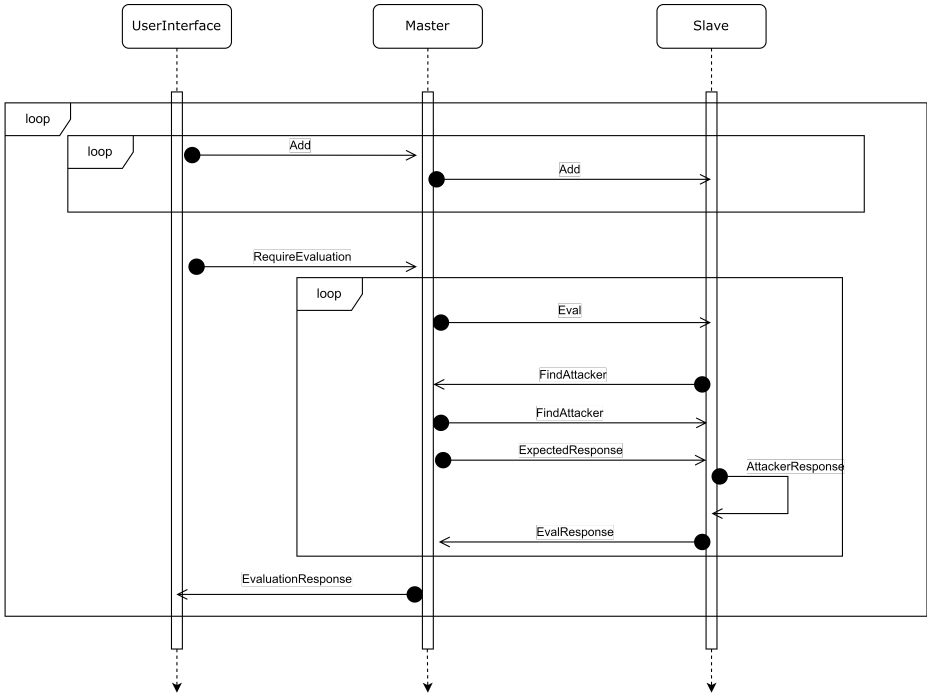

Fig. 2. Master-slave interaction for argument evaluation.

model implements both the parallelisation strategies described in Listing 1.2: the search for counterarguments is executed concurrently by all the worker nodes, as also the evaluation of the admissibility of arguments.

Example 4. Let us consider again the theory in Example 1. Let us assume a single `MasterActor` and the following order in the inclusion of the rules in the system: $r1$, $r3$, $r4$, $r2$.[5] As for the first three rules, the behaviour is the same. Since the rules are not chainable, it creates three distinct workers and sends a single rule to every one of them via the `Add` message. We now have `Worker 1`, `Worker 2`, and `Worker 3` with respectively $r1$, $r3$, and $r4$ in their knowledge bases. Then the inclusion of rule $r2$ is required, and both workers 1 and 3 results in having a chainable knowledge base. Rule $r2$ is, in fact, the missing link in the inference chain of $r1$ and $r4$. As a consequence, the `Master` stops the two workers, creates a new one, and then requires to it the inclusion of rules $r1$, $r4$, $r2$ via three `Add` messages. At the end of the distribution phase, we have two workers, one containing $r1$, $r2$, $r4$, and the other just $r3$. The dependency principle is thus respected. Going on with the example, we require the evaluation of claim b via the `RequireEvaluation` message: so, the `Master` sends an `Eval`

[5] The order of inclusion affects the steps required to converge, not the final state of the system.

message to all the actors. `Worker 1` succeeds in building an argument ($A1$) and sends to all the other `Workers` – also `Worker 1` is included in the list – a `FindAttacker` message requiring attackers evaluation—the broadcasting of the message is done by the `Master` actor. The master also communicates the number of responses that are expected (`ExpectedResponses` message)—only two in that case. `Worker 1` answers with a `AttackerResponse` communicating that there are no attacking arguments according to its knowledge, while `Worker 2` sends back a `AttackerResponse` with an `Und` result. Indeed, `Worker 2` is able to create a valid counterargument ($A2$), but a cycle is detected in the inference chain. According to the evaluation algorithm, receiving an `Und` response, `Worker 1` can finally label $A1$ as UND and let the master know that via a `EvalResponse` message.

3.2 Implementation: the parallel library

The model in Subsection 3.1 has been implemented as a library – the *Parallel* library – for the Arg2P framework.[6] The goal of the implementation is twofold: *(i)* providing a mechanism for the concurrent evaluation of a claim by a single Arg2P instance – actors in execution on a single machine can achieve real parallelisation thanks to multicore hardware architectures – *(ii)* enabling cooperative argumentation by allowing different Arg2P instances to create a single actor system, thus sharing their knowledge base or their hardware resources.

Among the available technologies for the implementation, we selected Akka.[7] [12] Akka is an open source middleware for programming concurrent and distributed actor systems based on the original Actor model by Hewitt [16]. Built upon the JVM platform, the framework offers an easy way of deploying network distributed systems observant of the original actor principles—e.g. reactivity, asynchronous communications, and absence of states of shared memory between actors. All these features made the Akka framework one of the reference technologies in the distributed landscape.

The final implementation makes use of the *Akka Clustering* features to enable the collaboration of different Arg2P instances. In particular, we rely on *Cluster Singletons*[8] to handle the `Master` actor lifecycle, and *Cluster Sharding*[9] for `Worker` nodes. The *Parallel* library makes available five directives:

- `join(Port)`, requesting the creation of an actor system on the local machine exposed on port `Port`;
- `join(Port, Address)`, to join an actor system on the machine at the given `Address`, exposed on port `Port`;
- `load`, requesting the distribution of the rules contained in the knowledge base of the local instance between all the members of the actor systems;
- `reset`, requesting the deletion of the data previously distributed in the actor system via the `load` directive;

[6] Sources available at https://github.com/tuProlog/arg2p-kt
[7] https://akka.io/
[8] https://doc.akka.io/docs/akka/current/typed/cluster-singleton.html
[9] https://doc.akka.io/docs/akka/current/typed/cluster-sharding.html

– `solve(Goal, In, Out, Und)`, requesting the evaluation of the `Goal` claim to the actor system according to the procedure in Fig. 2. Results are the set of facts matching the goal distributed in the three sets `IN`, `OUT`, and `UND`.

All the application scenarios can be modelled by using the directives above. We achieve a parallel evaluation of a claim on a single Arg2P instance in three steps: *(i)* creating a local actor system (`join(Port)`), *(ii)* distributing the theory between local actors (`load`), *(iii)* requiring the evaluation of a statement through the `solve(Goal, In, Out, Und)` directive. At the same time we could have others Arg2P instances offering their hardware resources (`join(Port, Address)`) or also participating in the resolution if they share their own knowledge (`load`).

4 Conclusion

In this work, given the relevance of issues such as pervasiveness and interconnection in the current technological landscape, we address the problem of distribution of the argumentation workload. We follow some insights from [5] and [22, 23]. In [5] the first proposal of a tuProlog-based is presented that exploits a dialogical argumentation mechanism—i.e., argumentation is performed across multiple processes proposing arguments and counterarguments. However, the argumentation algorithm distribution has not been addressed there. Conversely, in [22, 23] authors directly address the problem of enabling argumentation techniques in MAS. Nonetheless, their approach just depicts a general-purpose architectural solution for the multi-party argumentation problem in the MAS context, providing for neither an actual technology nor a precise model for the distribution and parallelisation of the argumentation process.

Overall, we believe that our approach is a step forward in the direction of a full argumentation-based MAS, and more in general of the diffusion of argumentation theories as a solid foundation for the engineering of complex intelligent systems. Yet, many issues are still to be considered. We should provide a complete analysis of the computational properties of the presented model – e.g., correctness, completeness, termination –, and also consider its relation with alternative distribution schemes (e.g., peer-to-peer). Moreover, an empirical evaluation of the performance of the system compared to traditional solvers should also be provided. Another topic of future investigations is the extension to different argumentation semantics. The main difference would be in the labelling conditions used to classify the arguments according to the different semantics. Moreover, a branching mechanism to allow the coexistence of multiple labellings should be devised in order to support the semantics with multiple extensions. However, most of the ideas behind the presented model should still remain applicable.

Acknowledgements

This work was supported by the H2020 ERC Project “CompuLaw” (G.A. 833647).

References

1. Andrighetto, G., Governatori, G., Noriega, P., van der Torre, L.W.: Normative multi-agent systems, Dagstuhl Follow-Ups, vol. 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013), http://www.dagstuhl.de/dagpub/978-3-939897-51-4
2. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. The Knowledge Engineering Review **26**(4), 365–410 (2011). https://doi.org/10.1017/S0269888911000166
3. Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L.: Handbook of Formal Argumentation. London, England: College Publications (2018), https://www.collegepublications.co.uk/handbooks/?00003
4. Besnard, P., García, A.J., Hunter, A., Modgil, S., Prakken, H., Simari, G.R., Toni, F.: Introduction to structured argumentation. Argument & Computation **5**(1), 1–4 (2014). https://doi.org/10.1080/19462166.2013.869764
5. Bryant, D., Krause, P.J., Vreeswijk, G.: Argue tuProlog: A lightweight argumentation engine for agent applications. In: Computational Models of Argument. Frontiers in Artificial Intelligence and Applications, vol. 144, pp. 27–32. IOS Press (2006), https://ebooks.iospress.nl/publication/2929
6. Calegari, R., Contissa, G., Lagioia, F., Omicini, A., Sartor, G.: Defeasible systems in legal reasoning: A comparative assessment. In: Araszkiewicz, M., Rodríguez-Doncel, V. (eds.) Legal Knowledge and Information Systems. JURIX 2019: The Thirty-second Annual Conference, Frontiers in Artificial Intelligence and Applications, vol. 322, pp. 169–174. IOS Press (11–13 December 2019). https://doi.org/10.3233/FAIA190320
7. Calegari, R., Contissa, G., Pisano, G., Sartor, G., Sartor, G.: Arg-tuProlog: a modular logic argumentation tool for PIL. In: Villata, S., Harašta, J., Křemen, P. (eds.) Legal Knowledge and Information Systems. JURIX 2020: The Thirty-third Annual Conference. Frontiers in Artificial Intelligence and Applications, vol. 334, pp. 265–268 (9-11 December 2020). https://doi.org/10.3233/FAIA200880
8. Calegari, R., Omicini, A., Sartor, G.: Computable law as argumentation-based MAS. In: Calegari, R., Ciatto, G., Denti, E., Omicini, A., Sartor, G. (eds.) WOA 2020 – 21st Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 2706, pp. 54–68. Sun SITE Central Europe, RWTH Aachen University, Aachen, Germany (October 2020), http://ceur-ws.org/Vol-2706/paper10.pdf, 21st Workshop "From Objects to Agents" (WOA 2020), Bologna, Italy, 14–16 September 2020. Proceedings
9. Calegari, R., Pisano, G., Omicini, A., Sartor, G.: Arg2P: An argumentation framework for explainable intelligent systems. Journal of Logic and Computation **32**(2), 369–401 (Mar 2022). https://doi.org/10.1093/logcom/exab089, Special Issue from the 35th Italian Conference on Computational Logic (CILC 2020)
10. Carrera, Á., Iglesias, C.A.: A systematic review of argumentation techniques for multi-agent systems research. Artificial Intelligence Review **44**(4), 509–535 (2015). https://doi.org/10.1007/s10462-015-9435-9
11. Ciatto, G., Calegari, R., Omicini, A.: 2P-Kt: A logic-based ecosystem for symbolic AI. SoftwareX **16**, 100817:1–7 (December 2021). https://doi.org/10.1016/j.softx.2021.100817
12. Cossentino, M., Lopes, S., Nuzzo, A., Renda, G., Sabatucci, L.: A comparison of the basic principles and behavioural aspects of Akka, JaCaMo and Jade development frameworks. In: Proceedings of the 19th Workshop "From Objects to

Agents". CEUR Workshop Proceedings, vol. 2215, pp. 133–141. CEUR-WS.org (2018), http://ceur-ws.org/Vol-2215/paper_21.pdf
13. Denti, E., Omicini, A., Ricci, A.: Multi-paradigm Java-Prolog integration in tuProlog. Science of Computer Programming **57**(2), 217–250 (2005). https://doi.org/10.1016/j.scico.2005.02.001
14. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77**(2), 321–358 (1995). https://doi.org/10.1016/0004-3702(94)00041-X
15. García, A.J., Simari, G.R.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming **4**(1-2), 95–138 (2004). https://doi.org/10.1017/S1471068403001674
16. Hewitt, C., Bishop, P.B., Steiger, R.: A universal modular ACTOR formalism for artificial intelligence. In: 3rd International Joint Conference on Artificial Intelligence. pp. 235–245. William Kaufmann (1973), http://ijcai.org/Proceedings/73/Papers/027B.pdf
17. Hulstijn, J., van der Torre, L.W.: Combining goal generation and planning in an argumentation framework. In: International Workshop on Non-monotonic Reasoning (NMR'04). pp. 212–218 (2004), https://www.pims.math.ca/science/2004/NMR/papers/paper28.pdf
18. Jung, H., Tambe, M., Kulkarni, S.: Argumentation as distributed constraint satisfaction: Applications and results. In: 5th International Conference on Autonomous Agents (Agents '01). pp. 324–331 (2001). https://doi.org/10.1145/375735.376322
19. Krippendorff, K.: Intrinsic motivation and human-centred design. Theoretical Issues in Ergonomics Science **5**(1), 43–72 (2004). https://doi.org/10.1080/1463922031000086717
20. Modgil, S., Caminada, M.: Proof theories and algorithms for abstract argumentation frameworks. In: Argumentation in Artificial Intelligence, pp. 105–129. Springer (2009). https://doi.org/10.1007/978-0-387-98197-0_6
21. Modgil, S., Prakken, H.: The $ASPIC^+$ framework for structured argumentation: a tutorial. Argument & Computation **5**(1), 31–62 (2014). https://doi.org/10.1080/19462166.2013.869766
22. Oliva, E., McBurney, P., Omicini, A.: Co-argumentation artifact for agent societies. In: Parsons, S., Rahwan, I., Reed, C. (eds.) Argumentation in Multi-Agent Systems, Lecture Notes in Computer Science, vol. 4946, pp. 31–46. Springer (April 2008). https://doi.org/10.1007/978-3-540-78915-4_3, 4th International Workshop (ArgMAS 2007), Honolulu, HI, USA, 15 May 2007
23. Oliva, E., Viroli, M., Omicini, A., McBurney, P.: Argumentation and artifact for dialog support. In: Rahwan, I., Moraitis, P. (eds.) Argumentation in Multi-Agent Systems, LNAI, vol. 5384, pp. 107–121. Springer (February 2009). https://doi.org/10.1007/978-3-642-00207-6_7, 4th International Workshop (ArgMAS 2008), Estoril, Portugal, 12 May 2008
24. Pisano, G., Calegari, R., Omicini, A., Sartor, G.: A mechanism for reasoning over defeasible preferences in Arg2P. In: Monica, S., Bergenti, F. (eds.) CILC 2021 – Italian Conference on Computational Logic. Proceedings of the 36th Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 3002, pp. 16–30. CEUR-WS, Parma, Italy (7-9 September 2021), http://ceur-ws.org/Vol-3002/paper10.pdf
25. Vasconcelos, W.W., Sabater, J., Sierra, C., Querol, J.: Skeleton-based agent development for electronic institutions. In: 1st International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2 (AAMAS '02). pp. 696–703. ACM, New York, NY, USA (2002). https://doi.org/10.1145/544862.544911