Operationalising Normative Rules in Autonomous Robotic Systems Through Context-Oriented Programming

Roberto Casadei[1] , **Martina De Sanctis**[2] , **Gianluca Filippone**[2] , **Sara Pettinari**[2] , **Gian Luca Scoccia**[2] , **Nicolas Troquard**[2]
[1]University of Bologna, Cesena, Italy
[2]Gran Sasso Science Institute, L'Aquila, Italy
roby.casadei@unibo.it, {martina.desanctis, gianluca.filippone, sara.pettinari, gianluca.scoccia, nicolas.troquard}@gssi.it

Abstract

The growing sensitivity to human aspects in autonomous systems engineering calls for principled approaches to embed ethical and normative concerns into their behaviour. Indeed, recent research has focused on expressing and validating sets of social, legal, ethical, empathetic, and cultural (SLEEC) concerns as rules. However, existing work is limited to rule specification or verification, leaving the problem of semantics-preserving operationalisation of ethical rules in autonomous systems largely unaddressed. Thus, we provide an operational solution for ethical-aware autonomous systems, applied to the realm of multi-service robots. Specifically, we devise a principled approach, named CO-SLEEC (Context-Oriented SLEEC), linking the normative setting of SLEEC rules to context-oriented programming (COP). CO-SLEEC enables runtime adaptation while preserving the semantics of SLEEC rules during robot task execution. It features two reusable Python libraries for (i) parsing SLEEC rules into contextual elements for operationalising them, and (ii) connecting the operational model to Robotic Operating System (ROS), respectively. We evaluate CO-SLEEC for correctness, efficiency, and maintainability over multi-service assistive robot scenarios.

1 Introduction

Autonomous and artificial intelligence (AI) systems interacting with or affecting humans with their behaviour come with associated risks that must be properly addressed. Namely, autonomous agents should transparently behave in accordance with recognised social and human values, and adapt ethically to different users in different contexts [Dignum, 2025]. At the time of global AI policy frameworks and regulations [Soler Garrido *et al.*, 2024; UNESCO, 2022; Radanliev, 2025], the mounting sensitivity on these aspects is driving research on *ethical AI* [Mittelstadt, 2019; AI HLEG, 2019] and so-called *ethical-aware autonomous systems (EthicASs)* [Autili *et al.*, 2026]. EthicASs can be framed as *normative* AI systems [Chopra *et al.*, 2018] where the behaviour of participants is regulated by contextual norms.

In this direction, [Townsend *et al.*, 2022] proposed a methodology to elicit *social, legal, ethical, empathetic, and cultural (SLEEC)* rules for autonomous systems. Then, recent research has provided ways for turning high-level normative principles into explicit, formalised, consistent sets of SLEEC rules [Yaman *et al.*, 2025; Troquard *et al.*, 2024], aimed at supporting automated and informed decision-making of EthicASs. For instance, [Troquard *et al.*, 2024] provide a mapping from SLEEC rules expressed in natural language to classical logic, enabling automated normative reasoning. [Yaman *et al.*, 2025] provide the SLEEC-TK toolkit, which offers a domain-specific language (DSL) for the specification of SLEEC rules, as well as tools for consistency validation of rulesets and conformance verification of design models formalised in tock-CSP [Miyazawa *et al.*, 2019]. However, such existing research targets specific formalisms and metamodels, while to foster the adoption of EthicASs it is key to support general-purpose programming environments.

Decoupling AI system design from norms is useful, as it enables ethics experts without programming skills to define the norms and accounts for the separate evolution of norms and systems. However, there is currently a lack of contributions on the actual *operationalisation* (i.e., mechanisms and procedures to turn principles into design and then implementations) of SLEEC rules in EthicASs. Indeed, existing approaches either (i) reason about ethical rules abstractly without executing them, or (ii) hard-code ethical behaviour, without ensuring the SLEEC semantics is preserved.

With these premises, we provide a solution to implement SLEEC rulesets in multi-service robots. Starting from the observation that SLEEC rules ground ethical concerns in specific execution contexts, we argue that their operationalisation requires programming abstractions that explicitly represent context and support runtime adaptation. Building on our preliminary work [Casadei *et al.*, 2026b], we propose *Context-Oriented SLEEC (CO-SLEEC)*, an approach that maps SLEEC rules to context abstractions and enables their semantics-preserving enactment at runtime using the *context-oriented programming (COP)* paradigm [Salvaneschi *et al.*, 2012; Elyasaf *et al.*, 2023].

CO-SLEEC provides (i) a principled mapping from

SLEEC rule constructs to COP mechanisms, and (ii) a reusable runtime approach for ROS-based robotic systems, based on two novel open-source Python libraries: `pysleec`, for parsing and managing SLEEC rules, and `sleec4ros`, for integrating ethical rule enactment into robot execution.

We evaluate CO-SLEEC in the scenario of a multi-service assistive robot working in the hospital and household domains. Results show that CO-SLEEC enacts domain-specific ethical rules correctly, with negligible runtime overhead, and grants maintainability on evolving ethical rulesets.

The paper is organised as follows. Section 2 provides background concepts and reference scenarios. Section 3 covers the contribution. Section 4 shows the implementation for robotic systems. Section 5 presents experiments and results. Section 6 covers related work. Section 7 concludes the work.

2 Background and Motivation

Here, we introduce SLEEC rules, COP and the reference scenarios of a multi-service care robot in two different settings.

2.1 SLEEC Rules

Autonomous agents interacting with humans are required to comply with the human contexts to avoid potential negative consequences arising from their own behaviour or decisions [Dignum, 2025]. To aid in the design of systems with this capability, [Townsend *et al.*, 2022] introduced *social, legal, ethical, empathetic, and cultural (SLEEC) rules* and a methodology for domain experts to elicit actionable constraints on the system behaviour from high-level norms and principles. It particularly builds on research on machine ethics [Anderson and Anderson, 2011] and defeasible reasoning [Horty, 2012]. Unlike deontic logics [Gabbay *et al.*, 2013], SLEEC rules do not aim to reason about normative systems. SLEEC rules also do not define the system's overall goal and behaviour but rather specify the system's response to contextual conditions at execution time. Specifically, a SLEEC rule consists of a single *default rule* possibly followed by one or more *hedge clauses* (a.k.a. *defeaters* [Brunero, 2022]). The default rule follows the structure "`WHEN` *trigger* `THEN` *response*". This specifies an event (the trigger) whose occurrence indicates the need to satisfy the constraints defined in the response [Yaman *et al.*, 2023]. However, a rule can be "defeated" by hedge clauses (exceptions) [Townsend *et al.*, 2022]. These are specified via construct `UNLESS` *condition* `IN WHICH CASE` *obligation*. Each hedge clause specifies a condition in which the original response should be preempted, with the obligation to perform an alternate response, thus enabling contextual reasoning. Notably, SLEEC rules are elicited and defined over Boolean propositions. Still, they can be applied to *fuzzy* contexts by instantiating fuzzy predicates to crisp values at evaluation time.

As an example, consider the rule from the assistive robotics scenario (introduced next in Section 2.3), which governs a robotic agent's response to user requests to open curtains:

`WHEN` CurtainOpenRequest `THEN` OpenCurtains
`UNLESS` UserUndressed `IN WHICH CASE` (RefuseRequest AND ExplainWhy) ①
`UNLESS` UserDistressed `IN WHICH CASE` (WarnUser AND OpenCurtains) ②

According to the default rule, the robot opens the curtains when asked. However, two context-dependent hedge clauses apply: if the user is undressed, the robot refuses and explains the reason, to respect privacy and cultural sensitivity ①; if the user is highly distressed, this clause is overridden and the curtains are opened ②, prioritising emotional state and autonomy. The order in which hedge clauses are listed specifies their importance and priority. In this work, we follow the evaluation order defined in [Troquard *et al.*, 2024], by which hedge clauses are evaluated in a top-down manner, with the last one whose condition holds taking precedence over the others. E.g., if the user is both undressed and distressed, then ② is activated, as it is the last one that evaluates to true.

2.2 Context-Oriented Programming (COP)

Context-oriented programming (COP) [Salvaneschi *et al.*, 2012] is a language-based paradigm for context-aware systems development supporting adaptation by explicit context modelling and management. Being SLEEC rules context-dependent, we find it natural to investigate this approach.

COP languages have abstractions and mechanisms for (i) modularising context-dependent behavioural variations, and (ii) supporting their dynamic activation upon context change. Though various flavours of COP exist [Elyasaf *et al.*, 2023], the most popular model is the one based on *layers* in object-oriented settings. A layer is a named modular unit that groups partial class or *method implementations*, and can be explicitly or implicitly activated at runtime (several activation mechanisms exist). Specifically, multiple implementation variants of a given method are bound to different layers to define behavioural variants. When a layer is activated, the default behaviour is overridden by the one prescribed by the implementation bound with the active layer. This way, COP improves modularisation w.r.t. if-then and object-oriented dispatch [Kamina *et al.*, 2016]. Also, when the target system is highly context-dependent (cf. EthicASs), COP helps to identify a *mapping* from requirements to implementation in explicit contextual terms, thus fostering understandability, inspection, and maintenance [Salvaneschi *et al.*, 2012].

In this work, we adopt ContextPy3 [Schubert *et al.*, 2015] as the supporting COP framework, as it provides a general and flexible layer activation mechanism and integrates naturally with ROS-based Python implementations. Consider the following minimal self-contained example of Listing 1.

Listing 1 Context-Oriented-Programming example

```
1  import contextpy3 as cop
2  PC_LAYER = cop.Layer('Polite Conversation')

3  @cop.base
4  def hello(): # base version
5    print('Hello!')

6  @cop.around(PC_LAYER)
7  def hello(): # when PC_LAYER is active
8    print('Hope you're having a wonderful day!')

9  hello() # Hello!  (No layer active)
10 cop.global_activate_layer(PC_LAYER)
11 hello() # Hope you're having a wonderful day!
```

It defines a layer `PC_LAYER` (instance of class `Layer`) and two implementation variants of the method `hello`.

The method implementation decorated with `@base` defines the default behavior to be selected when no layers are active. Instead, the implementation variant decorated with `@around(PC_LAYER)` (i.e., bound with `PC_LAYER` layer) is executed when `PC_LAYER` is active, to greet in a polite way. Layers can be *(de)*activated by calling `global_(de)activate_layer(L)`. Thus, context-dependent behaviour is accomplished.

2.3 Reference Scenario: Assistive Robotics

As a reference scenario, we target an *assistive multi-service humanoid care robot* aimed at providing assistance and companionship to humans across different contexts. As an EthicAS, it should behave in accordance with human values and ethical aspects while fulfilling its mission. SLEEC rules have been proposed precisely to organise and constrain system behaviour from high-level norms and ethical principles. Operationalising SLEEC rules would enable the robot to respect the ethical considerations of the different contexts in which it may be deployed. This work considers two representative domains for a humanoid care robot, each governed by multiple SLEEC rules, some of which are shared across contexts. To ground our scenario, we draw on the work in [Feng *et al.*, 2024], from which we extract a set of already elicited and validated SLEEC rules. Specifically, we selected eight rules: three per domain plus two cross-domain rules. The reference scenario intentionally relies on a ruleset defined and analysed in prior work [Feng *et al.*, 2024], where the authors apply a systematic methodology for eliciting SLEEC rules and verifying them through satisfiability checking. Thus, the rules we use have already been checked for conflict- and redundancy-freeness, rather than being "just assumed" to be correct. Here, we discuss two representative rules. The full ruleset, the CO-SLEEC implementation, all software artifacts, and data sets used is available online [Casadei *et al.*, 2026a].

Hospital Triage In the hospital domain, a care robot operates within a socio-technical system to support patients during clinical workflows. The robot assists patients in the triage process by gathering subjective information (e.g., self-reported symptoms) and objective data (e.g., blood pressure, temperature), and then guides them back to the waiting area.

The selected illustrative SLEEC rule specifies a context-dependent behavioural constraint during patient preparation for examination. The rule enforces that patient preparation is conducted in a private space. However, if the patient is below the legal age, the presence of a legal tutor is required, in accordance with clinical protocols. Formally:

`WHEN` PreparingExamination `THEN` EnsurePrivateSpace
`UNLESS` UserAge $<$ LegalAge `IN WHICH CASE` AskLegalPresence

Household Dress Assistance In the home domain, the care robot works to assist humans, e.g., the elderly or people with disabilities. It may collaborate with human caregivers or operate autonomously, with the main role of providing daily support (e.g., assisting in dressing). Beyond functional assistance, the robot should also contribute to the human's emotional well-being by offering companionship and comfort.

SLEEC Elements	COP Concepts
Default rule trigger	Event triggering a method call
Default rule response	Base method implementation
Hedge clause	Layer
Hedge clause condition	Runtime context
Hedge clause obligation	Layer-specific method implementation

Table 1: Mapping of SLEEC elements to COP concepts

As an example rule for this scenario, consider the rule previously introduced in Section 2.1, which governs the robot's response to user requests to open curtains during dressing.

3 The CO-SLEEC Approach

CO-SLEEC consists of a principled mapping from SLEEC rulesets to COP concepts. This section presents the mapping and the formal definition of the CO-SLEEC semantics.

3.1 Operationalising SLEEC Rules Through COP

The core idea of *CO-SLEEC* is to *map SLEEC rule elements to COP concepts*, as shown in Table 1. In this way, the developer can define behavioural variations through method implementation variants, each corresponding an obligation of the SLEEC rules. They are dispatched based on how the current context relates to the rules' triggers and hedge clause conditions. Specifically, hedge clauses are mapped to COP layers. This allows each layer to identify the method implementation for the behaviour prescribed by the associated hedge clause obligation. Then, when a layer is activated, its method implementation variant preempts the base method, like a hedge clause defeats the default rule response with its obligation.

In detail, the SLEEC elements are mapped as follows. Each SLEEC rule corresponds to a set of implementations of the same method (i.e., methods with the same name, signature, and scope). The *base method* implements the system's response to the trigger event (*default rule response* $\rightarrow$ *base method implementation*). The same method has a different implementation variant for each hedge clause associated to a COP layer (*hedge clause* $\rightarrow$ *COP layer*) implementing the clause's obligation. The method is dispatched when the trigger event occurs (*default rule trigger* $\rightarrow$ *event triggering a method call*), and COP will ensure the proper variant is selected (*hedge clause condition* $\rightarrow$ *runtime context* and *hedge clause obligation* $\rightarrow$ *layer-specific method implementation*).

At runtime, in CO-SLEEC, the SLEEC rules are evaluated against the current context conditions to determine the *active hedge clause* of each rule. An active hedge clause is the hedge clause to be enforced when, in the current runtime context (and according to the SLEEC semantics [Troquard *et al.*, 2024]), the default rule's trigger occurs. Thus, following the COP paradigm, the layer associated to the active hedge clause is activated, and the bound method is executed, instead of the base one. If no hedge clauses are active for a given SLEEC, when the trigger occurs, the base method is executed. For instance, considering the SLEEC rule in Section 2.1, when the user is both *undressed* and *distressed*, then the active hedge clause is ②. Here, the current runtime context is defined by the conditions *undressed* and *distressed* evaluating to true. If the user asks to open curtains (i.e., the default rule's trigger

occurs), the behaviour implemented by the layer associated to the hedge clause ② is run. This allows the system to behave according to the defined SLEEC ruleset.

Next, we provide, as an overview, a lightweight formalisation of SLEEC elements in the CO-SLEEC approach (more details are in Section 3.2). A SLEEC rule R is of the form: `WHEN` C_0 `THEN` O_0 `UNLESS` C_1 `IN WHICH CASE` O_1 ... `UNLESS` C_n `IN WHICH CASE` O_n. A context χ is a valuation of atomic variables that characterise a situation, and of which C_i ($i \in \{0, \ldots, n\}$) are Boolean combinations. We write $\chi \models C_i$ to indicate that a context χ makes a condition C_i true, and naturally extend it to conjunctions.

A hedge clause hc_i in a SLEEC rule R can be represented as a triplet $\langle \ell(R)_i, C_i, O_i \rangle$, where $\ell(R)_i$ is the layer label, while C_i and O_i denote the hedge clause condition and obligation, respectively. Given a SLEEC rule R and its set of hedge clauses $HC = \{hc_1, \ldots, hc_n\}$, a hedge clause $hc_i = \langle \ell(R)_i, C_i, O_i \rangle \in HC$ is said to be *active* if and only if one of the following holds:

$$\begin{cases} \chi \models C_1 \wedge \cdots \wedge C_i \wedge \neg C_{i+1} & \text{if } i < n \ , \\ \chi \models C_1 \wedge \cdots \wedge C_i & \text{if } i = n \ . \end{cases}$$

Intuitively, the active hedge clause is the last clause in the ordered sequence whose condition holds in the context χ, as defined in the semantics by [Troquard *et al.*, 2024].

Algorithm 1 SLEEC evaluation algorithm

1: **Input:** SLEEC rule R = `WHEN` C_0 `THEN` O_0 `UNLESS` C_1 `IN WHICH CASE` O_1 ... `UNLESS` C_n `IN WHICH CASE` O_n, and context χ
2: **Output:** active hedge clause label
3: **for** $i \in \{1, \ldots, n\}$ **do**
4: **if** $\chi \models C_i$ **then continue**
5: **else**
6: **if** $i = 1$ **then return** ϵ
7: **else return** $\ell(R)_{i-1}$
8: **return** $\ell(R)_n$

Operatively, Algorithm 1 presents the pseudocode of the SLEEC evaluation logic. This procedure is run upon each context change for every SLEEC rule and returns the label of the active hedge clause associated with that rule. The algorithm iterates over the entire set of hedge clauses, continuing until the condition of a hedge clause is evaluated to `false` (lines 3–7). Whenever a condition is `false` (lines 5–7), two cases can be identified. If the condition evaluated to `false` belongs to the first hedge clause, an empty label ϵ is returned, indicating that no hedge clauses are active (line 6). Otherwise, the algorithm returns the label of the previous hedge clause, which represents the layer to be activated (line 7). Finally, if all conditions evaluate to `true`, the label of the last hedge clause is returned (line 8).

3.2 Formal Aspects

Here we provide a formal presentation of SLEEC rules and their translation into CO-SLEEC instances.

Formal setting Let $\mathcal{T}$ be a finite set of atomic triggers. A trigger is considered true when the corresponding event occurs. Let $\mathcal{C}$ be a finite set of atomic conditions. They serve to build conditions in the left-hand side of hedge clauses. Let $\mathcal{O}$ be a finite set of atomic obligations and responses. They serve to build the right-hand side of clauses.

A context is an element of $2^{\mathcal{T} \cup \mathcal{C}}$, i.e., a set of elements taken from $\mathcal{T}$ and $\mathcal{C}$. A context χ specifies the set of atomic triggers and conditions that are considered true, while atomic triggers and conditions in the set $\overline{\chi} = (\mathcal{T} \cup \mathcal{C}) \setminus \chi$ are considered false in the context χ.

We use $\models$ to denote the classical logical entailment. A truth assignment, or *interpretation*, is a complete function $\mathsf{val} : \mathcal{T} \cup \mathcal{C} \cup \mathcal{O} \to \{\top, \bot\}$. It is extended to arbitrary Boolean combinations of $\mathcal{T} \cup \mathcal{C} \cup \mathcal{O}$—we also call them *formulas*—as it is usual in classical propositional logic. An interpretation val is a *model* of a formula ϕ if $\mathsf{val}(\phi) = \top$. The statement $S_1, \ldots, S_k \models \phi$ means that ϕ is entailed from the sets of formulas $S_1, \ldots, S_k$. More precisely, the statement holds exactly when, for every interpretation, if it is a model of all the formulas in $\cup_i S_i$, then it is also a model of ϕ.

SLEEC rules and obligation entailment A SLEEC rule is of the form: `WHEN` C_0 `THEN` O_0 `UNLESS` C_1 `IN WHICH CASE` O_1 ... `UNLESS` C_n `IN WHICH CASE` O_n, where n is a non-negative integer, C_0 is a trigger, for every $1 \leq i \leq n$, C_i is a formula over $\mathcal{C}$, and for every $0 \leq i \leq n$, O_i is a conjunction of elements in $\mathcal{O}$. Let $\mathcal{R}$ be the set of all SLEEC rules.

We will need to refer to specific SLEEC rules programmatically. A SLEEC rule is already uniquely identified by its syntactic form. But for code interpretability and maintenance it will also be convenient to let the developer specify more mnemonic names. Hence, every SLEEC rule R is given a label $\ell(R)$ that uniquely identifies it. Therefore, we assume that for all $R, R' \in \mathcal{R}$, if $\ell(R) = \ell(R')$ then $R = R'$.

We adapt the compilation of SLEEC rules into classical logic from [Troquard *et al.*, 2024]. When R = `WHEN` C_0 `THEN` O_0 `UNLESS` C_1 `IN WHICH CASE` O_1 ... `UNLESS` C_n `IN WHICH CASE` O_n, then we define $\mathsf{compile}(R) :=$

$$\left[\bigwedge_{0 \leq i \leq n-1} \left(\left(\bigwedge_{0 \leq j \leq i} (C_j) \wedge \neg C_{i+1} \right) \rightarrow O_i \right)\right] \wedge \left[\left(\bigwedge_{0 \leq j \leq n} (C_j) \right) \rightarrow O_n\right] \quad (1)$$

This effectively provides a semantics to SLEEC rules.

In the remainder, we let Γ be a finite set of N SLEEC rules (i.e., $N = |\Gamma|$). The SLEEC rules in Γ are of the form $R_j =$ `WHEN` C_0^j `THEN` O_0^j `UNLESS` C_1^j `IN WHICH CASE` O_1^j ... `UNLESS` $C_{n_j}^j$ `IN WHICH CASE` $O_{n_j}^j$ for every $j \in \{1, \ldots, N\}$.

Definition 1 (Obligation entailment from SLEEC rules)**.** *Given a finite set $\Gamma \subseteq \mathcal{R}$ of SLEEC rules, a context χ, and an obligation O, "Γ entails obligation to do O in χ" when*

$$\bigcup_{R \in \Gamma} \mathsf{compile}(R), \chi, \{\neg \iota \mid \iota \in \overline{\chi}\} \models O \ .$$

That is to say, an obligation is entailed in a context from a set of SLEEC rules, exactly when it is classically entailed by their logical representations.

CO-SLEEC instances A *CO-SLEEC instance* is a tuple $\langle M, L, MM, LM\rangle$ where M is a set of methods, L is a set of COP layers, $MM : \Gamma \to M$ is a *method manager* associating a method to every SLEEC rule, $LM : 2^{\mathcal{T}\cup\mathcal{C}} \to 2^{L\cup\{\epsilon\}}$ is a *layer manager* associating a set of layers to every context.

Definition 2 (CO-SLEEC instance execution)**.** *Given a CO-SLEEC instance $I = \langle M, L, MM, LM\rangle$, and a context χ, "I* executes *the code* impl *in context χ" when there is a rule $R_j \in \Gamma$ and a layer $\ell \in LM(\chi)$, such that $C_0^j \in \chi$ and an implementation variant of $MM(R_j)$ is bound with ℓ and contains* impl*.*

We write $\gamma(\Gamma)$ the translation of the SLEEC ruleset $\Gamma \subseteq \mathcal{R}$ into a CO-SLEEC instance. $\gamma(\Gamma)$ is the CO-SLEEC instance $\langle M, L, MM, LM\rangle$, such that:

- the set of methods is $M = \{\mathsf{method}(R) \mid R \in \Gamma\}$;
- the set of layers is $L = \{\ell(R_j)_i \mid 1 \le j \le N, \text{ and } 1 \le i \le n_j\} \cup \{\epsilon\}$;
- for every j, $MM : R_j \mapsto \mathsf{method}(R_j)$;
- for every context χ, $LM(\chi) = \{$**Algorithm 1**$(R_j, \chi) \mid R_j \in \Gamma$ and $C_0^j \in \chi\}$.

The implementation $\mathsf{impl}(O)$ of every obligation O occurring in the SLEEC rules must be placed inside the methods that are bound with corresponding layers, as exemplified in Listing 2. Formally, given a SLEEC rule $R_j \in \Gamma$, and given $O_i^j = O_0 \wedge \ldots \wedge O_k$, the code $\mathsf{impl}(O_0); \ldots; \mathsf{impl}(O_k)$ and nothing else will be placed in the implementation variant of $\mathsf{method}(R_j)$ bound with $\ell(R_j)_i$. We say that the implementation of SLEEC rule R_j is *properly structured*.

Proposition 1 (Semantics preservation)**.** *Let Γ be a set of SLEEC rules. Γ entails the obligation to do O in context χ iff $\gamma(\Gamma)$ executes* $\mathsf{impl}(O)$ *in context χ.*

It follows by mere construction, and the observation that Algorithm 1 lifts the SLEEC semantics represented by Equation (1) to the decision of layer activation.

4 CO-SLEEC for ROS-Based Systems

We implement CO-SLEEC on top of the ROS framework [Macenski *et al.*, 2022], which structures robotic systems as a set of independent processes (*nodes*) interacting via message-passing. Communication follows a *publish/subscribe* model, where nodes exchange messages via *topics*.

Figure 1 shows the proposed implementation architecture. It involves two main actors: (i) *SLEEC Experts*, responsible for eliciting SLEEC rules, and (ii) the *Developer*, responsible for developing CO-SLEEC-augmented ROS nodes.

To support the development, we provide two reusable libraries: (i) `pysleec`, and (ii) `sleec4ros` [Casadei *et al.*, 2026a]. The former is built on top of the `contextpy3` library (cf. Section 2.2). It provides a parser that translates textual rule definitions, expressed in the form described in Section 2.1, into corresponding Python objects, enabling their use within the system. Also, the library offers features for defining and evaluating SLEEC rules, as well as initialising and managing COP layers. The latter enables the integration of CO-SLEEC within ROS, by providing dedicated nodes, message types, and communication interfaces linking the CO-SLEEC-based reasoning with robotic components.

From SLEEC rules to layers SLEEC experts provide SLEEC rulesets for the target domains, checked for consistency and completeness according to formal approaches as in [Feng *et al.*, 2024]. When the robotic system loads a ruleset, the following steps occur: (1) given a rule, COP layers (hereafter also SLEEC layers) are defined for each hedge clause in it; (2) the rule is automatically mapped to the corresponding COP layers, through the *SLEEC Config* component. The layers are dynamically managed at runtime according to context changes and the SLEEC semantics.

Dynamic layers management Layer management in ROS is achieved via the *Context Manager Node*, according to the following event-driven, reactive logic. The *Domain Listener* subscribes to a */domain* ROS topic to determine the robot's current operational domain (e.g., hospital or house). This information is used by the *Hedge Clauses Manager* to filter the SLEEC rules for the current domain, ensuring that domain-specific rules and behaviours are correctly scoped and applied. To implement context-awareness, the *Conditions Updates Listener* subscribes to the */conditions* ROS topic to monitor system updates, such as user state (e.g., user undressed) or environmental changes (e.g., door opened). On a condition change, this component first triggers the *Hedge Clauses Manager*, which determines the set of currently active SLEEC layers and forwards them to the *Layer Updater*, then it publishes over the */layers_update* topic the list of layers to de/activate. The *Layer Listener* captures this list and passes it to the *COP Layer Manager*, responsible for the actual activation/deactivation. To achieve this, the *COP Layer Manager* employs the global activation mechanism (cf. Section 2.2) of `contextpy3` to support the activation and deactivation of SLEEC layers. Simultaneously, the *COP Layer Manager* maintains a shared knowledge base of active layers.

Normative-aware mission execution In robotic systems, the mission of robots is defined as a set of tasks. At runtime, the *Mission Manager Node* guides the mission by triggering tasks execution. Following ROS best practices, each task is encapsulated as a ROS node by defining *Task Manager Node*. When a task is constrained by one or more SLEEC rules, it must include multiple implementations (behavioural variations), overriding or complementing the default behaviour to accommodate hedge clauses. Thus, a *Task Implementation* is enabled by one of the currently active layers, thereby ensuring that the robot runs the selected implementation, adapting its behaviour to respect normative aspects. Listing 2 shows a ROS node implementing the SLEEC rule of the household dress assistance scenario of Section 2. `OpenCurtainNode` reacts to messages received on the */open_curtains_req* topic (line 4). It implements `open_curtains_request` in three variants: (i) a default case that opens the curtains upon request (lines 5–7); (ii) a variant declining the request if the user is undressed, offering an explanation (lines 8–10), cf. hedge clause ①; (iii) a variant for opening the curtains if the user

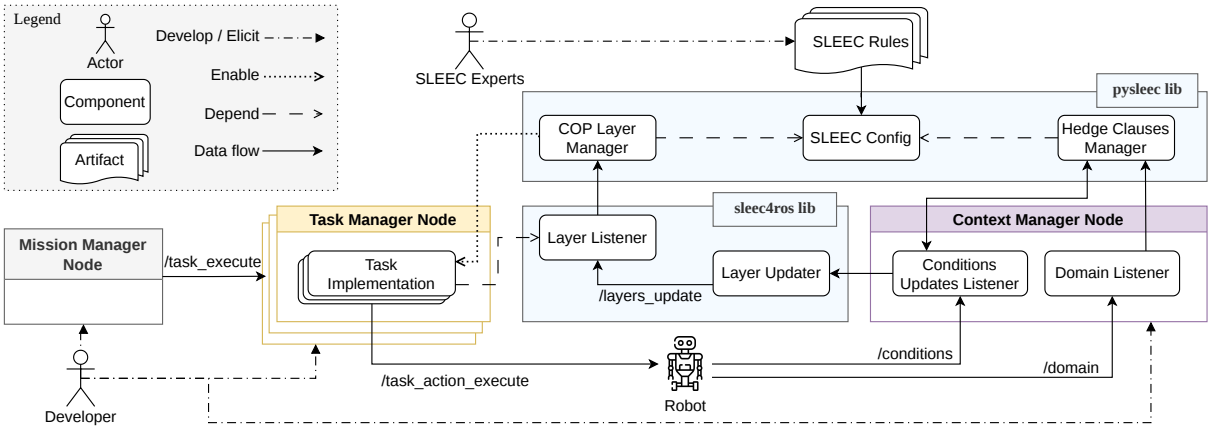

Figure 1: CO-SLEEC for ROS architecture

is distressed, after issuing a safety warning (lines 11–13), cf. hedge clause ②.

Importantly, this structure does not constrain the execution of a single obligation at a time. When multiple task triggers occur concurrently, or when a trigger occurs while an obligation is running, the execution of the corresponding actions follows the standard ROS execution model, which allows parallel task execution. The consistency of the enforced obligations is ensured by the conflict-freeness of the underlying SLEEC rules, while the responsibility for checking task executability in a given state is up to the developer.

Listing 2 CO-SLEEC Open Curtain ROS Node code snippet

```
1  class OpenCurtainNode(Node):
2    def __init__(self):
3      super().__init__('curtain_node')
4      self.create_subscription( Empty, 'open_curtains_req',
   self.open_curtains_request, 10)
5  @cop.base
6  def open_curtains_request(self, msg):
7    # implementation:  OpenCurtains
8  @cop.around(sleec_layers['dress_assist.hc1'])
9  def open_curtains_request(self, msg):
10   # implementation:  RefuseRequest AND ExplainReason
11 @cop.around(sleec_layers['dress_assist.hc2'])
12 def open_curtains_request(self, msg):
13   # implementation:  WarnUser AND OpenCurtains
```

5 Empirical Evaluation

We evaluate our CO-SLEEC implementation on the multi-service robot scenarios introduced in Section 2.3. Code, data, and results are available online [Casadei *et al.*, 2026a].

Goals Our evaluation assesses:

- *E1. Correctness*: the implemented system complies with the ethical rules, by respecting the SLEEC semantics.
- *E2. Efficiency*: CO-SLEEC does not add relevant runtime overhead compared to hardcoded implementations.
- *E3. Maintainability*: the evolution of SLEEC rulesets (e.g., normative requirements changes, addition of domains) requires little effort to be reflected in the CO-SLEEC-based system.

Experimental setting We considered the SLEEC rulesets of the two domains and generated 100 test cases with random assignments of rule conditions and active domain. For each case, we derived the ground truth as the expected obligation according to Definition 1. Overall, 800 rule triggers were executed (100 test cases $\times 8$ SLEEC rules). Experiments were run on a Turtlebot 4 equipped with a Raspberry Pi 4B (Ubuntu 22.04, ROS 2 Humble). To isolate CO-SLEEC overhead, robot actions were replaced with logging statements. Correctness is evaluated by comparing executed obligations against the ground truth. Efficiency is measured as the delay between rule triggering and task execution. Maintainability is assessed via lines of code (LoC) required to add or modify rules, and more qualitative observations.

Ablation study We compared the results obtained by running the test cases over CO-SLEEC with the ones obtained over two ablated system implementations: CS$-$*libs* and CS$-$*libs*$-$*CM*. **CS$-$*libs*** ("CO-SLEEC minus libraries") follows the same architecture of CO-SLEEC without using the `pysleec` and `sleec4ros` libraries (i.e., Figure 1 except the blue boxes). It leverages a *Context Manager Node* to evaluate the SLEEC rules when a context update is detected. SLEEC rules logic is manually implemented following the semantics of [Troquard *et al.*, 2024] and the selected hedge clauses are sent to the *Task Manager Nodes*. On their side, the *Task Manager Nodes* handle the selection of the actions associated with hedge clauses through if–else statements. **CS$-$*libs*$-$*CM*** ("CO-SLEEC minus libraries minus Context Manager node") uses neither the `pysleec` and `sleec4ros` libraries nor the *Context Manager Node* (i.e., Figure 1 except the blue and violet boxes). Instead, the SLEEC rules logic is directly implemented within the *Task Manager Nodes*. Specifically, the system includes only the *Mission Manager Node* and a set of *Task Manager Nodes*. Each Task Manager Node subscribes to domain and condition updates and evaluates them using if–then-else statements to trigger the actions of the default rule or hedge clauses.

Results On correctness (*E1*) of CO-SLEEC, in all 800 tests, the action run by the robot always matched the ground truth.

Concerning efficiency (*E2*), Table 2 reports the measured overhead time statistics across the three system variants. We

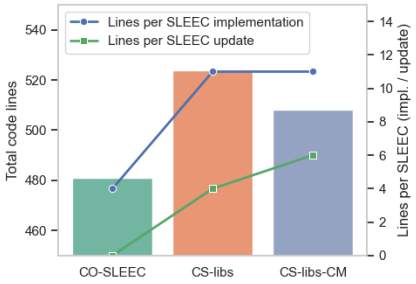

Figure 2: Overview of codelines across different variants

Variant	Mean	Std. dev.	Min	Median	Max
CO-SLEEC	4.81	1.01	3.19	4.60	11.9
CS−*libs*	4.69	0.962	3.11	4.50	10.4
CS−*libs*−*CM*	4.63	0.898	3.07	4.49	10.8

Table 2: Overhead stats (data in ms)

observe that CO-SLEEC adds *negligible overhead* (mean 4.81ms) compared to CS−*libs* (4.69ms) and CS−*libs*−*CM* (4.63ms). Even in the worst case, the measured overhead for CO-SLEEC is below 12ms. The differences in measured overhead among the three variants are statistically significant (Mann-Whitney U test [Conover, 1999], p-value $= 0.014$ for the comparison with CS−*libs*, p-value $= 0.001$ for the comparison with CS−*libs*−*CM*) but have small effect sizes (Cliff's $\delta = 0.071$ and 0.081 [Meissel and Yao, 2024]).

About maintainability (*E3*), Figure 2 reports (i) the total number of LoC in the codebases for each implemented variants, (ii) the number of LoC that the system developer needs to write in order to add a new SLEEC rule into the system, and (iii) the number of LoC that the system developer needs to modify to update an existing SLEEC rule. As shown in Figure 2, CO-SLEEC reduces total LoC by 6–9% compared to both baselines. Adding a new rule requires 4 LoC in CO-SLEEC versus 11 in the ablated variants. In addition, maintaining a CO-SLEEC system when the set of SLEEC rules changes is arguably simple, while both baselines require modifications to conditional logic. In CS−*libs* and CS−*libs*−*CM* edits imply the modification of nested stacks of if-then-else statements in different components of the architecture. This highlights the benefits of our solution in terms of modularity and separation of concerns.

6 Related Work

With the aim of designing socially responsible AI systems, [Townsend *et al.*, 2022] proposed a methodology to elicit SLEEC rules for multi-agent systems. Subsequent research has focused on formalising high-level normative principles into consistent SLEEC rulesets [Yaman *et al.*, 2025; Troquard *et al.*, 2024], supporting automated ethical decision-making. [Troquard *et al.*, 2024] demonstrated how natural-language SLEEC rules can be translated into classical logic for automated normative reasoning. [Getir Yaman *et al.*, 2024; Yaman *et al.*, 2025] introduced the SLEEC-TK toolkit: it provides a DSL for specifying SLEEC rules and tools for validating rulesets and verifying design model conformance in tock-CSP [Miyazawa *et al.*, 2019]. In [Mirani *et al.*, 2026], the authors extend FRET [Katis *et al.*, 2025] to manage SLEEC rule lifecycles and generate Datalog-based inference engines and runtime monitors for obligations. The work in [Anderson and Anderson, 2015] proposes teaching machines to behave ethically by combining agreed-upon general ethical principles with moral decision examples. [Solanki *et al.*, 2023] propose a framework that bridges abstract ethical principles and their practical implementation in healthcare AI, offering actionable guidance across the entire AI lifecycle from data management to deployment. However, there is still a lack of approaches enabling the *operationalisation* of ethical principles and rules in EthicASs.

Context-aware systems [Kapitsaki *et al.*, 2009] adapt based on context. In [Zeng *et al.*, 2018], context serves as key driver for decisions, allowing the argumentation-based framework to adapt its reasoning and ground explanations to the situation. [Banna *et al.*, 2025] present an LLM-based framework for context-aware, personality-adaptive human–robot interaction, showing that integrating user personality traits and contextual gestures greatly enhances communication satisfaction and social engagement. In [Bremner *et al.*, 2019], a layered architecture augments the traditional goal–task–action model [Vanderelst and Winfield, 2018] with an ethical layer. This layer, positioned above the robot controller, leverages a belief-desire-intention (BDI) framework to enable ethical reasoning in autonomous systems. However, operationalising values involves transforming abstract human values into actionable and testable software requirements [Mougouei *et al.*, 2018]. Yet, a definitive approach is still lacking, with most existing efforts providing only abstract solutions with little focus on practical methodologies, concrete implementations, and verifiable metrics [Mittelstadt, 2019].

7 Discussion and Conclusion

To support the implementation of EthicASs, we proposed CO-SLEEC, an approach based on context-oriented design and adaptation, targeting the ROS platform. The approach operationalises SLEEC rules in autonomous systems to foster socially responsible behaviour and transparent adaptation. CO-SLEEC has been evaluated in the context of a multi-service robot operating in two different settings, presenting different SLEEC concerns. Evaluation results show that the approach is correct, efficient, and maintainable.

CO-SLEEC relies on a properly structured coding to preserve the intended semantics of SLEEC obligations when operationalised via COP. While this structure is currently assumed during implementation, future work will explicitly enforce it through static analysis techniques. These mechanisms will enable automatic verification that method implementations conform to the expected structure. From an empirical perspective, future work will consider larger SLEEC rulesets and additional case studies to investigate the scalability of CO-SLEEC and to strengthen the generalisability of the results. Moreover, maintainability is currently assessed using LoC as a proxy measure, which may not fully capture developer effort. Hence, future work will include user studies with developers to provide deeper insights.

Ethical Statement

There are no ethical issues.

Acknowledgments

The authors acknowledge the support of the MUR (Italy) Department of Excellence 2023–2027 for GSSI. For UNIBO, the contribution is partially supported by the Italian Science Fund (FIS3) Starting Grant project FoMaSE – Foundations for Macro-programming-based Software Engineering (Grant No. FIS-2024-00174, CUP J53C25002170001).

References

[AI HLEG, 2019] AI HLEG. Ethics guidelines for trustworthy AI, 2019.

[Anderson and Anderson, 2011] Micheal Anderson and Susan Leigh Anderson, editors. *Machine Ethics*. Cambridge University Press, 2011.

[Anderson and Anderson, 2015] Michael Anderson and Susan Leigh Anderson. Toward ensuring ethical behavior from autonomous systems: a case-supported principle-based paradigm. *Ind. Robot*, 42(4):324–331, 2015.

[Autili *et al.*, 2026] Marco Autili, Martina De Sanctis, Paola Inverardi, Mashal Afzal Memon, Patrizio Pelliccione, and Sara Pettinari. A reference architecture for ethical-aware autonomous systems. *Journal of Systems and Software*, 235:112749, 2026.

[Banna *et al.*, 2025] Tahsin Tariq Banna, Sejuti Rahman, and Mohammad Tareq. Beyond words: Integrating personality traits and context-driven gestures in human-robot interactions. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 242–251, 2025.

[Bremner *et al.*, 2019] Paul Bremner, Louise Dennis, Michael Fisher, and Alan Winfield. On proactive, transparent, and verifiable ethical reasoning for robots. *Proc. IEEE*, 107(3):541–561, 2019.

[Brunero, 2022] John Brunero. Reasons and defeasible reasoning. *The Philosophical Quarterly*, 72(1):41–64, 2022.

[Casadei *et al.*, 2026a] Roberto Casadei, Martina De Sanctis, Gianluca Filippone, Sara Pettinari, Gian Luca Scoccia, and Nicolas Troquard. CO-SLEEC: Replication package, 2026. https://doi.org/10.5281/zenodo.20055346.

[Casadei *et al.*, 2026b] Roberto Casadei, Martina De Sanctis, Gianluca Filippone, Sara Pettinari, Gian Luca Scoccia, and Nicolas Troquard. Leveraging context-oriented programming to implement normative rules in autonomous systems. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2026*. ACM, 2026.

[Chopra *et al.*, 2018] Amit Chopra, Leendert van der Torre, Harko Verhagen, and Serena Villata, editors. *Handbook of Normative Multiagent Systems*. College Publications, London, UK, 2018.

[Conover, 1999] William Jay Conover. *Practical nonparametric statistics*. john wiley & sons, 1999.

[Dignum, 2025] Virginia Dignum. Responsible AI and autonomous agents: Governance, ethics, and sustainable innovation. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2025*, pages 1–2. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2025.

[Elyasaf *et al.*, 2023] Achiya Elyasaf, Nicolás Cardozo, and Arnon Sturm. A framework for analyzing context-oriented programming languages. *Journal of Systems and Software*, 198:111614, 2023.

[Feng *et al.*, 2024] Nick Feng, Lina Marsso, Sinem Getir Yaman, Yesugen Baatartogtokh, Reem Ayad, Victória Oldemburgo de Mello, Beverley A. Townsend, Isobel Standen, Ioannis Stefanakos, Calum Imrie, Genaína Nunes Rodrigues, Ana Cavalcanti, Radu Calinescu, and Marsha Chechik. Analyzing and debugging normative requirements via satisfiability checking. In *International Conference on Software Engineering*, pages 214:1–214:12. ACM, 2024.

[Gabbay *et al.*, 2013] Dov Gabbay, John Horty, Xavier Parent, Ron van der Meyden, and Leendert van der Torre, editors. *Handbook of Deontic Logic and Normative Systems*. College Publications, London, UK, 2013.

[Getir Yaman *et al.*, 2024] Sinem Getir Yaman, Pedro Ribeiro, Charlie Burholt, Maddie Jones, Ana Cavalcanti, and Radu Calinescu. Toolkit for specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents. *Science of Computer Programming*, 236:103118, September 2024.

[Horty, 2012] John Horty. *Reasons as defaults*. Oxford University Press, 2012.

[Kamina *et al.*, 2016] Tetsuo Kamina, Tomoyuki Aotani, Hidehiko Masuhara, and Tetsuo Tamai. *Context-Oriented Software Development with Generalized Layer Activation Mechanism*, page 3–40. Springer International Publishing, 2016.

[Kapitsaki *et al.*, 2009] Georgia Kapitsaki, George Prezerakos, Nikolaos Tselikas, and Iakovos Venieris. Context-aware service engineering: A survey. *Journal of Systems and Software*, 82(8):1285–1297, 2009.

[Katis *et al.*, 2025] Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, and Khanh Trinh. FRET: Formal Requirements Elicitation Tool, 2025.

[Macenski *et al.*, 2022] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), 2022.

[Meissel and Yao, 2024] Kane Meissel and Esther Yao. Using cliff's delta as a non-parametric effect size measure: an accessible web app and r tutorial. *Practical Assessment, Research, and Evaluation*, 29(1), 2024.

[Mirani *et al.*, 2026] Mahrokh Mirani, Paola Inverardi, Patrizio Pelliccione, Franco Raimondi, and Nicolas Troquard. Extending FRET with SLEEC rules: Formalization, obligation inference, and monitoring. In *Tools and Algorithms for the Construction and Analysis of Systems - 32nd International Conference, TACAS 2026*, Lecture Notes in Computer Science, pages 65–84. Springer, 2026.

[Mittelstadt, 2019] Brent Mittelstadt. Principles alone cannot guarantee ethical AI. *Nature machine intelligence*, 1(11):501–507, 2019.

[Miyazawa *et al.*, 2019] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. Robochart: modelling and verification of the functional behaviour of robotic applications. *Software & Systems Modeling*, 18(5):3097–3149, 2019.

[Mougouei *et al.*, 2018] Dena Mougouei, Harsha Perera, Wajeeha Hussain, Rukshan Shams, and Jon Whittle. Operationalizing human values in software: A research roadmap. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*, pages 780–784. ACM, 2018.

[Radanliev, 2025] Petar Radanliev. AI ethics: Integrating transparency, fairness, and privacy in AI development. *Applied Artificial Intelligence*, 39(1), February 2025.

[Salvaneschi *et al.*, 2012] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. Context-oriented programming: A software engineering perspective. *J. Syst. Softw.*, 85(8):1801–1817, 2012.

[Schubert *et al.*, 2015] Christian Schubert, Michael Perscheid, and Samuel Longchamps. Contextpy3 (github repository), 2015.

[Solanki *et al.*, 2023] Pravik Solanki, John Grundy, and Waqar Hussain. Operationalising ethics in artificial intelligence for healthcare: a framework for AI developers. *AI Ethics*, 3(1):223–240, 2023.

[Soler Garrido *et al.*, 2024] Josep Soler Garrido, Sarah De Nigris, Elias Bassani, Ignacio Sanchez, Tatjana Evas, Antoine-Alexandre André, and Thierry Boulangé. Harmonised standards for the European AI Act. Technical report, Seville (Spain), 2024.

[Townsend *et al.*, 2022] Beverley Townsend, Colin Paterson, T. T. Arvind, Gabriel Nemirovsky, Radu Calinescu, Ana Cavalcanti, Ibrahim Habli, and Alan Thomas. From pluralistic normative principles to autonomous-agent rules. *Minds and Machines*, 32(4):683–715, 2022.

[Troquard *et al.*, 2024] Nicolas Troquard, Martina De Sanctis, Paola Inverardi, Patrizio Pelliccione, and Gian Luca Scoccia. Social, legal, ethical, empathetic, and cultural rules: Compilation and reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(20):22385–22392, March 2024.

[UNESCO, 2022] UNESCO. Recommendation on the ethics of artificial intelligence, 2022.

[Vanderelst and Winfield, 2018] Dieter Vanderelst and Alan Winfield. An architecture for ethical robots inspired by the simulation theory of cognition. *Cognitive Systems Research*, 48:56–66, 2018.

[Yaman *et al.*, 2023] Sinem Getir Yaman, Charlie Burholt, Maddie Jones, Radu Calinescu, and Ana Cavalcanti. Specification and validation of normative rules for autonomous agents. In *International Conference on Fundamental Approaches to Software Engineering*, pages 241–248. Springer Nature Switzerland Cham, 2023.

[Yaman *et al.*, 2025] Sinem Yaman, Pedro Ribeiro, Ana Cavalcanti, Radu Calinescu, Colin Paterson, and Beverley Townsend. Specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents. *Journal of Systems and Software*, 220:112229, February 2025.

[Zeng *et al.*, 2018] Zhiwei Zeng, Xiuyi Fan, Chunyan Miao, Cyril Leung, Jing Jih Chin, and Yew-Soon Ong. Context-based and explainable decision making with argumentation. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018*, pages 1114–1122, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems / ACM.