A Process to Enforce Ethical Requirements of Autonomous Systems at Runtime

Martina De Sanctis
Gran Sasso Science Institute
L'Aquila, Italy
martina.desanctis@gssi.it

Gianluca Filippone
Gran Sasso Science Institute
L'Aquila, Italy
gianluca.filippone@gssi.it

Paola Inverardi
Gran Sasso Science Institute
L'Aquila, Italy
paola.inverardi@gssi.it

Raffaela Mirandola
Karlsruhe Institute of Technology
Karlsruhe, Germany
raffaela.mirandola@kit.edu

Sara Pettinari
Gran Sasso Science Institute
L'Aquila, Italy
sara.pettinari@gssi.it

Patrizia Scandurra
University of Bergamo
Bergamo, Italy
patrizia.scandurra@unibg.it

Abstract

This paper addresses the challenge of enforcing ethical behaviors at runtime in autonomous systems through a structured ethics assurance process. At the core of this process is a *subsystem* that rigorously operationalizes ethical rules, specifically social, legal, ethical, empathetic, and cultural (SLEEC) requirements. This subsystem enables the dynamic evaluation, adaptation, and enforcement of ethically compliant behavior within a formally defined runtime model. The proposed approach, named `SLEEC@run.time`, is demonstrated through a running example involving firefighter-uncrewed aerial vehicles. In addition, by leveraging its flexible runtime model, `SLEEC@run.time` accommodates changes such as the addition or removal of SLEEC rules, ensuring a robust and evolvable approach to ethical assurance in autonomous systems.

CCS Concepts

• **Software and its engineering**; • **Computer systems organization → Robotic autonomy**;

Keywords

SLEEC rules, Ethics Enforcement, Models@run.time, Autonomous Systems

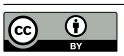
ACM Reference Format:
Martina De Sanctis, Gianluca Filippone, Paola Inverardi, Raffaela Mirandola, Sara Pettinari, and Patrizia Scandurra. 2026. A Process to Enforce Ethical Requirements of Autonomous Systems at Runtime. In *21st International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3788550.3794876

1 Introduction

As AI and autonomous systems become more widespread, concerns are growing about the potential harm that could result from their choices and behavior [11]. In this context, autonomous agents should act transparently in accordance with recognized ethical norms, guidelines, and principles [23, 29], possibly adapting their behavior ethically to different users and contexts. Townsend et al. [27] introduced a methodology for eliciting *social, legal, ethical, emphathetic, and cultural (SLEEC) rules* in autonomous systems. Building on this, recent studies (e.g., [14, 15, 19, 28, 30]) have proposed methods to translate high-level normative principles into explicit, formalized, and consistent SLEEC rulesets, supporting automated and informed decision-making in autonomous systems. However, while the existing work focuses primarily on the elicitation, formalization, validation, and verification of SLEEC rules, there is still a lack of contributions addressing their *operationalisation* at later stages, such as implementation and testing, that remain largely unexplored [26]. Specifically, we refer to solutions for translating ethical principles into concrete designs and implementations that can guide the runtime behavior of autonomous systems.

Building on these premises, this paper addresses the enforcement of ethical requirements (in the form of SLEEC rules) at runtime in autonomous systems. We present `SLEEC@run.time`, an ethics assurance process, for real autonomous systems and their environments, that covers all phases from the elicitation of ethics requirements to and throughout their runtime operation and evolution. In particular, to enable an autonomous system to satisfy a set of ethics requirements, we propose a runtime ethics enforcement approach that steers the system to behave ethically. The process considers the formal specification of SLEEC rules in terms of an *Abstract State Machine* (ASM) [8], provided as runtime/living model to an enforcement subsystem that leverages the ASMETA runtime simulator [6] for the model execution. Taking inspiration from [7, 12], `SLEEC@run.time` features the design and development of an ethics enforcement subsystem as an autonomic manager that wraps around the autonomous system with a MAPE-K (Monitor-Analyse-Plan-Execute over a Knowledge base) control loop architecture [18]. Moreover, the SLEEC rules are easily modifiable and adaptable, making evolutionary steps in the system straightforward to accommodate.

The rest of the paper is organized as follows: Section 2 presents the background of this work including related works. Section 3 describes the firefighter uncrewed aerial vehicle reference scenario. Section 4 presents the ethics assurance process. Finally, Section 5 concludes the work.



SEAMS '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2445-9/2026/04
https://doi.org/10.1145/3788550.3794876

2 Background

This section presents the ethical and theoretical foundations that motivate and support our enforcement process. It also presents related work to position our contribution with respect to the state of the art.

2.1 Social, Legal, Ethical, Emphathetic, and Cultural (SLEEC) Rules

In response to the concerns surrounding the behavior and impact of AI and autonomous systems, experts such as philosophers, lawyers, engineers, and ethicists are investigating new ways to guide and regulate these technologies. One approach to maintaining oversight involves formulating ethical principles as rules shaping how such systems operate and engage in interactions with humans. To support the design of systems with this capability, Townsend et al. [27] introduced the concept of *social, legal, ethical, empathetic, and cultural (SLEEC) rules*, and a methodology to elicit actionable constraints on the system behavior from high-level norms and principles. SLEEC rules delineate how the system should respond to specific contextual conditions encountered during execution, rather than determining the system's overarching objectives or behaviors. Rules are elicited by a group of SLEEC experts, built as a collaborative engagement of ethicists, philosophers, lawyers, and other domain experts [27]. Notably, SLEEC rules are defined with respect to the capabilities of the autonomous system, the application domain, and the stakeholders involved. Hence, SLEEC rules do not encode individual user preferences or personal moral judgments; rather, they reflect socially recognized values and norms. A SLEEC rule is made up of a single *default rule* possibly followed by one or more *hedge clauses* (triggered by "defeating conditions" [9, 17]). 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 [31]. However, a rule can be defeated or overcome by hedge clauses [27]. These are specified by means of the "`UNLESS` *condition* `IN WHICH CASE` *obligation*" construct. Each hedge clause specifies a condition in which the original response should be preempted, and there is an obligation to perform an alternate response, thus enabling ethical-oriented reasoning. Hedge clauses are meant to support SLEEC principles, including human dignity, autonomy, beneficence, non-maleficence, privacy, and cultural sensitivity. They do so by either prioritizing or limiting these principles depending on the context and promoting the user's well-being, thereby guiding the autonomous system's behavior to accommodate these principles.

A SLEEC rule governing an autonomous system's response to contextual changes will look like the following[1]:

`WHEN` C_0 `THEN` O_0
`UNLESS` C_1 `IN WHICH CASE` O_1 ①
`UNLESS` C_2 `IN WHICH CASE` O_2 ②

In this example, according to the default rule, when the condition C_0 is verified, the system should fulfill the obligation O_0. However, two context-dependent hedge clauses are present: ① and ②. The order in which hedge clauses are listed specifies their priority. In this work, we follow the evaluation order of linguistically justified rules as proposed in [28], according to which hedge clauses are evaluated in a top-down manner, with the last one taking priority over the others. Hence, referring to the previous example, the clause ① would be evaluated first, followed by the evaluation of clause ②. Assuming that $C_0 \wedge C_1$ are verified while C_2 is evaluated to false, then the clause ① would be activated and O_1 must be fulfilled. Whereas, if $C_0 \wedge C_1 \wedge C_2$ is evaluated to true, then the clause ② would be activated and O_2 must be fulfilled, as C_2 is the last condition that evaluates to true. A concrete example is given in Section 3.

2.2 Formal Models @runtime

Formal Models@run.time[3, 10] are used to maintain a formal, up-to-date representation of a software system during its execution. This live model serves as a foundation for monitoring, analyzing, and adapting the system at runtime to ensure system requirements.

Among runtime assurance techniques, *runtime enforcement*[7, 12] extends runtime verification monitors [32] by actively intervening to correct deviations. Runtime enforcement is crucial for systems where safety and security are paramount, such as medical devices and security-critical control access software (see approaches [7, 22, 24], to name a few). An *enforcement monitor* (or simply *enforcer*) is a component that observes the system's execution and modifies the system's behavior to ensure compliance with a predefined formal specification (*enforcement model*) of the enforcement strategy. Once validated/verified at design-time, the enforcement model can be deployed and used as runtime model [3, 10] by the enforcer. An example of such an approach is provided by the runtime enforcement framework presented in [7] for safety properties, where an *Abstract State Machine* (ASM) [8][2] is adopted as runtime enforcement model. The ASM is executed at runtime in tandem with the target system by the simulation engine of ASMETA [6], a suite of methods and tools based on the formal method ASM for the specification, validation, and verification of the behavior of discrete-event systems.

2.3 Related Work

In their vision paper, Boltz et al. [5] examine how self-adaptive socio-technical systems can be designed to empower humans while respecting diverse needs, values, and ethics. They emphasize the importance of human empowerment and ethical balance at individual, community, and societal levels. Specifically, the authors propose a high-level architectural framework to guide interactions and preserve human values.

Ethical oversight and well-designed regulatory structures are crucial for creating autonomous systems that act responsibly, maintain transparency, and reflect human values [11]. With this aim, Townsend et al. [27] proposed a methodology to elicit SLEEC rules as ethical requirements for autonomous systems. Since their introduction, multiple researchers have focused on formalizing high-level normative principles into consistent SLEEC rulesets, supporting automated ethical decision-making. Troquard et al. [28] showed how natural-language SLEEC rules can be converted into classical logic to enable automated normative reasoning. Mirani et al. [20]

[1] In [30], hedge clauses obligations are preceded by "THEN" instead of "IN WHICH CASE", while here we adopt the syntax used in [27].

[2] An ASM state is a first-order structure (an algebra), and ASM rules define state transitions by updating function interpretations.

propose encoding SLEEC rules in Datalog, a declarative logic programming language, to enable scalable computation of obligations. Yaman et al. introduced the SLEEC-TK toolkit [15, 30], which provides a DSL for specifying SLEEC rules and tools for validating rulesets and verifying design model conformance in tock-CSP [21]. Complementarily, Feng et al. [14] propose an alternative approach using Large Language Models (LLMs) to enhance automated reasoning techniques to better elicit, analyze, and ensure consistency of normative requirements. Kolyakov et al. [19] introduced LEGOS-SLEEC, a tool that assists interdisciplinary stakeholders in defining normative requirements as SLEEC rules and in checking and debugging their consistency. However, all these approaches remain limited to the elicitation, validation, and verification of SLEEC rules, without operationalizing them, namely, bringing them into the design, implementation, and execution within autonomous systems, supporting ethical decision-making.

Many approaches emphasize the importance of incorporating human values into software development. The survey in [26] notes that while current methods facilitate values operationalization during early stages (requirements and design), later stages, such as implementation and testing, remain underexplored. Bennaceur et al. [4] proposed *values@runtime*, an adaptive MAPE-K framework that helps users align decisions with their personal values, detect mismatches, receive recommendations, and reflect on behavior, illustrated via a shopping basket tool. Complementarily, our proposal focuses on adapting the system's behavior to align with ethical principles, rather than recommending alternative actions to users.

3 Running Example

As a running example, we consider here the firefighter uncrewed aerial vehicle (UAV) scenario from [30]. The firefighter UAV is supposed to help tackle wildfires and urban fires by interacting with human firefighters, bystanders and teleoperators. Its primary tasks are: detecting a potential warehouse fire using a thermal camera, localizing it via a depth camera and reporting it by sending video footage of the surveyed building to teleoperators, and deploying onboard water spraying to contain it until human firefighters arrive. Beyond these functional goals, the firefighter UAV must address SLEEC concerns arising from interactions with humans. For instance, an onboard loudspeaker alarm may raise social issues if triggered near people, and transmitting video footage to teleoperators may introduce legal or ethical privacy concerns when bystanders are present.

For the sake of presentation, we illustrate a representative rule of the firefighter UAV in Listing 1. In this rule, *CameraStart* denotes a teleoperator command to activate the camera and start recording, *SoundAlarm* triggers an onboard loudspeaker alarm, and *GoHome* represents the UAV's autonomous navigation capability to return to its home location. *CameraStart* normally leads to *SoundAlarm* (default rule). However, there are two hedge clauses: ① and ②. If *personNearby* is true, then the rule requires the UAV to go home (clause ① – *social* concern), so as to avoid the anti-social action of sounding an alarm near a person, likely a human firefighter. This is, however, defeated by clause ②: if the measured temperature is greater than 35 degrees Celsius (that is interpreted as evidence of a nearby fire), no normative response applies.

```
WHEN cameraStart THEN soundAlarm
UNLESS personNearby IN WHICH CASE goHome ①
UNLESS temperature > 35.0 ②
```

Listing 1: Firefighter UAV - respresentative SLEEC rule

The complete set of SLEEC rules for the firefighter UAV is available online at https://gssi-robotics.github.io/sleec-at-runtime/.

In general, a SLEEC rule allows the specification of multiple defeaters together with time constraints and timeouts [30]. This expressiveness makes the rule semantics non-trivial, and interactions between rules can yield unexpected outcomes.

4 `SLEEC@run.time` Ethics Assurance Process

This section presents the `SLEEC@run.time` ethics assurance process, spanning from the inception of SLEEC rules to and throughout their operation in the real environment. Figure 1 illustrates the core lifecycle phases of the ethics assurance process, spanning both the *offline* (`@design.time`) and *online* (`@run.time`) stages, from inception to system operation. The phases are color-coded to reflect their status within the proposed process: grey denotes phases that are established in the existing literature, green highlights phases that are newly introduced, while a grey-green striped pattern denotes an existing phase that has been extended to fit within the proposed process.

The process unfolds across four main phases. The first three phases, (1) *rule elicitation*, (2) *rule formal specification and analysis*, and (3) *Enforcement Subsystem development* are associated to the `@design.time` stage, while the (4) *Enforcement Subsystem operation* composes the `@run.time` stage. These four phases are executed sequentially, while, throughout the process, actions can be performed to revisit and evolve the SLEEC ruleset. This evolution step may be triggered by symptoms or signals emerging from any of the above phases, prompting reassessment and refinement of the normative framework. After rule reassessment, a refinement flow informs both the design and operational phases (phases (2) and (4), respectively), ensuring the system remains aligned with evolving normative expectations. In the following, we detail the four phases.

4.1 Rule elicitation process

The rule elicitation process aims at deriving the relevant SLEEC rules for the system domain, stakeholders, and context. It builds on the iterative process defined in [27], which allows deriving SLEEC rules aligned with system capabilities and ethical values. The rule elicitation process starts with the identification of high-level ethical norms and principles relevant to the application context, drawing from established moral philosophy and normative frameworks (e.g., *benevolence*, *non-maleficence*, *autonomy*). These principles are then mapped to the system capabilities, enabling the identification of initial ethical touch points where *base rules* can be formulated. Such rules are examined with domain experts and stakeholders to identify additional SLEEC concerns and potential conflicts. Rules are refined through the introduction of *hedge clauses* or new rules to manage newly identified concerns or conflicts. This refinement process is iterative, with rules re-evaluated until all the identified

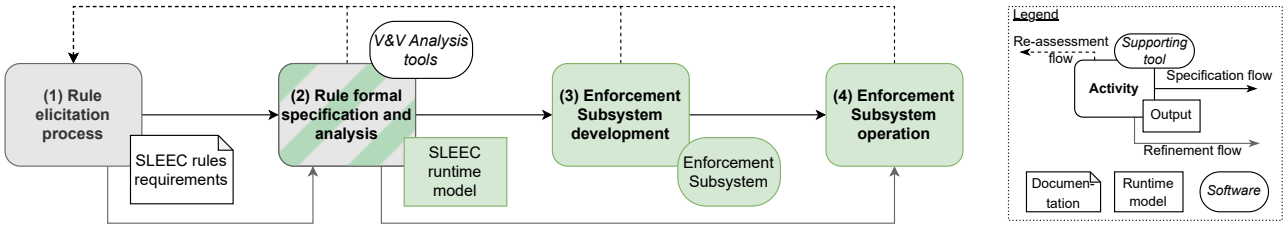


Figure 1: `SLEEC@run.time` ethics assurance process.

SLEEC concerns are addressed and all the conflicts are resolved. In this way, the SLEEC ruleset obtained through this process is consistent and conflict-free, e.g., there are no multiple rules that require the system to perform conflicting operations at the same time. The process terminates once a stable set of rules has been obtained.

4.2 Rule formal specification and analysis

To support the interdisciplinary stakeholders in specifying and analyzing the well-formedness of SLEEC requirements, a formal method for the specification and validation of SLEEC rules, is necessary.

For this purpose, any formal method that supports the defeasible logic [9, 17] underlying SLEEC rules and is capable of detecting conflicts and redundancies can be employed. Prior work has explored several approaches for formalizing SLEEC requirements, including the process algebra tock-CSP [21, 30], First-Order Logic with relational objects (FOL) [13, 19], Communicating Sequential Processes [16], as well as classical logic and logic programming [28]. The results of this formal consistency validation are then fed back to SLEEC experts for further refinement and adjustment of the requirements, potentially through multiple iterative cycles.

In our case, we chose ASMs [8] as it is an executable formalism and has proven effective for reasoning about consistency and conflict analysis in decentralized, multiple interacting MAPE-K loop architectures [1, 2], and for the formal specification and validation of functional requirements derived from use case models [25]. Moreover, ASMs are well suited to be adopted as formal models at runtime [7]. This latter aspect is fundamental for the implementation and operationalization of SLEEC requirements, complementing their elicitation and formal analysis, as pursued in our approach. More specifically, the set of SLEEC rules elicited in the previous phase is formally specified in ASM through ASMETA Language (AsmetaL), possibly using a model compiler from SLEEC requirements to AsmetaL[3] and subsequently analyzed using the ASMETA V&V techniques (including model animation, validation, verification, and review) [6]. This formal analysis ensures consistency, correctness, completeness, and minimality of the specification. The resulting SLEEC model is thus both formally validated and directly executable, constituting a ready-to-use *SLEEC runtime model*.

[3]ASMs supports the use of defeasible logic; rule defeaters are specified in terms of nested if-then-else statements with short-circuit priority of logic programming as in [28].

An excerpt of a SLEEC runtime model defining the SLEEC from the running example (Listing 1) in AsmetaL is presented in Listing 2. The SLEEC computation pattern, which follows the execution semantics prescribed in [28], is represented by the rule constructor r_SLEEC defined in AsmetaL as follows:

```
//SLEEC rule constructor for two hedge clauses
rule r_SLEEC($c0 in Boolean, $o0 in Rule, $c1 in Boolean, $o1 in Rule,
              $c2 in Boolean, $o2 in Rule) =
  if $c0 and not $c1 then $o0
  else if $c0 and $c1 and not $c2 then $o1
  else if $c0 and $c1 and $c2 then $o2 endif endif endif
```

where the boolean variable `c0` is the SLEEC's trigger condition, the boolean variables `c1` and `c2` are defeating conditions, the rule `o0` corresponds to the base obligation rule, and the rules `o1` and `o2` correspond to defeating obligations rules. These obligation rules are used to set, via the utility rule `r_setObligation`, the output function `outObligation($c)` of the SLEEC model to `true`, where `$c` ranges over `Capability`, an abstract domain whose elements represent the target agent's capabilities (e.g., `r_skip`, `r_soundAlarm`, `r_goHome` in Listing 2).

```
...
// obligations definition
rule r_skip = skip //no ASM state change (no prescribed obligation)
rule r_soundAlarm = r_setObligation[soundAlarm]
rule r_goHome = r_setObligation[goHome]
...
// rule definition
rule r_Rule = r_SLEEC[cameraStart, <<r_soundAlarm>>,
                      personNearby, <<r_goHome>>,
                      temperature > 35.0, <<r_skip>>]
```

Listing 2: Excerpt of a SLEEC model in ASMETA language

4.3 Enforcement Subsystem development

To manage the runtime enforcement of the SLEEC rules specified in the previous phase, the ethics assurance process prescribes the development of an *Enforcement Subsystem*. This subsystem represents an autonomic manager which leverages the SLEEC runtime model and wraps around the autonomous system to enforce its ethical behavior by following the *enforcement by monitoring and adaptation* pattern [7].

Figure 2 shows the high-level architecture of the Enforcement Subsystem. It features a MAPE-K loop that is distributed among

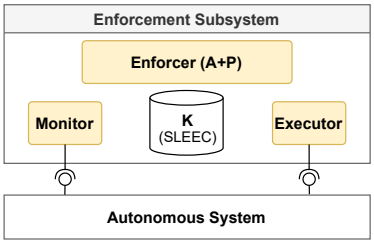


Figure 2: Enforcement Subsystem architecture.

three different components (*Monitor*, *Enforcer*, and *Executor*) and a shared *Knowledge*. The Knowledge consists of the SLEEC ruleset defined as an ASM runtime model. The *Monitor* component is in charge of acquiring the runtime conditions by leveraging the autonomous system's perception capabilities accessible through interfaces offered by the system. Once acquired, the Monitor translates the low-level sensor data into higher-level conditions relevant for the SLEEC rules evaluation, hence linking the data acquired from the system's perception layer to the evaluated rules conditions. The *Enforcer* component realizes the *Analyze* and *Plan* activities of the MAPE-K loop. It is responsible for detecting the rule's trigger conditions and formulating corrective actions in terms of SLEEC obligations to be enforced. These activities are performed by executing the SLEEC runtime model within the ASMETA simulation engine [6] embedded in the Enforcer component, which outputs the obligations resulting from the model execution. The *Executor* component translates these obligations into a set of concrete, executable tasks which are sent to the autonomous system through its interfaces.

4.4 Enforcement Subsystem operation

Once developed, the Enforcement Subsystem is deployed and connected to the targeted autonomous system. The SLEEC runtime model is therefore uploaded within the Enforcer component (cf. Figure 2) running the ASMETA engine, and the SLEEC runtime model is operationalized. Upon a condition change caught by the Monitor component, the Enforcer triggers the execution of the uploaded model to compute the obligations to be enforced. This enables a dynamic monitoring of the system status, to make behavioral adjustments as prescribed by SLEEC rules to ensure ethical compliance during system execution. Note that whenever a SLEEC runtime model is refined, the new version is uploaded to the running subsystem to enable the continuous evolution and adherence to ethical requirements.

5 Conclusion and Future Work

This paper presents the challenge of enforcing ethical principles at runtime in autonomous systems. It introduces `SLEEC@run.time`, an ethics assurance process that spans from the elicitation of ethical requirements (formalized as SLEEC rules) to their enactment during system operation. Leveraging the Abstract State Machine formalism and its supporting toolset, the approach enables the dynamic evaluation, adaptation, and enforcement of ethical behavior.

Further directions concern the application of `SLEEC@run.time` in multiple heterogeneous scenarios, with the involvement of different stakeholders to assess the actionability of the ethics assurance process.

Data Availability Statement

A replication package containing the SLEEC requirements, SLEEC model in ASMETA, all software artifacts, and data sets used to evaluate our approach is available online at https://gssi-robotics.github.io/sleec-at-runtime/.

Acknowledgments

This work has been partially funded by (a) the MUR (Italy) Department of Excellence 2023 - 2027, (b) the PRIN project P2022RSW5W - RoboChor: Robot Choreography, (c) the PRIN project 2022JKA4SL - HALO: etHical-aware AdjustabLe autOnomous systems, (d) the Helmholtz Association (HGF) with the KiKIT project, (e) the HGF Grant 46.23 (Engineering Secure Systems), and (f) the PRIN 2022 PNRR project SAFEST: truSt Assurance of digital twins For mEdical cyber-phySical sysTems (G53D23002770006 and F53D23004230006).

References

[1] Paolo Arcaini, Raffaela Mirandola, Elvinia Riccobene, and Patrizia Scandurra. 2020. MSL: A pattern language for engineering self-adaptive systems. *Journal of Systems and Software* 164 (2020), 110558. doi:10.1016/j.jss.2020.110558

[2] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, Paola Inverardi and Bradley R. Schmerl (Eds.). IEEE Computer Society, 13–23. doi:10.1109/SEAMS.2015.10

[3] Nelly Bencomo, Sebastian Götz, and Hui Song. 2019. Models@run.time: a guided tour of the state of the art and research challenges. *Softw. Syst. Model.* 18, 5 (Oct. 2019), 3049–3082. doi:10.1007/s10270-018-00712-x

[4] Amel Bennaceur, Diane Hassett, Bashar Nuseibeh, and Andrea Zisman. 2023. Values@Runtime: An Adaptive Framework for Operationalising Values. In *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Society, SEIS@ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 175–179. doi:10.1109/ICSE-SEIS58686.2023.00024

[5] Nicolas Boltz, Sinem Getir Yaman, Paola Inverardi, Rogério de Lemos, Dimitri Van Landuyt, and Andrea Zisman. 2024. Human empowerment in self-adaptive sociotechnical systems. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2024, Lisbon, Portugal, April 15-16, 2024*, Luciano Baresi, Xiaoxing Ma, and Liliana Pasquale (Eds.). ACM, 200–206. doi:10.1145/3643915.3644082

[6] Andrea Bombarda, Silvia Bonfanti, Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. 2024. ASMETA Tool Set for Rigorous System Design. In *Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14934)*. Springer, 492–517. doi:10.1007/978-3-031-71177-0_28

[7] Silvia Bonfanti, Elvinia Riccobene, and Patrizia Scandurra. 2023. A component framework for the runtime enforcement of safety properties. *J. Syst. Softw.* 198 (2023), 111605. doi:10.1016/J.JSS.2022.111605

[8] Egon Börger and Robert Stärk. 2003. *Abstract State Machines*. Springer Berlin Heidelberg. doi:10.1007/978-3-642-18216-7

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

[10] Radu Calinescu and Shinji Kikuchi. 2011. Formal Methods @ Runtime. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, Radu Calinescu and Ethan Jackson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–135. doi:10.1007/978-3-642-21292-5_7

[11] Virginia Dignum. 2025. 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, Detroit, MI, USA, May 19-23, 2025*. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 1–2. doi:10.5555/3709347.3743508

[12] Yliès Falcone, Leonardo Mariani, Antoine Rollet, and Saikat Saha. 2018. *Runtime Failure Prevention and Reaction*. Springer International Publishing, Cham, 103–134. doi:10.1007/978-3-319-75632-5_4

[13] Nick Feng, Lina Marsso, and Marsha Chechik. 2024. Diagnosis via Proofs of Unsatisfiability for First-Order Logic with Relational Objects. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, 1521–1532. doi:10.1145/3691620.3695522

[14] Nick Feng, Lina Marsso, Sinem Getir Yaman, Isobel Standen, Yesugen Baatartogtokh, Reem Ayad, Victória Oldemburgo de Mello, Beverley A. Townsend, Hanne Bartels, Ana Cavalcanti, Radu Calinescu, and Marsha Chechik. 2024. Normative Requirements Operationalization with Large Language Models. In *32nd IEEE International Requirements Engineering Conference, RE 2024, Reykjavik, Iceland, June 24-28, 2024*, Grischa Liebel, Irit Hadar, and Paola Spoletini (Eds.). IEEE, 129–141. doi:10.1109/RE59067.2024.00022

[15] Sinem Getir Yaman, Pedro Ribeiro, Charlie Burholt, Maddie Jones, Ana Cavalcanti, and Radu Calinescu. 2024. Toolkit for specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents. *Science of Computer Programming* 236 (Sept. 2024), 103118. doi:10.1016/j.scico.2024.103118

[16] Sinem Getir Yaman, Pedro Ribeiro, Charlie Burholt, Maddie Jones, Ana Cavalcanti, and Radu Calinescu. 2024. Toolkit for specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents. *Science of Computer Programming* 236 (2024), 103118. doi:10.1016/j.scico.2024.103118

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

[18] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (Jan. 2003), 41–50. doi:10.1109/MC.2003.1160055

[19] Kevin Kolyakov, Lina Marsso, Nick Feng, Junwei Quan, and Marsha Chechik. 2025. LEGOS-SLEEC: Tool for Formalizing and Analyzing Normative Requirements. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 33–36. doi:10.1109/ICSE-Companion66252.2025.00018

[20] Mahrokh Mirani, Franco Raimondi, and Nicolas Troquard. 2024. Towards Efficient Norm-Aware Robots' Decision Making Using Datalog (Short Paper). In *Proceedings of the 3rd Workshop on Bias, Ethical AI, Explainability and the role of Logic and Logic Programming co-located with the 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), Bolzano, Italy, November 26, 2024 (CEUR Workshop Proceedings, Vol. 3881)*. CEUR-WS.org, 50–59. https://ceur-ws.org/Vol-3881/paper6.pdf

[21] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. 2019. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Software & Systems Modeling* 18, 5 (Jan. 2019), 3097–3149. doi:10.1007/s10270-018-00710-z

[22] Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard Von Hanxleden. 2017. Runtime Enforcement of Cyber-Physical Systems. *ACM Transactions on Embedded Computing Systems* 16, 5s, Article 178 (Sept. 2017), 25 pages. doi:10.1145/3126500

[23] Petar Radanliev. 2025. AI Ethics: Integrating Transparency, Fairness, and Privacy in AI Development. *Applied Artificial Intelligence* 39, 1 (Feb. 2025). doi:10.1080/08839514.2025.2463722

[24] Oliviero Riganelli, Daniela Micucci, and Leonardo Mariani. 2019. Controlling Interactions with Libraries in Android Apps Through Runtime Enforcement. *ACM Transactions on Autonomous and Adaptive Systems* 14, 2 (2019), 8:1–8:29. doi:10.1145/3368087

[25] Patrizia Scandurra, Andrea Arnoldi, Tao Yue, and Marco Dolci. 2012. Functional requirements validation by transforming use case models into Abstract State Machines. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (Trento, Italy) *(SAC '12)*. Association for Computing Machinery, New York, NY, USA, 1063–1068. doi:10.1145/2245276.2231942

[26] Mojtaba Shahin, Waqar Hussain, Arif Nurwidyantoro, Harsha Perera, Rifat Shams, John Grundy, and Jon Whittle. 2022. Operationalizing human values in software engineering: A survey. *IEEE Access* 10 (2022), 75269–75295.

[27] Beverley Townsend, Colin Paterson, T. T. Arvind, Gabriel Nemirovsky, Radu Calinescu, Ana Cavalcanti, Ibrahim Habli, and Alan Thomas. 2022. From Pluralistic Normative Principles to Autonomous-Agent Rules. *Minds and Machines* 32, 4 (Oct. 2022), 683–715. doi:10.1007/s11023-022-09614-w

[28] Nicolas Troquard, Martina De Sanctis, Paola Inverardi, Patrizio Pelliccione, and Gian Luca Scoccia. 2024. Social, Legal, Ethical, Empathetic, and Cultural Rules: Compilation and Reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 20 (March 2024), 22385–22392. doi:10.1609/aaai.v38i20.30245

[29] UNESCO. 2022. Recommendation on the Ethics of Artificial Intelligence. https://www.unesco.org/en/artificial-intelligence/recommendation-ethics

[30] Sinem Yaman, Pedro Ribeiro, Ana Cavalcanti, Radu Calinescu, Colin Paterson, and Beverley Townsend. 2025. Specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents. *Journal of Systems and Software* 220 (Feb. 2025), 112229. doi:10.1016/j.jss.2024.112229

[31] Sinem Getir Yaman, Charlie Burholt, Maddie Jones, Radu Calinescu, and Ana Cavalcanti. 2023. Specification and Validation of Normative Rules for Autonomous Agents. In *Fundamental Approaches to Software Engineering*, Leen Lambers and Sebastián Uchitel (Eds.). Springer Nature Switzerland, Cham, 241–248. doi:10.1007/978-3-031-30826-0_13

[32] Xian Zhang, Martin Leucker, and Wei Dong. 2012. Runtime Verification with Predictive Semantics. In *NASA Formal Methods*, Alwyn E. Goodloe and Suzette Person (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 418–432. doi:10.1007/978-3-642-28891-3_37