# SLEECinFRET: A Tool to Manage Social, Legal, Ethical, Empathetic, and Cultural Requirements

Demonstration Track

Mahrokh Mirani
Gran Sasso Science Institute
L'Aquila, Italy
mahrokh.mirani@gssi.it

Franco Raimondi
Gran Sasso Science Institute
L'Aquila, Italy
franco.raimondi@gssi.it

Nicolas Troquard
Gran Sasso Science Institute
L'Aquila, Italy
nicolas.troquard@gssi.it

## KEYWORDS

Formal Requirement Elicitation Tool (FRET); SLEEC Requirements; Requirements Life Cycle (RLC); Requirement Analysis

**ACM Reference Format:**
Mahrokh Mirani, Franco Raimondi, and Nicolas Troquard. 2026. SLEECinFRET: A Tool to Manage Social, Legal, Ethical, Empathetic, and Cultural Requirements: Demonstration Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/XWPK8237

## 1 INTRODUCTION

Formal Requirement Elicitation Tool (FRET) is a tool for formalizing and managing requirements, especially for safety-critical systems. It prioritizes usability by offering GUI features and a domain-specific language, FRETish, close to natural language that removes ambiguity, to be used even by engineers and domain experts without strong formal methods backgrounds. Beyond formalization, it supports automated future and past time metric LTL encoding of requirements, realizability checking, simulation, and test-case generation [3, 4].

FRET has been used to manage requirements for safety-critical applications in avionics, healthcare, space exploration, etc. In these areas, multi-agent systems based on AI and ML are increasingly common, creating a growing demand for mechanisms to manage concerns related to these techniques, also driven by regulations such as the EU AI Act [7].

SLEEC rules were introduced by Townsend et al. to capture social, legal, ethical, empathetic, and cultural requirements [10], providing domain experts with a methodology to elicit these requirements. SLEEC rules were proposed as an instrument to ensure that the goals and behaviours of autonomous agents "can be assured to align with human values through their operation" [2]. The specification format consists of an obligation under a condition, with an arbitrary number of defeaters that override the obligation under more specific conditions, as common in normative requirements. It was intended that experts should be assisted during elicitation with technical means to check for consistency, non-redundancy, etc. Tools offering these services were already proposed [1, 12]. In particular, SLEECVAL and LEGOS-SLEEC are end-to-end software systems designed to support experts in SLEEC rule elicitation.

**Contribution.** We extend FRET to support the management of SLEEC rules alongside "standard" requirements, following requests from industry and project partners developing agent-based safety-critical systems. Our extension enables:
(1) the use of several temporal scopes and timing constraints in SLEEC rules. This goes beyond the 'within' timing constraint supported by existing tools (see Section 2);
(2) to inherit all the functionalities of FRET for requirement elicitation. This goes significantly beyond the support for well-formedness properties checking from existing tools.
In particular, our tool also supports realizability checking, simulation, and test-case generation for SLEEC requirements. The GUI also provides access to FRET's analytical tools and dashboards, and the management of the life-cycle of requirements, managing states such as "in progress", "paused", "deprecated", etc.

## 2 SLEEC REQUIREMENTS IN FRET

We extend the SLEEC syntax of [13] by introducing a scope and more timing constraints. Scopes can be one of[1]: in, before, after, notIn, onlyIn, onlyBefore, or onlyAfter, followed by some condition. Timings can be one of: immediately, next, always, never, eventually, until, before, for, within, or after, followed by some condition.

A SLEEC requirement with $n$ defeaters has therefore the form:

[SCOPE mode] IF $C_0$ THEN $[T_0]$ $O_0$,
UNLESS $C_1$ IWC $[T_1]$ $O_1$,
...
UNLESS $C_n$ IWC $[T_n]$ $O_n$

IWC stands for 'in which case'. Following [11], this requirement is translated into $n+1$ FRET requirements as follows:

1. [SCOPE mode] Whenever ($C_0$ & !$C_1$)
Component shall $[T_0]$ satisfy $O_0$
2. [SCOPE mode] Whenever ($C_0$ & $C_1$ & !$C_2$)
Component shall $[T_1]$ satisfy $O_1$
...
(n+1). [SCOPE mode] Whenever ($C_0$ & $C_1$& ... & $C_n$)
Component shall $[T_n]$ satisfy $O_n$

We refer to [3, 11] for the proof of correctness of the translation, for the formal semantics—which is the same as the one of FRETish requirements—and for complexity considerations, noting that the translation from SLEEC to FRETish adds only a linear number of new FRETish rules of linear size.

[1]We employ the same colouring of the FRET GUI: light brown for scope, blue for timing, etc.

The first author is a PhD student.

This work is licensed under a Creative Commons Attribution International 4.0 License.

*Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.* © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/XWPK8237

## 3 THE TOOL

Our extension, called SLEECinFRET, is available on GitHub[2]. The extension has been created as a fork of the official FRET tool at commit id `ba6e9d2`. Once compiled as standard for FRET, SLEECinFRET adds a drop-down menu to the CREATE button for new requirements, through which a user can choose to create a standard FRETish requirement or a SLEEC requirement. If the user selects SLEEC requirement, SLEECinFRET generates the corresponding FRETish requirements as described in Section 2, i.e., if a SLEEC requirement has $n$ defeaters, $n+1$ additional FRETish requirements are created. The new requirements are added to the same project as the SLEEC requirement. The generated requirements are marked as children of the SLEEC requirement and are read-only. If the parent is modified, they are recomputed accordingly. Aside from being read-only, they retain all standard FRET features, including LTL translation, simulation, and more importantly, realizability checking and test case generation.

In FRET, *"a set of requirements for a system component is realizable if an implementation of the component exists, such that it conforms to the requirements, given any input from the system's environment"* [5]. For instance, consider two requirements of the form (1) "whenever $a$, $X$ shall immediately satisfy $p$", and (2) "whenever $b$, $X$ shall immediately satisfy $!p$". These two requirements are consistent (meaning that they *can* be true at the same time, for instance if $a$ is true and $b$ is false, then $p$ should be true). However, the two requirements are not realizable: if $a$ and $b$ are input Boolean variables, an implementation does not exist *for all* combinations of $a$ and $b$, in particular when $a$ and $b$ are both true. Our SLEECinFRET allows for realizability checking of requirements that can involve both standard FRETish and SLEEC requirements.

In addition to realizability checking, FRET supports the generation of tests from requirements, following the approach presented in [4] and [9], which introduce a notion of *coverage* for requirements. Intuitively, given a requirement of the form "whenever $a$, $X$ shall satisfy $p$", one can generate tests in which $a$ is true and $p$ is false and expect a failure. By introducing temporal constraints, tests are represented by execution traces that provide the values of input variables and an expected output (typically, the requirement holds or the requirement is violated). Due to space limitations, we refer to [4] for additional details but we remark here that, to the best of our knowledge, SLEECinFRET is the first tool that enables automated test case generation from SLEEC requirements. As a concrete example, consider a robot in a nursing home, helping in taking care of elderly people. A general rule of the nursing home is that when the temperature is below a threshold, there cannot be an obligation to open the window of the room. This requirement can be modelled using the standard FRET as: *"whenever temperature_below_freezing Component shall satisfy !open_window"*. We employ here only Boolean variables for simplicity, but one could equally use numeric values and inequalities for temperature. In addition, a SLEEC requirement is introduced to respect the autonomy and privacy of the user. The robot has the obligation to open the window upon the user's request, unless the user is not dressed and their privacy is at risk, in which case the robot has the obligation to not open the window. This requirement can be modelled using SLEEC syntax: *"if user_asks then open_window unless user_undressed iwc do_not_open_window"*, where we introduce the explicit obligation *do_not_open_window* to highlight that there is a difference between having the obligation *do_not_open_window* and *not* having the obligation *open_window*. SLEECinFRET automatically breaks this rule into two FRETish requirements:
(1) *Whenever (user_asks & ! user_undressed) Component shall satisfy open_window*
(2) *Whenever (user_asks & user_undressed) Component shall satisfy do_not_open_window*
By creating these two requirements in our tool and running the realizability analysis it becomes evident that these two requirements are unrealizable. The tool also provides a counter example that shows unrealizability. In our example, it is possible that the user asks to open the window while the temperature is below the threshold. One quick fix to this issue is to change the SLEEC requirement so that the robot opens the window only when the user's demand is not in conflict with the temperature outside. The new requirement becomes: *"if user_asks & ! temperature_below_freezing then open_window unless user_undressed iwc do_not_open_window"*. This new requirement allows FRET to successfully pass the realizability checking, meaning that for all possible combinations of values of input variables there is an implementation that satisfies all the requirements. The tool also inherits test case generation from FRET. For the current example it generates 13 test cases. These test cases correspond to 13 traces that show how to satisfy or violate requirements as defined by the coverage metric in [4] and [9].

## 4 RELATED AND FUTURE WORK

Existing tools such as SLEECVAL [12] and LEGOS-SLEEC [6] enable the elicitation and analysis of SLEEC requirements. However, these tools mainly focus on the well-formedness of a set of requirements, not only in terms of syntax but also in terms of overall consistency, non-redundancy, etc. Compared to these tools, SLEECinFRET allows the management of the whole life-cycle of the requirements, simulation, and test case generation.

In previous work [8], we have investigated how to generate an obligation inference engine and monitors for a restricted syntax of SLEEC properties (no support for scopes nor timing). Compared to this work, we add support for scopes and timing, and for test-case generation, in addition to the re-use of FRET realizability checks.

In terms of limitations, LEGOS-SLEEC and SLEECVAL allow for additional checks of well-formedness properties, such as redundancy and sufficiency: these are not directly handled by our tool and we plan to incorporate or to interface with existing tools to add these functionalities. SLEECinFRET has currently no support for contrary-to-duty obligations (corresponding to the keyword 'otherwise' in other works). Compared to our work in [8], we do not have monitoring and obligation inference. In terms of implementation, SLEECinFRET does not have syntax highlighting yet, and some component names are hard-coded. We are currently working at improving these UI aspects for a funded EU project[3].

[2]The code is available at: https://github.com/mirgit/SLEECinFRET and a video for the demo is available at: https://youtu.be/88ShfVOqchs

[3]https://matisse-kdt.eu/

## REFERENCES


[1] 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. 2024. Analyzing and Debugging Normative Requirements via Satisfiability Checking. In *International Conference on Software Engineering*. ACM, 214:1–214:12. https://doi.org/10.1145/3597503.3639093

[2] Future of Life Institute. 2017. Asilomar AI Principles. Retrieved July, 2025, from https://futureoflife.org/open-letter/ai-principles/.

[3] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Julian Rhein, Johann Schumann, and Nija Shi. 2020. Formal Requirements Elicitation with FRET. In *Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2020), Pisa, Italy, March 24, 2020 (CEUR Workshop Proceedings, Vol. 2584)*. CEUR-WS.org. https://ceur-ws.org/Vol-2584/PT-paper4.pdf

[4] Andreas Katis, Anastasia Mavridou, and Tom Pressburger. 2025. A Streamlined, Formal Approach to Requirements-Based Testing. In *NASA Formal Methods: 17th International Symposium, NFM 2025, Williamsburg, VA, USA, June 11–13, 2025, Proceedings* (Hampton Roads, VA, USA). Springer-Verlag, Berlin, Heidelberg, 159–179. https://doi.org/10.1007/978-3-031-93706-4_10

[5] Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, and Khanh Trinh. 2025. FRET: Formal Requirements Elicitation Tool. https://github.com/NASA-SW-VnV/fret

[6] 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)*. IEEE Computer Society, Los Alamitos, CA, USA, 33–36. https://doi.org/10.1109/ICSE-Companion66252.2025.00018

[7] Tambiama André Madiega. 2024. Artificial intelligence act. https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf Think Tank European Parliament.

[8] Mahrokh Mirani, Paola Inverardi, Patrizio Pelliccione, Franco Raimondi, and Nicolas Troquard. 2026. Extending FRET with SLEEC Rules: Formalization, Obligation Inference, and Monitoring. In *To appear in Proceedings of TACAS 2026*.

[9] Charles Pecheur, Franco Raimondi, and Guillaume Brat. 2009. A formal analysis of requirements-based testing. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis* (Chicago, IL, USA) *(ISSTA '09)*. Association for Computing Machinery, New York, NY, USA, 47–56. https://doi.org/10.1145/1572272.1572279

[10] Beverley A. 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 Mach.* 32, 4 (2022), 683–715. https://doi.org/10.1007/S11023-022-09614-W

[11] Nicolas Troquard, Martina De Sanctis, Paola Inverardi, Patrizio Pelliccione, and Gian Luca Scoccia. 2024. Social, Legal, Ethical, Empathetic, and Cultural Rules: Compilation and Reasoning. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. AAAI Press, 22385–22392. https://doi.org/10.1609/AAAI.V38I20.30245

[12] 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 - 26th International Conference, FASE 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13991)*. Springer, 241–248. https://doi.org/10.1007/978-3-031-30826-0_13

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