Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Original software publication

Toolkit for specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents

Sinem Getir Yaman*, Pedro Ribeiro*, Charlie Burholt, Maddie Jones, Ana Cavalcanti, Radu Calinescu

Department of Computer Science, University of York, UK

ARTICLE INFO

ABSTRACT

A growing range of applications use AI and other autonomous agents to perform tasks that raise social, legal, ethical, empathetic, and cultural (SLEEC) concerns. To support a framework for the consideration of these concerns, we introduce SLEEC-TK, a toolkit for specification, validation, and verification of SLEEC requirements. SLEEC-TK is an Eclipse-based environment for defining SLEEC rules in a domain-specific language with a timed process algebraic semantics. SLEEC-TK uses model checking to identify redundant and conflicting rules, and to verify conformance of design models with SLEEC rules. We illustrate the use of SLEEC-TK for an assistive-care robot.

Code metadata

Code metadata description	
Current code version	v1.0.3
Permanent link to code/repository used for this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-23-00378
Permanent link to Reproducible Capsule	https://github.com/UoY-RoboStar/SLEEC-TK/releases/tag/v1.0.3
Legal Code License	Eclipse Public License 2.0
Code versioning system used	Git
Software code languages, tools, and services used	XTend, Java, Python
Compilation requirements, operating environments and dependencies	Java 11, Maven, FDR model checker
If available, link to developer documentation/manual	https://github.com/UoY-RoboStar/SLEEC-TK/blob/v1.0.3/README.md
Support email for questions	sinem.getir.yaman@york.ac.uk, pedro.ribeiro@york.ac.uk

1. Motivation and significance

The development of autonomous systems that interact with humans in high-stake application areas, such as healthcare and education, is on the rise. In this context, a new class of non-functional requirements related to social, legal, ethical, empathetic, and cultural (SLEEC) concerns [1] has gained increasing importance.

---

* Corresponding authors.
E-mail addresses: sinem.getir.yaman@york.ac.uk (S. Getir Yaman), pedro.ribeiro@york.ac.uk (P. Ribeiro).
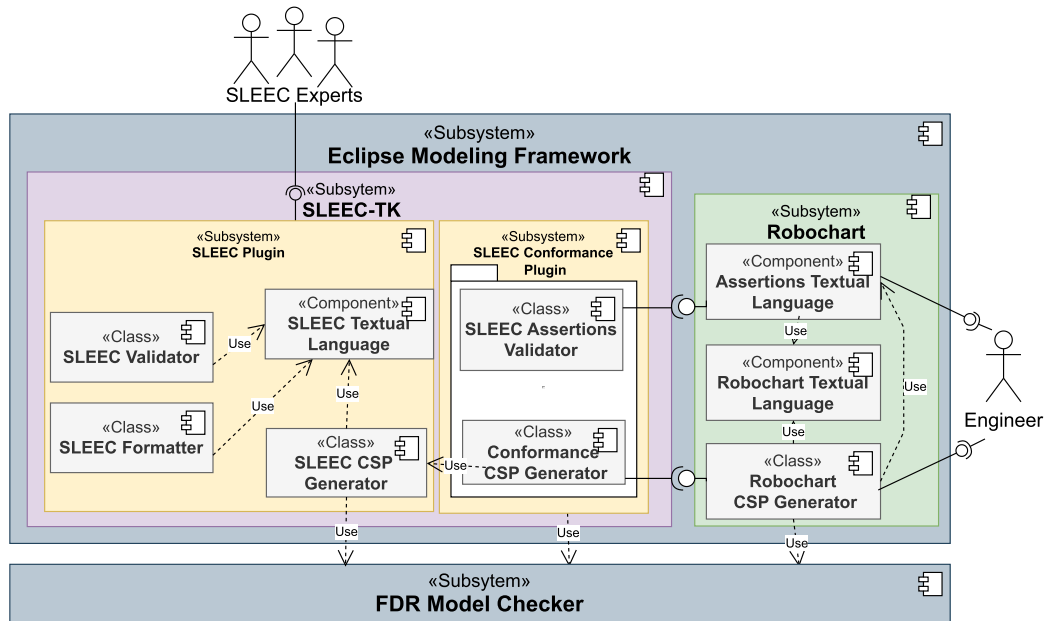
Fig. 1. Architecture of SLEEC-TK.

There is, however, minimal tool support for elicitation, specification, validation, and verification of SLEEC requirements. This is a challenging task due to the non-technical and varied background of the experts (ethicists, lawyers, regulators, end-users, and so on) that need to be involved in this process. Our software, SLEEC-TK, is a user-friendly toolkit employing formal methods to allow stakeholders to understand and address issues in the validation and verification of SLEEC requirements.

SLEEC-TK is a publicly available toolkit, usable by non-technical experts, to support the process and techniques in [1–3]. The SLEEC framework mechanised by SLEEC-TK includes a rules elicitation process [1], and specification, validation and verification techniques [2]. The technical report in [3] introduces our theoretical foundation and process for the specification, consistency validation, and verification of SLEEC requirements. It discusses the Domain Specific Language (DSL) used in the SLEEC-TK software and the formal semantics of this language, employing a timed version of CSP (Communicating Sequential Processes) [4]. An initial version of our tool, which supports only SLEEC language modelling, consistency, and redundancy validation of SLEEC rules, is described in [2]. The version we describe here implements an updated version of the semantics that provides increased scalability and has been extensively validated. Additionally, it has been enhanced with conformance verification of system models against SLEEC rules (i.e., the SLEEC Conformance Plugin demonstrated in Fig. 1 is a new component in our software). Furthermore, for SLEEC-TK we validated the rules and the language with 7 SLEEC specification files that cover 199 rules in conjunction with stakeholders.

There is significant work on development of autonomous systems from the perspective of normative ideas [5,6], including work on transparency [7], explainability, and data-driven personalised tools based on the moral choices of users [8]. The work on our SLEEC language has also considered alternative approaches to elicitation and debugging [9]. SLEEC-TK is concerned with the operationalisation of norms [1] [10], supporting automated processes for validating and verifying rules that capture these norms, via the mechanisation of its semantics described in *tock*-CSP, a timed process algebra [4,11].

SLEEC-TK is implemented as a set of plugins for the Eclipse environment, but includes a standalone version for SLEEC rule validation. The README.md file in the repository provides instructions for downloading, installing, and using the software, with examples. The definition of rules is via a graphical interface that provides guidance regarding any syntactic or typing issues. In the background, *tock*-CSP scripts are generated to support the checks of conflicts and redundancies. This is carried out at the push of a button, by using the CSP model checker FDR4 [12] in the background. Verification is carried out by integration with RoboTool[1] [13], a tool for modelling and verification of mobile and autonomous robots using a domain-specific notation, RoboChart. SLEEC rules can be included as part of a document defining properties of a RoboChart model, for automatic verification and production of a report.

2. Software description

The SLEEC framework has two main components: (1) SLEEC rule validation and (2) conformance checking, both supported by SLEEC-TK.

---

[1] https://robostar.cs.york.ac.uk/robotool/.

2.1. Software architecture

The overall software architecture of SLEEC-TK is described in Fig. 1 using a UML component diagram. There are six subsystems, with the top two being: Eclipse Modelling Framework, including our SLEEC-TK component and another subsystem called RoboChart (which is part of RoboTool), and the FDR model checker. Fig. 1 shows the RoboTool plugins that we extended to support verification of SLEEC rules, and FDR4 as a subsystem used by both the SLEEC-TK and the RoboChart subsystems.

SLEEC-TK has two plugins: SLEEC and SLEEC Conformance. The SLEEC plugin consists of a SLEEC textual language component, utilized by three classes: Validator, Formatter, and CSP Generator. A SLEEC expert can use this plugin through textual DSL editors. The CSP Generator class produces *tock*-CSP scripts suitable for the FDR model checker. The output of the SLEEC CSP Generator class can be checked by FDR for conflict and redundancy analysis independently from the SLEEC Conformance plugin.

The RoboChart subsystem operates independently from the SLEEC subsystem. To utilize this subsystem, an engineer must construct a RoboChart model and use the assertions textual language, where both behavioural properties and SLEEC conformance checks can be specified. If the user intends to employ conformance checking of SLEEC requirements, input from both the SLEEC experts and the engineer are necessary to employ the Conformance CSP Generator, which invokes FDR4 from the SLEEC plugin. We tested our software components with several models. For the RoboChart plug-in of RoboTool that is used in SLEEC-TK, we have a base of 52 models[2] that provide comprehensive coverage of the RoboChart notation. For the SLEEC notation, we have a base of 7 SLEEC documents covering in total 199 SLEEC rules that have been processed using SLEEC-TK.[3]

2.2. Software functionalities

SLEEC-TK implements three primary functionalities. Firstly, it empowers experts to encode SLEEC requirements using a DSL editor that provides syntax highlighting and type checking. Secondly, from a SLEEC specification (stored in a file with the extension .sleec), SLEEC-TK automatically calculates a formal semantics [3] in *tock*-CSP, via the generation of CSP_M code, the machine-readable version of CSP accepted by FDR4 (files with extension .csp).

The generated CSP files can be loaded into FDR4 to perform conflict and redundancy analysis, validating the specified SLEEC rules. Lastly, SLEEC-TK can be used to conduct conformance checking of RoboChart design models, constructed using the existing RoboTool software, against the rules from a SLEEC specification file. Following this analysis, an output file is generated, providing a report of the passed and failed rules within the Eclipse editor. Notably, all files, including the SLEEC specification and RoboChart model, are created within the same Eclipse project.

3. Illustrative example

Fig. 2 shows the definition of SLEEC rules for an assistive-care robot working in a home environment in the SLEEC editor. SLEEC specifications comprise two blocks. The first block (delimited by the keyword pair def_start. . .def_end) comprises declarations of events and measures that represent the functional capabilities and parameters of the agent.

The second block (delimited by the keywords rule_start. . .rule_end) defines the actual SLEEC rules in terms of those capabilities and parameters. A SLEEC rule has a unique id and the basic form 'when **trigger** then **response**'. The **trigger** defines an event whose occurrence indicates the need to satisfy the constraints defined in the **response**. For example, Rule1 applies when the event StartLunchTime occurs. The **response** defines requirements that need to be satisfied when the trigger holds, and may include deadlines and timeouts using the keywords within and otherwise, respectively. In Rule1, the response requires the occurrence of the event InformUser within 5 minutes.

A rule can be followed by one or more defeaters introduced by the unless construct. They specify circumstances that preempt the original response, providing an alternative. Examples are provided by Rule1, Rule2, and Rule3.

Whenever a SLEEC specification is saved, three files are automatically generated in a src-gen folder: an instantiations.csp file, where users can override a default bound for numeric types necessary for model checking; a file with the same name as the SLEEC specification but extension .csp that contains the CSP semantics of the SLEEC rules; and a file whose name includes the suffix -assertions. This third file contains assertions for checking conflicts and redundancies among pairs of rules from the SLEEC specification.

When the assertions file is analysed using FDR4 (either using its textual or graphical interface), an output is generated similar to that shown at the bottom in Fig. 2. The checks are generated just for rules that have overlapping alphabets of events and measures, and that, therefore, may be conflicting or redundant. So, in our example, there are assertions generated just for Rule2 and Rule4. The deadlock assertion given first fails, indicating that there is a conflict between Rule2 and Rule4. The remaining assertions, which correspond to the analysis of redundancy and different types of conflicts, indicate that there are no other inconsistencies. The formal semantics and theoretical framework, as well as its workflow are described in [2,3].

Fig. 3 illustrates the verification of a RoboChart design model of an assistive-care robot against behavioural properties and SLEEC rules. The top-left pane shows two timed assertions, A1 and A2, that check whether the ALMI model representing the system model of the assistive care robot is both deadlock free and deterministic, and three sleec assertions (S1 to S3) that check whether it

---

Fig. 2. Fragment of the SLEEC specification for an assistive-care robot and rule validation results.

conforms to SLEEC rules Rule1, Rule2 and Rule4. A subset of the graphical RoboChart design model is shown on the right, and a tabular report with results from the verification of the assertions is shown in the bottom-left pane.

4. Impact

SLEEC-TK orchestrates and supports the entire process of eliciting, validating, and verifying SLEEC requirements. With the availability of SLEEC-TK, we are currently working on the following questions:

- How can we enhance stakeholder engagement by offering diagnostic options and a constructive, user-friendly feedback mechanism for the validation and verification results?
- Can the availability of LLM (Large Language Models) contribute to comprehending SLEEC requirements and analysis results by SLEEC stakeholders with limited technical expertise?

SLEEC-TK, as demonstrated through user studies detailed in [2,9], has elevated the expressiveness and usability of SLEEC requirements.

The SLEEC elicitation process is iterative, involving stakeholders in the loop. Our software streamlines the writing and refinement of SLEEC rules in daily practice, providing a more systematic approach. It has been utilized by six experts with backgrounds in Philosophy, Law, Ethics, and Robotics, participating in a user study [3] encompassing nine different case studies in domains including healthcare, education, manufacturing, environment, and transport. Overall, these experts provided positive feedback regarding usability and its efficacy in reducing effort during the iterative SLEEC requirements engineering process.

5. Conclusion

We have presented SLEEC-TK, a publicly available software toolkit that supports an end-to-end process of elicitation, validation, and verification of SLEEC requirements.[4] SLEEC-TK employs a DSL for encoding timed SLEEC requirements in a user-friendly

---

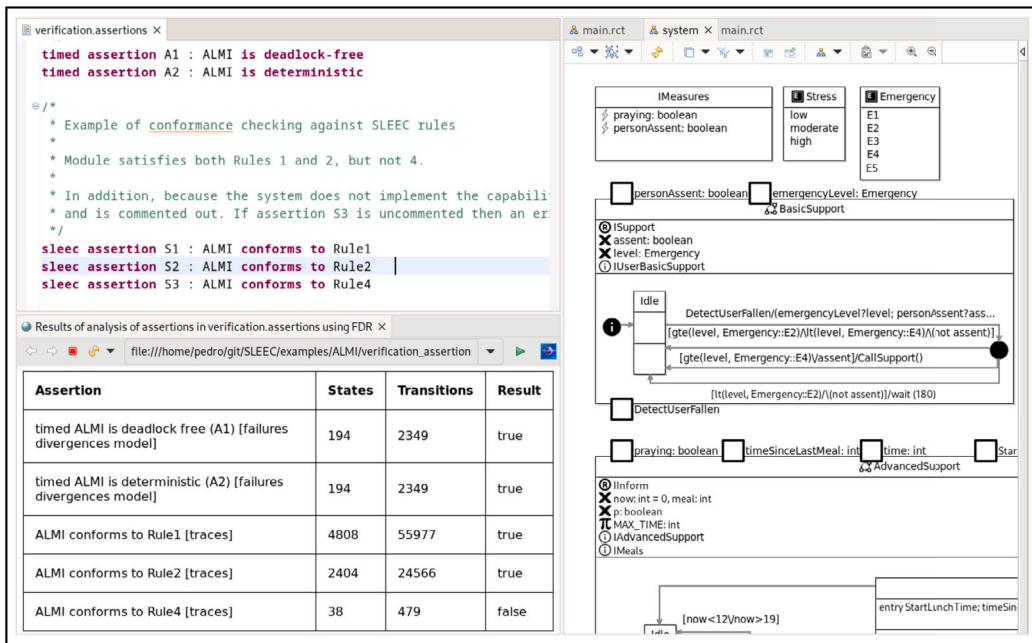[4] https://sleec-project.github.io/SLEEC-TK/index.html.

Fig. 3. Eclipse environment with assertions shown in the top-left pane and the results, shown below, of verifying the design model's conformance (shown on the right) against the assertions.

language, and augments them with automatically calculated formal timed process algebraic semantics, suitable for model-checking (including checks for rule conflicts and redundancy) as well as conformance checking of design models against these rules. In the future, we will extend the SLEEC DSL to consider uncertainty in the agents and their environments. Therefore, stakeholders can consider and encode alternative responses based on thresholds of certainty. Furthermore, we plan to validate SLEEC-TK in additional user studies, and to extend it with further SLEEC rule elicitation and design techniques. We also aim to augment SLEEC-TK with large language model based methods for explaining to stakeholders the SLEEC-TK analysis results, and for suggesting resolution actions for the conflicts and redundancies identified by SLEEC-TK.

CRediT authorship contribution statement

Sinem Getir Yaman: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Pedro Ribeiro:** Writing – review & editing, Validation, Software, Formal analysis, Conceptualization. **Charlie Burholt:** Software. **Maddie Jones:** Software. **Ana Cavalcanti:** Writing – review & editing, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Radu Calinescu:** Writing – review & editing, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

References

[1] B. Townsend, C. Paterson, T.T. Arvind, G. Nemirovsky, R. Calinescu, A. Cavalcanti, I. Habli, A. Thomas, From pluralistic normative principles to autonomous-agent rules, Minds Mach. 32 (4) (2022) 683–715, https://doi.org/10.1007/s11023-022-09614-w.
[2] S. Getir Yaman, C. Burholt, M. Jones, R. Calinescu, A. Cavalcanti, Specification and validation of normative rules for autonomous agents, in: 26th International Conference on Fundamental Approaches to Software Engineering, Springer-Verlag, 2023, pp. 241–248.

[3] S. Getir Yaman, A. Cavalcanti, R. Calinescu, C. Paterson, P. Ribeiro, B. Townsend, Specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents, arXiv:2307.03697, 2023.

[4] A.W. Roscoe, Understanding Concurrent Systems, Texts in Computer Science, Springer, 2011.

[5] L. Dennis, M. Fisher, A.F.T. Winfield, Towards verifiably ethical robot behaviour, ArXiv, arXiv:1504.03592 [abs], 2015.

[6] J. Fjeld, N. Achten, H. Hilligoss, A. Nagy, M. Srikumar, Principled artificial intelligence: Mapping consensus in ethical and rights-based approaches to principles for AI, Berkman Klein Center Research Publication 2020-1, 2020.

[7] A. Winfield, E. Watson, T. Egawa, E. Barwell, I. Barclay, S. Booth, L.A. Dennis, H. Hastie, A. Hossaini, N. Jacobs, M. Markovic, R.I. Muttram, L. Nadel, I. Naja, J. Olszewska, F. Rajabiyazdi, R.K. Rannow, A. Theodorou, M.A. Underwood, O. von Stryk, R.H. Wortham, IEEE standard for transparency of autonomous systems, https://doi.org/10.1109/IEEESTD.2022.9726144, 2022.

[8] C. Alfieri, P. Inverardi, P. Migliarini, M. Palmiero, Exosoul: ethical profiling in the digital world, in: S. Schlobach, M. Pérez-Ortiz, M. Tielman (Eds.), First International Conference on Hybrid Human-Artificial Intelligence, in: Frontiers in Artificial Intelligence and Applications, vol. 354, IOS Press, 2022, pp. 128–142.

[9] N. Feng, L. Marsso, S. Getir Yaman, B. Townsend, R. Calinescu, A. Cavalcanti, M. Chechik, Towards a formal framework for normative requirements elicitation, in: Proceedings of the 38th International Conference on Automated Software Engineering, ASE'2023, IEEE, Luxembourg, 2023, pp. 1776–1780.

[10] P. Solanki, J. Grundy, W. Hussain, Operationalising ethics in artificial intelligence for healthcare: a framework for AI developers, AI Ethics 3 (1) (2023) 223–240, https://doi.org/10.1007/s43681-022-00195-z.

[11] J. Baxter, P. Ribeiro, A. Cavalcanti, Sound reasoning in tock-CSP, Acta Inform. 59 (1) (2022) 125–162, https://doi.org/10.1007/s00236-020-00394-3.

[12] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, A.W. Roscoe, FDR3 - a modern refinement checker for CSP, in: Tools and Algorithms for the Construction and Analysis of Systems, 2014, pp. 187–201.

[13] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, J.C.P. Woodcock, RoboChart: modelling and verification of the functional behaviour of robotic applications, Softw. Syst. Model. 18 (5) (2019) 3097–3149, https://doi.org/10.1007/s10270-018-00710-z.