Runtime Verification for MAS

Rafael Ramildes Ferreira
DAS410059 - Sistemas Multiagentes
2024.3/4

Introduction

Multi-Agent Systems (MAS) are very complex system that may be sensitive to bugs and edge cases where the behavior is undesired

Mere testing and debugging may hide bugs that could be detrimental if happens in deployment

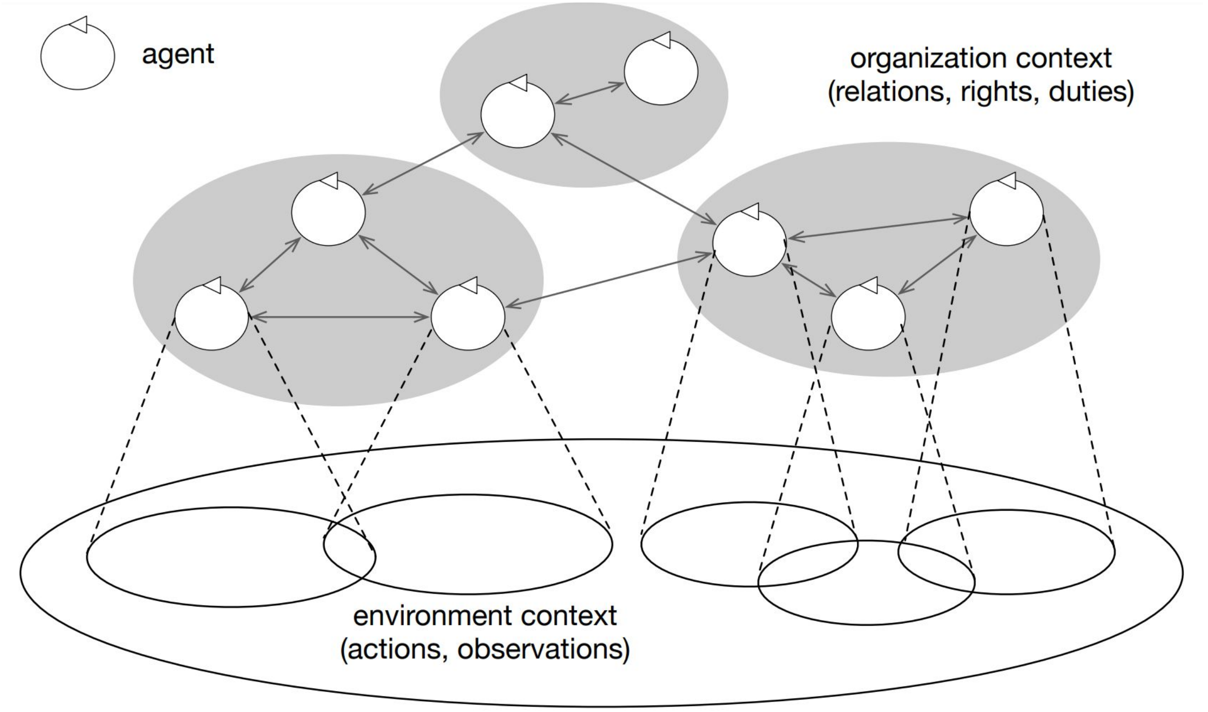
Formal Verification techniques are then researched for solving these problems and validate the expected behavior

However, it’s generally hard to adequately model a MAS, making model checking hard

Runtime Verification (RV) technology is than a viable alternative and will be discussed in this work

Multi-Agent Systems (MAS)

- MAS is a paradigm for modeling and developing complex systems
- The idea is to have autonomous *agents*, acting on a shared *environment* possibly being part of a *organization*



Multi-Agent Systems (MAS)

- The *agent* being autonomous means that de is capable of decision-make in the pursuit of an objective
- *Agents* can communicate and cooperate to active a shared goal
- The whole system is then decentralized, open, heterogene and adaptable
- Because of this, MAS are naturally promising solutions for robotic systems that operate in unpredictable environments

Problems for MAS validation

- This complex nature makes it hard to validate the intended behavior of the system
- Especially in safe critical systems, it’s important to guarantee that the system operates under certain constraints
- Testing on the real scenario may not be possible or prudent
- Basic testing and debugging generally are non-exhaustive and difficult for MAS
- Formal Verification methods could be the solution

Formal Verification

Validating systems with mathematical formalism

Formal Verification are technics use to check mathematically precise defined requisites, presented normally as Temporal Logic formulas, such as Computational Tree Logic (CTL) or Linear-Time Logic (LTL)

Formal Verification techniques

Verification in design-time

The most common method is via Model Checking

Benefits: It's very optimized and can prove the system's attributes

Drawbacks: Relies on a formal model and is just as good as the model

Runtime Verification (RV)

It's made by implementing monitors that verify the temporal formulas

Benefits: It's not reliant on a possibly faulty model. Verifies only the real world behavior

Drawbacks: Is only capable of checking the current execution

Runtime Verification

- Implementa monitores embarcados junto com a aplicação
- Monitores são geralmente sintetizados automaticamente com base em fórmulas temporais

$$\neg \mathbf{F}\ \text{fault}$$

$$\mathbf{G}(\ \text{new\_goal} \Rightarrow \mathbf{F}\ \text{acomplish\_goal}\ )$$

- The sequence of values used in the formula are called trace

Runtime Verification formulas

- For RV these formulas generally need a time bound, then languages like MLTL, used in R2U2, are more suited

$$\mathbf{G}(\text{ ask\_a\_question} \Rightarrow \mathbf{F}[1,3] \text{ get\_awnser })$$

- Other softwares may not use formulas expressed in Temporal Logic, but other formal representation of requirement
- For instance Runtime Monitoring Language (RML) expresses the set of expected events at any given time and yields a failed when the received event is not on the set

Runtime Verification in JaCaMo

RV4JaCa—Towards Runtime Verification of Multi-Agent Systems and Robotic Application

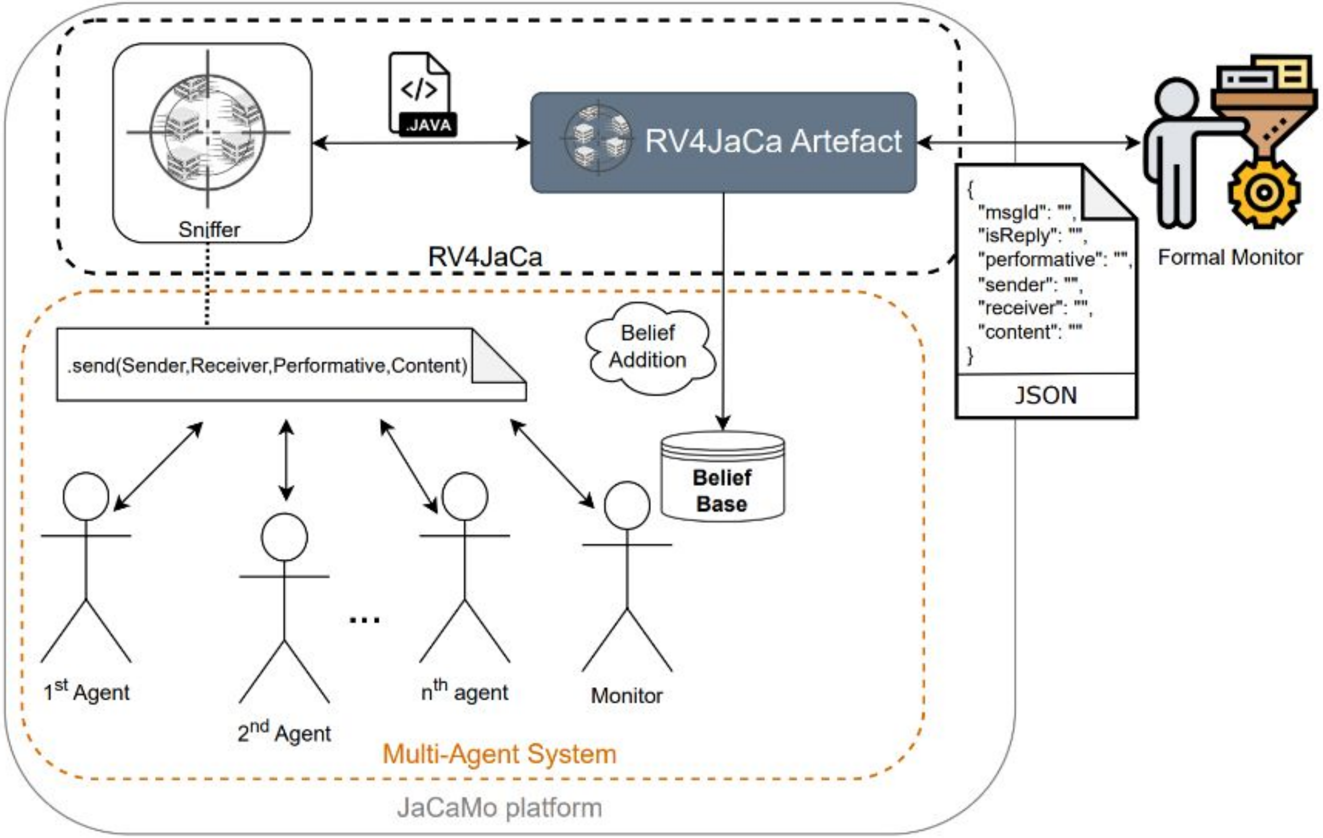
Debora C. Engelmann, Angelo Ferrando, Alison R. Panisson, Davide Ancona, Rafael H. Bordini and Viviana Mascardi

robotics

24 March 2023

Paper contribution

- Engelmann *et al.* (2023) implement a CArtAgO *artifact* called RV4JaCa
- It’s purpose is to interface between a external RV software, a Jason monitor *agent* and a *sniffer*
- The RV software could be anyone, but they test on the Runtime Monitoring Language (RML), as it supports the TCP/IP protocol used by the *artifact*
- The monitor *agent* can be used to trigger a response or just inform the *agents* about committed violations



RV4JaCa proposed architecture for RV in MAS

Monitor interface

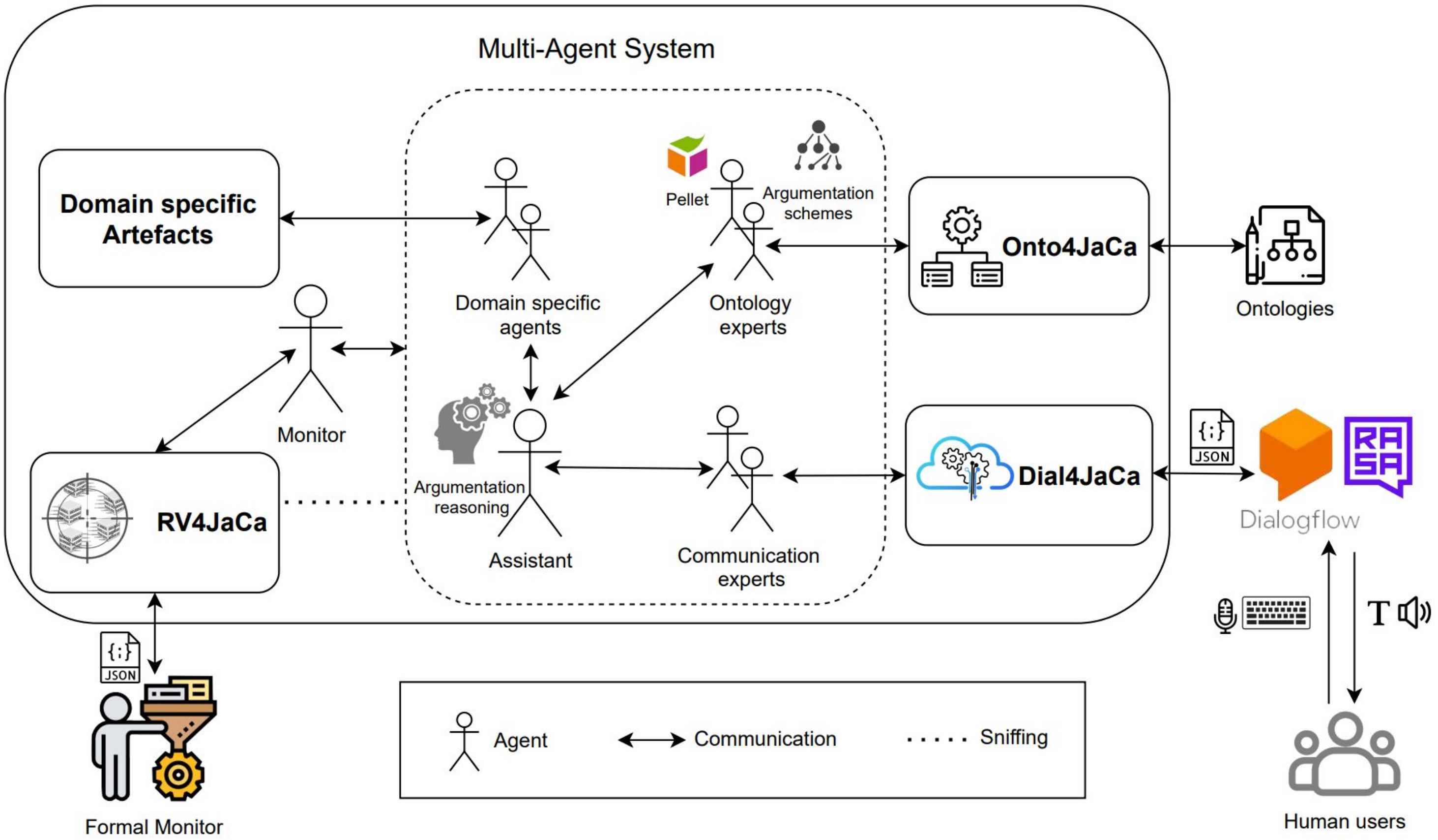
- Engelmann, D. C. et al. (2023) focus on verifying communication protocols between agents, hence why collecting the messages as trace data is natural
- But other works also propose to use a communication bus as interface for the monitor (MEREDITH et al., 2012),(PIKE et al., 2010),(GEIST; ROZIER; SCHUMANN, 2014)

These approaches aim in guaranteeing a non-obstruction attribute to the RV system, which is generally desired

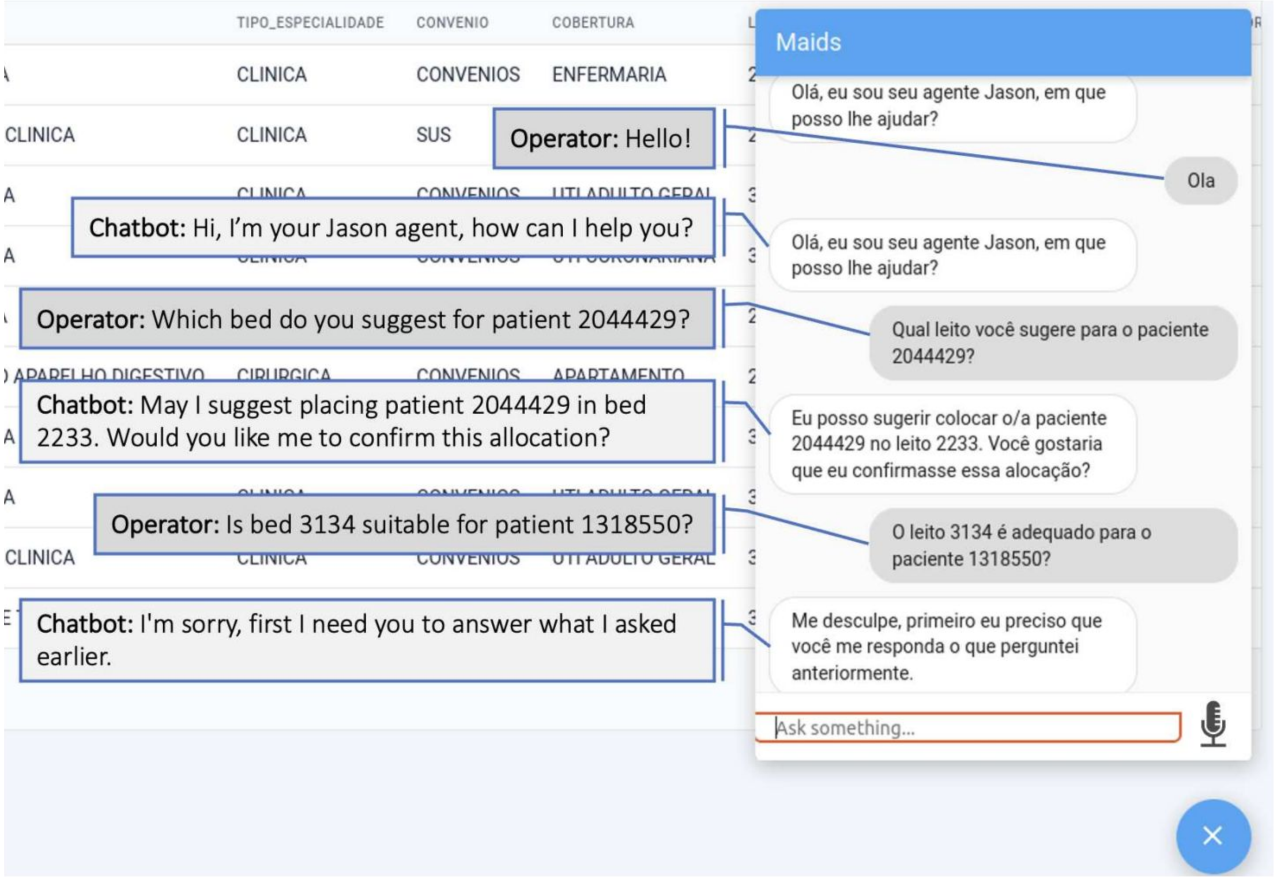
Case study

Bed allocation assistant for a hospital environment

The goal of the system is to assist humans in efficiently allocate beds for patients in a hospital. The RV systems verifies the communication between the agents and the operators, to spot unexpected changes in topic and communication protocol violations



Maids architecture



Chatbot reporting a communication protocol violation

Properties verification

- This is achieved by using Artificial Intelligence (AI) to identify topics in the user message, like `'allocValPatients'`, `'dontAllocValPatients'` between others
- Also topics like this can be translated to human language by the dialog *artifacts*
- Those identified topics are passed to the RV software for verification, in this case:

  After `'getValidationResult'` topic, it expects `'allocValPatients'` or `'dontAllocValPatients'`

Conclusion

RV is being exploited for MAS, focusing on a higher level, normally Agents Interaction Protocols (AIP)

The case of the paper that introduced RV4JaCa is a case that the protocol cannot be enforced on the human user, so the verification attempts to identify the human violation of the protocol to respond, in a manner that the *agent* keeps attempting to conclude it's goal

Idea to use for verifying the internal behavior of the *agents*, although not the main focus of research, is mentioned in papers

References

[1] BOISSIER, O. et al. **Multi-agent oriented programming: programming multi-agent systems using JaCaMo**. Cambridge, Massachusetts: The MIT Press, 2020.

[2] ENGELMANN, D. C. et al. RV4JaCa—Towards Runtime Verification of Multi-Agent Systems and Robotic Applications. **Robotics**, v. 12, n. 2, p. 49, 24 mar. 2023.

[3] GEIST, J.; ROZIER, K. Y.; SCHUMANN, J. Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems. Em: BONAKDARPOUR, B.; SMOLKA, S. A. (Eds.). **Runtime Verification**. Cham: Springer International Publishing, 2014. v. 8734p. 215–230.

[4] MEREDITH, P. O. et al. An overview of the MOP runtime verification framework. **International Journal on Software Tools for Technology Transfer**, v. 14, n. 3, p. 249–289, 2012.

[5] PIKE, L. et al. Copilot: A Hard Real-Time Runtime Monitor. Em: BARRINGER, H. et al. (Eds.). **Runtime Verification**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. v. 6418p. 345–359.

[6] ENGELMANN, D. et al. Dial4JaCa – A Communication Interface Between Multi-agent Systems and Chatbots. **Lecture Notes in Computer Science**, p. 77, 2021.

[7] ENGELMANN, D. C. **Intentional dialogues in multi-agent systems based on ontologies and argumentation**. Porto Alegre: PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL, 2023.

[8] FERRANDO, A.; MALVONE, V. Towards the Combination of Model Checking and Runtime Verification on Multi-agent Systems. Em: DIGNUM, F. et al. (Eds.). **Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection**. Cham: Springer International Publishing, 2022. v. 13616p. 140–152.