RV4JaCa – Runtime Verification for Multi-Agent Systems

Debora C. Engelmann
School of Technology – PUCRS, Brazil
DIBRIS – University of Genoa, Italy
debora.engelmann@edu.pucrs.br

Angelo Ferrando
DIBRIS – University of Genoa, Italy
angelo.ferrando@unige.it

Alison R. Panisson
Department of Computing – UFSC, Brazil
alison.panisson@ufsc.br

Davide Ancona
DIBRIS – University of Genoa, Italy
davide.ancona@unige.it

Rafael H. Bordini
School of Technology – PUCRS, Brazil
rafael.bordini@pucrs.br

Viviana Mascardi
DIBRIS – University of Genoa, Italy
viviana.mascardi@unige.it

This paper presents a Runtime Verification (RV) approach for Multi-Agent Systems (MAS) using the JaCaMo framework. Our objective is to bring a layer of security to the MAS. This layer is capable of controlling events during the execution of the system without needing a specific implementation in the behaviour of each agent to recognise the events. MAS have been used in the context of hybrid intelligence. This use requires communication between software agents and human beings. In some cases, communication takes place via natural language dialogues. However, this kind of communication brings us to a concern related to controlling the flow of dialogue so that agents can prevent any change in the topic of discussion that could impair their reasoning. We demonstrate the implementation of a monitor that aims to control this dialogue flow in a MAS that communicates with the user through natural language to aid decision-making in hospital bed allocation.

1 Introduction

A characteristic identified as essential in Artificial Intelligence (AI) is explainability, as it provides users with the necessary inputs to make it possible to understand the system's behaviour, and inspire confidence in the final outcome. Explainability becomes an essential feature in Multi-Agent Systems (MAS), as it is one of the most powerful paradigms for implementing complex distributed systems powered by artificial intelligence techniques. MAS are built upon core concepts such as distribution, reactivity, and individual rationality. Agents have been widely studied, and an extensive range of tools have been developed, such as agent-oriented programming languages and methodologies [12]. Thus, practical applications of multi-agent technologies have become a reality to solve complex and distributed problems [33]. In addition, it also allows the execution of various tasks and makes it possible the integration with various technologies.

Even though MAS solutions can be a natural choice for developing complex and distributed systems, like any other software development technique they are prone to errors and bugs (whether at the implementation or description level). Standard techniques such as testing and debugging can be deployed to tackle this problem. However, in case of MAS, the process of testing [35], debugging [36], and verifying [16] such systems can be quite complex. For this and other reasons, more lightweight approaches to guarantee the correct execution of the system are valuable. One technique that can be applied in

R.C. Cardoso, A. Ferrando, F. Papacchini,
M. Askarpour, L.A. Dennis (Eds.): Second
Workshop on Agents and Robots for reliable
Engineered Autonomy (AREA'22).
EPTCS 362, 2022, pp. 23–36, doi:10.4204/EPTCS.362.5

such cases is Runtime Verification (RV) [9, 25]. Differently from other verification techniques, RV is lightweight because it only concerns the analysis of the runtime execution of the system under analysis; which makes RV very similar to testing. However, rather than testing, RV is based on a specification formalism, as it happens in formal verification, to express the properties to be checked against the system's behaviour, and is particularly suitable to monitor control-oriented properties [2].

In this paper, we present RV4JaCa, an approach to perform RV of multi-agent systems developed using the JaCaMo framework [11]. RV4JaCa is obtained by extending MASs implemented in JaCaMo with a monitoring feature. In a nutshell, RV4JaCa handles all the engineering pipeline to introduce RV in JaCaMo, and to extract and check runs of the MAS against formal properties. In particular, RV4JaCa enables RV of agents' interactions (i.e., messages). As a proof of concept, we demonstrate a case study in a hospital bed allocation domain where RV is used to verify two different agent interaction protocols.

The paper is structured as follows. Section 2 reports the background needed to fully understand the contribution; here, the JaCaMo framework and the notion of RV are introduced. Section 3 presents RV4JaCa and its application in a bed allocation case study. Section 4 positions the contribution w.r.t. the state of the art. Finally, Section 5 summarises the contribution's results, and points out future developments.

2 Background

2.1 Multi-agent systems

Multi-agent systems (MAS) are systems composed of multiple agents. They seem to be a natural metaphor for building and understanding a wide range of artificial social systems and can be applied in several different domains [37]. There are two interlocking strands of work in multi-agent systems: the one that concerns individual agents and the one that concerns itself with the collections of these agents. In practice, agents rarely act alone, they usually inhabit an environment that contains other agents. Each agent can control, or partially control, parts of the environment, which is called its "sphere of influence". It may happen that these spheres of influence overlap, hence (parts of) the environment may be controlled jointly by more than one agent. In this case, to achieve the desired result, an agent must also consider how other agents may act. These agents will have some knowledge, possibly incomplete, about the other agents [13].

Wooldridge [37] believes that to be able to understand a multi-agent domain it is essential to understand the type of interaction that occurs between agents. For intelligent autonomous agents, the ability to reach agreements is extremely important, and for this, negotiation and argumentation skills are often necessary. In [20], the authors mention two types of studies in multi-agent systems; the first one is Agent-Centred Multi-Agent Systems (ACMAS), which study, at the level of an agent, states and the relationship between those states and their general behaviour and are projected in terms of the agent's states of mind. The second one is Organisation-Centred Multi-Agent Systems (OCMAS), which are systems whose foundations reside in the concepts of organisations, groups, communities, roles, and functions, among others. An OCMAS is not considered in terms of mental states but in capacities and constraints, which are considered organisational concepts, as well as functions, tasks, groups, and interaction protocols.

In a multi-agent system, the organisation is the collection of roles, relationships, and authority structures that govern agents' behaviour. Every multi-agent system has some form of organisation, even if it is implicit and informal. Organisations guide the mode of interaction between agents, which may influence data flows, resource allocation, authority relationships, and various other features of the system [23].

In [7], the authors argue that systems which heavily adopt AI techniques are increasingly available, and making them explainable is a priority. Explainable Artificial Intelligence (XAI) is a research field that aims "to make AI systems results more understandable to humans" [1]. These results must be clear (in non-technical terms) and provide justifications about decisions made [17]. In order to achieve a satisfactory level of explainability in multi-agent systems, much communication between agents and humans needs to be performed. However, this can generate an additional point of failure in the execution of the system since if, for example, the predefined communication protocol is not followed, or even if the human decides to change the conversation topic unexpectedly, this can lead to unexpected and probably inappropriate behaviour of the system.

2.2 JaCaMo framework

JaCaMo is a framework that allows for multi-agent oriented programming. This framework consists of the integration of three previously existing platforms: Jason – for programming autonomous agents, CArtAgO – for programming environmental artifacts, and Moise – for programming multi-agent organisations [11]. A multi-agent system programmed in JaCaMo has Jason agents that are organised and follow roles according to Moise's hierarchical structure. These agents work in environments based on distributed artifacts programmed using CArtAgO. Figure 1 shows an overview of the three dimensions of JaCaMo.

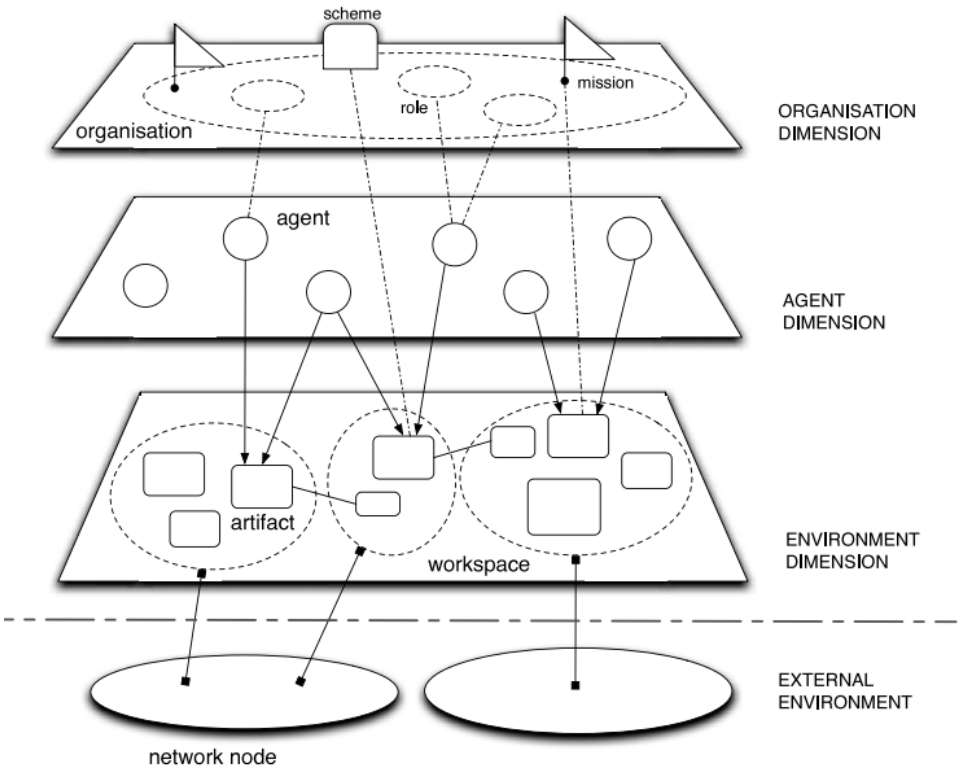

Figure 1: Overview of the three dimensions of JaCaMo [11]

Jason (`Agent dimension`) is an agent-oriented programming language that is an interpreter for an extended version of the AgentSpeak [13] language. Agents programmed in Jason use the Belief-

Desired-Intention (BDI) model [30]. The main idea of this approach is to model the process of deciding which action to take to achieve certain objectives [30]. Moise [24, 11] is related to the `organisation dimension`, where agents can be part of groups and follow specific roles. Also, with Moise, *schemes* are defined, that is, the structure of organisation goals is decomposed into sub-goals and grouped into missions. An organisation is specified in an XML (Extensible Markup Language) file. CArtAgO [31], the `environment dimension`, is used to simulate an environment or interface with a real one; this is where *artifacts* are defined. These artifacts define the environment's structure and behaviour, representing all resources that agents need. Agents can discover, create, and use artifacts at runtime [11]. Artifacts are programmed in Java. Combining these dimensions provides us with a complete framework for developing multi-agent systems through agents, organisations, and environments.

2.3 Runtime verification

Runtime Verification (RV) [10] is a kind of formal verification technique that focuses on checking the behaviour of software/hardware systems. With respect to other formal verification techniques, such as Model Checking [15] and Theorem Provers [27], RV is considered more dynamic and lightweight. This is mainly due to its being completely focused on checking how the system behaves, while the latter is currently running. This is important from a complexity perspective. RV does not need to simulate the system in order to check all possible execution scenarios; but, it only analyses what the system produces (i.e., everything that can be observed of the system). This is usually obtained through monitors, which are automatically generated from the specifications of the properties to be checked, and are nothing more than validation engines which, given a trace of events generated by the system execution, conclude the satisfaction (resp. violation) of the corresponding properties. In turn, a formal property is the formal representation of how we expect the system should behave. The monitor's job is to verify at runtime whether such a property holds.

Since monitors are usually deployed together with the system under analysis, they are well suited for checking properties that require to be continuously monitored. This is especially true in safety-critical scenarios, where a system's fault can cause injuries, loss of money and even deaths. A key example is autonomous and robotic systems, where *reliability* is vital [22], and the addition of monitors ensuring a correct behaviour is a valuable feature.

In the scenario envisaged in this contribution, we aim to use RV as a safety net for message exchange in JaCaMo. As pointed out elsewhere, the protocols involved in the communication amongst agents and human beings can be very complex, and hard to track. Moreover, agents are usually particularly focused on the reasoning and reactive aspects, while the consistency of the protocols is given for granted. However, above all in case of human being in the loop, such assumption cannot be made. RV is a suitable candidate to keep track of the protocols, to check whether the current agents' enactment is consistent (or not) with the expected protocol. Such consistency checking is extremely important in safety-critical scenarios, as in the healthcare domain, where a protocol violation can be costly.

2.4 Runtime Monitoring Language

Runtime Monitoring Language[1] (RML [6]) is a Domain-Specific Language (DSL) for specifying highly expressive properties in RV (such as non context-free ones). We use RML in this paper for its support of parametric specifications and its native use for defining interaction protocols. In fact, the low-level language on which RML is based upon was born for specifying communication protocols [4, 5].

[1] `https://rmlatdibris.github.io/`

Since RML is just a means for our purposes, indeed other formalisms can be as easily integrated into RV4JaCa, we only provide a simplified and abstracted view of its syntax and semantics. However, the complete presentation can be found in [6].

In RML, a property is expressed as a tuple $\langle t, ETs \rangle$, with t a term and $ETs = \{ET_1, \ldots, ET_n\}$ a set of event types. An event type ET is represented as a set of pairs $\{k_1 : v_1, \ldots, k_n : v_n\}$, where each pair identifies a specific piece of information (k_i) and its value (v_i). An event Ev is denoted as a set of pairs $\{k'_1 : v'_1, \ldots, k'_m : v'_m\}$. Given an event type ET, an event Ev matches ET if $ET \subseteq Ev$, which means $\forall (k_i : v_i) \in ET \cdot \exists (k_j : v_j) \in Ev \cdot k_i = k_j \wedge v_i = v_j$. In other words, an event type ET specifies the requirements that an event Ev has to satisfy to be considered valid.

An RML term t, with t_1, t_2 and t' as other RML terms, can be:

- ET, denoting a set of singleton traces containing the events Ev s.t. $ET \subseteq Ev$;
- $t_1 \ t_2$, denoting the sequential composition of two sets of traces;
- $t_1 \mid t_2$, denoting the unordered composition of two sets of traces (also called shuffle or interleaving);
- $t_1 \wedge t_2$, denoting the intersection of two sets of traces;
- $t_1 \vee t_2$, denoting the union of two sets of traces;
- $\{let\ x;\ t'\}$, denoting the set of traces t' where the variable x can be used (i.e., the variable x can appear in event types in t', and can be unified with values).
- $t'*$, denoting the set of chains of concatenations of traces in t'

Event types can contain variables. For example, $ET(ag1, ag2) = \{sender : ag1, receiver : ag2\}$, where we do not force any specific value for the sender (resp., receiver) of a message (in this case the events of interest would be messages). This event type matches all events containing sender and receiver. When an event matches an event type with variables, such as in this case, the variables get the values from the event. For instance, if the event observed would be $Ev = \{sender : \text{“}Alice\text{”}, receiver : \text{“}Bob\text{”}\}$, it would match ET by unifying its variables as follows: $ag1 = \text{“}Alice\text{”}$, and $ag2 = \text{“}Bob\text{”}$. This aspect is important because, as we are going to show in the bed allocation domain, we can use variables in RML terms to enforce a specific order of messages. For instance, in this very high-level example, we could say that when a message from $ag1$ to $ag2$ is observed, the only possible consequent message can be a message from $ag2$ to $ag1$. Since the first event has unified the two variables, the second event will have to be a message from *Bob* to *Alice* (otherwise this would be considered a violation). Naturally, this is only the intuition behind it, but it should help to grasp the expressiveness of RML and how variables can be exploited at the protocol level to enforce specific orders amongst the messages.

3 Engineering runtime verification for multi-agent systems

Our approach named RV4JaCa[2] allows the runtime verification of multi-agent systems based on the JaCaMo platform. In Figure 2, we present an overview of the entire approach. RV4JaCa is composed of: (i) a `Sniffer` class, developed in Java, responsible for observing all communication between agents in the MAS; (ii) a CArtAgO artifact named `RV4JaCa Artifact` responsible for analysing the messages observed by the `Sniffer`, transforming them into a JSON (JavaScript Object Notation) object and sending it as a REST (Representational State Transfer) request to the `RML Monitor`. Note that RV4JaCa is not in any way limited to a specific kind of monitor; we used RML simply because it was a most

[2]The source code is available at `https://github.com/DeboraEngelmann/RV4JaCa`

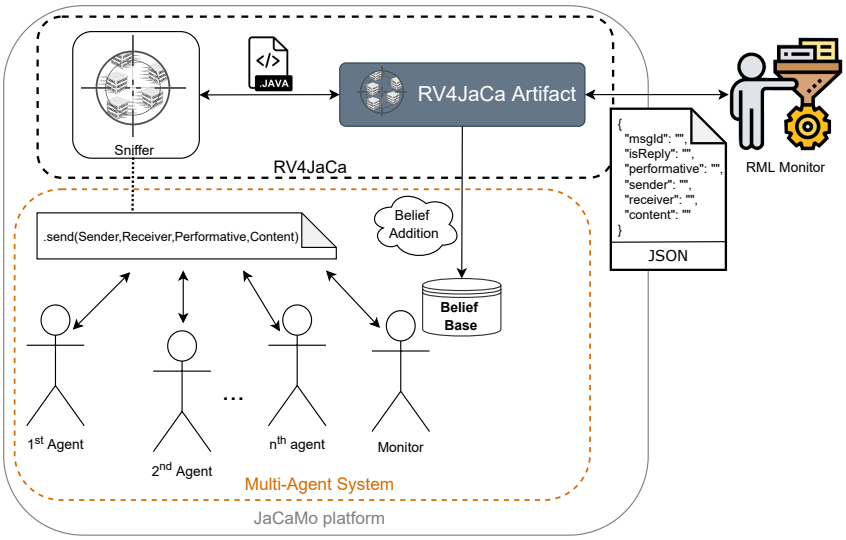

Figure 2: Approach overview for RV in MAS.

suitable candidate for specifying the protocols of our interest. Nonetheless, a different monitor could be as easily integrated as the RML one. In addition, when the `RV4JaCa Artifact` receives the response of the request made to the `RML Monitor` saying that there was a violation, it can add a belief in the `Monitor` agent belief base; (iii) the `RML Monitor` responsible for analysing the events sent by `RV4JaCa Artifact` and verifying the satisfaction or violation of a formal property of interest; and (iv) a `Monitor` agent, which can be added to the system if it is necessary to interfere with agents' behaviour at runtime. In this case, if there is a violation, the `RV4JaCa Artifact` adds a belief to the `Monitor`'s belief base. When the agent perceives this addition, it can react by sending a message to the interested agents warning about the violation. This may trigger some consequent recovery mechanism, which usually is fully domain dependent. On the other hand, the `Monitor` agent can also perform different activities depending on the needs of the system. To clarify how our approach works, we developed a case study in the field of bed allocation.

3.1 Bed allocation case study

We are working on a framework that supports the development of multi-agent applications to assist humans in decision making. One of the domains for which we are building an instance of this framework is hospital bed allocation. As it is shown in Figure 3, to provide interface with natural language processing platforms, such as Dialogflow[3], our framework relies on the use of Dial4JaCa [19]. The `Human user` can interact with the chatbot through text or voice. Dialogflow classifies the interaction intents and sends it to Dial4JaCa, making the request available to the `Communication expert` agent assigned to that specific user. One or more `Communication expert` agents can be instantiated, each one responsible for representing one particular `Human user`. It uses natural language templates [28] to translates the `Assistant` responses (the result of the MAS reasoning) into natural language and send them to its corresponding `Human user`. The `Assistant` agent performs argumentation reasoning [29] and is responsible for communicating with other agents in search of information. Several `Ontology expert` agents can be instantiated, allowing the MAS to consult different ontologies simultaneously given that

[3]https://cloud.google.com/dialogflow/es/docs

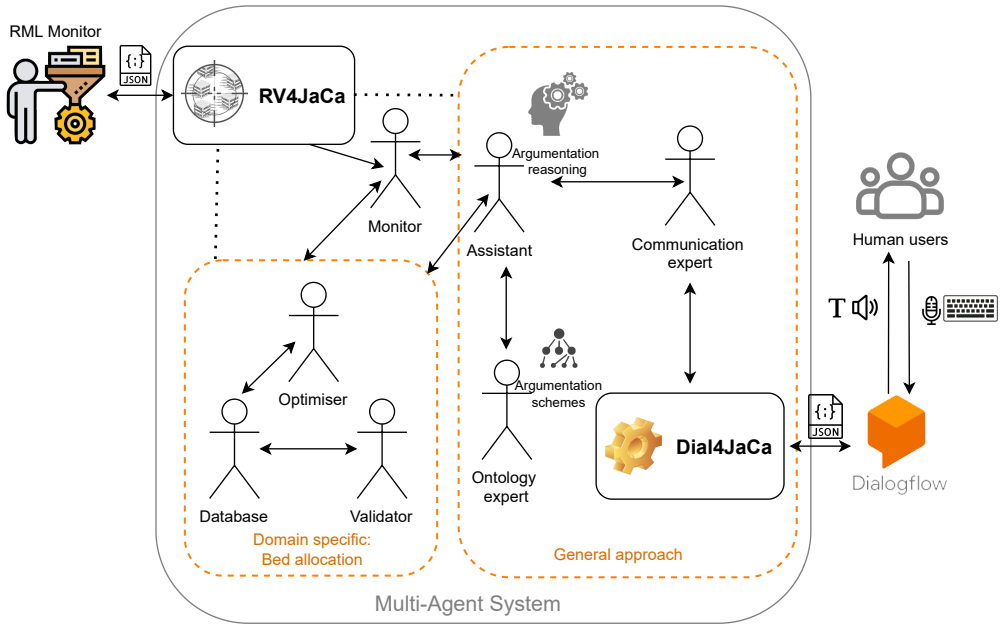

Figure 3: RV4JaCa in a bed allocation domain

each of these agents are able to interface with a specific ontology through a CArtAgO artifact. Such agents can also perform ontological reasoning. Dial4JaCa together with these three types of agents make up our `General approach`.

We can add domain-specific agents to the system to address the specificity of different application domains. For example, in the instance shown in Figure 3, we added specific agents for the bed allocation domain[4]. Among those domain-specific agents, the `Validator` agent is responsible for validating bed allocation plans made by the user (via our system interface) using a PDDL (Planning Domain Definition Language) plan validator; the `Optimiser` agent is responsible for making suggestions for optimised allocations using the GLPSol solver of GLPK[5] (GNU Linear Programming Kit), which is a free open source software for solving linear programming problems; and the `Database` agent is responsible for querying and updating the bed allocation system database.

RV4JaCa has been added to that MAS for collecting information about all messages exchanged between agents and sending them through a REST request to the RML monitor (where the properties that need to be checked are defined). After processing a received message, the monitor returns a result that states whether the message sent from one agent to the other violates any of the properties being checked by it. If a property is violated, RV4JaCa sends the information to the `Monitor` agent so it can add that information to its belief base and warns the agents involved in the exchange of messages that there has been a violation. This makes it possible for our agents to take action to recover from the failure that the breach caused.

The ability to monitor the messages exchanged between the agents in this case study is used for two different purposes. The first one is, according to the performative and content of each move, to verify whether the agents are following the predefined communication protocol. This aspect may be crucial for safety-critical and privacy-preserving aspects. For instance, in a healthcare domain, such as bed alloca-

[4]This scenario is detailed in [18]
[5]http://winglpk.sourceforge.net/

tion, the agents might be expected to follow some specific medical guidelines for the communication of personal information (even amongst themselves). Moreover, when in presence of multiple agents, each one having its own goals, it is common to specify the ideal expected outcome at a more abstract level, where is more natural to reason upon.

The second one is to check if a human participant changed the topic of the conversation without the proper conclusion of the previous topic. When we add humans to the agent-to-agent communication loop, the developer has limited control over the human's interactions in the dialogue. For example, in some cases, when the MAS is performing some specific task, it is important to be able to finish it before starting a new one. But when the completion of this task depends on some human interaction, we have no guarantee that the human will complete the necessary exchange. Naturally, it would be possible to do this check within each agent. However, when we have agents specialised in specific tasks, it is preferable to have all the agent plans related to the specific topic rather that other concerns such as verification, which significantly facilitates implementation and code maintenance.

Below, we report two example properties, written in RML, that have been checked for the bed allocation case study using RV4JaCa.

3.2 RML properties for the bed allocation domain

The first property, which is presented in Listing 1, concerns checking that the user does not change the topic before completing the one currently processed by the agents. In particular, the reported property cares about the `'getValidationResult'` topic. Such a topic relates to the user asking the assistant agent to validate a suggested bed allocation (the corresponding event type is expressed in lines 1-5). When this message is received by the assistant, the protocol goes on, causing a sequence of messages exchanged amongst the assistant, the optimiser, and the validator agents. Note that this part is not reported because it is not of interest for checking the property. After this step, the assistant agent sends back an answer to the user (the event type is in lines 6-11). If this answer is not empty (i.e., the event contains fields `arg1` and `arg2`), then the user is expected to conclude the communication with a certain content (listed in lines 12-17). For instance, the user could reply with an additional message containing `'allocValPatients'`, meaning that the user is fine with the result of the validation, and he/she wants to allocate the corresponding patient to the proposed bed. Naturally, the user might decline the allocation, in that scenario the message would have content `'dontAllocValPatients'`. Similar reasoning goes for the other possible options listed in lines 12-17.

Once the events corresponding to the messages previously mentioned have been specified (lines 1-27), the actual property can be expressed following the RML syntax (lines 28-31). In more detail, we may find in line 28 the definition of the main term denoting the property to check (which in RML is always called `Main`). In this scenario, the principal term corresponds to a sequence of subterms, named `Question`. Such term is defined in line 29, and starts with a question (as defined in lines 1-5). This means that, to comply with the protocol, the first event has to be a message containing a `'question'` from the `'operator'` to the `'assistant'` (in this case regarding the validation of a bed allocation). After that, the property goes on with the `Answer` term (line 30). Inside it, we find a disjunction between two possible alternatives in the protocol. On the left, we may observe an `answer_with_constraint` event, which means, according to lines 6-11, that the `'assistant'` replied to the `'operator'` with a validated result that the latter has to decide upon. On the right, we may observe an event corresponding to any other answer, which in this specific case denotes the case where the result is empty, meaning that no result is available to be sent to the `'operator'`. In the latter case (the right branch), the property ends this cycle, since the communication between the two agents is concluded and new messages concerning new topics can be

exchanged in the future. Instead, in the former case (the left branch), the current cycle is not ended, because the `'assistant'` is still waiting for an answer from the `'operator'` regarding the result sent. This last aspect is handled in the term in line 31, where no `'question'` is admissible from the `'operator'`, only one from those listed in lines 12-17. Upon receiving an event matching one such listed event types, the cycle of the property ends, and as for the right branch, the protocol can move on.

Now, before presenting another property of interest that we have analysed through RV4JaCa, it is important to explicit how a property can be violated. As we mentioned before, we presented which are the events that in certain points of the property are accepted, and why. An RML property is violated whenever given the current term denoting the current state of the property, and a new event, the property does not accept such event. For instance, in the property presented in Listing 1, an event which is different from an answer, after having observed a question, is not acceptable. This can be seen in line 30, where after consuming an event denoting a question, the only possible following events can be an answer requiring additional info (left branch), or a general answer (right branch). Thus, if the observed event is neither of the two, the term is stuck and cannot move on. In RML this translates into a violation of the property, which is then reported back to the monitor agent that in turn will trigger all mechanisms for the agents involved to properly react.

The second RML property we tested in the bed allocation domain is reported in Listing 2. Differently from the property reported in Listing 1, here we do not check the consistency amongst topics; instead, we care about checking that an agent always replies to a question, before posing a new one. As before, the first part of Listing 2 concerns the definition of which events are of interest for the property (lines 1-10). In this specific case, we have questions (lines 1-5), and answers (lines 6-10). Note that, differently from the previous RML property, here we exploit parameters inside the specification. In fact, the event types reported in lines 1-10 are all parametric w.r.t. the agents involved in the communication. This means that such event types do not focus on specific messages exchanged between predefined agents, as in the previous case, but are kept free (through RML parameters, we have late binding on the agents involved in the interaction). This makes the definition of the RML property in line 11 highly parametric, avoiding the need to update the property for each new agent added to the system. The property is defined in line 11, through the standard `Main` term in RML. Since the property is parametric, it starts with the `let` operator which defines the variables used in the term. In this case, the variables used are `ag1` and `ag2` (naturally any other name would have sufficed). After that, the property goes on expecting a question, followed by a corresponding answer. Here, note that in the first event (i.e., the question), the variables are bound to the agents involved in the communication, while in the second event (i.e., the answer), such variables are ground to the previously initialised values. In this way, a question is free to be sent by any possible agent `ag1`, to any possible agent `ag2` in the system (where `ag1` and `ag2` are bound to the observed agents involved in the communication); instead, an answer is constrained to be sent by agent `ag2` to agent `ag1` (with both variables already bound to the respective values through the previously observed question).

As before, also with this property we can ponder on which events can cause a violation. In particular, the property expressed in Listing 2 is violated when after a question between two agents (`ag1`→`ag2`), the following event is not the corresponding answer (`ag2`→`ag1`), but another message (for instance another question).

4 Related Work

In past years, some work has focused on formal verification from a more dynamic viewpoint. In [5], the authors presented a framework to verify Agent Interaction Protocols (AIP) at runtime. The formal-

```
question matches {
    performative:'question',
    sender:'operator', receiver:'assistant',
    content:{name:'getValidationResult'}
};
answer_with_constraint matches
{
    performative:'assert',
    sender:'assistant', receiver:'operator',
    content:{name:'answer', name:'result', arg1:_, arg2:_}
};
constrained_question matches
    {performative:'question', sender:'operator', receiver:'assistant',
    content:{name:'allocValPatients'}} |
    {performative:'question', sender:'operator', receiver:'assistant',
    content:{name:'getOptimisedAllocation'}} |
    {performative:'question', sender:'operator', receiver:'assistant',
    content:{name:'dontAllocValPatients'}} |
    {performative:'question', sender:'operator', receiver:'assistant',
    content:{name:'allocValidValPatients'}} |
    {performative:'question', sender:'operator', receiver:'assistant',
    content:{name:'allocValPatients'}};
a_question matches
{
    performative:'question',
    sender:'operator', receiver:'assistant'
};
an_answer matches
{
    performative:'assert',
    sender:'assistant', receiver:'operator'
};
Main = Question*;
Question = (question Answer);
Answer = (answer_with_constraint ConstrainedQuestion) \/ (an_answer);
ConstrainedQuestion = constrained_question Answer;
```

Listing 1: The RML specification for checking that no change of topic is observed after a validation result has been requested by the user.

ism used in this work allows the introduction of variables, that are then used to constrain the expected behaviour in a more expressive way. In [21], the same authors proposed an approach to verify AIPs at runtime using multiple monitors. This is obtained by decentralising the global specification (specified as a Trace Expression [4]), which is used to represent the global protocol, into partial specifications denoting the single agents' perspective. Both those approaches are based on the formalism serving as building block for RML's semantics. From this perspective, RV4JaCa is an evolution of these approaches in two ways: (i) it allows general-purpose verification since no constraint is assumed on the monitor side, except for being capable of receiving and sending JSON messages; (ii) to be self-contained, RV4JaCa natively supports RML monitors, and because of that it allows a more intuitive and high-level protocol specification. In [8, 32], other approaches to runtime verification of agent interactions are proposed, and in [26] a framework for dynamic adaptive MAS (DAMS-RV) based on an adaptive feedback loop is presented. Other approaches to MAS RV include the spin-off proposals from the SOCS project where the SCIFF computational logic framework [3] is used to provide the semantics of social integrity constraints. To model MAS interaction, an expectation-based semantics specifies the links between observed and expected events, providing a means to test runtime conformance of an actual conversation with respect to a given interaction protocol [34]. Similar work has been performed using commitments [14].

```
question(ag1, ag2) matches
{
    performative:'question',
    sender:ag1, receiver:ag2
};
answer(ag1, ag2) matches
{
    performative:'assert',
    sender:ag1, receiver:ag2
};
Main = {let ag1, ag2; question(ag1, ag2) answer(ag2, ag1)}*;
```

Listing 2: The RML specification for checking that an agent always replies before sending messages about something else.

To the best of our knowledge, RV4JaCa is the first approach that integrates RV within JaCaMo for verifying agent interaction protocols.

5 Conclusions and Future Work

Communications between agents play a key role in the functioning of a multi-agent system since, in practice, agents rarely act alone, they usually inhabit an environment that contains other agents. Therefore, an extra layer of security that allows us to verify key aspects of this message exchange adds great value, in addition to great possibilities for improvement, since certain aspects do not need to be considered when developing each of the agents. Using this type of formal verification at runtime allows us to standardise the interaction between agents through previously defined protocols that all agents must follow and, if they do not, react in a way that the execution is not negatively affected by the effects that were caused by this protocol deviation.

On the other hand, the checks done with RV are not limited to protocol validation. More specific properties of each application domain can also be verified once the monitor has access to the content of the exchanged messages. Even the execution of certain routines or functions according to the direction in which the conversations between the agents go can be done. For example, recording the results obtained during the agents' reasoning in a database without agents having the responsibility to carry out the registrations themselves. Or even sending an automatic email to a supervisor if any property identified by an agent and communicated to another is outside certain parameters. Therefore, depending on the MAS's domain, there is a range of possibilities in which RV can be used.

Based on that, we proposed RV4JaCa, an approach to integrate multi-agent systems and runtime verification. Our approach was built using JaCaMo and RML, and it provides significant progress toward obtaining guarantees of the correct execution of the MAS. The case study presented in this paper demonstrates the use of RV4JaCa in practice, also showing promising preliminary results. In addition, it is important to note that our approach can be applied to different scenarios in different MAS. Based on the presented case study, we created two distinct properties to be checked at runtime in the system. The first one is to allow agents to be alerted if there is an unexpected change in the topic of conversation by the human user who is interacting with the system. And the second one is capable of verifying that the previously defined communication protocol between agents is being followed correctly (in particular, question-answer relations).

As MAS have been used to build systems focused on explainability, RV's extra safety layer is certainly useful, since to achieve explainability in MAS much communication between agents and humans

needs to be carried out. Also, we need to take into account that the developer does not have complete control over the interactions that human users can make during a dialogue. In this sense, RV allows us to avoid unexpected and probably inappropriate system behaviour.

As future work, we plan to further extend RV4JaCa to more than interaction protocols. For instance, it would be relevant to check the agents' state of mind as well. Moreover, through RV4JaCa we also want to create a library of verifiably correct agent interaction protocols to be used in different scenarios involving both agents and humans in the loop.

References

[1] Amina Adadi & Mohammed Berrada (2018): *Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI)*. *IEEE Access* 6, pp. 52138–52160, doi:10.1109/ACCESS.2018.2870052.

[2] Wolfgang Ahrendt, Jesús Mauricio Chimento, Gordon J. Pace & Gerardo Schneider (2017): *Verifying data- and control-oriented properties combining static and runtime verification: theory and tools*. *Formal Methods Syst. Des.* 51(1), pp. 200–265, doi:10.1007/s10703-017-0274-y.

[3] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello & Paolo Torroni (2005): *The SCIFF Abductive Proof-Procedure*. In Stefania Bandini & Sara Manzoni, editors: *AI*IA 2005: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence, Milan, Italy, September 21-23, 2005, Proceedings*, *Lecture Notes in Computer Science* 3673, Springer, pp. 135–147, doi:10.1007/11558590_14.

[4] Davide Ancona, Angelo Ferrando & Viviana Mascardi (2016): *Comparing Trace Expressions and Linear Temporal Logic for Runtime Verification*. In Erika Ábrahám, Marcello M. Bonsangue & Einar Broch Johnsen, editors: *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science* 9660, Springer, pp. 47–64, doi:10.1007/978-3-319-30734-3_6.

[5] Davide Ancona, Angelo Ferrando & Viviana Mascardi (2017): *Parametric Runtime Verification of Multiagent Systems*. In Kate Larson, Michael Winikoff, Sanmay Das & Edmund H. Durfee, editors: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, ACM, pp. 1457–1459, doi:10.5555/3091125.3091328.

[6] Davide Ancona, Luca Franceschini, Angelo Ferrando & Viviana Mascardi (2021): *RML: Theory and Practice of a Domain Specific Language for Runtime Verification*. *Science of Computer Programming* 205, p. 102610, doi:10.1016/j.scico.2021.102610.

[7] Sule Anjomshoae, Amro Najjar, Davide Calvaresi & Kary Främling (2019): *Explainable agents and robots: Results from a systematic literature review*. In: *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1078–1088, doi:10.5555/3306127.3331806.

[8] Najwa Abu Bakar & Ali Selamat (2013): *Runtime Verification of Multi-agent Systems Interaction Quality*. In Ali Selamat, Ngoc Thanh Nguyen & Habibollah Haron, editors: *Intelligent Information and Database Systems - 5th Asian Conference, ACIIDS 2013, Kuala Lumpur, Malaysia, March 18-20, 2013, Proceedings, Part I*, *Lecture Notes in Computer Science* 7802, Springer, pp. 435–444, doi:10.1007/978-3-642-36546-1_45.

[9] Ezio Bartocci, Yliès Falcone, Adrian Francalanza & Giles Reger (2018): *Introduction to Runtime Verification*. In Ezio Bartocci & Yliès Falcone, editors: *Lectures on Runtime Verification - Introductory and Advanced Topics*, *Lecture Notes in Computer Science* 10457, Springer, Cham, pp. 1–33, doi:10.1007/978-3-319-75632-5_1.

[10] Ezio Bartocci, Yliès Falcone, Adrian Francalanza & Giles Reger (2018): *Introduction to Runtime Verification*. In Ezio Bartocci & Yliès Falcone, editors: *Lectures on Runtime Verification - Introductory and Advanced Topics*, *Lecture Notes in Computer Science* 10457, Springer, pp. 1–33, doi:10.1007/978-3-319-75632-5_1.

[11] Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci & Andrea Santi (2013): *Multi-agent oriented programming with JaCaMo*. Science of Computer Programming 78(6), pp. 747–761, doi:10.1016/j.scico.2011.10.004.

[12] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix & Amal El Fallah Seghrouchni, editors (2009): *Multi-Agent Programming, Languages, Tools and Applications*. Springer, doi:10.1007/978-0-387-89299-3.

[13] Rafael H Bordini, Jomi Fred Hübner & Michael Wooldridge (2007): *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, doi:10.1002/9780470061848.

[14] Federico Chesani, Paola Mello, Marco Montali & Paolo Torroni (2009): *Commitment Tracking via the Reactive Event Calculus*. In Craig Boutilier, editor: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 91–96, doi:10.5555/1661445.1661461. Available at `http://ijcai.org/Proceedings/09/Papers/026.pdf`.

[15] Edmund M Clarke (1997): *Model checking*. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer, pp. 54–56, doi:10.1007/BFb0058022.

[16] Louise A. Dennis, Michael Fisher, Matthew P. Webster & Rafael H. Bordini (2012): *Model checking agent programming languages*. Autom. Softw. Eng. 19(1), pp. 5–63, doi:10.1007/s10515-011-0088-x.

[17] Ivan Donadello, Mauro Dragoni & Claudio Eccher (2020): *Explaining reasoning algorithms with persuasiveness: a case study for a behavioural change system*. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 646–653, doi:10.1145/3341105.3373910.

[18] Débora C. Engelmann, Lucca Dornelles Cezar, Alison R. Panisson & Rafael H. Bordini (2021): *A Conversational Agent to Support Hospital Bed Allocation*. In André Britto & Karina Valdivia Delgado, editors: *Intelligent Systems - 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29 - December 3, 2021, Proceedings, Part I*, *Lecture Notes in Computer Science* 13073, Springer, pp. 3–17, doi:10.1007/978-3-030-91702-9_1.

[19] Débora C. Engelmann, Juliana Damasio, Tabajara Krausburg, Olimar Teixeira Borges, Mateus da Silveira Colissi, Alison R. Panisson & Rafael H. Bordini (2021): *Dial4JaCa - A Communication Interface Between Multi-agent Systems and Chatbots*. In Frank Dignum, Juan Manuel Corchado & Fernando de la Prieta, editors: *Advances in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection - 19th International Conference, PAAMS 2021, Salamanca, Spain, October 6-8, 2021, Proceedings*, *Lecture Notes in Computer Science* 12946, Springer, pp. 77–88, doi:10.1007/978-3-030-85739-4_7.

[20] Jacques Ferber, Olivier Gutknecht & Fabien Michel (2003): *From Agents to Organizations: An Organizational View of Multi-agent Systems*. In Paolo Giorgini, Jörg P. Müller & James Odell, editors: *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, *Lecture Notes in Computer Science* 2935, Springer, pp. 214–230, doi:10.1007/978-3-540-24620-6_15.

[21] Angelo Ferrando, Davide Ancona & Viviana Mascardi (2017): *Decentralizing MAS Monitoring with DecAMon*. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, ACM, pp. 239–248, doi:10.5555/3091125.3091164.

[22] Michael Fisher, Viviana Mascardi, Kristin Yvonne Rozier, Bernd-Holger Schlingloff, Michael Winikoff & Neil Yorke-Smith (2021): *Towards a framework for certification of reliable autonomous systems*. Auton. Agents Multi Agent Syst. 35(1), p. 8, doi:10.1007/s10458-020-09487-2.

[23] Bryan Horling & Victor R. Lesser (2004): *A survey of multi-agent organizational paradigms*. Knowl. Eng. Rev. 19(4), pp. 281–316, doi:10.1017/S0269888905000317.

[24] Jomi F Hubner, Jaime S Sichman & Olivier Boissier (2007): *Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels*. International Journal of Agent-Oriented Software Engineering 1(3-4), pp. 370–395, doi:10.1504/IJAOSE.2007.016266.

[25] Martin Leucker & Christian Schallhart (2009): *A brief account of runtime verification*. J. Log. Algebraic Methods Program. 78(5), pp. 293–303, doi:10.1016/j.jlap.2008.08.004.

[26] Yoo Jin Lim, Gwangui Hong, Donghwan Shin, Eunkyoung Jee & Doo-Hwan Bae (2016): *A runtime verification framework for dynamically adaptive multi-agent systems*. In: *2016 International Conference on Big Data and Smart Computing, BigComp 2016, Hong Kong, China, January 18-20, 2016*, IEEE Computer Society, pp. 509–512, doi:10.1109/BIGCOMP.2016.7425981.

[27] Donald W. Loveland (1978): *Automated theorem proving: a logical basis*. *Fundamental studies in computer science* 6, North-Holland. Available at `https://www.worldcat.org/oclc/252520243`.

[28] Alison R. Panisson, Débora C. Engelmann & Rafael H. Bordini (2021): *Engineering Explainable Agents: An Argumentation-Based Approach*. In Natasha Alechina, Matteo Baldoni & Brian Logan, editors: *Engineering Multi-Agent Systems - 9th International Workshop, EMAS 2021, Virtual Event, May 3-4, 2021, Revised Selected Papers*, *Lecture Notes in Computer Science* 13190, Springer, pp. 273–291, doi:10.1007/978-3-030-97457-2_16.

[29] Alison R Panisson, Felipe Meneguzzi, Renata Vieira & Rafael H Bordini (2014): *An approach for argumentation-based reasoning using defeasible logic in multi-agent programming languages*. In: *11th International Workshop on Argumentation in Multiagent Systems*, pp. 1–15.

[30] Anand S Rao & Michael P Georgeff (1995): *BDI agents: from theory to practice*. In: *Icmas*, 95, pp. 312–319.

[31] Alessandro Ricci, Michele Piunti, Mirko Viroli & Andrea Omicini (2009): *Environment programming in CArtAgO*. In: *Multi-agent programming*, Springer, pp. 259–288, doi:10.1007/978-0-387-89299-3_8.

[32] Chittra Roungroongsom & Denduang Pradubsuwun (2015): *Formal Verification of Multi-agent System Based on JADE: A Semi-runtime Approach*. In: *Recent Advances in Information and Communication Technology 2015*, Springer, pp. 297–306, doi:10.1007/978-3-319-19024-2_30.

[33] Daniela Schmidt, Alison R. Panisson, Artur Freitas, Rafael H. Bordini, Felipe Meneguzzi & Renata Vieira (2016): *An Ontology-Based Mobile Application for Task Managing in Collaborative Groups*. In Zdravko Markov & Ingrid Russell, editors: *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016, Key Largo, Florida, USA, May 16-18, 2016*, AAAI Press, pp. 522–526. Available at `http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/view/12956`.

[34] Paolo Torroni, Pinar Yolum, Munindar P. Singh, Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma & Paola Mello (2009): *Modelling Interactions via Commitments and Expectations*. In Virginia Dignum, editor: *Handbook of Research on Multi-Agent Systems - Semantics and Dynamics of Organizational Models*, IGI Global, pp. 263–284, doi:10.4018/978-1-60566-256-5.ch011.

[35] Michael Winikoff (2017): *BDI agent testability revisited*. *Auton. Agents Multi Agent Syst.* 31(5), pp. 1094–1132, doi:10.1007/s10458-016-9356-2.

[36] Michael Winikoff (2017): *Debugging Agent Programs with Why?: Questions*. In Kate Larson, Michael Winikoff, Sanmay Das & Edmund H. Durfee, editors: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, ACM, Richland, SC, pp. 251–259, doi:10.5555/3091125.3091166.

[37] Michael Wooldridge (2002): *An Introduction to MultiAgent Systems*. John wiley & sons Ltd.