Distributed Runtime Verification of JADE and Jason Multiagent Systems with Prolog*

Daniela Briola, Viviana Mascardi, and Davide Ancona

DIBRIS, Genoa University, Italy
daniela.briola,viviana.mascardi,davide.ancona@unige.it

1 Introduction

Verifying properties of interactions taking place inside open, complex, distributed, dynamic systems is of paramount importance for most applications and is mandatory for safety-critical ones. Verification can take place at design time (offline or static verification) or at runtime (online or dynamic). For runtime verification some layer between the monitor executing the verification engine and the system under monitoring must exist, so that actual interactions can be intercepted and the compliance of each one against the protocol can be checked. A common way to improve efficiency and fault tolerance of the runtime verification is to distribute it among many monitors. This requires that the protocol is projected onto subsets of participants.

If the system has been engineered as a multiagent system (MAS), which is a good option when openness, complexity, distribution, dynamics are characterizing features, then the choice of either JADE, http://jade.tilab.com/, or Jason http://jason.sourceforge.net/, as the platform for implementing it may be a very natural one. JADE, implemented in Java, is the state-of-the-art tool for MAS development and has been used for many real industrial applications. Jason, implemented in Java as well and based on a Prolog engine built from scratch by its developers, is one of the most widely used frameworks when the agents under development are designed according to the Belief, Desire, Intentions (BDI) architecture.

Due to the wide range of possible application fields of Jason and to the large amount of real use cases of JADE, being able to verify interactions taking place in MASs implemented in one of these two frameworks is a concrete step towards making MASs more reliable and enhancing their industrial and commercial usability. In this demo we show our contribution for the achievement of this goal.

We have in fact designed and implemented a framework for distributed runtime verification of MASs and ad hoc interfaces for monitoring JADE and Jason interactions. The framework consists of four layers: **(1)** a formalism for describing agent interaction protocols (AIPs) based on constrained global types [1] and their extension with attributes [7]; **(2)** a mechanism for projecting AIPs onto subsets of agents, obtaining a new protocol in the same formalism of constrained global types [2]; **(3)** a mechanism for verifying that interactions are compliant

* Paper presented at the CILC 2014 Demo Session, based on published material [3, 4].

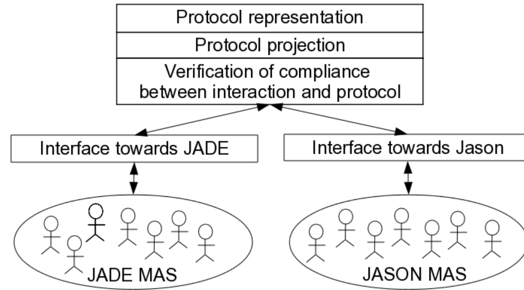

Fig. 1. Our modular framework for distributed runtime verification of MASs.

with the AIP [3]; and **(4)** a mechanism for intercepting messages involving the agents under monitoring, be them JADE or Jason ones, in a way as transparent as possible.

The strength of our framework, represented in Figure 1, is its high modularity and potential for code reuse: the first three layers are independent from the actual MAS under monitoring and have been implemented in Prolog. The "protocol representation" and "compliance verification" layers have been tested and improved over time, reaching an almost stable version now, whereas the projection layers works under the assumption that the protocol contains no attributes (namely, it is a "plain" constrained global type) and has not been tested extensively yet. The fourth layer depends on the MAS framework under monitoring: nothing prevents us from adding new agent frameworks at the bottom of our architecture by developing ad hoc mechanisms for message interception, still leaving the first three layers unchanged. By exploiting the components offered by our stacked framework it is possible to implement both monitors external to the MAS, implemented as completely separate processes that do not intervene in the observed system, and agents which are able to monitor the protocol executions and have the power to intervene when they detect a violation. Associating the "compliance verification" capability with an artifact (as in the first case) or with an agent (as in the second case) are two different design choices, each with pros and cons. We experimented both approaches, as discussed in Section 3 where monitoring is performed by a Java artifact that does not intervene in the MAS activity, and in Section 4 where the Jason agent in charge of the monitoring activity can prevent other agents from sending non compliant messages.

Whatever the choice, compliance verification should be an efficient process. Although efficiency issues are still to be explored, distributing the runtime verification by projecting onto subsets of agents could be a way to balance the load of the monitoring activity among more entities.

2 Background

Global types [6] are a behavioral type and process algebra approach to the problem of specifying and verifying multiparty interactions between distributed components. We took inspiration from global types to propose a formalism,

constrained global types, suitable for representing AIPs. Because of space constraints we cannot go into the details of the formalism which can be found in [7]. Since attribute global types are interpreted coinductively, it is possible to specify protocols that are not allowed to terminate like for example the SERVER protocol defined by the equation

```
SERVER = (receive_request,0):(serve_request,0):SERVER
```

where `SERVER` is a logical variable which is unified with a recursive (or cyclic, or infinite) Prolog term consisting of a `receive_request` producer interaction type, followed by a `serve_request` producer interaction type (both requiring 0 consumers), followed by the term itself. This protocol models the (infinite) behavior of a server which is always ready to receive and serve requests; the only valid interaction trace is the infinite sequence `receive_request serve_request receive_request serve_request ...`.

By means of attribute global types we were able to concisely represent protocols which are well known in the concurrent systems and MAS communities like the Alternating Bit Protocol (en.wikipedia.org/wiki/Alternating_bit_protocol) and the FIPA Iterated Contract Net Protocol (fipa.org/specs/fipa00030). FYPA (Find Your Path, Agent! [5]) is not as well known, but is a negotiation protocol for a real MAS used by Ansaldo STS, and is far more complex than the two others. We exploited our formalism to model FYPA as well [8].

Constrained global types can be easily expressed as a set of Prolog equations like the one defining `SERVER`. Attributes and constraints on their values are represented as additional Prolog facts. In order to allow agents to verify only a sub-protocol of the global interaction protocol, we designed a projection algorithm that takes a constrained global type and a set of agents Ags as input, and returns a constrained global type which contains only interactions involving agents in Ags as output [2]. Projection can be described as a function $\Pi : \mathcal{CT} \times \mathcal{P}(AGS) \to \mathcal{CT}$ where $\mathcal{CT}$ is the set of constrained global types and AGS is the set of agents. The intuition besides the algorithm is that interactions that do not involve agents in Ags are removed from the projected constrained global type.

Whatever the protocol to be monitored (global one or projection) and the framework (JADE or Jason), a monitor keeps track of the runtime evolution of the protocol by saving its current state (which is an attribute global type) and checking that each interaction taking place in the MAS is allowed by the current state (namely, can lead to a new state by means of the transition function which defines the semantics of attribute global types, implemented in Prolog). If the interaction is not allowed, an error is reported. The monitor also checks agents responsiveness by means of a time-out whose value can be set by the user: if the current state of the monitor corresponds to the empty protocol (that is, the protocol must terminate), then the monitor reports an error as soon as an interaction is detected (independently of the time-out); if the current state is not final (that is, the protocol is not allowed to terminate), then the monitor reports a warning as soon as the time-out expires, if no interaction is detected (and of

course an error is reported in case an invalid interaction is detected before the time-out).

3 Runtime Verification of JADE MASs

In order to verify the interactions taking place in a JADE MAS, we have designed a monitor meeting the following requirements for non intrusiveness and code reuse [4]: **(1)** the monitor must be able to supervise the execution of the MAS *without interfering with it*, **(2)** the monitor activity must require *no changes to the agents' code*, **(3)** the formalism for representing the AIP must be *attribute global types*, and **(4)** the Prolog code already developed for implementing verification of interactions w.r.t. attribute global types and for protocol projection *must be re-used as it is*.

To meet requirements 1 and 2 we extended the JADE Sniffer agent which is able to sniff all the messages exchanged during the execution of the MAS in a non intrusive way: JADE is developed under the LGPL (Lesser General Public License) and the Java source code is available to the research community, so we were able to modify it to achieve our goals.

To meet requirements 3 and 4 we exploited the JLP library[1] providing a bidirectional interface between Java and SWI Prolog. As all our previous work on attribute global types was implemented in Prolog, allowing our JADE Monitor to consult Prolog programs and to call Prolog predicates was the best way to ensure reusability.

The monitor reads a file containing the Prolog code implementing verification and projection, and a configuration file listing the agents to be monitored, and onto which the protocol projection will be performed. A log file is written as the monitoring goes on. The Prolog file contains definitions for three predicates:

– `initialize(LogFile, SniffedAgents)` which sets LogFile as the file where writing the outcome of the verification, and projects the global protocol - which must be defined by the global_type/1 predicate -, onto SniffedAgents.

– `remember(ParsedMsg)` which inserts the Prolog representation of the JADE sniffed message into a message list, where messages are ordered by time stamp (if they have a time stamp, which is not mandatory) or in order of arrival.

– `verify(CurrentTime)` which verifies the compliance of each message remembered in the message list and whose time stamp is lower than `CurrentTime` to the global protocol.

These predicates are called in different methods of the monitor code, implementing the wanted behavior.

With this approach no changes are required to the monitored agents and hence existing MASs can be monitored without accessing to their code, but the monitor detects a violation only after it took place and even in case of a protocol violation the MAS execution goes on.

[1] http://www.swi-prolog.org/packages/jpl/java_api/

4 Runtime Verification of Jason MASs

Since Jason agents can integrate Prolog facts and rules for defining their knowledge, the Jason monitor [3] can be generated in a trivial way by integrating directly into its code the Prolog code for protocol specification, monitoring and projection that the JADE monitor must instead read from the Prolog file. The way interactions are sniffed in Jason depends on some assumption on the agents' code and requires some modifications to it: we assume that agents interact via asynchronous exchange of messages with `tell` performatives; their original code must be modified in the following way:

1. the `.send` built-in action for message delivery is replaced by `!my_send` and
2. two plans must be added for managing the interaction with the monitor.

The first plan is triggered by the `!my_send` internal goal; `my_send` has the same signature as the `.send` internal action, but, instead of sending a message with performative `Perf` and `Content` to `Receiver`, it sends a `tell` message to the monitor with content `msg(Sender, Receiver, Perf, Content)`. When received, this message will be checked by the monitor against the attribute global type representing the protocol, as briefly explained in Section 2.

The second plan is triggered by the reception of the monitor's message that allows the agent to actually send `Content` to `Receiver`. In reaction to the reception of such a message, the agent sends the corresponding message to the expected agent.

With this approach the code of the monitored agents must be conceived and implemented in a way that makes monitoring possible, but the monitor detects a violation prior than the actual message is sent and can stop the agent violating the protocol by not allowing it to send the "wrong" message.

References

1. D. Ancona, M. Barbieri, and V. Mascardi. Constrained global types for dynamic checking of protocol conformance in multi-agent systems. In *SAC*. ACM, 2013.
2. D. Ancona, D. Briola, A. E. F. Seghrouchni, V. Mascardi, and P. Taillibert. Efficient verification of MASs with projections. In *EMAS Pre-proceedings*, 2014.
3. D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In *DALT X*, volume 7784 of *LNAI*. Springer, 2012.
4. D. Briola, V. Mascardi, and D. Ancona. Distributed runtime verification of JADE multiagent systems. In *IDC*, Studies in Computational Intelligence. Springer, 2014.
5. D. Briola, V. Mascardi, M. Martelli, R. Caccia, and C. Milani. Dynamic resource allocation in a MAS: A case study from the industry. In *WOA*, 2009.
6. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP*, LNCS, pages 2–17. Springer, 2007.
7. V. Mascardi and D. Ancona. Attribute global types for dynamic checking of protocols in logic-based multiagent systems. *TPLP*, 13(4-5-Online-Supplement), 2013.
8. V. Mascardi, D. Briola, and D. Ancona. On the expressiveness of attribute global types: The formalization of a real multiagent system protocol. In *AI*IA*, 2013.