Coping with Bad Agent Interaction Protocols When Monitoring Partially Observable Multiagent Systems

Davide Ancona, Angelo Ferrando, Luca Franceschini, and Viviana Mascardi(✉)

Università degli Studi di Genova, Genoa, Italy
{davide.ancona,angelo.ferrando,luca.franceschini,
viviana.mascardi}@dibris.unige.it

Abstract. Interaction Protocols are fundamental elements to provide the entities in a system, be them actors, agents, services, or other communicating pieces of software, a means to agree on a global interaction pattern and to be sure that all the other entities in the system adhere to it as well. These "global interaction patterns" may serve different purposes: if the system does not yet exist, they may specify the allowed interactions in order to drive the system's implementation and execution. If the system exists before and independently from the protocol, the protocol may still specify the allowed interactions, but it cannot be used to implement them. Its purpose in this case is to monitor that the actual system does respect the rules (runtime verification). Tagging some protocols as good ones and others as bad is common to all the research communities where interaction is crucial, and it is not surprising that some protocol features are recognized as bad ones everywhere. In this paper we analyze the notion of good, bad and ugly protocols in the MAS community and outside, and we discuss the role that bad protocols, despite being bad, may play in a runtime verification scenario where not all the events and interaction channels can be observed.

Keywords: Bad agent interaction protocols · Monitoring
Distributed Runtime Verification · Partial observability

1 Introduction

Interaction Protocols are a key ingredient in MASs as they explicitly represent the agents expected/allowed communicative patterns and can be used either to check the compliance of the agent actual behavior w.r.t. expected one [1,10] or to drive the agent behavior itself [4].

Interaction protocols are also crucial outside the MAS community: what we name an "Agent Interaction Protocol", AIP, is referenced as a "Choreography" in the Service Oriented Computing (SOC) community [36] and as a "Global Type" in the multiparty session types one [15,31]. In the MAS community, AIPs describe interaction patterns characterizing the system as a whole. This global

Y. Demazeau et al. (Eds.): PAAMS 2018, LNAI 10978, pp. 59–71, 2018.
https://doi.org/10.1007/978-3-319-94580-4_5

viewpoint is supported by many formalisms and notations such as AUML [32], commitment machines and their extensions [11,12,22,40,42], the Blindingly Simple Protocol Language (BSPL) and its Splee extension [21,37], the Hierarchical Agent Protocol Notation (HAPN) [41], trace expressions [7]. A systematic comparison of these approaches – apart from AUML which is not formal, and not supported by a textual notation – can be found in [9].

When moving from the specification to the execution stage, the AIP must be enacted by agents in the MAS: besides the global description of the protocol, the "local" description of the AIP portion each agent is in charge of, is required to run the AIP. The AIP enactment is usually left to Computer-Aided Software Engineering tools that move from AIP diagrams directly into agent skeletons in some concrete agent oriented programming language [23,30], or to algorithms that translate the AIP textual representation to some abstract, intermediate formalism for modeling the local viewpoint [19,26]. Such intermediate formalisms are not perceived as the main target of the MAS research and no standardization effort has been put on them. In the SOC community, on the contrary, formalisms exist for modeling both the global and the local perspectives. As observed by [35], WS-CDL[1] follows an interaction-oriented ("global") approach, whereas in BPEL4Chor[2] the business process of each partner involved in a choreography is specified using an abstract version of BPEL[3]: BPEL4Chor follows a process-oriented ("local") approach. In the multiparty session types community, the main emphasis is on type-checking aspects: the formalism used to represent global types is relevant, as well as its expressive power, but even more relevant are the properties of the "global to local" projection function w.r.t. type-safety [25]. Finally, in the Distributed Runtime Verification (DRV) community [29], one of the most relevant issues is how to automatically generate a monitor (or a set of monitors) from a given protocol, and to use it (them) to dynamically monitor the existing system's behaviour.

Our work lies in the intersection of the MAS and DRV research areas. We are interested in exploring under which conditions a good AIP can turn into a bad one, if it is possible to exploit such bad protocols anyway for DRV purposes, and in providing an implemented tool for automatically generating a set of MAS monitors under these conditions. To this aim, we developed RIVERtools [27] which supports the development of AIPs modeled using the trace expressions formalism [3,7] via a user-friendly GUI. RIVERtools implements static controls to check if a protocol is good, ugly or bad, and supports the user in finding out how to (partially) decentralize the runtime verification process also in the last case [28]. The adoption of RIVERtools for this purpose in exemplified in the companion demo paper [6], which addresses the most practical aspects of our investigation: this paper is mainly centered around the notions of bad, good, and

[1] Web Services Choreography Description Language Version 1.0 W3C Candidate Recommendation 9 November 2005, https://www.w3.org/TR/ws-cdl-10/.

[2] BPEL4Chor Choreography Extension for BPEL, http://www.bpel4chor.org/.

[3] Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11 April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

ugly protocol in the literature, and on the observability-driven transformation of AIPs which, under known environmental/network constraints, may turn a good protocol into a bad one which can nevertheless serve for DRV purposes.

2 The Good, the Bad and the Ugly

The Good. Let us consider the following interaction protocol expressed in natural language:

1. Alice sends a whatsApp message to her mother Barbara asking her to buy a book (plus some implausible excuse for not doing it herself...);
2. Barbara sends an email message to her friend Carol, responsible for the Book Shop front end, to reserve that book;
3. Carol receives Barbara email and sends a whatsApp message to Dave, the responsible of the Book Shop warehouse, to check the availability of the book and order it if necessary;
4. Dave checks if the book is available in the warehouse;
   (a) if it is,
      (i.) he sends a whatsApp message to Emily who is in charge for physically managing the books and informing the clients if their requests can be satisfied immediately;
      (ii.) Emily takes the book to the front end and sends a confirmation email to Barbara telling that the book is already there;
   (b) otherwise,
      i. Dave sends an email to Frank, the point of contact for the publisher of the required book, and orders it;
      ii. Frank sends a confirmation to Barbara via whatsApp telling her that the book will be available in two days.

For readability, we express the protocol using a more compact and formal syntax where $a \overset{mns,cnt}{\Longrightarrow} b$ stands for "agent a sends a message with content cnt via communication means mns to agent b". The symbol: stands for the prefix operator ($int{:}P$ is the protocol whose first allowed interaction is int, and the remainder is the protocol P) and has precedence over the other operators, $\vee$ stands for mutual exclusiveness, and ϵ represents the empty protocol:

$$P1 = alice \overset{wa,buy}{\Longrightarrow} barbara{:}barbara \overset{em,reserve}{\Longrightarrow} carol{:}carol \overset{wa,checkAvail}{\Longrightarrow} dave{:}$$
$$(dave \overset{wa,take2shop}{\Longrightarrow} emily{:}emily \overset{em,availNow}{\Longrightarrow} barbara{:}\epsilon \ \vee$$
$$dave \overset{em,order}{\Longrightarrow} frank{:}frank \overset{wa,avail2Days}{\Longrightarrow} barbara{:}\epsilon)$$

P1 receives the unanimous appreciation whatever the research community. In fact, two very intuitive properties are met: 1. apart from the first one, the message that each agent sends is a reaction to the message it just received, and there is an evident cause-effect link between two sequential messages; 2. in case some mutually exclusive choice must be made, the choice is up to only

one agent involved in the protocol, and hence it is feasible. These good properties take different names depending on the research communities and on the authors. The first one is named, for example, "sequentiality" [20], "connectedness for sequence" [35], "explicit causality" [37]; the second "knowledge for choice" [20], "local choice" [34], "unique point of choice" [35]. Meeting these two properties is closely related to the absence of covert channels; they ensure that all communications between different participants are explicitly stated, and rule out those protocols whose implementation or enactment relies on the presence of secret/invisible communications between participants: a protocol description must contain all and only the interactions used to implement it [20].

The Bad. Those protocols that do not respect the connectedness for sequence and unique point of choice properties are bad, and it is not difficult to see why. Let us consider protocol $P2$:

$$P2 = alice \overset{wa,buy}{\Longrightarrow} barbara{:}carol \overset{wa,checkAvail}{\Longrightarrow} dave{:} \\ (dave \overset{wa,take2shop}{\Longrightarrow} emily{:}\epsilon \ \vee \ frank \overset{wa,avail2Days}{\Longrightarrow} barbara{:}\epsilon)$$

The protocol states that *carol* can send a *checkAvail* message to *dave* only after *alice* has sent a *buy* message to *barbara*, but how can *carol* know if and when *alice* sent that message? Also, the protocol states that either *dave* sends a message to *emily*, or *frank* sends a message to *barbara*: how can *frank* know if he is allowed to send a message to *barbara*, without coordinating with *dave* via some covert channel not shown in the protocol?

The Ugly. Protocols which are not syntactically correct are ugly, and are ignored by all the authors. However, some protocols may be ugly even if they are syntactically correct, for example if they are characterized by:

- "Causality unsafety": consider the two shuffled sequences $carol \overset{wa,buy}{\Longrightarrow} dave{:}\epsilon \mid alice \overset{wa,buy}{\Longrightarrow} barbara{:}\epsilon$, where $\mid$ models interleaving (a.k.a. shuffling) between two protocol branches; suppose we are only able to observe what *alice* sends, and what *dave* receives. If *alice* sends *buy* and *dave* receives *buy*, we might think that the protocol above is respected. However, that observation might be due to *alice* sending *buy* to *dave*, which is not an allowed interaction: the protocol is not causality safe [35].
- "Non-Determinism": given an interaction taking place in some protocol state, we might want to deterministically know how to move to the next state. For example, if *alice* asks her mother to buy a book, and the protocol is

$$alice \overset{wa,buy}{\Longrightarrow} barbara : barbara \overset{wa,reserve}{\Longrightarrow} carol : \epsilon \ \vee \\ alice \overset{wa,buy}{\Longrightarrow} barbara : barbara \overset{wa,buyItYourself}{\Longrightarrow} alice : \epsilon$$

  we could move on either branch. If we opt to move on the first branch, the next expected action is that *barbara* asks *carol* to reserve a book. If, instead, *barbara* tells *alice* to buy the book by herself, we have to backtrack to the

previous protocol state in order to check that this interaction is allowed as well; this is extremely inefficient and should be avoided [7]. While the notions of good and bad protocols are universally recognized, ugliness also depends on the formalism and its expressive power.

Can We Simply Ignore Bad Protocols? Let us suppose that the protocol is used for monitoring purposes: it does not need to be implemented or enacted. The agents in the MAS are already there, and they are heterogeneous black boxes behaving according to their own policies and goals, in full autonomy. The monitor observes messages that agents exchange, in a completely non obtrusive way, checks their compliance with the protocol ruling the MAS, and reports violations to some other entity or to the human(s) in charge for the security and safety of the monitored system.

Let us also suppose that the MAS protocol is $P1$ and the human(s) who set up the monitoring process know in advance that email messages cannot be observed by the monitor, for infrastructural, legal, or other reasons. Keeping interactions taking place via email in the protocol that the monitor will check would lead to false positives, as the monitor would look for messages foreseen by the protocol that it cannot see and would hence detect a protocol violation, but removing them from $P1$ leads to $P2$: a bad protocol! If the monitor observation ability is not perfect – which is an extremely realistic situation – there is no gain in struggling against bad protocols: unobservable interactions are there and generate the same problems of covert channels. We may state that what is "bad" in this situation is the monitor, since it is not able to observe everything should be observed, and not the AIP. This is another way to analyze the situation, but the problem still remains: a monitor with imperfect observation capability cannot be driven by a "perfect" protocol which includes unobservable messages. Rather, it might need to be driven by what we classified a "bad" protocol, describing all and only the messages that the monitor can actually observe.

Since we cannot ignore bad protocols, we opted for deepening our acquaintance with them. We developed RIVERtools, a tool able to (1) generate bad protocols starting from good ones with known unobservable channels, to correctly drive monitors with limited observation ability, and (2) explore the usage of bad protocols for Distributed Runtime Verification under suitable conditions. The next two sections introduce trace expressions and the algorithm implemented by RIVERtools to address the first point above. The second issue, namely how to partially decentralize the runtime monitoring process also when the AIP is bad, has been addressed in [28].

3 Background: Modeling AIPs with Trace Expressions

The general mechanism we propose for taking unobservable events into account during MAS monitoring, discussed in Sect. 4, can be applied to any formalism. However, to make our proposal more practical, we will discuss the algorithm we implemented for a specific formalism, trace expressions [3,5,7], and we briefly

introduce it in the sequel. Trace expressions are based on the notions of *event* and *event type*. We denote by $\mathcal{E}$ the fixed universe of events subject to monitoring. An event trace $\bar{e}$ over $\mathcal{E}$ is a possibly infinite sequence of events in $\mathcal{E}$, and a trace expression over $\mathcal{E}$ denotes a set of event traces over $\mathcal{E}$. Trace expressions are built on top of event types (chosen from a set $\mathcal{ET}$), each specifying a subset of $\mathcal{E}$.

The semantics of event types is specified by the function *match*: if e is an event, and ϑ is an event type, then $match(e, \vartheta)$ holds if and only if event e matches event type ϑ; hence, the semantics $[\![\vartheta]\!]$ of an event type ϑ is the set $\{e \in \mathcal{E} \mid match(e, \vartheta) \text{ holds}\}$.

A trace expression $\tau \in \mathcal{T}$ represents a set of possibly infinite event traces, and is defined on top of the following operators:

- ϵ (empty trace), denoting the singleton set $\{\epsilon\}$ containing the empty event trace ϵ.
- $\vartheta{:}\tau$ (*prefix*), denoting the set of all traces whose first event e matches the event type ϑ, and the remaining part is a trace of τ.
- $\tau_1 \cdot \tau_2$ (*concatenation*), denoting the set of all traces obtained by concatenating the traces of τ_1 with those of τ_2.
- $\tau_1 \wedge \tau_2$ (*intersection*), denoting the intersection of the traces of τ_1 and τ_2.
- $\tau_1 \vee \tau_2$ (*union*), denoting the union of the traces of τ_1 and τ_2.
- $\tau_1 | \tau_2$ (*shuffle*), denoting the set obtained by shuffling the traces of τ_1 with the traces of τ_2.

Trace expressions support recursion through cyclic terms expressed by finite sets of recursive syntactic equations, as supported by modern Prolog systems. If $match(alice \overset{wa,buy}{\Longrightarrow} barbara, buy)$, $P4 = buy{:}(\epsilon \vee P4)$ denotes the protocol where *alice* may send one *buy* request to *barbara* and either terminate (ϵ) or start the protocol again ($\vee\ P4$). The traces denoted by $P4$ are

$\{alice \overset{wa,buy}{\Longrightarrow} barbara,\ alice \overset{wa,buy}{\Longrightarrow} barbara\ \ alice \overset{wa,buy}{\Longrightarrow} barbara,\ ...,\ (alice \overset{wa,buy}{\Longrightarrow} barbara)^n,\ ...,(alice \overset{wa,buy}{\Longrightarrow} barbara)^\omega\}$

namely, traces consisting of n instances of event $alice \overset{wa,buy}{\Longrightarrow} barbara$, with $n \geq 1$, plus the infinite trace. Infinite traces allow us to model systems that ought not to terminate, such as those involving a "service provider agent" that must be always be ready to answer the requests of its clients. We represent the infinite trace in an explicit way, to distinguish it from finite traces of any length.

Some Considerations on Events and Event Types. For presentation purposes, the protocols shown in Sect. 2 did not include event types but events. A simplified variant of $P1$ with event types is $P5$, where

$$P5 = bookReservationReq : availabilityCheckReq :$$
$$(availableNow : \epsilon\ \ \vee\ \ order : okOrder : available2Days : \epsilon)$$

and, for example,

$match(barbara \overset{em,reserve}{\Longrightarrow} carol, bookReservationReq)$
$match(barbara \overset{wa,reserve}{\Longrightarrow} carol, bookReservationReq)$
$match(carol \overset{wa,checkAvail}{\Longrightarrow} dave, availabilityCheckReq)$
$match(hillary \overset{wa,checkAvail}{\Longrightarrow} dave, availabilityCheckReq)$

The sequence $barbara \overset{em,reserve}{\Longrightarrow} carol \quad hillary \overset{wa,checkAvail}{\Longrightarrow} dave$ (namely $barbara \overset{em,reserve}{\Longrightarrow} carol$ followed by $hillary \overset{wa,checkAvail}{\Longrightarrow} dave$) is correct w.r.t. $P5$ since the first event matches *bookReservationReq*, after which an event matching *availabilityCheckReq* is expected and in fact the second event matches it. Clearly, this sequence of messages does not make sense: we would like to state that the receiver of the first message must be the sender of the second one, but we cannot express such a constraint if the event type and protocol languages do not support variables. Parametric trace expressions [8] overcome this limitation by introducing parameters in the trace expression formalism. For space constraints we do not discuss them, but parameters are the element of the formalism that allows us, for example, to support multiple concurrent conversations and correctly track which message instance belongs to which conversation. The algorithm presented in Sect. 4 works for parametric trace expressions as well.

As far as observability is concerned, an event type *bookReservationReq* that models reservation requests whatever the communication means used to send them (whatsApp or email) makes sense in most situations, apart those where the observability of messages is different depending on the communication means.

The sequence $barbara \overset{em,reserve}{\Longrightarrow} carol \quad carol \overset{wa,checkAvail}{\Longrightarrow} dave$ is correct w.r.t. $P5$, and the same holds for $barbara \overset{wa,reserve}{\Longrightarrow} carol \quad carol \overset{wa,checkAvail}{\Longrightarrow} dave$, but they might lead to different monitoring outcomes if the likelihood of $barbara \overset{em,reserve}{\Longrightarrow} carol$ to be observed by the monitor is different from that of $barbara \overset{wa,reserve}{\Longrightarrow} carol$.

In Sect. 4 we will limit our investigation to non-deterministic and contractive trace expressions (for example, definitions like $P = P \vee P$ are not contractive, as there is no means to "consume" interactions from P while rewriting it) and we will assume that event types model only sets of events whose observability likelihood is equivalent.

4 Partial Observability: When the Good Becomes Bad

In this section we discuss how a good protocol may become a (possibly) bad one, due to unobservability or partial observability of events. The human beings in charge for the system monitoring process may use the algorithm we present if they are aware of limited observability of events or channels, in order to generate a monitor which will check the system's compliance with the protocol output by the algorithm (possibly different from the original one, and possibly "bad") and avoid raising false alarms. Given a trace expression modeling an AIP, the monitor that dynamically verifies it can be generated automatically both for Jade [14] and for Jason [16], as discussed in [17] and [5] respectively.

For sake of readability, let us suppose that we are in charge for the monitoring process. We must associate with each event foreseen by the protocol, its "observability likelihood", namely the likelihood that the event can be observed by the monitor. Of course, we must know this parameter, which depends on the system to be monitored and on its context. If, when the event takes place, the monitor can always observe it, we associate 1 with the event. If the monitor can never observe the event (for example, the monitor can sniff whatsApp messages only, and the event is an email message), we associate 0 with it. If the event is transmitted over an unreliable or leaky channel, we may associate a number between 0 and 1, excluding the extremes, with it.

Let us consider $P1$ again, and let us suppose that:

(1) the observability likelihood of messages exchanged via email is 0;
(2) the observability likelihood of whatsApp messages sent by *frank* is 0.95;
(3) the observability likelihood of the other whatsApp messages is 1.

Condition 1 forces us to remove all messages exchanged via email from the protocol, and condition 3 forces us to keep all the other whatsApp messages but those sent from *frank*. The first and last conditions would lead to protocol $P2$. The second condition, however, requires a special treatment. In fact, message $frank \overset{wa,okOrder}{\Longrightarrow} dave$ could be either observed or not and both cases would be correct, even if the first one should be much more frequent than the second.

The subprotocol where $frank \overset{wa,okOrder}{\Longrightarrow} dave$ can either take place or not can be modeled by $frank \overset{wa,okOrder}{\Longrightarrow} dave : \epsilon \ \vee \ \epsilon$. The transformation from $P1$ to the protocol which takes observability likelihood into account requires the following steps:

(1) since the observability likelihood of messages exchanged via email is 0, remove them by $P1$;
(2) since the observability likelihood of the whatsApp message sent by *frank* is 0.95, substitute it with the corresponding subprotocol where the message can take place or not, and concatenate this subprotocol with the remainder;
(3) since the observability likelihood of the other whatsApp messages is 1, keep them all.

The result is

$$P3 = alice \overset{wa,buy}{\Longrightarrow} barbara : carol \overset{wa,checkAvail}{\Longrightarrow} dave : \\ (dave \overset{wa,take2shop}{\Longrightarrow} emily : \epsilon \ \vee \ (frank \overset{wa,okOrder}{\Longrightarrow} dave : \epsilon \ \vee \ \epsilon) \ \cdot \ \epsilon)$$

which can be simplified into the equivalent protocol

$$P3' = alice \overset{wa,buy}{\Longrightarrow} barbara : carol \overset{wa,checkAvail}{\Longrightarrow} dave : \\ (dave \overset{wa,take2shop}{\Longrightarrow} emily : \epsilon \ \vee \ (frank \overset{wa,okOrder}{\Longrightarrow} dave : \epsilon \ \vee \ \epsilon))$$

Since dealing with likelihoods in $(0, 1)$ results into a more complex protocol, as the original protocol must be extended with the choice between observing the event or not, we might want to collapse likelihoods greater than a given

threshold to 1, to avoid proliferation of choices: we may set a threshold above which events will be considered fully observable. Let Th be such threshold and P be the protocol to transform.

P' is obtained by P applying the following rules; L is the observability likelihood of interaction *int*

(1) if $L > Th$, *int* is kept;
(2) if $0 < L \leq Th$, *int* is transformed into the subprotocol where *int* can either take place or not, and suitably concatenated with the remainder;
(3) if $L = 0$, *int* is discarded.

Since different monitors might observe different events and observability might change over time, causing an evolution of the observable protocol, modeling the good global protocol and then transforming it based on contingencies is a better engineering approach than directly modeling the partial, observable protocol. However, even if we start from a good P protocol, P' might be bad or even ugly.

Observability-Driven Transformation of Trace Expressions. We implemented the algorithm sketched above for protocols modeled as trace expressions. The code has been developed in SWI-Prolog (http://www.swi-prolog.org/) and, for space constraints, is made available in a longer version of this paper available at http://www.disi.unige.it/person/MascardiV/Download/PAAMS-long18.pdf. The code for `filter_events` implementing the observability-driven transformation is 36 lines long and – provided a basic knowledge of logic programming – is self-explaining. Despite its simplicity, it can operate on very complex parametric and recursive (also non terminating) protocols. The magic behind the "invisible" management of non terminating protocols like $P4$ is the use of the SWI-Prolog coinduction library (http://www.swi-prolog.org/pldoc/doc/_SWI_/library/coinduction.pl) which allows to cope with infinite terms without entering into loops. RIVERtools [6,27] supports the development of AIPs modeled as trace expressions via a user-friendly GUI and implements static controls to check if a protocol is ugly, bad or good, suggesting how to decentralize the runtime monitoring process even in presence of bad protocols.

Experiments. We have experimented the filtering algorithm on a parametric trace expressions modeling the English Auction where the auctioneer proposes to sell an item for a given price and the bidders either accept or reject the proposal; as long as more than one bidder accepts, the price – which is a parameter of the protocol – is raised and another negotiation round is made. The protocol is consistent with the existing descriptions of the English Auction that can be found online, even if it slightly differs from the English Auction FIPA specification. The protocol description, as well as its code, can be downloaded from http://parametricTraceExpr.altervista.org/. We initially assumed that the only partially observable event was `buy(X)` with observability likelihood `0.5`. By

setting the threshold to `0.7`, all occurrences of `buy(X)` became optional, while with a threshold equal to `0.4` they were all kept in the protocol. By setting the observability likelihood of `buy(X)` to `0`, any occurrence of `buy(X)` was removed from the protocol.

The filtering algorithm was also run on a variant of the Alternating Bit Protocol [25] with 6 agents. Different observability likelihoods and different thresholds were set with both protocols, to test the algorithm in an exhaustive way.

5 Related Work and Conclusions

To the best of our knowledge, MAS monitoring under partial or imperfect observability has been addressed in the context of normative multiagent organizations only, and by just a few works. Among them, [2] spun off from [18] and shows how to move from the heaven of ideal norms to the earthly condition of approximate norms. The paper focuses on conditional norms with deadlines and sanctions [39]; ideal norms are those that can be perfectly monitored given a monitor, and optimality of a norm approximation means that any other approximation would fail to detect at least as many violations of the ideal norm. Given a set of ideal norms, a set of observable properties, and some relationships between observable properties and norms, the paper presents algorithms to automatically synthesize optimal approximations of the ideal norms defined in terms of the observable properties. Even if the purpose of our work is in principle similar to that of [2,18], the approaches used to model AIPs are too different – also in expressive power – to compare them. A more recent work in the normative agents area is [24] that proposes information models and algorithms for monitoring norms under partial action observability by reconstructing unobserved actions from observed actions. While we assume to know in advance which events cannot be observed and we transform the ideal protocol into a monitorable one based on this information, the authors of [24] "guess" the actions that the monitor could not observe, but that must have taken place because of their visible effects. Outside the MAS community, partial ability of monitors to observe events is a well studied problem in many contexts including command and control [43] and runtime verification. In [38] the authors address the problem of gaps in the observed program executions. To deal with the effects of sampling on runtime verification, they consider event sequences as observation sequences of a Hidden Markov Model (HMM), and use an HMM of the monitored program to fill in sampling-induced gaps in observation sequences, and extend the classic forward algorithm for HMM state estimation to compute the probability that the property is satisfied by an execution of the program. Similarly to [24], that work complements ours by estimating the likelihood of an event to occur, whereas we assume to know that likelihood, and we transform the protocol – and hence the expected sequence of observed events – based on this knowledge. Other works pursuing the objective of suitably dealing with "lossy traces" in the runtime verification area are [13,33].

As part of our future work, we will evaluate the possibility to integrate our approach with those that complement it, like [24,38]. We expect that this might

lead to some interesting result: on the observation (by the monitor) that the system is compliant with the bad AIP P', after many observations we might be able to state that "probably" the system is also compliant with the good AIP P. This research issue is very challenging: our roadmap involves deepening our understanding of the techniques that other researchers exploit to estimate the likelihood of unobserved events, and then merging those techniques into the algorithms supported by RIVERtools. We will also consider if and how our approach could be used to evaluate the protocol robustness to the presence of leaky and unreliable channels.

References

1. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The *S*CIFF abductive proof-procedure. In: Bandini, S., Manzoni, S. (eds.) AI*IA 2005. LNCS (LNAI), vol. 3673, pp. 135–147. Springer, Heidelberg (2005). https://doi.org/10.1007/11558590_14
2. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Proceedings of AAMAS 2014, pp. 117–124. IFAAMAS/ACM (2014)
3. Ancona, D., Bono, V., Bravetti, M., Campos, J., Castagna, G., et al.: Behavioral types in programming languages. Found. Trends Program. Lang. **3**(2–3), 95–230 (2016)
4. Ancona, D., Briola, D., Ferrando, A., Mascardi, V.: Global protocols as first class entities for self-adaptive agents. In: Proceedings of AAMAS, pp. 1019–1029. ACM (2015)
5. Ancona, D., Drossopoulou, S., Mascardi, V.: Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In: Baldoni, M., Dennis, L., Mascardi, V., Vasconcelos, W. (eds.) DALT 2012. LNCS (LNAI), vol. 7784, pp. 76–95. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37890-4_5
6. Ancona, D., Ferrando, A., Franceschini, L., Mascardi, V.: Managing bad AIPs with RIVERtools. In: Demazeau, Y., et al. (eds.) PAAMS 2018, LNAI, vol. 10978, pp. 296–300. Springer, Cham (2018)
7. Ancona, D., Ferrando, A., Mascardi, V.: Comparing trace expressions and linear temporal logic for runtime verification. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 47–64. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30734-3_6
8. Ancona, D., Ferrando, A., Mascardi, V.: Parametric runtime verification of multiagent systems. In: Proceedings of AAMAS 2017, pp. 1457–1459. ACM (2017)
9. Ancona, D., Ferrando, A., Mascardi, V.: Improving flexibility and dependability of remote patient monitoring with agent-oriented approaches. In: IJAOSE. (2018, to appear)
10. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Verification of protocol conformance and agent interoperability. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 265–283. Springer, Heidelberg (2006). https://doi.org/10.1007/11750734_15
11. Baldoni, M., Baroglio, C., Capuzzimati, F.: A commitment-based infrastructure for programming socio-technical systems. ACM Trans. Internet Techn. **14**(4), 1–23 (2014)

12. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Exploiting social commitments in programming agent interaction. In: Chen, Q., Torroni, P., Villata, S., Hsu, J., Omicini, A. (eds.) PRIMA 2015. LNCS (LNAI), vol. 9387, pp. 566–574. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25524-8_39
13. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring compliance policies over incomplete and disagreeing logs. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 151–167. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_17
14. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley, Hoboken (2007)
15. Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85361-9_33
16. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. Wiley, Hoboken (2007)
17. Briola, D., Mascardi, V., Ancona, D.: Distributed runtime verification of JADE multiagent systems. In: Camacho, D., Braubach, L., Venticinque, S., Badica, C. (eds.) Intelligent Distributed Computing VIII. SCI, vol. 570, pp. 81–91. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10422-5_10
18. Bulling, N., Dastani, M., Knobbout, M.: Monitoring norm violations in multi-agent systems. In: Proceedings of AAMAS 2013, pp. 491–498. IFAAMAS (2013)
19. Casella, G., Mascardi, V.: West2East: exploiting web service technologies to engineer agent-based software. IJAOSE **1**(3/4), 396–434 (2007)
20. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multiparty session. Log. Methods Comput. Sci. **8**(1) (2012)
21. Chopra, A.K., Christie, S., Singh, M.P.: Splee: a declarative information-based language for multiagent interaction protocols. In: Proceedings of AAMAS 2017, pp. 1054–1063. ACM (2017)
22. Chopra, A.K., Singh, M.P.: Cupid: commitments in relational algebra. In: Proceedings of AAAI 2015, pp. 2052–2059. AAAI Press (2015)
23. Cossentino, M.: From requirements to code with the PASSI methodology. Agent-Orient. Methodol. **3690**, 79–106 (2005)
24. Criado, N., Such, J.M.: Norm monitoring under partial action observability. IEEE Trans. Cybern. **47**(2), 270–282 (2017)
25. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28869-2_10
26. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Trans. Softw. Eng. **31**(12), 1015–1027 (2005)
27. Ferrando, A.: RIVERtools: an IDE for runtIme VERification of MASs, and beyond. CEUR Workshop Proc. **2056**, 13–26 (2017)
28. Ferrando, A., Ancona, D., Mascardi, V.: Decentralizing MAS monitoring with DecAMon. In: Proceedings of AAMAS 2017, pp. 239–248. ACM (2017)
29. Francalanza, A., Pérez, J.A., Sánchez, C.: Runtime verification for decentralised and distributed systems. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 176–210. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_6

30. García-Ojeda, J.C., DeLoach, S.A., Robby: AgentTool III: from process definition to code generation. In: Proceedings of AAMAS 2009, pp. 1393–1394. IFAAMAS (2009)
31. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proceedings of POPL 2008, pp. 273–284. ACM (2008)
32. Huget, M., Odell, J.: Representing agent interaction protocols with agent UML. In: Proceedings of AAMAS 2004, pp. 1244–1245. IEEE Computer Society (2004)
33. Joshi, Y., Tchamgoue, G.M., Fischmeister, S.: Runtime verification of LTL on lossy traces. In: Proceedings of SAC 2017, pp. 1379–1386. ACM (2017)
34. Ladkin, P.B., Leue, S.: Interpreting message flow graphs. Formal Aspects Comput. **7**(5), 473–509 (1995)
35. Lanese, I., Guidi, C., Montesi, F., Zavattaro, G.: Bridging the gap between interaction-and process-oriented choreographies. In: Proceedings of ICSEFM 2008, pp. 323–332. IEEE (2008)
36. Papazoglou, M.P.: Service -oriented computing: concepts, characteristics and directions. In: Proceedings of WISE 2003, p. 3. IEEE Computer Society (2003)
37. Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In: Proceedings of AAMAS 2011, pp. 491–498. IFAAMAS (2011)
38. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15
39. Tinnemeier, N.A.M., Dastani, M., Meyer, J.C., van der Torre, L.W.N.: Programming normative artifacts with declarative obligations and prohibitions. In: Proceedings of IAT 2009, pp. 145–152. IEEE Computer Society (2009)
40. Winikoff, M., Liu, W., Harland, J.: Enhancing commitment machines. In: Leite, J., Omicini, A., Torroni, P., Yolum, I. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 198–220. Springer, Heidelberg (2005). https://doi.org/10.1007/11493402_12
41. Winikoff, M., Yadav, N., Padgham, L.: A new hierarchical agent protocol notation. Auton. Agent. Multi-Agent Syst. **32**(1), 59–133 (2018)
42. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 235–247. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45448-9_17
43. Yukish, M., Peluso, E., Phoha, S., Sircar, S., Licari, J., Ray, A., Mayk, I.: Limits of control in designing distributed C^2 experiments under imperfect communications. In: Military Communications Conference MILCOM 1994. IEEE (1994)