Agent Interaction Protocols: The Good, the Bad and the Ugly

Davide Ancona, Angelo Ferrando, Luca Franceschini, Viviana Mascardi

Università degli Studi di Genova, Genova, Italy
davide.ancona@unige.it, angelo.ferrando@dibris.unige.it,
luca.franceschini@dibris.unige.it, viviana.mascardi@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. 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 explore the possibility that bad protocols are not that bad, after all. In particular, we concentrate on the problem of MAS monitoring under the assumption of partial observability: even if the global protocol ruling the MAS is a good one, partial or no observability of some events therein can make the actually monitorable protocol a bad one, with covert channels due to unobservable events. An observability-driven protocol transformation algorithm is presented, and its implementation and experiments with the trace expression formalism are discussed.

Keywords: Good Agent Interaction Protocols, Bad Agent Interaction Protocols, Monitoring, 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, 8] 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 community [31] and as a "Global Type" in the multiparty session types one [13, 25].

In the MAS community, AIPs describe interaction patterns characterizing the system as a whole. This global viewpoint is supported by many formalisms and notations such as AUML [26], commitment machines and their extensions [9, 10, 18, 35, 37], the Blindingly Simple Protocol Language (BSPL) and its Splee

extension [17, 32], the Hierarchical Agent Protocol Notation (HAPN) [36], trace expressions [6].

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 [19, 24, 30], or to algorithms that translate the AIP textual representation to some abstract, intermediate formalism for modeling the local viewpoint [15, 22]. Such intermediate formalisms are not perceived as the main target of the 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 [29], 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 issues:

> *Multiparty session types are a type discipline that can enforce strong communication safety for distributed processes, via a choreographic specification (called global type) of the interaction between several peers. Global types are then projected to end-point types (called local types), against which processes can be statically type-checked. Well-typed processes are guaranteed to interact correctly, following the global protocol [21].*

Whatever the research area, assumptions on the protocols are made which allow them to be classified as good, bad and ugly. Whereas the classification of ugly protocols may depend on the protocol purpose and on the formalism used to express it, almost all the authors agree on tagging as "bad" the same classes of protocols based on problems they might raise during the projection and/or enactment stages.

In this paper we first discuss what is generally perceived as a bad, good or ugly protocol and we motivate why bad protocols as not necessarily as bad (Section 2). In Section 3 we provide some background knowledge on trace expressions and in Section 4 we present an algorithm for observability-driven transformation of AIPs. This algorithm, implemented for trace expressions, might transform a good protocol into a bad or even a ugly one, hence moving "from heaven to hell". Finally, in Section 5, we discuss the related works and conclude.

[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`.

2 The Good, the Bad and the Ugly

2.1 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, which is not relevant for our work);
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 they 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". 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" [16], "connectedness for sequence" [29], "explicit causality" [32]; the second "knowledge for choice" [16], "local choice" [28], "unique point of choice" [29].

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 [16].

2.2 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 $P2$:

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

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?

Many authors observe that bad protocols cannot be used for designing Interaction-Oriented Programming systems where interactions are first-class concepts:

> *The flow of causality is reflected in the flow of information: there are no hidden flows of causality because there are no hidden flows of information. Indeed, if there were any hidden flows, then the very idea of protocols as a basis for Interaction-Oriented Programming would be called into question [32].*

2.3 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:

- "Causality unsafety": consider the two shuffled sequences $\mathit{carol} \overset{wa,buy}{\Longrightarrow} \mathit{dave}{:}\epsilon \mid \mathit{alice} \overset{wa,buy}{\Longrightarrow} \mathit{barbara}{:}\epsilon$, 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 above is not causality safe [29].

– "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 [6].
– "Non-Contractivenness": recursive definitions are a very powerful feature to express complex protocols in a compact way, but – if we set no rules – we might end up with definitions like $P = P \vee P$ which are not "contractive", as there is no means to "consume" interactions from P while rewriting it. This makes writing algorithms that halt on such protocols unfeasible [6].

While the notions of good and bad protocols are universally recognized, ugliness also depends on the formalism and its expressive power. In the sequel we will concentrate on bad protocols.

2.4 Yin and Yang

Bad protocols have very few defenders, but there is always something good in the bad, and something bad in the good. Are there situations where using bad protocols makes sense?

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. However, they act inside a society and they must respect the society rules, expressed as an interaction protocol. The monitor is in charge of observing messages that agents exchange, in a completely non obtrusive way, and check if they are compliant with the protocol ruling the MAS.

Let us suppose that the MAS protocol is $P1$. If the monitor is able to observe any kind of message, transmitted over any kind of communication means, we are in a standard situation, with a monitor in charge for verifying the compliance of actual interactions with a good protocol.

But what if the monitor cannot observe email messages? The protocol ruling the MAS is still $P1$, and it is still a good protocol, but from the monitor point of view it contains covert channels: the unobservable interactions taking place via email. Keeping them in the protocol 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.

Nevertheless, if there is a single, centralized monitor for the whole MAS, bad protocols are harmless. Let us consider $P2$: the monitor knows that after observing *alice* $\overset{wa,buy}{\Longrightarrow}$ *barbara* it must observe *carol* $\overset{wa,checkAvail}{\Longrightarrow}$ *dave* and nothing else. Either it succeeds, or it fails. Covert channels have no impact on centralized monitoring.

The situation where monitoring bad protocols becomes really bad is when we want to decentralize the monitoring activity, by having different monitors in charge for disjoint subsets of the agents in the MAS. Let us consider $P2$ again: if there is one monitor $M1$ observing *alice* and *barbara*, and another monitor $M2$ observing *carol* and *dave*, the sequentiality between *alice* $\overset{wa,buy}{\Longrightarrow}$ *barbara* and *carol* $\overset{wa,checkAvail}{\Longrightarrow}$ *dave* cannot be verified by any of them. Should we then discard bad protocols from our investigation, because of the problems they raise in a decentralized monitoring setting? No: a solution for partially decentralizing the monitoring activity also in case of bad protocols has been recently proposed [23], which makes bad protocols harmless (or at least, less dangerous) even in that context.

Given that good protocols where unobservable interactions have been removed can become bad protocols and that perfect observability cannot be always given for granted, we claim that – at least for monitoring purposes – bad protocols can be necessary to suitably model observable protocols.

3 Background: Modeling AIPs with Trace Expressions

The general mechanism we propose for taking unobservable events[4] into account during MAS monitoring, discussed in Section 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, 6], and we briefly introduce it in the sequel. Trace expressions are a protocol specification formalism expressly designed for runtime verification, inspired by initial work on monitoring agent interactions in MASs [5]. They are based on the notions of *event* and *event type*.

Events and event types

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 $\mathcal{E}$.

[4] Even if in this paper we mainly address agent interaction protocols characterized by interaction events, trace expressions can be used to model events of any kind. In the sequel we will use the term "event" instead of the more specific "interaction" or "communicative event" ones to stress the generalizability of the approach.

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 $\{e \in \mathcal{E} \mid match(e, \vartheta) \text{ holds}\}$. For generality, we leave unspecified the formalism used for defining event types; in practice, in our implementation *match* is defined by a Prolog predicate.

Syntax

A trace expression $\tau \in \mathcal{T}$ is defined on top of the following operators:[5]
• 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.

Semantics

The semantics of trace expressions is specified by a transition relation $\delta \subseteq \mathcal{T} \times \mathcal{E} \times \mathcal{T}$, where $\mathcal{T}$ and $\mathcal{E}$ denote the set of trace expressions and of events, respectively. We write $\tau_1 \xrightarrow{e} \tau_2$ to mean $(\tau_1, e, \tau_2) \in \delta$; the transition $\tau_1 \xrightarrow{e} \tau_2$ expresses the property that the system under monitoring can safely move from the state specified by τ_1 into the state specified by τ_2 when event e is observed.

Figure 1 defines the transition system, together with the auxiliary predicate $\epsilon(\_)$ checking whether a trace expression is allowed to contain the empty trace; if $\epsilon(\tau)$ holds, then it means that the monitored system can safely terminate.

[5] Binary operators associate from left, and are listed in decreasing order of precedence, that is, the first operator has the highest precedence.

$$\text{(prefix)}\frac{}{\vartheta{:}\tau \xrightarrow{e} \tau}\ match(e,\vartheta) \qquad \text{(or-l)}\frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_1'} \qquad \text{(or-r)}\frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1 \vee \tau_2 \xrightarrow{e} \tau_2'}$$

$$\text{(and)}\frac{\tau_1 \xrightarrow{e} \tau_1' \quad \tau_2 \xrightarrow{e} \tau_2'}{\tau_1 \wedge \tau_2 \xrightarrow{e} \tau_1' \wedge \tau_2'} \qquad \text{(shuffle-l)}\frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1' | \tau_2} \qquad \text{(shuffle-r)}\frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1 | \tau_2 \xrightarrow{e} \tau_1 | \tau_2'}$$

$$\text{(cat-l)}\frac{\tau_1 \xrightarrow{e} \tau_1'}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_1' \cdot \tau_2} \qquad \text{(cat-r)}\frac{\tau_2 \xrightarrow{e} \tau_2'}{\tau_1 \cdot \tau_2 \xrightarrow{e} \tau_2'}\ \epsilon(\tau_1)$$

$$\text{(ϵ-empty)}\frac{}{\epsilon(\epsilon)} \qquad \text{(ϵ-or-l)}\frac{\epsilon(\tau_1)}{\epsilon(\tau_1 \vee \tau_2)} \qquad \text{(ϵ-or-r)}\frac{\epsilon(\tau_2)}{\epsilon(\tau_1 \vee \tau_2)} \qquad \text{(ϵ-others)}\frac{\epsilon(\tau_1) \quad \epsilon(\tau_2)}{\epsilon(\tau_1\ op\ \tau_2)}\ op \in \{|, \cdot, \wedge\}$$

Fig. 1. Transition system for trace expressions

Some considerations on events and event types

For presentation purposes, the protocols shown in Section 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,

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

and so on.

The sequence $barbara \overset{em,reserve}{\Longrightarrow} carol \quad hillary \overset{wa,checkAvail}{\Longrightarrow} dave$ 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 [7] overcome this limitation by introducing parameters in the trace expression formalism. Without parameters, we need to design the protocol in a different way, to explicitly account that events involving different agents should belong to different event types. In the previous sections we did not provide the syntax and semantics of parametric trace expressions for sake of presentation clarity, as they raise some technical details that would

require space to be properly addressed. However, the algorithm presented in Section 4.1 works for trace expressions that might have parameters inside.

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, but those where the observability of messages is different depending on the communication means.

Both sequences

$$barbara \overset{em,reserve}{\Longrightarrow} carol \quad carol \overset{wa,checkAvail}{\Longrightarrow} dave$$

and

$$barbara \overset{wa,reserve}{\Longrightarrow} carol \quad carol \overset{wa,checkAvail}{\Longrightarrow} dave$$

are correct w.r.t. $P5$, but 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 Section 4 we will limit our investigation to non-deterministic and contractive[6] trace expressions and we will assume that protocols are designed in a correct way: event types model only sets of events whose observability likelihood is equivalent.

4 From Heaven to Hell

In this section we discuss how to transform a good protocol to a (possibly) bad one, due to unobservability or partial observability of events in the original protocol.

First of all, we need to associate with each event foreseen by the protocol, its "observability likelihood", namely the likelihood that the event can be observed by the monitor. 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. The higher this number, the more likely the monitor will be able to observe the event when it takes place.

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.

[6] See Section 2.3.

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. For this reason we assume that the protocol designer can 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.

4.1 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[7] and is shown below, along with comments starting with `%` that explain each clause.

The predicate that implements the algorithm is `filter_events(ProtocolToFilter, FilteredProtocol, Th, PrId)`. Since in our setting protocols are first class entities which can be analyzed, manipulated, and exchanged among agents, they are characterized by a unique name, `PrId`. `ProtocolToFilter` is the Prolog representation of the trace expression where unobservable events must be filtered out, `FilteredProtocol` is the transformation result, `Th` is the threshold above which an event is considered fully observable and hence kept in `FilteredProtocol`.

Each event type must be associated with its likelihood to be observed, thanks to the `observable(ET, Lkl, PrId)` predicate.

As an example, if we had a parametric English Auction interaction protocol where the `buy(X)` parametric event type observability is `0.5`, and the observability of all the other event types is `1`, we would write

```
observable(buy(X), 0.5, english_auct) :- !.
observable(E, 1, english_auct) :- E \= buy(_).
```

where `!` prevents the Prolog interpreter from backtracking when it is executed[8] and `\=` stands for "cannot unify with". Uppercase symbols represent logical variables, and `p :- q, r, s.` should be read as "if q, r, s hold, then p holds".

`filter_events` operates according to the trace expression syntax. We omit the rule for dealing with parameters.

```
filter_events(epsilon, epsilon, _, _) :- !.
% empty trace expression epsilon is transformed into epsilon

filter_events(ET:T, TFiltered, Th, PrId) :-
  observable(ET, 0, PrId), !,
  filter_events(T, TFiltered, Th, PrId).
% trace expression ET:T where observable(ET, 0, PrId) is
% transformed into TFiltered if T is transformed into
% TFiltered (ET is removed)

filter_events(ET:T, TFiltered, Th, PrId) :-
  observable(ET, Lkl, PrId), Lkl > Th, !,
  filter_events(T, T1, Th, PrId),
  TFiltered = ET:T1, !.
% trace expression ET:T where observable(ET, Lkl, PrId) and
% Lkl > Th is transformed into ET:T1, where T1 results from
% transforming T (ET is kept)
```

[7] `http://www.swi-prolog.org/`.

[8] The functioning of "cut" is more complex than this, but we can ignore the details.

```
filter_events(ET:T, TFiltered, Th, PrId) :-
  observable(ET, Lkl, PrId),
  filter_events(T, T1, Th, PrId), Lkl =< Th,
  TFiltered = ((ET:epsilon) \/ epsilon) * T1), !.
% in trace expression ET:T where observable(ET, Lkl, PrId) and
% Lkl <= Th, ET becomes optional.
% ET:T becomes ((ET:epsilon)\/epsilon)*T

filter_events(T1\/T2, TFiltered, Th, PrId) :-
  filter_events(T1, TFiltered1, Th, PrId),
  filter_events(T2, TFiltered2, Th, PrId),
  TFiltered = (TFiltered1 \/ TFiltered2), !.
% filtering T1\/T2 means filtering T1, filtering T2, and
% joining the results with the \/ operator.
% The same holds for the other operators, |, *, /\ (not shown)
```

The code for `filter_events` 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[9] which allows to cope with infinite terms without entering into loops.

4.2 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/`.

As anticipated, 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 algorithm was also run on a variant of the Alternating Bit Protocol [21] with 6 agents.

Different observability likelihoods and different thresholds were set with both protocols, to test the algorithm in an exhaustive way.

[9] `http://www.swi-prolog.org/pldoc/doc/_SWI_/library/coinduction.pl`.

5 Related Work and Conclusions

In this paper we have analyzed the notions of good, bad and ugly protocols inside and outside the MAS community, and we have motivated the reason why bad protocols are not that bad by considering runtime monitoring of MASs with unobservable events.

To the best of our knowledge, MAS monitoring under partial or imperfect observability has been addressed in the context of normative multi-agent organizations only, and by just a few works. Among them, [2] spun off from [14] 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 [34]; 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, 14], 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 [20] that proposes information models and algorithms for monitoring norms under partial action observability by reconstructing unobserved actions from observed actions. The reconstruction process entails: (i) searching for the actions that have been performed by unobserved agents; and (ii) using the actions found to increase the knowledge about the state of the world. That paper proposes an approach that complements ours. While we assume to know in advance which events cannot be monitored, and we transform the ideal protocol into a monitorable one based on this information, the authors of [20] "guess" the actions that the monitor could not observe, but that must have taken place because of their visible effects.

Whereas we are not aware of proposals to monitor agent interactions using commitment machines, BSPL, Splee, or HAPN under partial observability assumptions, we could mention dozens of works tackling this problem outside the MAS community.

Indeed, partial ability of monitors to observe events is a well studied problem in many contexts including command and control [38] and runtime verification. In [33] 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 model 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 [20], 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 [11, 27].

As part of our future work, we will further explore the relationships between our proposal and related works in the RV field: we just started our state-of-the-art analysis and we feel that many interesting works are still to be discovered. Also, we plan to create a proof of concept of our work by implementing a JADE [12] MAS where some events cannot be observed, and showing the full process of moving from a good protocol to a bad, but monitorable, one, and actually monitoring it. In fact, MAS monitoring of parametric trace expressions is already integrated on top of JADE [7] and the event filtering based on their observability can be easily integrated as well. We need to wrap these pieces up and associate a clear engineering methodology with them, to make our approach available to the research community.

References

1. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The *S*ciff abductive proof-procedure. In: AI*IA 2005: Advances in Artificial Intelligence, Proceedings. LNCS, vol. 3673, pp. 135–147. Springer (2005)
2. Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2014. pp. 117–124. IFAAMAS/ACM (2014)
3. Ancona, D., Bono, V., Bravetti, M., Campos, J., Castagna, G., Deniélou, P.M., Gay, S.J., Gesbert, N., Giachino, E., Hu, R., Johnsen, E.B., Martins, F., Mascardi, V., Montesi, F., Neykova, R., Ng, N., Padovani, L., Vasconcelos, V., Yoshida, N.: Behavioral types in programming languages. Foundations and Trends in Programming Languages 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: Weiss, G., Yolum, P., Bordini, R.H., Elkind, E. (eds.) International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015. Proceedings. 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: Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Revised Selected Papers. LNCS, vol. 7784, pp. 76–95. Springer (2012)
6. Ancona, D., Ferrando, A., Mascardi, V.: Theory and Practice of Formal Methods: Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday, chap. Comparing Trace Expressions and Linear Temporal Logic for Runtime Verification, pp. 47–64. Springer (2016), http://dx.doi.org/10.1007/978-3-319-30734-3_6
7. Ancona, D., Ferrando, A., Mascardi, V.: Parametric runtime verification of multi-agent systems. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017. pp. 1457–1459. ACM (2017)
8. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Verification of protocol conformance and agent interoperability. In: Computational Logic in Multi-Agent Systems VI, 2005, Revised Selected and Invited Papers. LNCS, vol. 3900, pp. 265–283. Springer (2005)
9. Baldoni, M., Baroglio, C., Capuzzimati, F.: A commitment-based infrastructure for programming socio-technical systems. ACM Trans. Internet Techn. 14(4), 23:1–23:23 (2014), http://doi.acm.org/10.1145/2677206

10. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Exploiting social commitments in programming agent interaction. In: PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Proceedings. LNCS, vol. 9387, pp. 566–574. Springer (2015)
11. Basin, D.A., Klaedtke, F., Marinovic, S., Zalinescu, E.: Monitoring compliance policies over incomplete and disagreeing logs. In: Runtime Verification, Third International Conference, RV 2012, Revised Selected Papers. LNCS, vol. 7687, pp. 151–167. Springer (2012)
12. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
13. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: CONCUR 2008 - Concurrency Theory, 19th International Conference. Proceedings. LNCS, vol. 5201, pp. 418–433. Springer (2008)
14. Bulling, N., Dastani, M., Knobbout, M.: Monitoring norm violations in multi-agent systems. In: International conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2013. pp. 491–498. IFAAMAS (2013)
15. Casella, G., Mascardi, V.: West2East: exploiting web service technologies to engineer agent-based software. IJAOSE 1(3/4), 396–434 (2007), https://doi.org/10.1504/IJAOSE.2007.016267
16. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. Logical Methods in Computer Science 8(1) (2012), https://doi.org/10.2168/LMCS-8(1:24)2012
17. Chopra, A.K., Christie, S., Singh, M.P.: Splee: A declarative information-based language for multiagent interaction protocols. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017. pp. 1054–1063. ACM (2017)
18. Chopra, A.K., Singh, M.P.: Cupid: Commitments in relational algebra. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 2052–2059. AAAI Press (2015)
19. Cossentino, M.: From requirements to code with the PASSI methodology. Agent-oriented methodologies 3690, 79–106 (2005)
20. Criado, N., Such, J.M.: Norm monitoring under partial action observability. IEEE Trans. Cybernetics 47(2), 270–282 (2017), https://doi.org/10.1109/TCYB.2015.2513430
21. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012. Proceedings. LNCS, vol. 7211, pp. 194–213. Springer (2012)
22. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Trans. Software Eng. 31(12), 1015–1027 (2005), https://doi.org/10.1109/TSE.2005.140
23. Ferrando, A., Ancona, D., Mascardi, V.: Decentralizing MAS monitoring with DecAMon. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017. pp. 239–248. ACM (2017)
24. García-Ojeda, J.C., DeLoach, S.A., Robby: AgentTool III: from process definition to code generation. In: 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Volume 2. pp. 1393–1394. IFAAMAS (2009), http://doi.acm.org/10.1145/1558109.1558311
25. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL. pp. 273–284. ACM (2008)

26. Huget, M., Odell, J.: Representing agent interaction protocols with agent UML. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004). pp. 1244–1245. IEEE Computer Society (2004)
27. Joshi, Y., Tchamgoue, G.M., Fischmeister, S.: Runtime verification of LTL on lossy traces. In: Proceedings of the Symposium on Applied Computing, SAC 2017. pp. 1379–1386. ACM (2017)
28. Ladkin, P.B., Leue, S.: Interpreting message flow graphs. Formal Aspects of Computing 7(5), 473–509 (1995)
29. Lanese, I., Guidi, C., Montesi, F., Zavattaro, G.: Bridging the gap between interaction-and process-oriented choreographies. In: 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods. pp. 323–332. IEEE (2008)
30. Padgham, L., Winikoff, M.: Prometheus: A methodology for developing intelligent agents. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1. pp. 37–38. AAMAS '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/544741.544749
31. Papazoglou, M.P.: Service -oriented computing: Concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering. pp. 3–. WISE '03, IEEE Computer Society (2003), http://dl.acm.org/citation.cfm?id=960322.960404
32. Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In: 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Volume 1-3. pp. 491–498. IFAAMAS (2011)
33. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Runtime Verification - Second International Conference, RV 2011, Revised Selected Papers. LNCS, vol. 7186, pp. 193–207. Springer (2011)
34. 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 the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009. pp. 145–152. IEEE Computer Society (2009)
35. Winikoff, M., Liu, W., Harland, J.: Enhancing commitment machines. In: Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, Revised Selected Papers. LNCS, vol. 3476, pp. 198–220. Springer (2004)
36. Yadav, N., Padgham, L., Winikoff, M.: A tool for defining agent protocols in HAPN: (demonstration). In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015. pp. 1935–1936. ACM (2015)
37. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.C., Tambe, M. (eds.) Intelligent Agents VIII, 8th International Workshop, ATAL 2001, Revised Papers. vol. 2333, pp. 235–247. Springer (2002)
38. 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, 1994. MILCOM '94. IEEE (1994)