Agents Interoperability via Conformance Modulo Mapping

Davide Ancona, Angelo Ferrando, and Viviana Mascardi
DIBRIS, University of Genova, Italy, name.surname@dibris.unige.it

Abstract—We present an algorithm for establishing a flexible conformance relation between two local agent interaction protocols (LAIPs) based on mappings involving agents and messages, respectively. Conformance is in fact computed “modulo mapping”: two LAIPs \( \tau \) and \( \tau' \) may involve different agents and use different syntax for messages, but may still be found to be conformant provided that a given map from entities appearing in \( \tau \) to corresponding entities in \( \tau' \) is applied. LAIPs are modelled as trace expressions whose high expressive power allows for the design of protocols that could not be specified using finite state automata or equivalent formalisms. This expressive power makes the problem of stating if \( \tau \) conforms to \( \tau' \) undecidable. We cope with this problem by over-approximating trace expressions that may lead to infinite computations, obtaining a sound but not complete implementation of the proposed conformance check.

Index Terms—Agent Interaction Protocols, Conformance, Mappings, Trace Expressions

I. INTRODUCTION

We open the paper by means of an example. The example allows us to explain the research question we address and to introduce the trace expressions formalism for representing agent interaction protocols in a gentle way, before their formal presentation in the body of the paper.

The example scenario is the following: the company AIA4Tour develops chatbots interacting with human beings in their daily working activities. AIA4Tour business is in the touristic sector and chatbots support touristic operators.

A typical conversation between a touristic agency TourAgency and the chatbot TravelChat starts with the request of whether a plane landed\(^1\) or a cruise ship docked, or a train/bus reached the main city station; the chatbot, by accessing some database or web service in the backend, answers either “yes”, “not yet”, or “canc” (for canceled), and then becomes available to answer new questions.

The global agent interaction protocol (GAIP) \( \tau \) which norms the simple multiagent system mas involving TA (for TourAgency) and TC (for TravelChat) might look like

\[
\tau = (TA \text{ landed} \rightarrow TC; :c \lor TA \text{ docked} \rightarrow TC; :c \lor TA \text{ train arrived} \rightarrow TC; :c \lor TA \text{ bus arrived} \rightarrow TC; :c) .
\]

For sake of clarity, we disregard the facts that a flight is characterized by a code which should be supplied as a parameter to the query, and that when the chatbot answers and becomes ready to manage a new query, it might be able to interact with a travel agency different from TourAgency. The trace expressions formalism supports parameters both at the data level (to model messages which only differ for the flight code) and at the agent level (to model multiple concurrent conversations among different agents), but taking parameters into account would make the presentation more complex and we opted for keeping it as simple as possible.
The intuition behind the “mappings” the AI4Tour software engineers are looking for should be clear. We formally define them as a map $M_M : M_1 \rightarrow M_2$ from the messages that appear in an interaction protocol $\tau_{ag2}$ to those that appear in $\tau_{ag1}$, and a map $M_A : A_1 \rightarrow A_2$ from the agents that appear in $\tau_{ag1}$ to those that appear in $\tau_{ag2}$, respectively. To answer their “substitutability” question, the engineers must:

1. Move from the global description $\tau$ of how $TA$ and $TC$ interact, to $TC$’s local agent interaction protocol $\tau_{TC}$ (LAIP):

$$\tau_{TC} = (\begin{array}{ll}
       \text{landed} & TA : \epsilon \\
       \text{docked} & TA : \epsilon \\
       \text{train\ arrived} & TA : \epsilon \\
       \text{bus\ arrived} & TA : \epsilon \\
     \end{array}) \cdot \begin{array}{ll}
       \text{yes} & TA : \epsilon \\
       \text{not\ yet} & TA : \epsilon \\
       \text{canc} & TA : \epsilon \\
     \end{array} \cdot \tau_{TC}$$

In $\tau_{TC}$ we omit to write $TC$ as sender or receiver, as this information is implicit. Also, if there were messages in $\tau$ that involved TravelChat neither as the sender not as the receiver, they would not appear in $\tau_{TC}$.

2. Move from the global description $\tau'$ of how citizens and $MC$ interact, to $MC$’s LAIP, $\tau_{MC}$:

$$\tau_{MC} = (\begin{array}{ll}
       \text{docked} & C : \epsilon \\
       \text{train\ in\ station} & C : \epsilon \\
       \text{bus\ in\ station} & C : \epsilon \\
     \end{array}) \cdot \begin{array}{ll}
       \text{yes} & C : \epsilon \\
       \text{in\ 1\ h} & C : \epsilon \\
       \text{in\ 2\ h} & C : \epsilon \\
       \text{not\ in\ 2\ h} & C : \epsilon \\
     \end{array} \cdot \tau_{MC}$$

3. Check whether $\tau_{TC}$ is conformant to $\tau_{MC}$: this is achieved by looking for mappings $M_A$ among agents and mappings $M_M$ among messages involved in $\tau_{TC}$ and $\tau_{MC}$, such that TravelChat can play the role of MovingChat in $\tau'$, still ensuring that the GAIP $\tau'$ is respected.

4. Select one couple of mappings among those computed in step (3), $(M_M, M_A)$, based on their semantics pragmatics.

5. Implement a means to allow TravelChat and the citizens to interact, by forcing TravelChat to apply the selected mappings when interacting with them.

Agent TourAgency in mas must be necessarily mapped to C in mas'. From a semantic and pragmatic point of view, the most reasonable message mapping is the one that maps docked $\in$ mas into docked $\in$ mas' (we abuse notation, and we write msg $\in$ mas to mean that msg is one of the messages exchanged by agents belonging to mas); train_arrived into train_in_station; bus_arrived into bus_in_station; yes $\in$ mas into yes $\in$ mas'; not_yet into in_2_h; and canc into not_3_h. The landed message is mapped into no message: when “pretending to be MovingChat”, TravelChat will never receive a message whose meaning is close to landed, as $\tau'$ does not support it. On the other hand, TravelChat is not able to discriminate between trains and buses arriving in one or two hours. The mapping of not_yet into in_2_h is a cautious choice and the citizen will never receive the message in_1_h, even if it would be supported by $\tau'$.

From a purely syntactic point of view, and considering protocol specifications only – hence, disregarding the actual services and actions that are triggered by reception of messages –, many other mappings would respect the protocol conformance, including the one that maps canc into yes $\in$ mas' and yes $\in$ mas into not_3_h.

The research question that we address in this paper is the one in step (3) above. We point out that such research question cannot be answered by using ontology matching algorithms [1]. Ontology matching techniques could indeed be exploited in step (4) of the process, as we discuss in the Conclusions, but not in step (3): an ontology represents static knowledge, not dynamic behaviour. An agent interaction protocol represents dynamic behaviour, not static knowledge. Checking whether a protocol is conformant to another must necessarily take such dynamics into account, which is not required in an ontology matching process and which raises many subtle issues. For example, when moving from $\tau$ to $\tau'$ to substitute $ag'$, $ag$ must be capable to react at least to all the “passive events” (for example, receiving a message) that $ag'$ can address, and to perform at most all the “active events” (for example, sending a message) that $ag'$ can perform, at any stage of the protocol. This requirement cannot be satisfied by an ontology matching approach, where it does not even make sense, whereas it is well known in the protocol conformance literature. Depending on the expressiveness of the language used to specify GAIPs, verifying that $ag$ can actually substitute $ag'$ in a safe way may be more or less complex, or even impossible to perform in an exact way. As an example, recursive protocol definitions are usually disregarded in the literature as they are extremely complex to manage. The formalism we use for modeling GAIPs and LAIPs supports recursion, and this is enough to make existing conformance checking algorithms not powerful enough for our needs.

Our contribution is an algorithm for addressing step (3) above when GAIPs are specified as trace expressions [2], [3], [4], [5], [6], [7], [8], [9]. To demonstrate the feasibility of our approach, we present an example implemented in JADE [10].

II. RELATED WORK

The works closer to our proposal come from Baldoni and Baroglio who, together with their colleagues, introduced the notion of syntactic conformance in the context of interaction protocols for MAS and Service Oriented Computing (SOC) scenarios, starting from 2004. Conformance is based on the notion of interoperability among the entities’ policies (e.g. a BPEL process [11], similar to some extent to our LAIPs) with respect to interaction protocols (e.g. a WS-CDL choreography [12], similar to our GAIPs), through the use of finite state automata. While in [13], [14], [15] protocols were limited to involve two entities only, [16] presents an extension supporting multiple parties. A further extension is presented in [17] where decision points are explicitly represented.

Besides the fact that we address the conformance between LAIPs, there are other differences between those works and ours: first, they assume that entities/messages involved in the
policy and in the protocol respectively, are exactly the same in order to conformance check to have some chance to succeed: no notion of mapping is foreseen; second, the expressive power of trace expressions is higher than the expressive power of WS-CDL/BPEL. The presence of expansive subtraces, introduced later on, makes trace expressions able to recognize context-free and non-context-free languages, and raises technical problems that do not show up when less expressive formalisms are used.

Among the works by Baldoni and Baroglio’s team, however, the most inspiring for our research is [18], recently improved and extended in [19]. That work presents an agent typing system, where types are defined as commitments [20]. The typing includes a notion of compatibility, based on subtyping, which allows for the safe substitution of agents to roles along an interaction that is ruled by a commitment-based protocol. The proposal is implemented in the 2COMM framework [21], which is based on the Agent & Artifact meta-model [22], and exploits JADE and CArtAgO [23]. Considering the LAIP associated with an agent as its “communicative type” is an almost natural idea in our approach also. The LAIP makes the communicative interface of an agent explicit and can be used both to type check an agent w.r.t. the possibility of entering a MAS normed by some GAIP, and to define a subtyping relation which we name “is conformant to” relation. The main difference between our approach and the one discussed in [18] lies in the adopted formalism and the generality: commitments without mappings there, trace expressions with mappings here.

Many other works besides those mentioned above aim at defining and testing conformance in the SOC community, including [24], [25], [26], [27]. None of them uses formalisms which are as powerful as context-free grammars, or more, and none integrates the notion of agents and messages mappings. Also, some of them are limited to two-party protocols.

When moving to the MAS realm, we can devise the same differences between our approach and the others as those identified for SOC approaches: lower expressive power of the adopted formalisms and less generality, due to the absence of mappings in the conformance definition. Among the most notable contributions to protocol conformance, we may mention [28] where Endriss et al. identify three levels of conformance, weak, exhaustive, and robust, and explore how a specific class of logic-based agents can exploit an AIP formalism based on simple if-then rules to check conformance a priori or enforce it at runtime. In a similar way, Alberti et al. exploit the SCIFF abductive proof-procedure [29] for both a priori and runtime verification of compliance of agent interactions [30]. In [31], Chopra and Singh formalize the notions of conformance, coverage and interoperability. In [32] a formal interoperability test for agents is presented. That work considers the presence of two agents only, but in an open scenario where agents can behave differently from the protocol specification. Finally, in [33], Giordano and Martelli address the problem of conformance between an agent and a protocol through an automata-based technique, when the specification of the protocol is given in a temporal action logic.

III. BACKGROUND AND BASIC DEFINITIONS

a) Trace expressions: Trace expressions are based on events and event types and can be combined with various operators. For sake of presentation, in this paper we do not distinguish between events and event types, and we trace the last ones back to the notion of interactions.

An interaction is represented by $S \xrightarrow{msg} R$ where $S$ is the sender, $msg$ is the message, and $R$ is the receiver. We define $MSG$ as the function which, given an interaction, returns its message: $MSG(S \xrightarrow{msg} R) = msg$.

A trace expression $\tau$ represents a set of possibly infinite interaction traces and is defined on top of the following operators:

- $\epsilon$ (empty trace), denoting the singleton set $\{\epsilon\}$ containing the empty interaction trace $\epsilon$.
- $int_1$ (prefix), denoting the set of all traces whose first interaction is $int$ and the remainder is a trace of $\tau$.
- $\tau_1 \cdot \tau_2$ (concatenation), denoting the set of all traces obtained by concatenating the traces of $\tau_1$ with those of $\tau_2$.
- $\tau_1 \cap \tau_2$ (intersection), denoting the intersection of the traces of $\tau_1$ and $\tau_2$.
- $\tau_1 \cup \tau_2$ (union), denoting the union of the traces of $\tau_1$ and $\tau_2$.
- $\tau_1 \bowtie \tau_2$ (shuffle), denoting the set obtained by shuffling the traces in $\tau_1$ with the traces in $\tau_2$.

To support recursion without introducing an explicit construct, trace expressions are regular terms and can be represented by a finite set of syntactic equations.

As an example, $T = int : T$ is equivalent to the infinite but regular term $int ; int ; int ; \ldots$ The only trace represented by $T$ is $int^\omega$: trace expressions are interpreted in a coinductive way to represent infinite traces of interactions [34].

The semantics of trace expressions is specified by a transition relation $\delta \subseteq T \times I \times T$, where $T$ and $I$ denote the set of trace expressions and of interactions, respectively. Notation $\tau_1 \xrightarrow{int_1} \tau_2$ means $(\tau_1, int_1, \tau_2) \in \delta$; the transition $\tau_1 \xrightarrow{int_1} \tau_2$ expresses the property that the system can safely move from the state specified by $\tau_1$ into the state specified by $\tau_2$ when interaction $int$ takes place. Trace expressions model GAIPs.

b) Expansive trace expressions: The expressive power of trace expressions is due to the presence of expansive terms.

Def. 3.1: A trace expression $\tau$ is expansive iff $\tau = \tau_1 \cdot \tau_2$ and $\tau_1$ is a cyclic term containing $\tau$; or $\tau = \tau_1 \cdot \tau_2$ and either $\tau_1$ or $\tau_2$ is a cyclic term containing $\tau$; or $\tau = \tau_1 \cdot \tau_2$ and either $\tau_1$ or $\tau_2$ is a cyclic term containing $\tau$; or it contains a subtrace that is expansive.

Expansive subtraces allow the trace expression formalism to recognize more than context-free languages. Given a trace expression $\tau$, $exp(\tau)$ is true if $\tau$ is expansive.

c) Trace expression over-approximation: Given a trace expression $\tau$, $\overline{\tau}$ is an over-approximation of $\tau$ iff $\tau$ is not expansive and $\overline{\tau} = \tau$; or $\tau$ is expansive and $\overline{\tau}$ is a trace expression equivalent to a regular expression representing a superset of the traces recognized by $\tau$.

Since $\overline{\tau}$ is equivalent to a regular expression, it is not expansive. Given an expansive trace expression $\tau$, there may be
many ℰ that over-approximate it. The algorithm that computes one of these non-expansive over-approximations is discussed in [35].

d) Projection: Let A be a set of agents. Projection ℰ is a function \( \Pi : \tau \times \mathcal{P}(A) \to \tau \). Given a trace expression \( \tau \) and a set of agents \( \text{ags} \subseteq A \) as input, \( \Pi \) returns a trace expression \( \tau_{\text{ags}} \) which contains only interactions involving agents in \( \text{ags} \); interactions that do not involve agents in \( \text{ags} \) are removed from \( \tau_{\text{ags}} \). Since in this paper we are interested in projecting onto a single agent at a time, we will write \( \tau_{ag} \) instead of \( \tau_{\{ag\}} \) to denote the projection of \( \tau \) onto agent \( ag \).

When projected onto \( S \), interaction \( S \xrightarrow{msg} R \) is represented by \( \xrightarrow{msg} \) R (“sending interaction”); when projected onto \( R \), it is represented by \( \xrightarrow{msg} \) S (“receiving interaction”). In projected interactions, we omit to write the agent onto which the projection is performed. We extend the \( MSG \) function introduced for interactions, to sending and receiving interactions: \( MSG(\xrightarrow{msg} R) = MSG(\xrightarrow{msg} S) = msg \). Projected trace expressions model LAIPs.

e) GAIPs, agents, interactions, and messages: Let \( mas \) be a multiagent system governed by some GAIP modeled by trace expression \( \tau \). We define \( GAIP(mas) \) as \( \tau \). Let \( \tau \) be a trace expression involving all and only agents \( A \) and interactions \( I \). We define \( AG(\tau) \) as \( A, INT(\tau) \) as \( I \), and \( MSG(\tau) \) as \( \{msg \mid int \in INT(\tau) \land msg = MSG(int)\} \).

The definitions of \( AG, INT \) and \( MSG \) hold for both trace expressions and projected trace expressions.

IV. LAIP Conformance Modulo Mapping

We first give a simpler, but stronger, definition of compliance which does not allow renaming of messages and agents.

Def. 4.1: Given two LAIPs \( \tau_{ag1} \) and \( \tau'_{ag2} \), we say that \( \tau_{ag1} \) is conformant to \( \tau'_{ag2} \), written \( \tau_{ag1} \leq_{ag2} \tau'_{ag2} \), iff the following conditions are coinductively verified:

- \( \forall msg, ag \) if \( \exists \tau''_{ag1} \) s.t. \( \tau_{ag1} \xrightarrow{msg} \tau''_{ag1} \), then \( \exists \tau''_{ag2} \) s.t. \( \tau''_{ag2} \xrightarrow{msg} \tau''_{ag2} \land \tau''_{ag2} \leq_{ag2} \tau''_{ag1} \);
- \( \forall msg, ag \) if \( \exists \tau''_{ag2} \) s.t. \( \tau''_{ag2} \xrightarrow{msg} \tau''_{ag2} \), then \( \exists \tau''_{ag1} \) s.t. \( \tau_{ag1} \xrightarrow{msg} \tau''_{ag1} \land \tau''_{ag1} \leq_{ag2} \tau''_{ag2} \);
- \( \{\tau''_{ag1} \mid \exists \tau_{ag1}, ag \xrightarrow{msg} \tau_{ag1} \} \neq \emptyset \) implies \( \{\tau''_{ag2} \mid \exists \tau_{ag2}, ag \xrightarrow{msg} \tau_{ag2} \} \neq \emptyset \).

In the following formalization we assume that \( ag1 \) is an agent in \( mas \), and \( ag2 \) an agent in \( mas' \), and define \( \tau = GAIP(mas) \), \( \tau' = GAIP(mas') \), \( \tau_{ag1} = \Pi(\tau_{ag1}) \), \( \tau_{ag2} = \Pi(\tau'_{ag2}) \), \( A_1 = AG(\tau_{ag1}) \), \( A_2 = AG(\tau'_{ag2}) \), \( M_1 = MSG(\tau_{ag1}) \), \( M_2 = MSG(\tau_{ag2}) \).

As introduced in Section I we consider a map \( {M_A : M_A : M_1 \rightarrow M_2} \) from the messages that appear in \( \tau_{ag1} \) to those that appear in \( \tau'_{ag2} \); and a map \( {M_A : A_1 \rightarrow A_2} \) from the agents that appear in \( \tau_{ag1} \) to those that appear in \( \tau'_{ag2} \).

A more general conformance relation modulo mappings can be defined in terms of the basic conformance relation of Definition 4.1.

Def. 4.2: Given two LAIPs \( \tau_{ag1} \) and \( \tau'_{ag2} \), and two mappings \( M_M \) and \( M_A \) on messages and agents, respectively, we say that \( \tau_{ag1} \) is conformant to \( \tau'_{ag2} \) modulo \( M_M \) and \( M_A \), written \( \tau_{ag1} \leq_{(M_M, M_A)} \tau'_{ag2} \) iff \( (M_M, M_A)(\tau_{ag1}) \leq_{ag2} \tau'_{ag2} \).

With \( (M_M, M_A)(\tau_{ag1}) \) we denote the trace expression obtained from \( \tau_{ag} \) by replacing all the interactions \( \xrightarrow{msg}_{ag} \) and \( \xrightarrow{msg}_{ag} \) with \( M_M(msg) \) and \( M_A(msg) \) respectively.

Intuitively, the relation \( \tau_{ag1} \leq_{(M_M, M_A)} \tau'_{ag2} \) ensures that \( ag1 \) can safely substitute \( ag2 \) in \( mas' \), provided that mappings \( M_M \) and \( M_A \) are applied to messages and agents in \( \tau_{ag1} \), respectively.

An algorithm for conformance. Given the definitions 4.1 and 4.2 a first question that may arise is whether there exists an algorithm for deciding if the compliance relation holds for a pair of trace expressions, and, in case of the more general notion of conformance modulo mappings, if such mappings can be computed. Unfortunately, the problem is undecidable even for the simpler conformance relation of Definition 4.1: this can be derived by the fact that a context-free grammar can be encoded into a trace expression, and that the problem of inclusion between context-free languages (which is known to be undecidable) can be reduced to the conformance problem between two trace expressions. Despite this negative result, it is still interesting to investigate the existence of algorithms which are sound (even though not complete) w.r.t. the definition of conformance between trace expressions.

We define the merging of two maps in the following way: let \( M_A : M_1 \rightarrow M_2 \) and \( M'_A : M'_1 \rightarrow M'_2 \) be two maps among messages:

- if \( \exists msg \in M_1 \cap M'_1 \). \( M_M(msg) \neq M'_M(msg) \)
  then \( \text{merge}(M_M, M'_M) = \emptyset \)
- else \( \text{merge}(M_M, M'_M) = M_M \cup M'_M \)
  such that \( M'_M = M_M \cup M'_M \).

In other words, merging two maps consists in computing the union of the elements in the maps, unless there is some conflict, namely, some element is mapped to two different elements in the two maps. In this case, the maps cannot be merged (the merged map is empty). For instance, if \( M_M = \{msg_1 \mapsto msg_2, msg_3 \mapsto msg_4\} \) and \( M'_M = \{msg_3 \mapsto msg_4, msg_5 \mapsto msg_6\} \), the merged map is \( M''_M = \{msg_1 \mapsto msg_2, msg_3 \mapsto msg_4, msg_5 \mapsto msg_6\} \).

If \( M_M = \{msg_1 \mapsto msg_2\} \) and \( M'_M = \{msg_1 \mapsto msg_3\} \), their merged map is empty.

The same definition can be adopted for merging maps of agents.

Given two maps \( M_M \) and \( M_A \), a sending interaction \( msg \xrightarrow{R} \) can substitute a sending interaction \( msg \xrightarrow{R'} \) in the context of \( M_M \) and \( M_A \) iff \( \text{merge}(\{msg \mapsto msg'\}, M_M) \neq \emptyset \) and \( \text{merge}(\{R \mapsto R'\}, M_A) \neq \emptyset \). The definition for a receiving interaction \( msg \xrightarrow{S} \) substituting a receiving interaction \( msg \xrightarrow{S'} \) is similar.

The computation of \( \tau_{ag1} \leq_{(M_M, M_A)} \tau'_{ag2} \) is carried out by
the “isConformant” algorithm. The algorithm starts from two initial agent and message maps, and incrementally adds to them those mappings which are necessary to ensure agents interoperability. Consequently, it is possible to obtain partial maps where some messages and agents have not been mapped to anything at the end of the computation. Partial maps must be completed (namely, they must become total maps and be defined on all the elements in their domain), in order to be used in practice. Completion can be achieved by adding dummy elements in the range, and associate the elements in the domain that had no corresponding element in the range, with such dummy elements. The completion step is necessary to ensure that, when actually used to substitute ag1 ∈ mas to ag2 ∈ mas’, the maps returned by the algorithm can be applied to all the agents and messages appearing in τag1. The isConformant algorithm operates by cases, and the following implication holds:

\[ \tau_{ag1} \subseteq (c_M, c_M) \overset{\tau_{ag2}}{\rightarrow} (M_M, M_A) = \text{isConformant}(\tau_{ag1}, \tau_{ag2}, \emptyset, \{ag1 \mapsto ag2\}) \]

and \( (M_M, M_A) \neq (\emptyset, \emptyset) \) and \( \text{complete}(M_A) = c_M \) and \( \text{complete}(M_A) = c_M \), where complete is the map completion step sketched above.

Conformance can be lifted to global protocols:

\[ \tau \subseteq \tau' \iff \forall ag_i \in mas, \exists ag'_i \in mas', \tau_{ag_i} \subseteq \tau'_{ag'_i} \]

Conjecture: Soundness. The conformance algorithm is sound w.r.t. Definition 4.2

Claim: No Completeness. The algorithmic implementation of the conformance test is not complete.

For space constraints, the pseudo-code of isConformant along with the sketch of the claim proof is available at [https://www.disi.unige.it/person/MascardiV/Download/supplementalConformanceModuloMapping.pdf](https://www.disi.unige.it/person/MascardiV/Download/supplementalConformanceModuloMapping.pdf)

V. EXAMPLES AND IMPLEMENTATION

Given two MASs mas and mas' ruled by \( \tau = GAIP(mas) \) and \( \tau' = GAIP(mas') \) respectively, the steps for using an agent involved in mas inside mas' are the following: (i) identify the agent ag1 ∈ mas to be used in mas'; (ii) generate the set of agents and maps that ensure ag1 conformance \( \{ag2, (M_M, M_A)\} | ag2 \in mas', \tau_{ag1} \subseteq (M_M, M_A) \tau_{ag2} \} \); (iii) select one couple of agents and maps \( \{ag2, (M_M, M_A)\} \) from the set, based on domain-dependent criteria that might involve message similarity, similar behaviours of the mapped agents, and so on; (iv) generate an interface i for ag1 driven by \( (ag2, (M_M, M_A)) \); (v) substitute ag2 in mas' with the “interfaced version” of ag1: the agents in the MAS obtained via this substitution still respect \( \tau' \), because of step (ii).

The notion of interface introduced in step (iv) is related to the actual use of the maps generated during the conformance check and it is meant as a logical component offering a “bridging service”, that can be implemented in many different ways. Hence, the interface i generated in step (iv) realizes the map-driven translations needed by agents in mas and mas' to interoperate. Each time ag1 ∈ mas performs the action of sending a message msg1 to an agent ag1r ∈ mas, the interface i “intercepts” (from a logical point of view) msg1, translates it into \( M_M(msg1) = msg2 \), and forwards msg2 to \( M_A(ag1r) \in mas' \). In the same way, each time an agent ag2r ∈ mas' sends a message msg2 to ag2 ∈ mas', the interface intercepts msg2, looks for a message msg4 that ag1 can receive in the current protocol state s.t. \( M_M(msg4) = msg3 \), translates msg3 into msg4, and forwards it to ag1.

As an example, let us consider a GAIP \( \tau \) representing the protocol where an agent Buyer (Buy) asks to an agent Seller (Sel) for a resource and if the resource is available, the Seller can give it in exchange of money, otherwise the Seller informs the Buyer of the unavailability.

\[ \tau = (Buy \overset{res?}{\rightarrow} Sel): \]

\[ (Sel \overset{res}{\rightarrow} Buy:Buy \overset{money}{\rightarrow} Sel: \overset{Sel: no}{\rightarrow} Buy:\tau) \]

Let us consider another similar GAIP \( \tau' \) defining a book-shop protocol where an agent Client (Cl) asks for a book to an agent BookShop, and again if the book is available the BookShop (Shop) agent sells it for a given amount of euros, otherwise a no avail message is returned.

\[ \tau' = (Cl \overset{book?}{\rightarrow} Shop): \]

\[ ((Shop \overset{book}{\rightarrow} Cl: Cl: \overset{euros}{\rightarrow} Shop: \overset{euros}{\rightarrow} Cl: \tau') \]

We want to check if \( \tau \) is conformant to \( \tau' \), \( \tau \leq \tau' \). From the definition of conformance between global protocols, for each agent \( ag \in \tau \) we must find at least one agent \( ag' \in \tau' \) s.t. \( \tau_{ag} \subseteq (M_M, M_A) \tau_{ag'} \). First of all, we generate the local perspectives LAPIs of \( \tau \) and \( \tau' \) through projection.

\[ \tau_{Buy} = (\overset{res?}{\rightarrow} Sel):((\overset{res}{\rightarrow} Sel: \overset{money}{\rightarrow} Sel \overset{no}{\rightarrow} Sel: \overset{Buy}{\rightarrow} Bu))) \]

\[ \tau_{Sel} = (\overset{res?}{\rightarrow} Buy):((\overset{res?}{\rightarrow} Bu: \overset{money}{\rightarrow} Bu: \tau') \overset{no}{\rightarrow} Bu: \overset{Sel}{\rightarrow} Sel)) \]

\[ \tau_{Cl} = (\overset{book?}{\rightarrow} Shop):((\overset{book}{\rightarrow} Shop: \overset{euros}{\rightarrow} Shop: \tau') \overset{no}{\rightarrow} Shop: \overset{no}{\rightarrow} Cl: \tau')) \]

\[ \tau_{Shop} = (\overset{book?}{\rightarrow} Cl):((\overset{book}{\rightarrow} Cl: \overset{euros}{\rightarrow} Cl: \tau') \overset{no}{\rightarrow} Cl: \tau')) \]

Then we apply the rules deriving from Definition 4.2, obtaining that (Figure 1) \( \tau_{Buy} \leq (M_M, M_A) \tau_{Cl} \) with \( M_M = \{res? \rightarrow book?, res \rightarrow book, money \rightarrow euros, no \rightarrow no\} \) and \( M_A = \{Buy \rightarrow Cl, Sel \rightarrow Shop\} \) and (Figure 4) \( \tau_{Sel} \leq (M_M, M_A) \tau_{Shop} \) with the same maps. From this, we derive that \( \tau \leq (M_M, M_A) \tau' \).

In this example we had no prior knowledge on possibly correct mappings. In many real scenarios, however, some of the correct mappings among messages and agents, respectively, are known in advance. The values \( \{ag1 \mapsto ag2\} \) used to initialize the maps in the definition of \( \tau_{ag1} \subseteq (c_M, c_M) \tau_{ag2} \) if \( (M_M, M_A) = \text{isConformant}(\tau_{ag1}, \tau_{ag2}, \emptyset, \{ag1 \mapsto ag2\}) \) correspond to the worst case where the developer has no knowledge at all about the possible correct mappings, and wants to generate all of them for further inspection. If the developer knows that the initial associations modelled by
To demonstrate how to exploit the maps generated by the conformance testing to substitute JADE (http://jade.tilab.com) agents with other JADE agents, we adopted an automatic source code translation approach. The methodology we followed consists in three steps:

a) 1st step, conformance checking (corresponding to steps (i),(ii) and (iii) presented at the very beginning of this Section): The algorithm presented in Section [V] is fully implemented in SWI-Prolog (http://www.swi-prolog.org). Prolog has been chosen because of its built-in support to handle cyclic terms, coinduction, and for the possibility to use backtracking for generating all the existing maps. The implementation of the algorithm is < 400 LOC.

b) 2nd step, substitution (corresponding to step (iv)): Let us suppose that $ag_1 \in mas'$ with maps $M_A$ and $M_M$. For demonstrating how substitution can be put into practice we have opted for a basic approach where we apply the maps to the source code of $ag_1$, and we use the modified source code $map(ag_1)$ instead of the source code of $ag_2$ in $mas'$; we operate on the Java file containing the JADE class implementing $ag_1$ and we substitute all the occurrences of $ag_1$ with $M_A(ag_1)$ and all occurrences of $msg_i$ with $M_M(msg_i)$. This substitution step is also implemented in SWI-Prolog (< 30 LOC).

c) 3rd step, execution (corresponding to step (v)): We recompile $map(ag_1)$ and we execute $mas'$ with $map(ag_1)$ instead of $ag_2$. Despite being simple and applicable only when the source code is available, this approach demonstrates how we can actually use the maps generated during the conformance check, and has been adopted for all the examples shown in this paper.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a conformance modulo mapping algorithm suitable for checking conformance between local protocols specified as (projected) trace expressions, together with its implementation and usage example. The paper presents a general solution to the problem with as few constraints as possible, to make it reusable in as many situations as possible, but the actual scenarios where we believe that our approach can be more profitably exploited involve conformance between different versions of the same LAIP or LAIPs which are known to be similar, like the ones presented in Sections [I] and [V]. As another example, a self-driving car may interact with other cars, lights, etc., according to the current road norms (LAIP $\tau_1$). Norms change and the new LAIP to which the car must conform, becomes $\tau_2$. Which transformations (mappings) should we implement over $\tau_1$ to ensure it is syntactically conformant to $\tau_2$? The developer in charge of migrating $\tau_1$ to $\tau_2$ can use our algorithm for having guarantees on the syntactic compliance, although she cannot have guarantees that semantics is preserved: a human is required to finally select and validate the produced mappings. We think that semantic compliance will never be fully automated, and for this reason we expect that our
algorithm should be used in scenarios where LAIPs should not be re-aligned frequently.

W.r.t. the five steps introduced in Section 1, we exploited achieved results for steps (1-2), we devoted the entire paper to step (3), and we demonstrated how to tackle step (5). More sophisticated approaches could be followed for step (5), each with pros and cons. Experimenting some of them, such as introducing a mediator agent between mas and mas' acting as the i interface and generating wrappers for the agents that must substitute other agents, will be explored in the future.

Step (4) is an open problem which falls outside the scope of our investigation: in this paper we do not face the issue of "semantic/pragmatic conformance", but only that of "syntactic conformance". In case some constraints on interactions are know, for example commitments that must be fulfilled and that can drive the choice of the most suitable mapping, we might exploit them. Otherwise, by interpreting messages as words or sentences in some natural language, we might take advantage of semantic techniques similar to those used for matching ontological concepts [37]. Ontology matching could hence be exploited in the global process we have presented, in step (4). We remark that if we knew in advance which are the semantically correct message and agents mappings, we could feed our algorithm with them and use it as a "plain conformance checker" like those mentioned above, rather than a "conformant mappings builder". Even if we knew all the correct mappings in advance, however, we could not run any of the existing conformance checking algorithms on trace expressions, because of their higher expressiveness.

Finally, an extremely challenging issue would be to identify mappings where one message used in one MAS corresponds to a sequence of messages used in another MAS, and to consider message inputs and outputs as well. This issue will drive our future directions of investigation.

REFERENCES