Constrained Global Types for Dynamic Checking of Protocol Conformance in Multi-Agent Systems

Davide Ancona Matteo Barbieri Viviana Mascardi

DIBRIS - Università di Genova, Via Dodecaneso 35, 16146 Genova, Italy
{davide.ancona,matteo.barbieri,viviana.mascardi}@unige.it

Global types are behavioral types for specifying and verifying multiparty interactions between distributed components, inspired by the process algebra approach.

In this paper we extend the formalism of global types in multi-agent systems resulted from our previous work with a mechanism for easily expressing constrained shuffle of message sequences; accordingly, we extend the semantics to include the newly introduced feature, and show the expressive power of these "constrained global types".

1. INTRODUCTION

In open, dynamic, unpredictable and heterogeneous systems as Multi-Agent Systems (MASs) are, ensuring conformance of the agents' actual behavior to a given interaction protocol is of paramount importance to guarantee the participants' interoperability and security.

In our previous work [1, 2] we proposed a formalism of global session types – global types for short, [5, 6] – for specifying multiparty interactions in a MAS, based on cyclic Prolog terms. Global types represented in that formalism can be directly exploited in the definition of a monitor agent implemented in the Jason logic-based programming language [4] for Belief-Desire-Intention (BDI) agents [8], that validates messages with respect to a given protocol at runtime.

Even if that formalism is expressive enough to represent many interesting protocols in a compact notation, there are some protocols, such as the Alternating Bit one proposed by Deniélou and Yoshida [7], whose representation would grow exponentially in the number of different message types involved in the communication. In this paper we extend our formalism with a mechanism for easily expressing constrained shuffle of message sequences, thus reaching the expressive power required to represent complex protocols like the Alternating Bit in a very compact way.

The paper is organized in the following way: Section 2 describes the extension of [1, 2] with constraints, Section 3 presents the implementation of our framework and shows some examples, and Section 4 concludes.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

2. CONSTRAINED GLOBAL TYPES

Global types τ are built on top of the following constructs: *sending actions* which occur between two agents and are 4-tuple consisting of two agent identifiers (sender and receiver), the speech act or performative (for example, "tell", "ask", "refuse"), and the actual content of the message; *predicates on sending actions* ($p \in P$) that model which kind of message pattern is expected at a certain point of the conversation; *empty type* (λ) that represents the empty sequence of sending actions, and models the fact that a protocol is allowed to be terminated at a certain point of the conversation; *sequence* ($p : \tau$) that allows the definition of types composed by a predicate on a sending action followed by another global type; *choice* ($\tau_1 + \tau_2$) allowing the definition of global types where more than one option is available; *fork* ($\tau_1 \mid \tau_2$) that specifies two interactions that can be interleaved; and *concatenation* ($\tau_1 \cdot \tau_2$), used to append global types together. To specify loops we exploit recursive global types; for instance, the protocol consisting of infinite sending actions where first `alice` sends a message with performative `tell` and content `ping` to `bob`, and then `bob` replies with a `tell` performative and `pong` content to `alice`, can be represented by the recursive type $\tau = \text{pi} : \text{po} : \tau$ where pi is a predicate holding on sending action (*alice*, *bob*, `tell`, ping) and po holds on (*bob*, *alice*, `tell`, pong).

A constrained global type is a triple (τ, Γ, V), where τ is a global type and Γ and V are defined below.

$V{:}P \to 2^P$ represents the *static constraints*: for each predicate $\text{p} \in P$, $V(\text{p})$ denotes the transitions that must occur before p may take place. In fact, $V(\text{p})$ specifies the constraints on p; in this way it is possible to impose that a sending action specified in the left-hand side of a fork must occur strictly after a sending action specified in the right-end side.

$\Gamma{:}P \to 2^P$ is the *dynamic constraints store* (DCS) which associates each predicate symbol p with the set of predicate symbols corresponding to those transitions that must still occur before p can take place. Therefore, a transition p is enabled if and only if $\Gamma(\text{p}) = \emptyset$. Initially Γ specifies the enabled transitions, hence there must exist at least one p such that $\Gamma(\text{p}) = \emptyset$.

Figure 1 gives the formal interpretation of constrained global types. Intuitively, a constrained global type represents a state from which several transitions to other states (i.e. other constrained global types) are possible; for each constrained global type there is a corresponding set of messages which cause a transition to a new state. A transition is "fired" when a message that belongs to that set is exchanged.

$$\text{(seq)}\ \frac{\mathrm{p}(a) \wedge \Gamma(\mathrm{p}) = \emptyset \quad \forall \mathrm{p}' \in P\ \Gamma'(\mathrm{p}') = \begin{cases} V(\mathrm{p}) & \text{if } \mathrm{p}' = \mathrm{p} \\ V(\mathrm{p}') \setminus \{\mathrm{p}\} & \text{if } \mathrm{p}' \neq \mathrm{p} \end{cases}}{(\mathrm{p} : \tau, \Gamma) \xrightarrow{a} (\tau, \Gamma')} \qquad \text{(choice)}\ \frac{(\tau_i, \Gamma) \xrightarrow{a} (\tau_i', \Gamma')}{(\tau_1 + \tau_2, \Gamma) \xrightarrow{a} (\tau_i', \Gamma')}\ i \in \{1,2\}$$

$$\text{(fork-l)}\ \frac{(\tau_1, \Gamma) \xrightarrow{a} (\tau_1', \Gamma')}{(\tau_1 | \tau_2, \Gamma) \xrightarrow{a} (\tau_1' | \tau_2, \Gamma')} \qquad \text{(cat-l)}\ \frac{(\tau_1, \Gamma) \xrightarrow{a} (\tau_1', \Gamma')}{(\tau_1 \cdot \tau_2, \Gamma) \xrightarrow{a} (\tau_1' \cdot \tau_2, \Gamma')} \qquad \text{(cat-r)}\ \frac{(\tau_2, \Gamma) \xrightarrow{a} (\tau_2', \Gamma')}{(\tau_1 \cdot \tau_2, \Gamma) \xrightarrow{a} (\tau_2', \Gamma')}\ \epsilon(\tau_1)$$

Figure 1: Transition rules for constrained global types.

For space constraints, we omitted rule (fork-r), which is symmetric to rule (fork-l). Since the component V specifying the static constraints is not affected by transitions, for simplicity we omitted it from the rules as well, but it is implicit that all transitions have shape $(\tau, \Gamma, V) \xrightarrow{a} (\tau', \Gamma', V)$.

The main change w.r.t. the rules for unconstrained global types described in [1] concerns rule (seq), whereas the other rules are extended in a straightforward way, and their meaning is intuitive. A transition from $(\mathrm{p} : \tau, \Gamma)$ with sending action a is allowed only if $\mathrm{p}(a)$ holds and, furthermore, if $\Gamma(\mathrm{p}) = \emptyset$ (that is, p is enabled). The new state is the pair (τ, Γ') where Γ' is obtained by Γ by resetting p by letting $\Gamma'(\mathrm{p})$ to equal $V(\mathrm{p})$, and removing p from all sets $\Gamma(\mathrm{p}')$, for all $\mathrm{p}' \neq \mathrm{p}$.

3. IMPLEMENTATION AND EXAMPLES

Our approach can be considered as a first step towards the development of a unit testing framework for MASs where testing, types, and – more generally – formal verification can be reconciled in a synergistic way: given a Jason implementation of a MAS, our approach allows automatic generation of an extended MAS that can be run on a set of tests to detect possible deviations of the behavior of a system from a given protocol, and to check agent responsiveness.

To achieve this goal, rules given in Figure 1 have been implemented by means of the `next` predicate described in [2]. Such an approach is highly modular, since other notions of global types can be considered by just changing the definition of predicate `next`. Furthermore, the developer is required to provide, besides the MAS to be verified, the following additional definitions:

- The interpretation of predicates in PS defined by the predicate `holds(msg(S,R,Perf,C), P)`, for each `P` $\in PS$.
- The constrained global type (τ, Γ, V) specifying the protocol to be tested, given by a collection of unification equations defining τ, a collection of facts of shape `g(P, [P1, ..., Pn])` and `v(P, [P1, ..., Pn])` defining the initial dynamic constraints store and the static constraints, respectively.

Given these elements, a centralized *monitor* agent is automatically generated to verify that a conversation among any number of participants is compliant with the specified constrained global type, and warns the developer if the MAS is not responsive. The code of the agents in the original MAS requires minimal changes that can be performed in an automatic way as well. The monitor keeps track of the runtime evolution of the protocol by saving its current state (corresponding to a constrained global type), and checking that each message that a participant would like to send, is allowed by the current state. If so, the monitor allows the participant to send the message by explicitly sending an acknowledgment to it. The participants are expected to interact via asynchronous exchange of messages characterized by `tell` performatives, and only two changes are required to their code:

1. the Jason standard internal action `.send` is replaced by the `!my_send` internal goal that, instead of sending the actual message with performative `Perf` and content `Content` to `Receiver`, sends a `tell` message to the monitor in the format `msg(Sender, Receiver, Perf, Content)`. When received, this message will be checked by the monitor against the constrained global type.
2. Two more plans are added for managing the interaction with the monitor, one for sending the message to the actual receiver after the monitor has checked its compliance to the protocol, or to block the execution otherwise.

As an example, we show how the alternating bit protocol can be specified by a constrained global type. We need four predicate symbols, $\mathrm{m}_1, \mathrm{m}_2, \mathrm{a}_1$, and a_2, such that $\mathrm{m_i}$ holds for sending actions of the form $(a, b, \texttt{tell}, \mathrm{msg_i})$ and $\mathrm{a_i}$ holds for sending actions of the form $(b, a, \texttt{tell}, \mathrm{ack_i})$, for $i = 1, 2$. The alternating bit protocol basically consists of two parallel infinite loops where messages of one loop have to be synchronized with the other one. The two loops can be represented by the global types $\tau_1 = \mathrm{m}_1 : \mathrm{a}_1 : \tau_1$ and $\tau_2 = \mathrm{m}_2 : \mathrm{a}_2 : \tau_2$; the main global type for the protocol consists of these two types composed with a *fork*: $\tau_{ABP} = \tau_1 \mid \tau_2$.

The function V must reflect the fact that, for every iteration, $\mathrm{m}_{2,\mathrm{i}}$ must[1] follow $\mathrm{m}_{1,\mathrm{i}}$ and $\mathrm{m}_{1,\mathrm{i}+1}$ must follow $\mathrm{m}_{2,\mathrm{i}}$. The following specification models these constraints:

$$V = \{(\mathrm{m}_1, \{\mathrm{m}_2\}), (\mathrm{m}_2, \{\mathrm{m}_1\}), (\mathrm{a}_1, \emptyset), (\mathrm{a}_2, \emptyset)\}$$

Finally, the initial content of the DCS Γ has to enable sending actions of type m_1, a_1, a_3, but not of type m_2.

$$\Gamma = \{(\mathrm{m}_1, \emptyset), (\mathrm{m}_2, \{\mathrm{m}_1\}), (\mathrm{a}_1, \emptyset), (\mathrm{a}_2, \emptyset)\}$$

This constrained global type has a one-to-one representation as a set of Prolog facts and unification equations.

Suppose the protocol is in its initial state: in this state, two transitions are expected, m_1 and m_2 but only one of them is actually enabled (the initial DCS contains the element `g(m2, [m1])` which disables transition m_2). At this point, if a sending action s such that $\mathrm{m}_2(s)$ holds occurs, the system notifies a violation because the only applicable clause for predicate *next* corresponds to rule (seq), but $\Gamma(\mathrm{m}_2) \neq \emptyset$, therefore the body of the clause fails. An error message is displayed, showing the value of function V for disabled transition and the content of the DCS for debugging purposes.

The following example shows that *predicates* on sending actions make the formalism more expressive also w.r.t. the kind of constrains that can be enforced. Let us consider the protocol shown in Figure 2 where two sequences of messages are exchanged in an interleaved way; after that, a final message with content `end` concludes the protocol. However the sending action $(a, b, \texttt{tell}, \mathrm{msg}_2)$ (corresponding to predicate

[1]The notation $\mathrm{m}_{2,\mathrm{i}}$ indicates "transition m_2 at the i-th iteration".

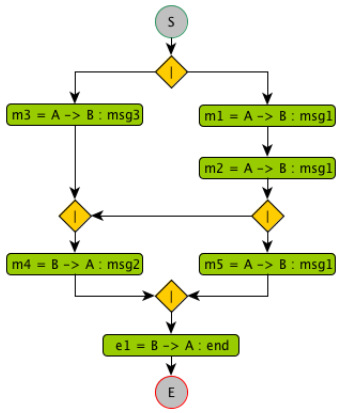

Figure 2: "Constrained fork" example.

m4) can occur only after the sending action $(a, b, \texttt{tell}, \text{msg}_1)$ has occurred twice: if constraints were specified using sending actions, it would be impossible to distinguish between the three transitions on the right side of the diagram which are labeled by the same sending action.

The protocol is specified by the constrained global type (τ_0, Γ, V), assuming that $\text{m}_1(a, b, \texttt{tell}, \text{msg}_1)$, $\text{m}_2(a, b, \texttt{tell}, \text{msg}_1)$, $\text{m}_3(a, b, \texttt{tell}, \text{msg}_3)$, $\text{m}_4(b, a, \texttt{tell}, \text{msg}_2)$, $\text{m}_5(a, b, \texttt{tell}, \text{msg}_1)$ and $\text{e}_1(b, a, \texttt{tell}, \text{end})$ hold.

$$\begin{array}{lcllcl} \tau_0 & = & \tau_f \cdot \text{e}_1 : \lambda & \tau_f & = & (\tau_l \mid \tau_r) \\ \tau_l & = & \text{m}_3 : \text{m}_4 : \lambda & \tau_r & = & \text{m}_1 : \text{m}_2 : \text{m}_5 : \lambda \\ \Gamma & = & \{(\text{m}_1, \emptyset), (\text{m}_2, \emptyset), (\text{m}_3, \emptyset), (\text{m}_4, \{\text{m}_2\})(\text{m}_5, \emptyset), (\text{e}_1, \emptyset)\} & & & \\ V & = & \Gamma & & & \end{array}$$

Finally, the protocol shown in Figure 3 expresses the fact that sequences $m_5 m_6$, $m_1 m_2$ and $m_3 m_4$ may be freely shuffled, but transition m_6 must occur after both m_1 and m_4.

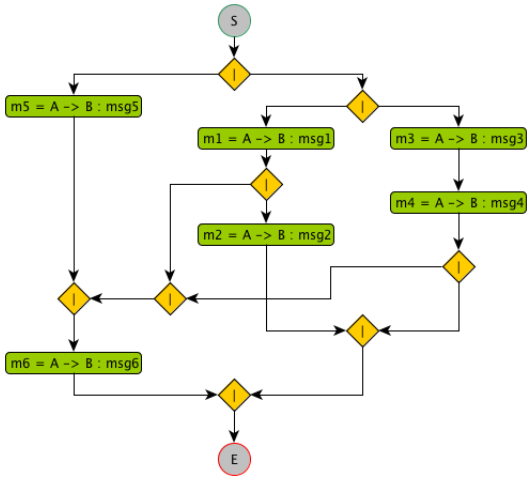

Figure 3: "Ordering of sending actions" example

The three sequences are composed with the `fork` construct, and constraints are used to impose the ordering. The protocol is specified by the constrained global type (τ_0, Γ, V) and $\text{m}_\text{i}(a, b, \texttt{tell}, \text{msg}_\text{i})$ holds for $i = 1..6$.

$$\begin{array}{lcllcl} \tau_0 & = & \tau_1 \mid \tau_2 \mid \tau_3 & \tau_1 & = & \text{m}_5 : \text{m}_6 : \lambda \\ \tau_2 & = & \text{m}_1 : \text{m}_2 : \lambda & \tau_3 & = & \text{m}_3 : \text{m}_4 : \lambda \\ \Gamma & = & \{(\text{m}_i, \emptyset) \mid i = 1..5\} \cup \{(\text{m}_6, \{\text{m}_2, \text{m}_4\})\} & & & \\ V & = & \Gamma & & & \end{array}$$

4. CONCLUSIONS AND FUTURE WORK

Although the literature on static and dynamic verification of protocol conformance within MASs is extremely rich, as the survey included in [2] demonstrates, to the best of our knowledge no previous attempts of using global types to dynamically verify MASs in some widespread agent oriented programming languages had been made before. On the other hand, using constraints to express (or restrict) interaction protocols is not new. For instance, the notion of constraint automata [3] is very relevant for our work; one of the most interesting properties of constraint automata is compositionality, hence the formalism deserves a deeper comparison with ours.

There are other interesting directions we are considering.

Constrained global types can be used not only for dynamic verification, but also for supporting self-recovering MASs. For such kinds of systems the role of the monitor agent is more complex: besides communicating to the other agents the type of messages allowed at a certain point of the conversation, the monitor should also be able to select a default recovery action in case the communication gets stuck.

Constrained global types can be used for static verification of protocol implementations as well. In this case the problem is clearly more challenging, but static verification offers stronger correctness guarantees in comparison with the dynamic approach. A preliminary step towards static verification requires studying how a constrained global type can be projected to a single agent.

Finally, performance issues that have been neglected so far, should be taken into consideration in order to make the approach usable even in those domains, such as ambient intelligence applications and embedded agents, where computational resources are strictly bounded.

5. REFERENCES

[1] D. Ancona, M. Barbieri, and V. Mascardi. Global types for dynamic checking of protocol conformance of multi-agent systems (extended abstract). In *ICTS 2012*, pages 39–43, 2012.

[2] D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic Generation of Self-Monitoring MASs from Multiparty Global Session Types in Jason. In *DALT 2012, Workshop Notes*, pages 1–17, 2012.

[3] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in Reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.

[5] M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP'07*, pages 2–17. Springer, 2007.

[6] G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On Global Types and Multi-Party Sessions. *Logical Methods in Computer Science*, 8:1–45, 2012.

[7] P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP'12*, LNCS. Springer, 2012.

[8] A. S. Rao and M. P. Georgeff. BDI agents: from theory to practice. In *ICMAS*, pages 312–319, S. Francisco, CA, June 1995.