Norms for Typing MAS Multi-Agent Systems

This is the author's manuscript

Original Citation:
Norms for Typing MAS Multi-Agent Systems / Matteo Baldoni; Cristina Baroglio; Federico Capuzzimati. - ELETTRONICO. - (2013).

Availability:
This version is available http://hdl.handle.net/2318/145391 since 2016-06-28T08:43:07Z

Terms of use:
Open Access
Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)
Normative Multi-Agent Systems: NorMAS 2013

Norms for Typing MAS

M. Baldoni\textsuperscript{1}  C. Baroglio\textsuperscript{1}  F. Capuzzimati\textsuperscript{1}

\textsuperscript{1}Dipartimento di Informatica, Università degli Studi di Torino

Leiden, August 19-23, 2013
Overview

1. Vision and motivation
2. Types and MAS
3. Commitment Protocols
4. Reifying commitment protocols into artifacts
5. Type checking via commitments
6. Discussion and conclusions
B2B, cross-business, open environment systems

- Software infrastructures: more and more global, pervasive and autonomic
- Computing is becoming ubiquitous, with embedded and distributed devices interacting with each other
- MAS have been recognized to be a promising paradigm for this kind of scenarios
**Typing Systems**

**HOWEVER**

The more the complexity of programming these systems will increase, the more the need for effective tools for reasoning on properties of programs is noticed.

**Types**

provide abstractions to perform sophisticated forms of program analysis and verifications that help programmers to face the complexity of their job.
Typing Systems

**HOWEVER**

The more the complexity of programming these systems will increase, the more the need for effective tools for reasoning on properties of programs is noticed.

**Types**

provide abstractions to perform sophisticated forms of program analysis and verifications that help programmers to face the complexity of their job.
Typing Systems

- Enable compile time/runtime error checking
- Conceptual and abstraction tools for modeling
- Documentation
- Conformance and compliance
- Reasoning about programs and components
- Type checking as a simple form of (a priori/runtime) verification
Typing Systems for MAS

- Here we focus on two more recent proposals:

**Global Session Types in Jason**
- By D. Ancona, S. Drossopoulou, and V. Mascardi
  - [Ancona et al., 2012, Ancona et al., 2013]
- Behavioral types for multiparty interactions
- Monitoring agent that verifies (dynamically) the conformance of interacting agents w.r.t. a global session type

**simpAL**
- By A. Ricci and A. Santi
  - [Ricci and Santi, 2012a, Ricci and Santi, 2012b]
- An agent-oriented programming language with types checking inspired by main stream OO languages
- Static type checking for error detection
Typing Systems for MAS

- Here we focus on two more recent proposals:

**Global Session Types in Jason**

- By D. Ancona, S. Drossopoulou, and V. Mascardi
  [Ancona et al., 2012, Ancona et al., 2013]
- Behavioral types for multiparty interactions
- Monitoring agent that verifies (dynamically) the conformance of interacting agents w.r.t. a global session type

**simpAL**

- By A. Ricci and A. Santi
  [Ricci and Santi, 2012a, Ricci and Santi, 2012b]
- An agent-oriented programming language with types checking inspired by mainstream OO languages
- Static type checking for error detection
Typing Systems for MAS

- Here we focus on two more recent proposals:

**Global Session Types in Jason**
- By D. Ancona, S. Drossopoulou, and V. Mascardi
  - [Ancona et al., 2012, Ancona et al., 2013]
- Behavioral types for multiparty interactions
- Monitoring agent that verifies (dynamically) the conformance of interacting agents w.r.t. a global session type

**simpAL**
- By A. Ricci and A. Santi
  - [Ricci and Santi, 2012a, Ricci and Santi, 2012b]
- An agent-oriented programming language with types checking inspired by mainstream OO languages
- Static type checking for error detection
Global Session Types in Jason

- Inspired by Scribble, in [Ancona et al., 2012, Ancona et al., 2013] protocols are the key aspect
- Protocols are expressed by means of global session types
- Jason is extended in order to automatically generate an agent monitor for dynamic conformance (compliance) verification
- Cyclic Prolog terms
Global Session Types in Jason

- Global session types are “procedural” types (process abstraction), so they do not:
  - respect autonomy of agents
  - clearly express what is expected from a role and what is possible for a role

- Who is the agent monitor? Who trusts it? Should all messages/actions be notified to it? How to guarantee this fact?

- Lack of a normative characterization of coordination [Castelfranchi, 1997, Singh, 1999], so that the publicly acceptance of the regulation allows reasoning about agents’ behavior [Conte et al., 1998]
Inspired by mainstream OO languages, in [Ricci and Santi, 2012a, Ricci and Santi, 2012b] static type checking for error detection is the key aspect.

Builds on the experience of JaCaMo.

role, usage-interface, org-model: interfaces

agent-script, artifact, org: implementations
 simpAL

Role Interface

Task
- talks-about
- understand

Task
- talks-about
- input-params

Agent Implementation

Plan-for
- assign-task
- do-task
- tell

how

assignee

Agent Implementation

Plan-for
- assign-task
- do-task
- tell

assigner

implements

what
Static vs dynamic type systems: is compile time checking the key point? Sometimes this is not good also for OO languages (eg. downcasting)

Types or ontological reasoning?

Are roles mere agent interfaces?

What is the semantics of types? Is type checking only a syntactic matching?
Commitment

\[ C(x, y, r, p) \]

represents the engagement from \( x \) to \( y \), to bring about the consequent condition \( p \) when the antecedent condition \( r \) holds.

- Commitments have a normative nature: agents are liable for the violation of the commitments they have taken.
- Commitment protocols allow for flexible behaviours: \( x \) is free to choose its actions.
- The agent’s compliance can be verified by observing the interaction.
A commitment-based protocol is a set of actions whose meaning in terms of effects on the social state is agreed upon by all the interacting agents.

**Actions definition**

\[
\text{action means effects if condition}
\]

The *means* construct captures which *physical* events count as which *social* events

- **means**: introduces the social effects
- **if**: condition for the action to have the intended meaning
Commitment-based protocols

- An agent/initiator should be able to accept or refuse a proposal.
- An agent/participant should be able to complete the assigned task (done) or to communicate its failure.
Commitment-based protocols

Organizational Roles
[Baldoni et al., 2007, Boella and van der der Torre, 2007]

- **Foundation, definitional dependence, and institutional empowerment**
- **Requirements** (ability to satisfy own commitments) and **powers** (action with a institutional meaning)
Integrating JADE with support for indirect communication

According to Keil and Goldin, indirect communication fosters the collaboration and the coordination inside open systems.

The adoption of programmable communication channels allows the specification of a normative facet that applies to the agents that are involved in the interaction.
Mercurio [Baldoni et al., 2011]

- Artifact abstraction [Weyns et al., 2007, Omicini et al., 2008]: first-class entity, i.e. non-agent dynamic and programmable resource, that an agent can use, perceive, observe
- Artifacts can provide mediated, indirect communication to agents
- Designers can leverage artifacts to explicitly model interaction protocols, defining a social agreement accepted by agents using them
Artifacts, as programmable indirect communication channels, can reify and implement normative characterization and social expectation.

Such an artifact entails mutual, social dependencies between agents using it.

We model social dependencies as commitments.
2COMM [Baldoni et al., 2013]

● 2COMM: reifying commitment protocols in JADE by means of CArtAgO framework [Ricci et al., 2009]
An agent can use a communication artifact enacting a role
- A role represents the interface between the artifact and the agent using it
- When acting as a certain role, an agent’s actions impact on the interaction state
Type checking via commitments
Type checking with commitments

- Type checking by means of Java Annotations
- Dynamic type checking: behaviors must comply to requirements (java reflection is used)
- Type system as a logic “theory” of commitments
- Commitment to regulation [Marengo et al., 2011]: regulating how
Conclusion

- Dynamic vs static type checking
- Conformance as logic entailment
- Compliance: programmable communication channel with monitoring functionalities
- A normative value thanks to commitment-based approach
- Flexibility and openness typical of MAS
- Modularity and compositionality typical of design and development methodologies
- Enable a business level architecture as fostered in [Chopra and Singh, 2009]


