Compliance Verification of Agent Interaction: a Logic-based Software Tool

Federico Chesani, Paola Mello, Paolo Torroni
University of Bologna
Marco Alberti, Marco Gavanelli, Evelina Lamma
University of Ferrara

Vienna, April 14th, 2004

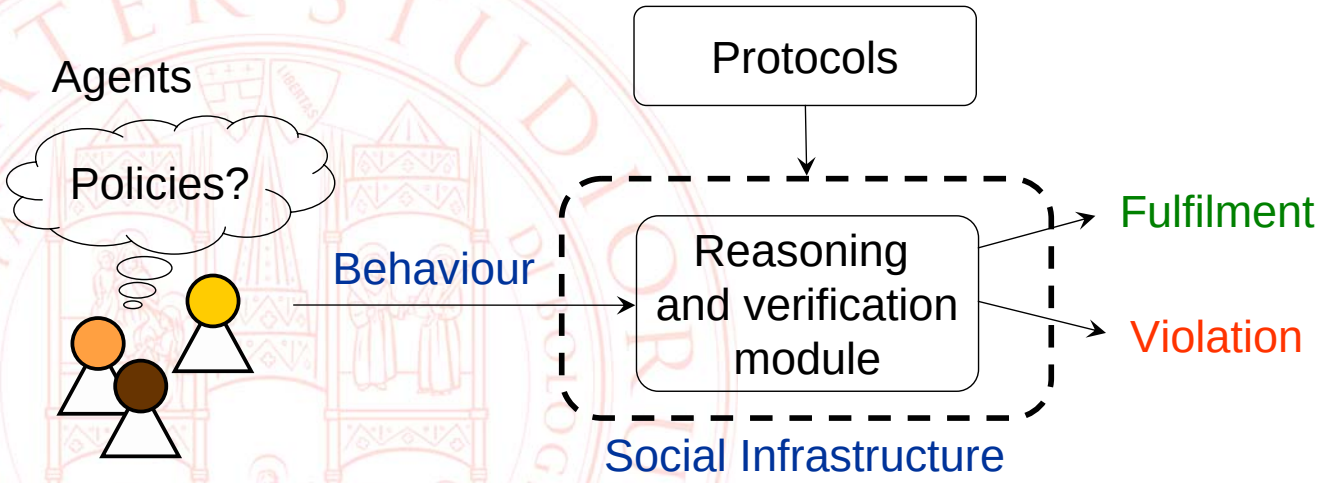
Motivations of this work

- **Open Societies of Agents**
  - agents are heterogeneous
  - no assumption on the internals of agents
  - no assumptions on the behaviour of agents
    → observation of the external behaviour of agents (interactions, exchanged messages)
- Interaction
  - agent communication language
  - interaction protocols
- **issues:**
  1. formal specification of interactions (protocol definition)
  2. verification of compliance

Structure of this presentation

- Introduction to the formal framework
- The Society Infrastructure tool (*SOCS-SI*)
- The SCIFF: generation of expectation, as well as detection of fulfillment and violation
- The Graphical User Interface developed for the *SOCS-SI* tool
- Conclusions and future work

Compliance Verification



Interaction specification

- Use of a uniform, based on abductive logic, declarative formalism and computational model for the specification of ACL and protocols
- Agents interact by exchanging messages (mapped onto events)
- According to interaction protocols, expectations are generated about the agent behaviour
- Protocols are represented using ***Social Integrity Constraints (ICs)***

Social Integrity Constraints (ICs)

- **Example of Social Integrity Constraints: the politeness protocol**

  If an agent A ask something to an agent B, B is supposed to be polite, and to answer back yes or no (but not both):

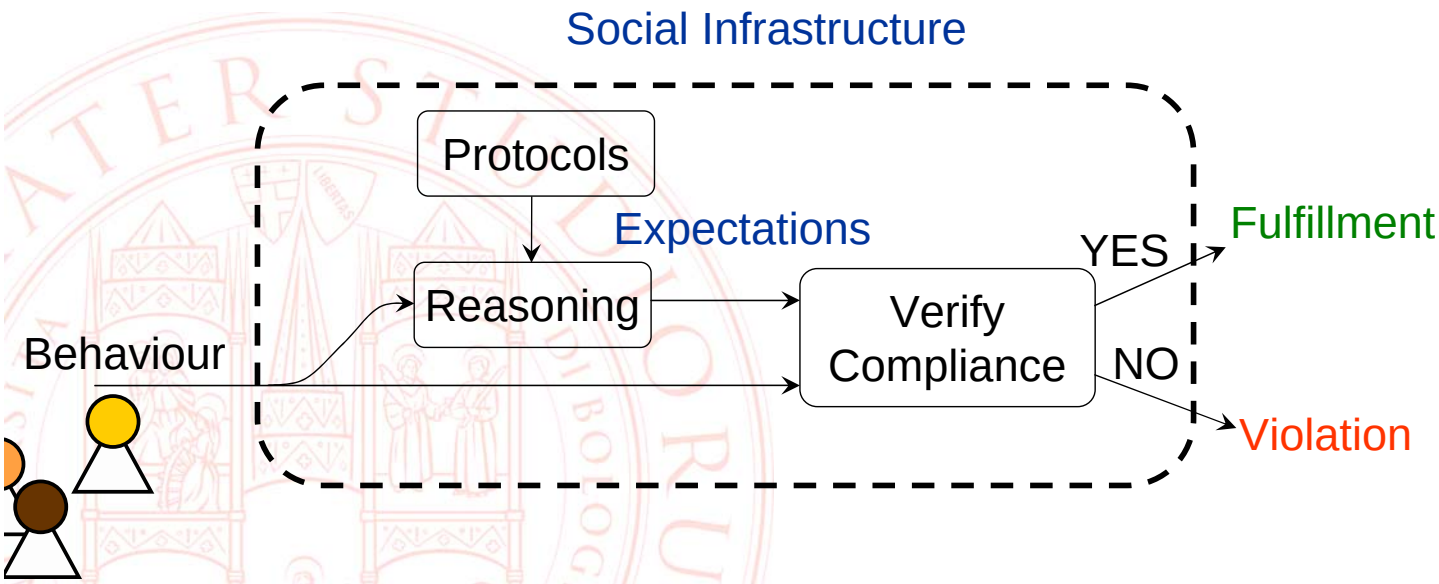
  (1) If an agent A ask something to agent B, B is supposed to answer yes or no

  $$H(\text{tell}(A, B, \text{ask}(\text{Something}), T)) \rightarrow E(\text{tell}(B, A, \text{yes}(\text{Something}), T'), T' \geq T \vee E(\text{tell}(B, A, \text{no}(\text{Something}), T'), T' \geq T$$

  (2) An agent X cannot say yes and no in answer at the same request

  $$H(\text{tell}(B, A, \text{yes}(\text{Something}), T) \rightarrow EN(\text{tell}(B, A, \text{no}(\text{Something}), Tr), Tr \geq T$$

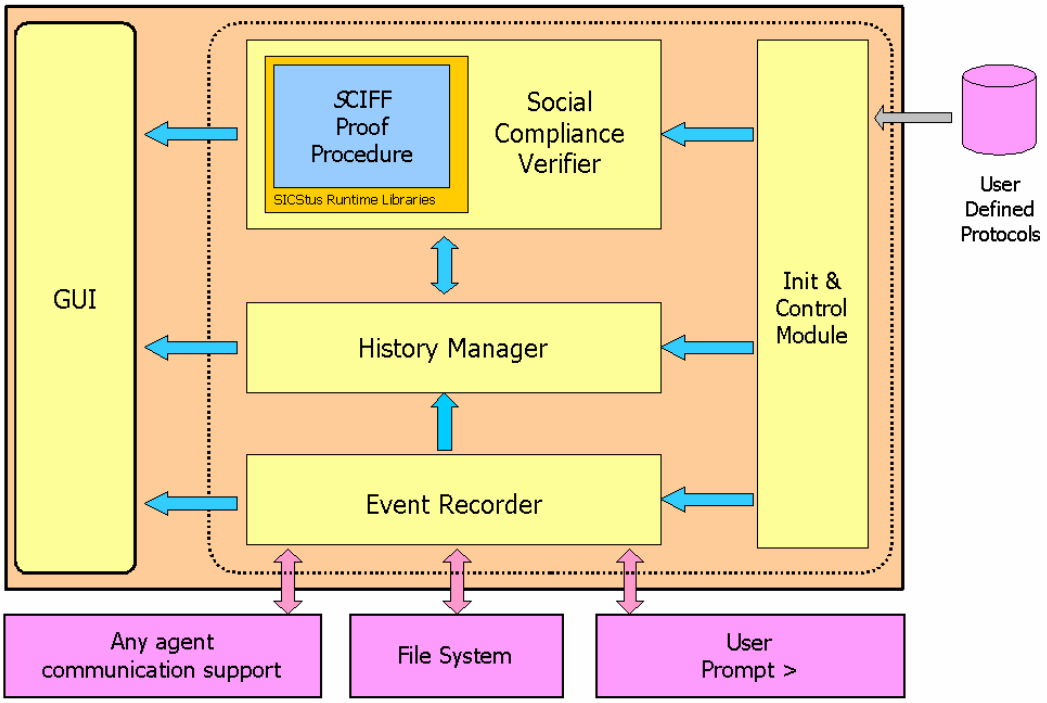
Social infrastructure



The *SOCS-SI* tool

- A software tool for verifying the compliance of agent behavior in respect to given protocols.
- The tool has been developed within the european IST SOCS project. More precisely, *SOCS-SI* is the implementation of the abductive logic framework for verification.
- Within the same project, a complete platform for agent development, PROSOCS, has been defined and implemented (earlier presentation this morning by Kostas).
- *SOCS-SI* was initially intended as the social infrastructure for the PROSOCS platform.
- However it can be easily used with other platforms. The integration with JADE and Tucson, for example, is currently under development.

SOCS-SI - Overall Architecture



The SCIFF Proof Procedure

- It is an abductive proof procedure, where:
  - Expectation (**E** and **EN**) are mapped as abducibles
  - Social Intergrity Constrains (***ICs***) are represented as the Integrity Constraints of the abductive framework
- Extends the **IFF** proof procedure:
  - The set of facts grows **dynamically**
  - Deals with **CLP** constraints (constraints also onto the abducible variables)
  - Concepts of **fulfilment** and **violation**
- Implemented using the **SICStus Prolog** and the Constraint Handling Rules (**CHR**) library

The SCIFF Proof Procedure

- The SCIFF Proof Procedure processes the events: for each event it looks for a possible "unification" with the body of one (or more) *ICs*.
- For each IC whose "body" is verified by the events, the expectations defined in the head are generated.
- The expectations will then be checked for **fulfilment** or **violation**

body

H(tell(A, B, ask(Something), T) →
E(tell(B, A, yes(Something), T'), T' ≥T ∨
E(tell(B, A, no(Something), T'), T' ≥T

head

Example (generation of expectations)

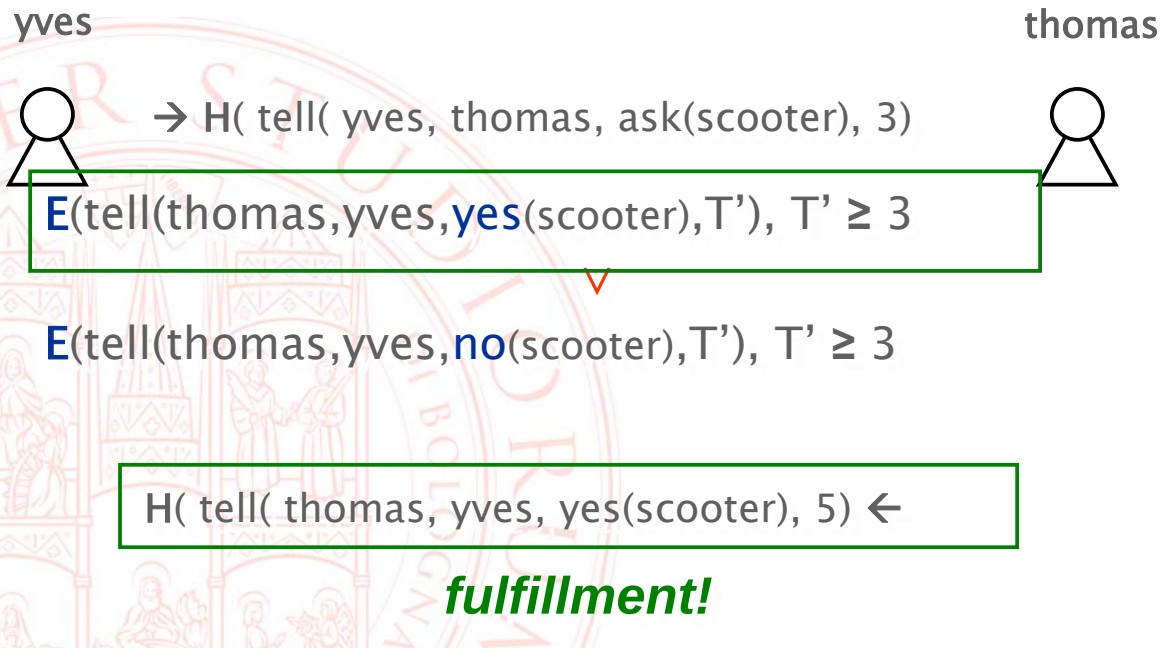
yves → H(tell(yves, thomas, ask(scooter), 3)

thomas

H(tell(A, B, ask(Something), T) →
E(tell(B, A, yes(Something), T'), T' ≥ 3
E(tell(B, A, yes(Something), T'), T' ≥T ∨
E(tell(B, A, no(Something), T'), T' ≥T

E(tell(thomas, yves, yes(scooter), T'), T' ≥ 3

Example (fulfilment of an expectation)



fulfillment!

Expectations and Violations

Expectations can be violated in two different ways:

1. **Something happened that was expected NOT to happen**
2. **Something that was expected to happen didn't happen (either because a deadline has expired, or because it is assumed that no more events can happen anymore)**

Example (violation of an expectation)

yves

thomas

→ H(tell(yves, thomas, ask(scooter), 3)

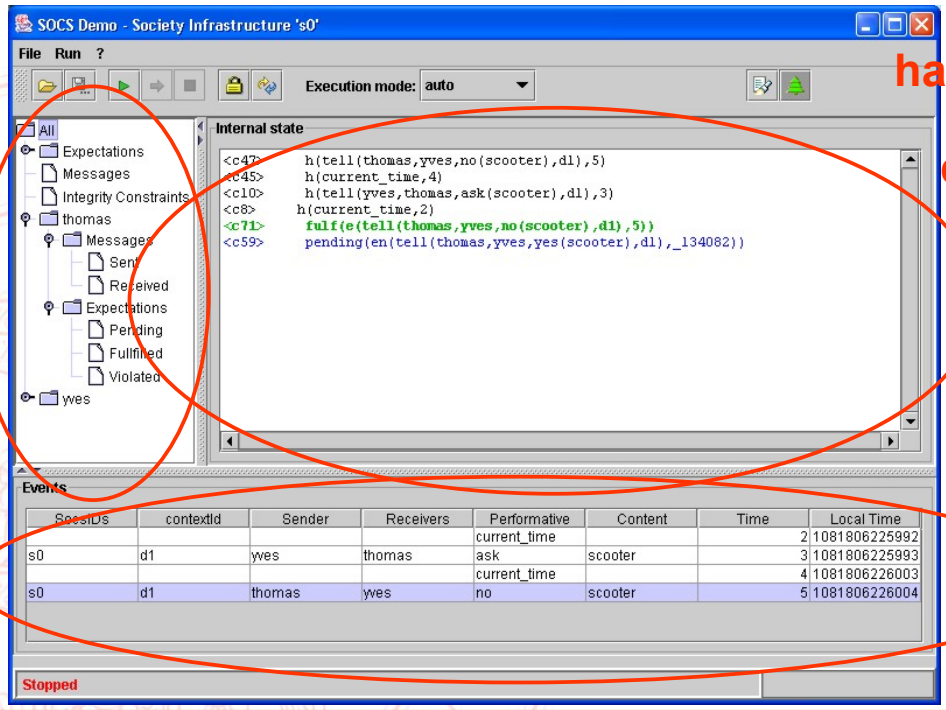
H(tell(thomas, yves, yes(scooter), 5) ←

EN(tell(thomas, yves, no(scooter), T'), T' ≥ 5

H(tell(thomas, yves, no(scooter), 9) ←

violation!

The Graphical User Interface



list of agents

happened events and expectations

list of events

The Tree Viewer

- Protocols specify which is the next action, in response to a certain event. More often, a protocol specify **alternative** (sets of) actions.
- ICs represent alternatives as a disjunction of sets of expectations.
- The "politeness protocol":

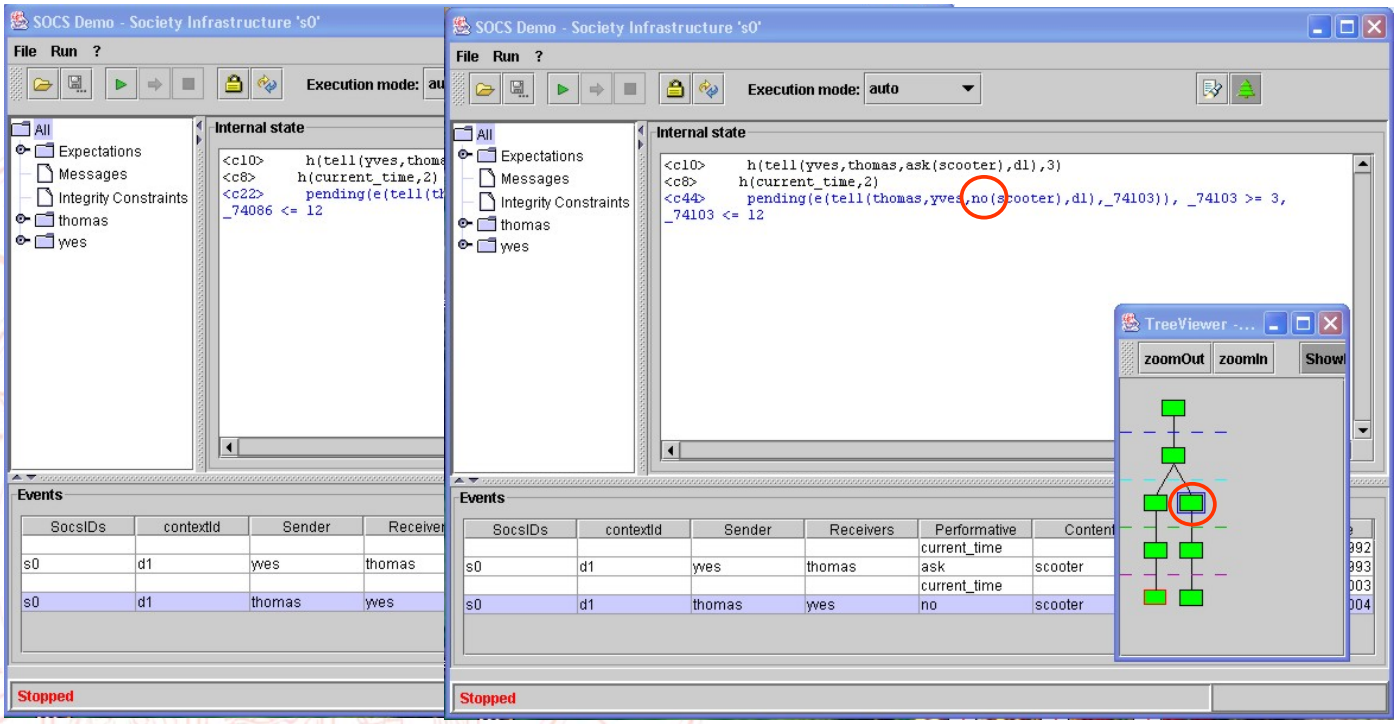
$$H(\text{tell}(A, B, \text{ask}(\text{Something}), T) \rightarrow E(\text{tell}(B, A, \text{yes}(\text{Something}), T'), T' \geq T \vee E(\text{tell}(B, A, \text{no}(\text{Something}), T'), T' \geq T$$

The Tree Viewer

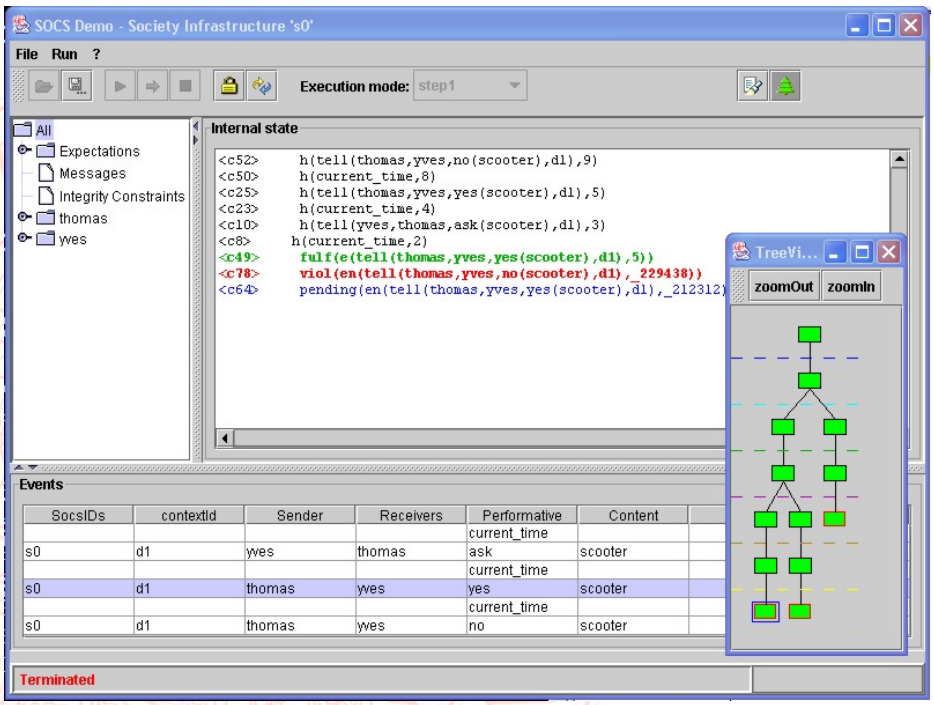
- The more intuitive way to represent them is a tree structure.
- Each node represents the facts happened until now (i.e. the messages exchanged), as well as a set of expectations about the future events.
- Nodes at the same level are alternatives (defined by the protocol).

From Agent Theory to Agent Implementation
17th European Meeting on Cybernetics and System Research
Vienna, April 13 – 16, 2004

The Tree Viewer



How expectations are rendered



Conclusions

- SOCS is a software tool for verification of agent compliance to interaction protocols
- Interactions, as well as protocols, are expressed by means of a declarative logic formalism
- Main uses of the tool:
  - Checks for conformance of a static dialogue (logged onto a file)
  - Runtime checks of conformance within agents platforms (mainly PROSOCS, but also JADE and TUCSON)
  - As a "test tool" for protocol designers

Future work

- To extend the number of supported agent platforms
- To investigate the generation and the management of an agents *reputation*
- To suggest agents what they are (not) expected to do

This work is partially funded by IST program of the European Commission, under the IST-2001-32530 SOCS Project.

This work is also partially funded by the national MIUR COFIN 2003 projects "Sviluppo e verifica di sistemi multi-agente basati sulla logica" and "La gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici ed informatici".