A verifiable logic-based agent architecture

Marco Alberti[1] Federico Chesani[2], Marco Gavanelli[1], Evelina Lamma[1], and Paola Mello[2]

[1] ENDIF, University of Ferrara, Italy
[2] DEIS, University of Bologna, Italy

Abstract. In this paper, we present the $\mathcal{S}$CIFF platform for multi-agent systems. The platform is based on Abductive Logic Programming, with a uniform language for specifying agent policies and interaction protocols. A significant advantage of the computational logic foundation of the $\mathcal{S}$CIFF framework is that the declarative specifications of agent policies and interaction protocols can be used directly, at runtime, as the programs for the agent instances and for the verification of compliance.
We also provide a definition of conformance of an agent policy to an interaction protocol (i.e., a property that guarantees that an agent will comply to a given protocol) and a operational procedure to test conformance.

1 Introduction

In the last few years, the multi agent paradigm has been proposed as an effective conceptual and technological tool for enhancing and automating interaction between humans in areas such as workflow management and electronic commerce.

However, in order for the users to be willing to delegate critical functions to software agents, it is necessary for the agent technology to be highly reliable.

Of course, users' trust will be much higher if the reliability of Multi-agent Systems (also MAS in the following) is not only demonstrated by running systems, but also proved formally, by expressing and proving formal properties of a MAS.

The properties of a MAS that can be expressed and proved obviously depend on the amount of information that is available about the MAS and the agents that compose it.

Guerin and Pitt [1] distinguish three possible types of verification, depending on the available information:

- *Type 1*: verify that an agent will always comply;
- *Type 2*: verify compliance by observation;
- *Type 3*: verify protocol properties.

Type 1 verification can be performed at design time. Given a representation of the agent, by means of some proof technique (such as *model checking* [2]) it proves that the agent will always exhibit the desired behaviour.

Type 2 verification can be performed at runtime. It checks that the *actual* agent behaviour being observed is compliant to some specification. It does not require any knowledge about the agent internals, but only the observability of the agent behaviour.

Type 3 verification can be performed at design time. It proves that some property will hold in the society, provided that the agents follow the interaction protocols (i.e., behave accordingly to the interaction specification).

In previous work, we focused on Type 2 and 3: for Type 2 verification, we developed the $\mathcal{S}$CIFF abductive framework and proof procedure [3], and we integrated it into the SOCS-SI system [4]; for Type 3 verification, we developed the g-$\mathcal{S}$CIFF extension of $\mathcal{S}$CIFF[5].

In this paper, we present the $\mathcal{S}$CIFF agent platform, and we tackle Type 1 verification. For this type of verification, a representation of the agent internal policy is necessary. We developed an agent model, inspired by that proposed by Kowalski and Sadri [6] where the agent internal policies are expressed by means of the same formalism (the $\mathcal{S}$CIFF language) that we use for specifying interaction protocols. This choice, besides letting us exploit the $\mathcal{S}$CIFF operational machinery to implement agent systems, as we show in this paper, also makes it simpler to express and verify that an agent will be compliant to an interaction protocol.

The paper is structured as follows. In Sect. 2, we introduce the $\mathcal{S}$CIFF agent platform architecture, and the language that can be used for specifying agent policies and interaction protocols. In Sect. 3, we give a definition of conformance, and propose an operational procedure to verify conformance. Discussion of related work and conclusions follow.

2 The $\mathcal{S}$CIFF agent platform

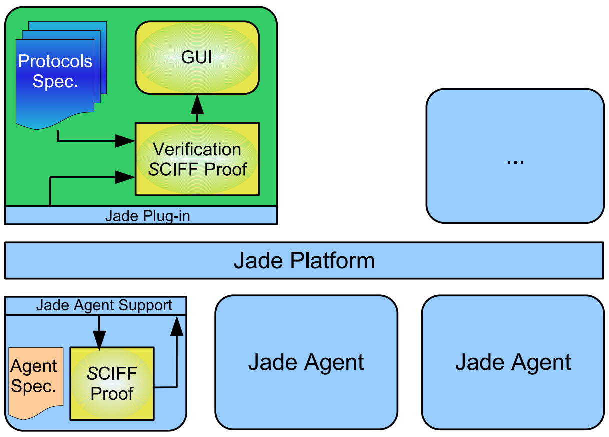

Fig. 1. The $\mathcal{S}$CIFF agent platform architecture.

The $\mathcal{S}$CIFF agent platform, represented in Fig. 1 has been implemented on top of the JADE agent platform [7].

The $\mathcal{S}$CIFF agent platform is composed of these main components: the *SOCS-SI*[3] tool, which can be used for the on-line verification, the $\mathcal{S}$CIFF agent, whose instances will populate the MAS, and, optionally, other agents which can interact with the $\mathcal{S}$CIFF agents in the MAS.

The agents communicate through a communication channel, and the messages that they exchange

The *SOCS-SI* tool, whose purpose is to observe the agent interaction and to check that they happen according to the interaction protocols, has been discussed in previous publications [4] and, for lack of space, we do not describe it here. However, in the following we briefly introduce the $\mathcal{S}$CIFF abductive framework, which we use to specify both agent policies and interaction protocols.

2.1 The $\mathcal{S}$CIFF language

We specify the agent policies, as well as interaction protocols, by means of an abductive logic program (ALP, for short, in the following) [8].

Abductive Logic Programs. An ALP is a triple $\langle P, A, IC \rangle$, where P is a logic program, A is a set of distinguished predicates named *abducibles*, and IC is a set of integrity constraints. Reasoning in abductive logic programming is usually goal-directed (being G a goal), and corresponds to find a set of abduced hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ and $P \cup \Delta \models IC$. Suitable proof procedures (e.g., Kakas-Mancarella [9], IFF [10], SLDNFA [11], etc.) have been proposed to compute such set Δ, in accordance with the chosen declarative semantics.

Events *Events* are our abstraction to represent the agent behaviour.

We consider two types of events: *happened* events (denoted by the functor $\mathbf{H}$) and *expected* events (denoted by $\mathbf{E}$, and also called *expectations*). Both are abducible, and represent hypotheses on, respectively, which events have happened and which are expected to happen: $\mathbf{H}(m_x(\ldots), T_x)$, expresses the fact that a message m_x has been exchanged between two agents at time T_x, whereas $\mathbf{E}(m_x(\ldots), T_x)$ says that the message m_x is expected to be exchanged at time T_x.

Protocol specification An interaction protocol specifies the allowed interactions among agents from an external viewpoint, i.e., it specifies the desirable agent observable behaviour.

We specify an interaction by means of an abductive logic program. The specification, besides representing a declarative representation of the protocol, is also used directly in the *SOCS-SI* tool as the program for the verification, at run time, that the agents actually comply to the protocol.

A protocol specification $\mathcal{P}_{prot}$ is defined by the tuple:

$$\mathcal{P}_{prot} \equiv \langle KB_{prot}, \mathcal{E}_{prot}, \mathcal{IC}_{prot} \rangle$$

[3] The name stands for SOCS Social Infrastructure. SOCS is the acronym of the European project (IST-2001-32530) which originally supported the research for the $\mathcal{S}$CIFF framework.

- KB_{prot} is the *Knowledge Base*,
- $\mathcal{E}_{prot}$ is the set of *abducible predicates*, and
- $\mathcal{IC}_{prot}$ is the set of *Integrity Constraints*.

KB_{prot} specifies declaratively pieces of knowledge about the interaction protocol, such as role descriptions and the list of participants. It is expressed in the form of clauses (a logic program) that may also contain in their body expectations about the behaviour of participants.

The *abducible predicates* are those that can be hypothesized in our framework, namely happened events ($\mathbf{H}$) and expectations ($\mathbf{E}$).

Integrity Constraints are forward rules, of the form *body* → *head*, whose *body* can contain literals and (happened and expected) events, and whose *head* can contain (disjunctions of) conjunctions of expectations. The syntax of $\mathcal{IC}_{prot}$ is the same defined for the SOCS Integrity Constraints [12].

In order to support goal directed interactions, we let the user specify a *goal*, which has the same syntax as the body of the clauses in KB_{prot}.

Specification 2.1 Integrity Constraints for the *query_ref* interaction protocol.

$$\begin{array}{rl}
 & \mathbf{H}(m_x(\mathit{A},\mathit{B},\mathit{query\_ref}(\mathit{Info})),\mathit{T}) \;\wedge \\
 & \mathit{qr\_deadline}(\mathit{TD}) \\
\rightarrow & \mathbf{E}(m_x(\mathit{B},\mathit{A},\mathit{inform}(\mathit{Info},\mathit{Answer})),\mathit{T1}) \;\wedge \\
 & \mathit{T1} < \mathit{T} + \mathit{TD} \\
\vee & \mathbf{E}(m_x(\mathit{B},\mathit{A},\mathit{refuse}(\mathit{Info})),\mathit{T1}) \;\wedge \\
 & \mathit{T1} < \mathit{T} + \mathit{TD}
\end{array}$$

$$\begin{array}{rl}
 & \mathbf{H}(m_x(\mathit{A},\mathit{B},\mathit{inform}(\mathit{Info},\mathit{Answer})),\mathit{Ti}) \\
\rightarrow & \mathbf{EN}(m_x(\mathit{A},\mathit{B},\mathit{refuse}(\mathit{Info})),\mathit{Tr})
\end{array}$$

Spec. 2.1 shows the integrity constraints for the FIPA *query_ref* interaction protocol [13].

Intuitively, the first IC means that if agent A sends to agent B a *query_ref* message, then B is expected to reply with either an *inform* or a *refuse* message by TD time units later, where TD is defined in the Social Knowledge Base by the *qt_deadline* predicate (with the example in Spec. 2.2, the value of TD would be 10).

The second IC means that, if an agent sends an *inform* message, then it is expected not to send a *refuse* message at any time.

2.2 The $\mathcal{S}$CIFF agent

The same language used for the specification of the interaction protocols can be used, as we show in Sect. 2.2, for the specification of agent policies. In fact, it can also be used

Specification 2.2 Social Knowledge Base for the *query_ref* social specification

$$\textit{qr\_deadline}(\textit{10}).$$

directly as the implementation language of reactive agents, following the Kowalski-Sadri schema [6], as we show in Sect. 2.2.

Agent specification An Agent Specification $\mathcal{P}_{ag}$ is an ALP

$$\mathcal{P}_{ag} \equiv \langle KB_{ag}, \mathcal{E}_{ag}, \mathcal{IC}_{ag} \rangle$$

– KB_{ag} is the *Knowledge Base* of the agent,
– $\mathcal{E}_{ag}$ is the set of *abducible predicates*, and
– $\mathcal{IC}_{ag}$ is the set of *Integrity Constraints* of the agent.

KB_{ag} and $\mathcal{IC}_{ag}$ are completely analogous to their counterparts in the protocol specification, except that they represent an individual, rather than global, perspective: they represent, respectively, the declarative knowledge and the policies of the agent.

$\mathcal{E}_{ag}$ is the set of abducible predicates: as for the protocols, it contains both expectations and happened events. The expectations can be divided into two significant subsets:

– expectations about messages where ag is the sender (of the form $\mathbf{E}_{ag}(m_x(ag, A, Content))$), i.e., actions that ag intends to do;
– expectations about messages uttered by other participants to ag (of the form $\mathbf{E}_{ag}(m_x(A, ws, Content))$, with $A \neq ag$), which can be intended as the messages that ag is able to understand.

Specification 2.3 A simple train timetable agent.

$$\begin{aligned} & \mathbf{H}(m_x(\textit{A}, \textit{me}, \textit{query\_ref}(\textit{trainTime(TrainCode)})), \textit{T}) \;\wedge \\ & timeTable(TrainCode, TTime) \\ \rightarrow\; & \mathbf{E}(m_x(\textit{me}, \textit{A}, \textit{inform}(\textit{trainTime(TrainCode)}, \textit{TTime})), \textit{T1}). \end{aligned}$$

In Spec. 2.3 a simple agent specification is shown: such agent, upon the request of an information about a train (identified by the train code), always answers back with the time of that train. In the example, we assume that the `timeTable/2` predicate is defined in the knowledge base of the agent; for lack of space, we have omitted the knowledge base. We also assume that the keyword `me` is used in the expectations to identify actions that must be executed by the agent.

Architecture The $\mathcal{S}$CIFF agent has been modeled on the basis of the Kowalsky-Sadri cycle for intelligent agents [6]. In particular, the phases of *observe* and *act* have been implemented directly in Java, by extending a JADE agent. The *think* phase instead has been realized using the $\mathcal{S}$CIFF proof procedure, that provides instructions on the basis of the happened events.

A schematic representation of the blocks composing the $\mathcal{S}$CIFF agent is shown in Fig. 2.2.

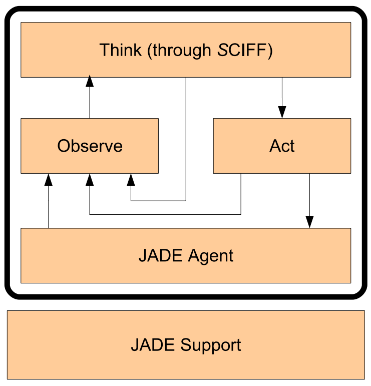

Fig. 2. Schematic diagram of the $\mathcal{S}$CIFF Agent

More in detail, the observation step consists on analyzing all the events that happened since the observation step of the previous cycle. All the events are registered in an *Event Buffer*, a repository that keeps trace of the new events. Three types of events are considered:

$i)$ Events corresponding to received messages.
$ii)$ Events corresponding to sent messages.
$iii)$ Events corresponding to internal state changes.

The “think” step consists of using the $\mathcal{S}$CIFF proof procedure to elaborate the happened events, and to generate a set of alternative expected behaviours. By applying a further selection function to all the alternatives, only one behaviour is selected, and it is passed on to the execution block.

Finally, the execution step consists on interpreting the expected behaviour generated by the $\mathcal{S}$CIFF proof procedure. The expectations about the behaviour of other agents will not be considered (it will be a task of the $\mathcal{S}$CIFF procedure to understand if such expectations have been satisfied or not, possibly providing further behaviours). Instead, expectations that regards actions to be done by the agent itself, will be interpreted as orders to be executed. If the expectation, e.g., is about sending a certain message, then the execution block will send such message. Then, for each action executed, the execution block generates a corresponding event and updates the buffer of the happened events:

in this way it is possible to specify Integrity Constraints in such a way that $\mathcal{S}$CIFF procedure can reason also about the actions made by the agent itself.

3 Conformance

In this section, we define the conformance of an agent to an interaction protocol, and we propose an operational procedure to check conformance.

3.1 Definitions

The intuitive meaning of the notion of conformance is the ability of an agent to interact with other agents as specified by an interaction protocol.

Definition 1. *Given an agent specification $\mathcal{P}_{ag} \equiv \langle KB_{ag}, \mathcal{E}_{ag}, \mathcal{IC}_{ag} \rangle$ and an interaction protocol specification $\mathcal{P}_{prot} \equiv \langle KB_{prot}, \mathcal{E}_{prot}, \mathcal{IC}_{prot} \rangle$, and given the abductive program $\langle KB_U, \mathcal{E}_U, \mathcal{IC}_U \rangle$,*

– $KB_U \triangleq KB_{prot} \cup KB_{ag}$
– $\mathcal{E}_U \triangleq \mathcal{E}_{prot} \cup \mathcal{E}_{ag}$
– $\mathcal{IC}_U \triangleq \mathcal{IC}_{prot} \cup \mathcal{IC}_{ag}$

a possible interaction *is a pair* $(\mathbf{HAP}, \mathbf{EXP})$ *where* $\mathbf{HAP}$ *is a set of events (atoms with functor* $\mathbf{H}$*), and* $\mathbf{EXP}$ *is a set of expectations (atoms with functors in* $\mathcal{E}_U$*) such that*

$$KB_U \cup \mathbf{HAP} \cup \mathbf{EXP} \models G \tag{1}$$
$$KB_U \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_U \tag{2}$$
$$\mathbf{E}_{ag}(m_x(ag, R, M)) \rightarrow \mathbf{H}(m_x(ag, R, M)) \tag{3}$$
$$\mathbf{E}_{prot}(m_x(A, B, M)) \wedge A \neq ag \rightarrow \mathbf{H}(m_x(A, B, M)) \tag{4}$$

Conditions (1) and (2) express the requirement, usual in abductive frameworks, that the set of generated abducibles entail both the goal and the integrity constraints.

Condition (3) expresses the fact that the agent will fulfill the expectations that it has about its own behaviour. Condition (4) formalizes the idea that the other agents will behave according to the protocol. These two conditions will be used as generative rules in the operational semantics.

G is the goal of the derivation.

Example 1. Given the specifications in Specs. 2.1 and 2.3, a possible interaction is given by

$$\mathbf{HAP} = \{\mathbf{H}(m_x(b, a, query\_ref(trainTime(10))), 1), \tag{5}$$
$$\mathbf{H}(m_x(a, b, inform(trainTime(12))), 2)\} \tag{6}$$
$$\mathbf{EXP} = \{\mathbf{E}_{ag}(m_x(a, b, inform(trainTime(I'))), T'), \tag{7}$$
$$\mathbf{E}_{prot}(m_x(a, b, inform(trainTime(I'))), T')\} \tag{8}$$

We can now provide a definition of *conformance* based on the possible interactions, selecting those that indeed respect the protocol and agent specifications.

Definition 2 ($\exists$-Conformance). *An agent specification $\mathcal{P}_{ag}$ is* existentially conformant *to a protocol specification $\mathcal{P}_{prot}$ if there exists at least one possible interaction $(\mathbf{HAP}^{py}, \mathbf{EXP}^{py})$ such that the following implications hold:*

$$\mathbf{E}_{ag}(m_x(a, A, M)) \rightarrow \mathbf{H}(m_x(a, A, M)) \quad (9)$$
$$\mathbf{E}_{prot}(m_x(S, A, M)) \rightarrow \mathbf{H}(m_x(S, A, M)) \quad (10)$$

Condition (9) requires that if a had an expectation about sending a message, the corresponding event should have happened. If this event is not present in $\mathbf{HAP}^{py}$, from Def. 1 we can infer that the event was unexpected from the protocol viewpoint. This situation corresponds to the case where a is determined to utter a message that is not envisaged by the protocol: in this case there is not conformance between a and the protocol.

Condition (10) requires that every message expected by the protocol indeed happens. Together with conditions (3) and (4), this means that all expectations involving the agent being tested are matched by a corresponding expectation of the protocol. This condition is false if there exists an expectation of the protocol without a corresponding expectation of the agent, i.e., either if ag may receive (during interactions specified by the protocol) a message that it is not able to understand, or if the aget failed to utter a message that the protocol expects.

Example 2. The agent specified in Spec. 2.3 is $\exists$-conformant to the protocol specified in Spec. 2.1, because the possible interaction in Eq. (8) also respects the conditions in Eqs. (9) and (10).

Definition 3 ($\forall$-Conformance). *An agent specification $\mathcal{P}_{ag}$ is* universally conformant *to a protocol specification $\mathcal{P}_{prot}$ if for all pairs $(\mathbf{HAP}^{py}, \mathbf{EXP}^{py})$ of Def. 1 the conditions (9-10) hold.*

When Definitions 2 and 3 hold together for an agent ag, we say that ag is *conformant*.

3.2 Operational test

The operational semantics is based on the two versions of the $\mathcal{S}$CIFF proof procedure developed in the SOCS project. The $\mathcal{S}$CIFF proof procedure was proven sound [14] and complete [15]; $\mathcal{S}$CIFF terminates [14] for acyclic programs. The $\mathcal{S}$CIFF proof procedure considers the $\mathbf{H}$ events as a predicate defined by a set of incoming atoms, and is devoted to generate expectations corresponding to a given history and to check that expectations indeed match with happened events. $\mathcal{S}$CIFF was developed to check the compliance of agents to protocols [3]. The $\mathcal{S}$CIFF proof procedure is based on a rewriting system transforming one node to another (or to others) as specified by rewriting steps called *transitions*.

The g-$\mathcal{S}$CIFF proof procedure, instead, considers $\mathbf{H}$ as an abducible predicate and aims at finding both the set of expectations and the history that fulfils some property requested as goal. g-$\mathcal{S}$CIFF has been used to prove properties of protocols, such as security protocols [17], and others. It contains the same rules in $\mathcal{S}$CIFF; in the version adopted in this paper, we also added as integrity constraints the rules (3) and (4).

$$\begin{tikzcd}
 & \boxed{\mathcal{S}\text{CIFF}} \arrow[dr] & & \\
\boxed{\text{g-}\mathcal{S}\text{CIFF}} \arrow[ur, "\mathbf{HAP}_1"] \arrow[r, "\mathbf{HAP}_2"] \arrow[dr, "\mathbf{HAP}_3"] & \boxed{\mathcal{S}\text{CIFF}} \arrow[r] & \wedge \arrow[r] & \text{conformance} \\
 & \boxed{\mathcal{S}\text{CIFF}} \arrow[ur] & &
\end{tikzcd}$$

Fig. 3. The conformance test.

In order to prove conformance, we apply the two proof procedures to the two phases implicitly defined in the previous section, as shown in Fig. 3.2. We decompose the proof of feeble conformance into a *generative* phase and a *test* phase. In the generative phase, we generate, by means of g-$\mathcal{S}$CIFF, all the possible histories. Of course, those histories need not be generated as ground histories (the set of ground histories can be infinite), but intensionally: the $\mathbf{H}$ events can contain variables, possibly with constraints à la Constraint Logic Programming [16].

In the test phase, we check with $\mathcal{S}$CIFF the compliance of the generated histories both with respect to the agent and the protocol specifications. If all the histories are conformant, the agent is feeble conformant to the protocol: at runtime, all the possible agent instances will behave accordingly to the protocol . Otherwise, if there exists at least one history that is not conformant, the agent is not conformant.

4 Related work

Our work is highly inspired by Baldoni et al. [18], who adopt a Multi-agent Systems point of view in defining a priori conformance in order to guarantee interoperability of Web Services whose interactions are specified by choreographies. As in [18], we give an interpretation of the a-priori conformance as a property that relates two formal specifications: the global one determining the interactions allowed by the interaction protocols and the local one related to the single agent policies. But, while in [18] a specification is represented as a finite state automation, we claim that the formalisms and technologies developed in the area of Computational Logic in providing a declarative representation of the social knowledge, could be applied also in the context of interaction protocol with respect to the conformance checking of agents. For example, a difference between our work and [18] can be found in the number of parties as they can manage only 2-party choreographies while we do not impose any such limit on the number of interacting agents.

In [19, 20], the authors apply a formalism based on computational logic to the a-priori conformance in the MAS field. Their formalism is similar to the one we propose, but they restrict their analysis to a particular type of protocols (named *shallow protocols*). Doing this, they address only 2-party interactions, without the possibility of expressing conditions over the content of the exchanged messages.

The use of abduction for verification was also explored in other work. Noteworthily, Russo et al. [21] use an abductive proof procedure for analysing event-based requirements specifications. Their method uses abduction for analysing the correctness of specifications, while our system is more focussed on the check of compliance/conformance of a set of agents.

5 Conclusions

In this paper, we introduced an agent platform based on the $\mathcal{S}$CIFF abductive framework. The main features of the framework are the possibility to define and check the conformance of an agent to a protocol, and the fact that the specification of an agent can be used, directly, as its implementation as a reactive system.

We described the architecture of the system, the language for the specification of the agent policies and the interaction protocols, a declarative definition of conformance of an agent to a protocol, and an operational procedure to verify conformance.

Acknowledgements

This work has been partially supported by the MIUR PRIN 2005 project *Specification and verification of agent interaction protocols*.

References

1. Guerin, F., Pitt, J.: Proving properties of open agent systems. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, Bologna, Italy, ACM Press (2002) 557–558
2. Merz, S.: Model checking: A tutorial overview. In Cassez, F., Jard, C., Rozoy, B., M.D.Ryan, eds.: Modeling and Verification of Parallel Processes. Number 2067 in Lecture Notes in Computer Science, Springer-Verlag (2001) 3–38
3. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The sciff abductive proof-procedure. In: Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005. Volume 3673 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2005) 135–147

4. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based tool. In Trappl, R., ed.: Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4), Vienna, Austria, Austrian Society for Cybernetic Studies (2004) 570–575 Extended version to appear in a special issue of Applied Artificial Intelligence, Taylor & Francis, 2006.
5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Security protocols verification in abductive logic programming: a case study. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: ESAW 2005 workshop notes, Kusadasi, Aydin, Turkey (2005)
6. Kowalski, R.A., Sadri, F.: From logic programming towards multi-agent systems. Annals of Mathematics and Artificial Intelligence **25** (1999) 391–419
7. : (Java Agent DEvelopment framework) Home Page: `http://sharon.cselt.it/projects/jade/`.
8. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation **2** (1993) 719–770
9. Kakas, A.C., Mancarella, P.: On the relation between Truth Maintenance and Abduction. In Fukumura, T., ed.: Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan, Ohmsha Ltd. (1990) 438–443
10. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
11. Denecker, M., Schreye, D.D.: SLDNFA: an abductive procedure for abductive logic programs. Journal of Logic Programming **34** (1998) 111–167
12. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The SOCS computational logic approach for the specification and verification of agent societies. In Priami, C., Quaglia, P., eds.: Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers. Volume 3267 of Lecture Notes in Artificial Intelligence. Springer-Verlag (2005) 324–339
13. : FIPA Query Interaction Protocol (2001) Published on August 10th, 2001, available for download from the FIPA website, `http://www.fipa.org`.
14. Gavanelli, M., Lamma, E., Mello, P.: Proof of properties of the SCIFF proof-procedure. (Technical Report CS-2005-01)
15. Gavanelli, M., Lamma, E., Mello, P.: Proof of completeness of the SCIFF proof-procedure. (Technical Report CS-2005-02)
16. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
17. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Security protocols verification in Abductive Logic Programming: A case study. In Pettorossi, A., Proietti, M., Senni, V., eds.: CILC 2005 - Convegno Italiano di Logica Computazionale, Università degli Studi di Roma Tor Vergata (2005)
18. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying the conformance of web services to global interaction protocols: A first step. In Bravetti, M., Kloul, L., Zavattaro, G., eds.: EPEW/WS-FM. Volume 3670 of Lecture Notes in Computer Science., Springer (2005)
19. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Protocol conformance for logic-based agents. In Gottlob, G., Walsh, T., eds.: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico (IJCAI-03), Morgan Kaufmann Publishers (2003)
20. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Logic-based agent communication protocols. In Dignum, F., ed.: Advances in Agent Communication. Volume 2922 of LNAI. Springer-Verlag (2004) 91–107

21. Russo, A., Miller, R., Nuseibeh, B., Kramer, J.: An abductive approach for analysing event-based requirements specifications. In Stuckey, P., ed.: Logic Programming, 18th International Conference, ICLP 2002. Volume 2401 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2002) 22–37