The SOCS Computational Logic Approach to the Specification and Verification of Agent Societies

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

[1] Dip. di Ingegneria - Università di Ferrara - Via Saragat, 1 - 44100 Ferrara, Italy.
{malberti|mgavanelli|elamma}@ing.unife.it
[2] DEIS - Università di Bologna - Viale Risorgimento, 2 - 40136 Bologna, Italy.
{fchesani|pmello|ptorroni}@deis.unibo.it

Abstract. This article summarises part of the work done during the first two years of the SOCS project, with respect to the task of modelling interaction amongst CL-based agents. It describes the SOCS social model: an agent interaction specification and verification framework equipped with a declarative and operational semantics, expressed in terms of abduction. The operational counterpart of the proposed framework has been implemented and integrated in SOCS-SI, a tool that can be used for on-the-fly verification of agent compliance with respect to specified protocols.

1 Introduction

Computees are Computational Logic-based entities interacting in the context of global and open computing systems [1]. They are abstractions of the entities that populate Global Computing (GC) environments [2]. These entities can form complex organizations, that we call *Societies Of ComputeeS* (SOCS, for short) [3]. The main objective of Global Computing, rephrased in terms of SOCS, is to provide a solid scientific foundation for the design of societies of computees, and to lay the groundwork for achieving effective principles for building and analyzing such systems. Between January 2002 and March 2004, the project developed a society formal model to satisfy the high-level objectives derived directly from the GC vision of an *open* and *changing* environment.

In this context, by "open" environment we mean, following Hewitt's work [4] about information systems and then Artikis et al.'s [5] about computational societies, an environment or society where the following properties hold:

(*i*) the behavior of members and their interactions are unpredictable (i.e., the evolution of the society is non-deterministic);
(*ii*) the internal architecture of each member is neither publicly known nor observable (thus, members may have heterogeneous architectures);

(iii) members of the society do not necessarily share common goals, desires or intentions (i.e., each member may conflict with others when trying to reach its own purposes).

This definition of openness is based on externally observable features within the society. It caters for heterogeneous and possibly non-cooperative members. Therefore, our model of society will not constrain the ways computees join or leave a society, it will emphasize the presence of heterogeneous computees in the same society, and it will assume that the internal structure of computees is not guaranteed to be observable, or their social behaviour predictable.

The SOCS social model specifies a social knowledge which interprets and gives a social meaning to the members' social behavior. It supports the notion of *social goal*, allowing for both goal-directed and non-goal-directed societies.

In our approach, we believe that the knowledge and technologies acquired so far in the area of Computational Logic provide a solid ground to build upon. At the society level, the role of Computational Logic is to provide both a declarative and an operational semantics to interactions. The advantages of such an approach are to be found:

(i) in the design and specification of societies of computees, based on a formalism which is declarative and easily understandable by the user;
(ii) in the possibility to detect undesirable behavior, through *on the fly* control of the system based on the computees' observable behavior (communication exchanges) and dynamic conformance check of such behaviour with the constraints posed by the society. Interestingly, as we will see, this can be achieved by exploiting a suitable proof procedure which is the operational counterpart of the mentioned formalism;
(iii) in the possibility to (formally) prove properties of protocols and societies.

Therefore, in our approach, we define the (semantics of) protocols and communication languages as logic-based integrity constraints over social events (e.g., communicative acts), called Social Integrity Constraints (ic_S) [6].

The ideal "correct" behaviour of a society is modelled as expectations about events. ic_S define the expectations stemming from a certain history of events and possibly a set of goals. Expectations and ic_S are the formalism used to define the "social semantics" of agent communication languages and interaction protocol: a semantics which is verifiable without having any knowledge about the agents' internals.

The syntax of ic_S and of the society in general are those of a suitably extended logic program. In fact, we define the "social knowledge" by assimilating it to abductive logic programs [7], and we define a notion of *expected social events*, by expressing them as abducible predicates, while using ic_S to constrain the "socially admissible" communication patterns of computees (i.e., those who match the expectations).

The society infrastructure is devoted to checking the compliance of the society members' behaviour, with respect to its expectations.

The compliance check is based on a proof-procedure called $\mathcal{S}$CIFF. $\mathcal{S}$CIFF, standing for "IFF, augmented with Constraints, for handling agent $\mathcal{S}$ocieties", is an extension of the well known IFF abductive logic programming proof-procedure, defined by Fung and Kowalski [8]. The $\mathcal{S}$CIFF extends the IFF in a number of directions: it provides both a richer syntax of abductive theories (programs and integrity constraints), it caters for interactive event assimilation, it supports fulfillment check and violation detection, and it embodies CLP-like constraints [9] in the ic_S.

The $\mathcal{S}$CIFF has been proven sound with respect to the declarative semantics of the society model, in its ALP interpretation [10].

The $\mathcal{S}$CIFF has been implemented and integrated into a Java-Prolog-CHR based tool, named SOCS-SI (SOCS Social Infrastructure [11]). This implementation can be used to verify that agents comply to a Social Integrity Constraints-based specification. The intended use of SOCS-SI is in combination with agent platforms, such as PROSOCS [12], for on-the-fly verification of compliance to protocols. In SOCS-SI, $\mathcal{S}$CIFF is part of an integrated environment, provided with interface modules to allow for such a combination, and with a graphical user interface to observe the actual behaviour of the society members with respect to their expected behaviour, and to detect possible deviations.

The main innovative contribution of the SOCS social model, under a Global Computing perspective, resides in the foundational aspects of the SOCS society model and in its direct link with its implementation, SOCS-SI.

The present work is meant to survey the activity undergone within the first two years of the SOCS project, with respect to the society infrastructure, and in the context of Global Computing. For a more detailed description of specific aspects, the reader can refer to the articles cited in the bibliography.

The paper is structured as follows. In Section 2, we present the formal model for societies, and its declarative semantics. Section 3 presents the $\mathcal{S}$CIFF proof procedure. Its implementation, and the overall tool SOCS-SI is described in Section 4. We discuss related work in Section 5, and we conclude and outline future work in Section 6.

2 SOCS social model

The SOCS model describes the knowledge about society in a declarative way. Such knowledge is mainly composed of two parts: a *static* part, defining the society organizational and "normative" elements, and a *dynamic* part, describing the "socially relevant" events, that have so far occurred. In most of our examples, events will be communicative acts, in line with most work done on software agents. However, this is not necessarily the case. Depending on the context in which this model is instantiated, socially relevant events could indeed be physical actions or transactions, such as electronic payments. In addition to these two categories of knowledge, information about social *goals* is also maintained.

In our model, the society is time by time aware of social events that dynamically happen in the social environment (*happened* events). The "normative elements" are encoded in what we call ic_S, as we will show below. Based on the available history of events, on its specification of ic_S and its goals, the society can define what the "expected social events" are and what the social events that are expected *not* to happen. The expected events, from a normative perspective, reflect the "ideal" behaviour of the computees. We call these events *social expectations*.

2.1 Representation of the society knowledge.

The knowledge in a society S is given by the following components:

- a (static) *Social Organization Knowledge Base*, denoted $SOKB$;
- a (static) set of *Social Integrity Constraints* (IC_S), denoted $\mathcal{IC}_S$; and
- a set of *Goals* of the society, denoted by $\mathcal{G}$.

In the following, the terms *Atom* and *Literal* have the usual Logic Programming meaning [13].

A society may evolve, as new events happen, giving rise to sequence of society instances, each one characterised by the previous knowledge components and, in addition, a (dynamic) *Social Environment Knowledge Base*, denoted $SEKB$.

In particular, $SEKB$ is composed of:

- *Happened events*: atoms indicated with functor **H**;
- *Expectations*: events that should (but might not) happen (atoms indicated with functor **E**), and events that should not (but might indeed) happen (atoms indicated with functor **EN**).

In our context, "happened" events are not all the events that have actually happened, but only those observable from the outside of agents, and relevant to the society. The collection of such events is the history, **HAP**, of a society instance. Events are represented as ground atoms

$$\mathbf{H}(\mathit{Event}[, \mathit{Time}]).$$

Expectations can be

$$\mathbf{E}(\mathit{Event}[, \mathit{Time}]) \qquad \mathbf{EN}(\mathit{Event}[, \mathit{Time}])$$

for, respectively, positive and negative expectations. **E** is a positive expectation about an event (the society expects the event to happen) and **EN** is a negative expectation, (the society expects the event not to happen[3]). Explicit negation ($\neg$) can be applied to expectations.

The arguments of expectation atoms can be non-ground terms. Intuitively, if an $\mathbf{E}(X)$ atom is in the set of expectations generated by the society, $\mathbf{E}(X) \in \mathbf{EXP}$, "**E**" indicates a wish about an event $\mathbf{H}(Y) \in \mathbf{HAP}$ which unifies with it:

[3] **EN** is a shorthand for **E** *not*.

X/Y. One such event will be enough to fulfill the expectation: thus, variables in an **E** atom are always existentially quantified.

For instance, in an auction context such as the one exemplified in [14], the following atom:

$$\mathbf{E}(\mathit{tell}(\mathit{Auctioneer}, \mathit{Bidders}, \mathit{openauction}(\mathit{Item}, \mathit{Dialogue})), T_{open})$$

could stand for an expectation about a communicative act *tell* made by a computee ($\mathit{Auctioneer}$), addressed to a (group of) computees ($\mathit{Bidders}$), with subject $\mathit{openauction}(\mathit{Item}, \mathit{Dialogue})$, at a time T_{open}.

The following scope rules and quantifications are adopted:

– variables in **E** atoms are always existentially quantified with scope the entire set of expectations
– the other variables, that occur only in **EN** atoms are universally quantified (the scope of universally quantified variables is not important, as $\forall X.p(X) \wedge q(X)$ is logically equivalent to $\forall X.p(X) \wedge \forall Y.q(Y)$).

The $SOKB$ defines structure and properties of the society, namely: goals, roles, and common knowledge and capabilities. $SOKB$ can change from time to time. However, this knowledge can be seen as *static* since it describes the organization of a society which changes more slowly than the way the $SEKB$ does. The SOKB is a logic program, consisting of clauses

$$\begin{aligned} \mathit{Clause} &::= \mathit{Atom} \leftarrow \mathit{Body} \\ \mathit{Body} &::= \mathit{ExtLiteral}\ [\ \wedge \mathit{ExtLiteral}\ ]^{\star} \\ \mathit{ExtLiteral} &::= \mathit{Literal} \mid \mathit{Expectation} \mid \mathit{Constraint} \\ \mathit{Expectation} &::= [\neg]\mathbf{E}(\mathit{Event}\ [,T]) \mid [\neg]\mathbf{EN}(\mathit{Event}\ [,T]) \end{aligned} \tag{1}$$

In a clause, the variables are quantified as follows:

– Universally, if they occur only in literals with functor **EN** (and possibly constraints), with scope the body;
– Otherwise universally, with scope the entire *Clause*.

We call *definite* the predicates for which there exists a definition; i.e., a predicate whose name occurs in at least the head of a clause.

The following is a sample clause:

$$\begin{aligned} &\mathit{sold}(\mathit{Item}) \leftarrow \\ &\qquad \mathbf{E}(\mathit{tell}(\mathit{Auctioneer}, \mathit{Bidders}, \mathit{openauction}(\mathit{Item}, \mathit{Dialogue})), T_{open}) \end{aligned} \tag{2}$$

It says that one way to fulfill the goal: "to have a certain *item* sold," could be to have some computee acting as an auctioneer and telling a set of possible bidders that an auction is open for the item.

The goal $\mathcal{G}$ of the society has the same syntax as the *Body* of a clause in the SOKB, and the variables are quantified accordingly.

As an example, we can consider a society with the goal of selling items. In order to sell an item, the society might expect some computee to embody the role of auctioneer. The goal of the society could be

$$\leftarrow sold(nail)$$

and the society might have, in the $SOKB$, a rule such as Eq. 2. Indeed, there could be more clauses specifying other ways of achieving the same goal, like expecting some computee to advertise a sale on some public channel, or generating an expectation about a request of that item by some potential customer agent. The protocol of the auction (i.e., the way the auctioneer and the bidders are expected to interact, and in particular, to interact in a socially meaningful way) can be then specified by means of ic_S.

Social Integrity Constraints are in the form of implications. We report here, for better readability, the characterizing part of their syntax:

$$\begin{aligned}
ic_S &::= \chi \rightarrow \phi \\
\chi &::= (HEvent|Expectation)\ [\wedge BodyLiteral]^\star \\
BodyLiteral &::= HEvent|Expectation|Literal|Constraint \\
\phi &::= HeadDisjunct\ [\ \vee HeadDisjunct\ ]^\star|\bot \\
HeadDisjunct &::= Expectation\ [\ \wedge (Expectation|Constraint)]^\star \\
Expectation &::= [\neg]\mathbf{E}(Event\ [,T])\ |\ [\neg]\mathbf{EN}(Event\ [,T]) \\
HEvent &::= [\neg]\mathbf{H}(Event\ [,T])
\end{aligned} \tag{3}$$

Given an ic_S $\chi \rightarrow \phi$, χ is called the *body* (or the *condition*) and ϕ is called the *head* (or the *conclusion*).

The rules of scope and quantification are as follows:

1. Any variable in an ic_S must occur in at least an *Event* or in an *Expectation*.
2. The variables that occur both in the body and in the head are quantified universally with scope the entire ic_S.
3. The variables that occur only in the head (and must occur in at least one *Expectation*, by rule 1)
   (a) if they occur in literals $\mathbf{E}$ or $\neg\mathbf{E}$ are quantified existentially and have as scope the disjunct they belong to;
   (b) otherwise they are quantified universally.
4. The variables that occur only in the body are quantified inside the body as follows:
   (a) if they occur only in conjunctions of $\neg\mathbf{H}$, $\mathbf{EN}$, $\neg\mathbf{EN}$ or *Constraint*s are quantified universally;
   (b) otherwise are quantified existentially.
5. The order of the quantifiers is indeed meaningful. In our syntax, the quantifier $\forall$ cannot be followed by $\exists$.

The following ic_S models one of the auction rules, stating that each time a bidding event happens, the auctioneer should have sent before an *openauction* event (to all bidders).

$$\begin{aligned}
&\mathbf{H}(current\_time, T_c), \mathbf{H}(tell(S, R, bid(Item, P), A_{number}), T_{bid}), T_{bid} < T_c \\
&\rightarrow \mathbf{E}(tell(R, Bidders, openauction(Item, A_{number})), T_{open}), T_{open} \leq T_c
\end{aligned}$$

2.2 ALP Interpretation of the Society

SOCS social model has been interpreted in terms of Abductive Logic Programming [7], and an abductive semantics has been proposed for it [15]. Abduction has been widely recognised as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge [16,17,18,19]. Incomplete knowledge is handled by labeling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. More formally, given a theory T and a formula G, the goal of abduction is to find a (possibly minimal) set of atoms Δ which together with T "entails" G, with respect to some notion of "entailment" that the language of T is equipped with.

An Abductive Logic Program (ALP, for short) [7] is a triple $\langle KB, \mathcal{A}, IC \rangle$ where KB is a logic program, (i.e., a set of clauses), $\mathcal{A}$ is a set of predicates that are not defined in KB and that are called *abducibles*, IC is a set of formulae called *Integrity Constraints*. An abductive explanation for a goal G is a set $\Delta \subseteq \mathcal{A}$ such that $KB \cup \Delta \models G$ and $KB \cup \Delta \models IC$, for some notion of entailment $\models$.

In our social model, the idea is to exploit abduction for defining expected behaviour of the computees inhabiting the society, and an abductive proof procedure such as the $\mathcal{S}$CIFF to dynamically *generate* the expectations and perform the *compliance check*. By "compliance check" we mean the procedure of checking that the ic_S are not violated, together with the function of detecting fulfillment and violation of expectations.

Before we give the declarative semantics of the SOCS social model, we formalise better the notions of *instance* of a society, and *closure* of an instance of a society.

Definition 1. *An* instance $\mathcal{S}_{\mathbf{HAP}}$ *of a society* $\mathcal{S}$ *is represented as an ALP, i.e., a triple* $\langle P, \mathcal{E}, \mathcal{IC}_S \rangle$ *where:*

- P *is the* $SOKB$ *of* $\mathcal{S}$ *together with the history of happened events* $\mathbf{HAP}$*;*
- $\mathcal{E}$ *is the set of* abducible predicates*, namely* $\mathbf{E}$, $\mathbf{EN}$, $\neg\mathbf{E}$, $\neg\mathbf{EN}$*;*
- $\mathcal{IC}_S$ *are the social integrity constraints of* $\mathcal{S}$*.*

The set $\mathbf{HAP}$ characterises the instance of a society, and represents the set of *observable* and *relevant* events for the society which have already happened. Note that we assume that such events are always ground.

A society instance is closed, when its characterizing history has been closed under the Closed World Assumption (CWA), i.e., when no further event might occur. In the following, we indicate a closed history by means of an overline: $\overline{\mathbf{HAP}}$.

2.3 Declarative semantics

We give semantics to a society instance by defining those sets of expectations which, together with the society's knowledge base and the happened events, imply an instance of the goal—if any—and *satisfy* the integrity constraints.

In our definition of integrity constraint satisfaction we will rely upon a notion of entailment in a three-valued logic, being it more general and capable of dealing with both open and closed society instances. Therefore, in the following, the symbol $\models$ has to be interpreted as a notion of entailment in a three-valued setting [20].

Throughout this section, as usual when defining declarative semantics, we always consider the ground version of social knowledge base and integrity constraints, we do not consider CLP-like constraints. Moreover, we omit the time argument in events and expectations.

We first introduce the concept of *$\mathcal{IC}_S$-consistent set of social expectations*[4]. Intuitively, given a society instance, an $\mathcal{IC}_S$-consistent set of social expectations is a set of expectations about social events that are compatible with P (i.e., the $SOKB$ and the set $\mathbf{HAP}$), and with $\mathcal{IC}_S$.

Definition 2. ($\mathcal{IC}_S$-consistency) *Given a (closed/open) society instance $\mathcal{S}_{\mathbf{HAP}}$, an $\mathcal{IC}_S$-*consistent *set of social expectations* $\mathbf{EXP}$ *is a set of expectations such that:*

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_S \tag{4}$$

(Notice that for closed instances $\mathbf{HAP}$ *has to be read* $\overline{\mathbf{HAP}}$*).*

In definition 2 (and in the following definitions 5, 6, 7 and 8), for open instances we refer to a three-valued completion where only the history of events has not been completed. Therefore, for open instances,

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_S$$

is a shorthand for:

$$Comp(SOKB \cup \mathbf{EXP}) \cup \mathbf{HAP} \cup CET \models \mathcal{IC}_S$$

where $Comp()$ is three-valued completion [20] and CET is Clark's equational theory.

For closed instances, instead,

$$SOKB \cup \overline{\mathbf{HAP}} \cup \mathbf{EXP} \models \mathcal{IC}_S$$

is a shorthand for:

$$Comp(SOKB \cup \mathbf{EXP} \cup \mathbf{HAP}) \cup CET \models \mathcal{IC}_S$$

since also the (closed) history of events needs to be completed.

$\mathcal{IC}_S$-consistent sets of expectations can be self-contradictory (e.g., both $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ may belong to a $\mathcal{IC}_S$-consistent set). In particular, among the $\mathcal{IC}_S$-consistent sets of expectations, we are interested in those which are also consistent from the viewpoint of our intended use of expectations, i.e., in relation to

[4] With abuse of terminology, we call this notion $\mathcal{IC}_S$-consistency though it corresponds to the theoremhood view rather than to the consistency view defined in [8].

the semantics of interactions. We will say that a set $\mathbf{EXP}$ is E-consistent if it does not contain both a positive and a negative expectation of the same event, and that it is $\neg$-consistent if it does not contain both an expectation and its explicit negation:

Definition 3. (E-consistency) *A set of social expectations* $\mathbf{EXP}$ *is* E-consistent *if and only if for each (ground) term p:*

$$\{\mathbf{E}(p), \mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}$$

Definition 4. ($\neg$-consistency) *A set of social expectations* $\mathbf{EXP}$ *is* $\neg$-consistent *if and only if for each (ground) term p:*

$$\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{EXP} \qquad \textit{and} \qquad \{\mathbf{EN}(p), \neg\mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}.$$

Given a closed (respectively, open) society instance, a set of expectations is called *closed* (resp. *open*) *admissible* if and only if it satisfies Definitions 2, 3 and 4, i.e., if it is $\mathcal{IC}_S$-, E- and $\neg$-consistent.

Definition 5. (Fulfillment) *Given a (closed/open) society instance* $\mathcal{S}_{\mathbf{HAP}}$*, a set of social expectations* $\mathbf{EXP}$ *is* fulfilled *if and only if for all (ground) terms p:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \to \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \to \neg\mathbf{H}(p)\} \not\models \bot \tag{5}$$

Notice that Definition 5 requires, for a closed instance of a society, that each positive expectation in $\mathbf{EXP}$ has a corresponding happened event in $\mathbf{HAP}$, and each negative expectation in $\mathbf{EXP}$ has no corresponding happened event. This requirement is weaker for open instances, where a set $\mathbf{EXP}$ is not fulfilled only when a negative expectation occurs in the set, but the corresponding event happened (i.e., the implication $\mathbf{EN}(p) \to \neg\mathbf{H}(p)$ is false).

Symmetrically, we define violation:

Definition 6. (Violation) *Given a (closed/open) society instance* $\mathcal{S}_{\mathbf{HAP}}$*, a set of social expectations* $\mathbf{EXP}$ *is* violated *if and only if there exists a (ground) term p such that:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \to \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \to \neg\mathbf{H}(p)\} \models \bot \tag{6}$$

Finally, we give the notion of goal achievability and achievement.

Definition 7. Goal achievability *Given an open instance of a society,* $\mathcal{S}_{\mathbf{HAP}}$*, and a ground goal G, we say that G is* achievable *(and we write* $\mathcal{S}_{\mathbf{HAP}} \models\!\!\approx_{\mathbf{EXP}} G$*) iff there exists an (open) admissible and fulfilled set of social expectations* $\mathbf{EXP}$*, such that:*

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models G \tag{7}$$

(which, as explained earlier, is a shorthand for $Comp(SOKB \cup \mathbf{EXP}) \cup \mathbf{HAP} \cup CET \models G$*).*

Definition 8. Goal achievement *Given a closed instance of a society, $\mathcal{S}_{\overline{\mathbf{HAP}}}$, and a ground goal G, we say that G is* achieved *(and we write $\mathcal{S}_{\overline{\mathbf{HAP}}} \vDash_{\mathbf{EXP}} G$) iff there exists a (closed) admissible and fulfilled set of social expectations* **EXP**, *such that:*

$$SOKB \cup \overline{\mathbf{HAP}} \cup \mathbf{EXP} \vDash G \tag{8}$$

(i.e., $Comp(SOKB \cup \mathbf{HAP} \cup \mathbf{EXP}) \cup CET \models G$).

3 Operational framework

The $\mathcal{S}$CIFF proof procedure is inspired to the IFF proof procedure [8]. As the IFF, it is based on a transition system, that rewrites a formula into another, until no more rewriting transitions can be applied (quiescence). Each of the transitions generates one or more children from a node. As an extension of the IFF, the $\mathcal{S}$CIFF also has to deal with (*i*) universally quantified variables in abducibles (*ii*) dynamically incoming events (*iii*) consistency, fulfillment and violations (*iv*) CLP-like constraints.

Each node of the proof procedure is represented by the tuple

$$T \equiv \langle R, CS, PSIC, \mathbf{PEXP}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle$$

where

- R is a conjunction (that replaces the *R*esolvent in SLD resolution); initially set to the goal G, the conjuncts can be atoms or disjunctions (of conjunctions of atoms)
- CS is the constraint store (as in Constraint Logic Programming [9])
- $PSIC$ is a set of implications, representing *partially solved social integrity constraints*
- **PEXP** is the set of pending expectations
- **HAP** is the history of happened events
- **FULF** is a set of fulfilled expectations
- **VIOL** is a set of violated expectations

Initial Node and Success A derivation D is a sequence of nodes $T_0 \rightarrow T_1 \rightarrow \ldots \rightarrow T_{n-1} \rightarrow T_n$. Given a goal G, a set of integrity constraints $\mathcal{IC}_S$, and an initial history $\mathbf{HAP}^i$ (that is typically empty) the first node is: $T_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ i.e., the conjunction R is initially the query ($R_0 = \{G\}$) and the partially solved integrity constraints $PSIC$ is the whole set of social integrity constraints ($PSIC_0 = \mathcal{IC}_S$). The other nodes $T_j, j > 0$, are obtained by applying one of the transitions of the proof procedure, until no further transition can be applied (we call this last condition *quiescence*).

Let us now give the definition of successful derivation, both in the case of an open society instance (where new events may be added to the history further on) and of a closed society instance.

Definition 9. *Given an initial history* $\mathbf{HAP}^i$ *that evolves toward a final history* $\mathbf{HAP}^f$ *(with* $\mathbf{HAP}^f \supseteq \mathbf{HAP}^i$*), and an open society instance* $\mathcal{S}_{\mathbf{HAP}^i}$*, there exists an* open successful derivation *for a goal* G *iff the proof tree with root node*

$$\langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$$

has at least one leaf node

$$\langle \emptyset, CS, PSIC, \mathbf{PEXP}, \mathbf{HAP}^f, \mathbf{FULF}, \emptyset \rangle$$

where CS *is consistent (i.e., there exists a ground variable assignment such that all the constraints are satisfied).*

Analogously, there exists a closed successful derivation *iff the proof tree has at least one leaf node*

$$\langle \emptyset, CS, PSIC, \mathbf{PEXP}, \overline{\mathbf{HAP}^f}, \mathbf{FULF}, \emptyset \rangle$$

where CS *is consistent, and* $\mathbf{PEXP}$ *contains only negative literals* $\neg\mathbf{E}$ *and* $\neg\mathbf{EN}$*.*

From each non-failure leaf node N, answers can be extracted in a very similar way to the IFF proof procedure. Answers of the $\mathcal{S}$CIFF proof procedure, called *expectation answers*, are composed of an answer substitution and a set of abduced expectations. First, an answer substitution σ' is computed such that (i) σ' replaces all variables in N that are not universally quantified by a ground term (ii) σ' satisfies all the constraints in the store CS_N. Notice that, by definition 9, there must be a grounding of the variables satisfying all the constraints. In other words, we assume that the solver is (theory) complete [21], i.e., for each set of constraints c, the solver always returns *true* or *false*, and never *unknown*. Otherwise, if the solver is incomplete, σ' may not exist. The non-existence of σ' is discovered during the answer extraction phase. In such a case, the node N will be marked as a failure node, and another leaf node can be selected (if it exists).

Definition 10. *Given a non-failure node* N*, let* σ' *be the answer substitution extracted from* N*.*

Let $\sigma = \sigma'|_{vars(G)}$ *be the restriction of* σ' *to the variables occurring in the initial goal* G*. Let* $\mathbf{EXP}_N = (\mathbf{FULF}_N \cup \mathbf{PEXP}_N)\sigma'$*. The pair* $(\mathbf{EXP}_N, \sigma)$ *is the* expectation answer *obtained from the node* N*.*

3.1 Transitions

The transitions are based on those of the IFF proof procedure, augmented with those of CLP [9], and with specific transitions accommodating the concepts of fulfillment, dynamically growing history and consistency of the set of expectations with respect to the given definitions (Definitions 2, 3, and 4).

Due to lack of space, we do not list all the transitions, but we informally describe the main ones, and we give the formal definition of one (*Violation* **EN**), in order to give the taste of how the proof procedure works. The full list of transition can be found in [10].

IFF-like transitions The $\mathcal{S}$CIFF proof procedure inherits the transitions of the IFF proof procedure. The IFF proof procedure starts with a formula (that replaces the concept of *resolvent* in logic programming) built as a conjunction of the initial query and the ICs. Then it repeatedly applies one of the following *inference rules*:

unfolding replaces resolution: given a node with a definite atom, it replaces it with one of its definitions;
propagation propagates ic_S: given a node containing $A \wedge B \rightarrow C$ and an atom A' that unifies with A, it replaces the implication with $(A = A') \wedge B \rightarrow C$;
splitting distributes conjunctions and disjunctions, making the final formula in a sum-of-products form;
case analysis if the body of an ic_S contains $A = A'$, case analysis nondeterministically tries $A = A'$ or $A \neq A'$,
factoring tries to reuse a previously made hypothesis;
rewrite rules for equality use the inferences in the Clark Equality Theory;
logical simplifications try to simplify a formula through equivalences like $[A \wedge false] \leftrightarrow false$, $[true \rightarrow A] \leftrightarrow A$, etc.

Thanks to these inference rules, each node is always translated into a (disjunction of) conjunctions of atoms and implications; e.g., it can look like:

$$\begin{gathered}(A_1 \wedge A_2 \wedge [B_1 \wedge B_2 \rightarrow A_3] \wedge [B_3 \wedge B_4 \rightarrow A_4]) \\ \vee\ (A_i \wedge A_j \wedge A_k \wedge [B_y \rightarrow A_z] \wedge [B_5 \rightarrow false])\end{gathered}$$

the atoms have a similar meaning to those in the resolvent in LP, while the implications are (partially-propagated) integrity constraints.

Given a formula, it is always clear the quantification of the variables by the following rules:

- if a variable is in the initial query, then it is *free*;
- *else* if it appears in an atom, it is existentially quantified;
- *else* (it appears only in implications) it is universally quantified.

CLP-like The $\mathcal{S}$CIFF proof procedure also deals with constraints. It contains the CLP transitions [9] of *Constrain* (moves a constraint from R to the constraint store CS), *Infer* (infers new constraints given the current state of CS) and *Consistent* (checks if the constraint store is satisfiable). The solver has been extended to deal with unification and disunification of existentially and universally quantified atoms.

Dynamically incoming events We assume to have an external set of events that happen in the society; the events in this external set are inserted in the history **HAP** by a transition *Happening*. Other transitions deal with non-happening of events and closure of the history.

Consistency, Fulfillment and Violation In order to rule out nodes that are either inconsistent with respect to the declarative semantics or contain violations, we defined transitions that nondeterministically try to unify/disunify the terms in atoms. For instance, in order to detect a violation of **EN** atoms, we need to check if one happened event unifies with it. We have the transition:

Violation **EN** Given a node N_k as follows:

– $\mathbf{PEXP}_k = \mathbf{PEXP}' \cup \{\mathbf{EN}(E_1)\}$
– $\mathbf{HAP}_k = \mathbf{HAP}' \cup \{\mathbf{H}(E_2)\}$

violation **EN** produces two nodes N^1_{k+1} and N^2_{k+1}, where

N^1_{k+1}	N^2_{k+1}
$\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k \cup \{\mathbf{EN}(E_1)\}$	$\mathbf{VIOL}_{k+1} = \mathbf{VIOL}_k$
$CS_{k+1} = CS_k \cup \{E_1 = E_2\}$	$CS_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Example 1. Suppose that $\mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}$ and $\exists X \forall Y \mathbf{PEXP}_k = \{\mathbf{EN}(p(X,Y))\}$. *Violation* **EN** will produce the two following nodes:

$$\begin{tikzcd}
& \begin{array}{c}\exists X\forall Y \mathbf{PEXP}_k = \{\mathbf{EN}(p(X,Y))\}\\ \mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}\end{array} \arrow[dl, dash] \arrow[dr, dash] & \\
\begin{array}{c} X = 1 \wedge Y = 2\\ \mathbf{VIOL}_{k+1} = \{\mathbf{EN}(p(1,2))\}\end{array} & & X \neq 1 \vee Y \neq 2 \arrow[d, dash] \\
& & X \neq 1
\end{tikzcd}$$

where the last simplification in the right branch is due to the rules of the constraint solver [10].

3.2 Sample Derivation

Let us consider a simple protocol: if a computee is asked for some information, it should either provide the information or refuse, but not both.[5] The protocol definition is given by means of the following Social Integrity Constraints:

IC_1: $\mathbf{H}(\mathit{tell}(A, B, \mathit{query\text{-}ref}(\mathit{Info}), D), T) \Rightarrow$
$\mathbf{E}(\mathit{tell}(B, A, \mathit{inform}(\mathit{Info}, \mathit{Answer}), D), T_1), T_1 < T + 10 \vee$
$\mathbf{E}(\mathit{tell}(B, A, \mathit{refuse}(\mathit{Info}), D), T_1), T_1 < T + 10$

IC_2: $\mathbf{H}(\mathit{tell}(A, B, \mathit{inform}(\mathit{Info}, \mathit{Answer}), D), T) \Rightarrow$
$\mathbf{EN}(\mathit{tell}(A, B, \mathit{refuse}(\mathit{Info}), D), T_1), T_1 > T$

IC_3: $\mathbf{H}(\mathit{tell}(A, B, \mathit{refuse}(\mathit{Info}), D), T) \Rightarrow$
$\mathbf{EN}(\mathit{tell}(A, B, \mathit{inform}(\mathit{Info}, \mathit{Answer}), D), T_1), T_1 > T$

and let us suppose that the history evolves from an empty history to a final history $\mathbf{HAP}^f$ composed of only two events:

$\mathbf{H}(\mathit{tell}(\mathit{yves}, \mathit{david}, \mathit{query\text{-}ref}(\mathit{train\_info}), d_1), 1)$
$\mathbf{H}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \text{“}\textit{departs(sv,rm,10:15)}\text{”}), d_1), 2)$

[5] This protocol is inspired to the FIPA query-ref interaction protocol [22].

The first node of the derivation tree is $N_0 \equiv \langle \emptyset, \emptyset, \mathcal{IC}_S, \emptyset, \emptyset, \emptyset, \emptyset \rangle$. The only applicable transition is *Happening* with one of the events in the external set of happened events; in this example, we will consider the events in chronological order:

$$N_1 \equiv \langle \emptyset, \emptyset, PSIC, \emptyset, \{\mathbf{H}(\mathit{tell}(\mathit{yves}, \mathit{david}, \mathit{query\text{-}ref}(\mathit{train\_info}), d_1), 1)\}, \emptyset, \emptyset \rangle.$$

Now transition *Propagation* is applicable to IC_1.

$$\begin{aligned} PSIC_2 = \{\ & IC_1, IC_2, IC3, \\ & A' = \mathit{yves}, B' = \mathit{david}, \mathit{Info}' = \mathit{train\_info}, D' = d_1, T' = 1 \\ & \rightarrow \mathbf{E}(\mathit{tell}(B', A', \mathit{inform}(\mathit{Info}', \mathit{Answer}'), D'), T_1'),\ T_1' < T' + 10 \\ & \vee\ \ \mathbf{E}(\mathit{tell}(B', A', \mathit{refuse}(\mathit{Info}'), D'), T_1'), \qquad\qquad T_1' < T' + 10 \\ & \} \end{aligned}$$

Each of the equalities in the body of the implication is dealt with by *case analysis*. Concerning $A' = \mathit{yves}$, case analysis generates two nodes: in the first $A' = \mathit{yves}$ and in the second $A' \neq \mathit{yves}$ is put in the constraint store. Since A' is universally quantified, the constraint $A' = \mathit{yves}$ succeeds when applying transition *Consistent*, and $A' \neq \mathit{yves}$ fails.

$$\begin{tikzcd}[column sep=small] & \bullet \arrow[dl, dash] \arrow[dr, dash] & \\ \begin{array}{c}\textbf{Case Analysis}\\ CS_3 = \{A' = \mathit{yves}\}\end{array} \arrow[d, dash] & & \begin{array}{c}\textbf{Case Analysis}\\ CS_3 = \{A' \neq \mathit{yves}\}\end{array} \arrow[d, dash] \\ \begin{array}{c}\textbf{Infer+Consistent}\\ PSIC_4 = \{IC_1, IC_2, IC_3, IC_1'\}\end{array} & & \begin{array}{c}\textbf{Consistent}\\ \text{fail}\end{array} \end{tikzcd}$$

where

$$IC_1' = \left\{ \begin{array}{l} B' = \mathit{david}, \mathit{Info}' = \mathit{train\_info}, D' = d_1, T' = 1 \\ \qquad \rightarrow \mathbf{E}(\mathit{tell}(B', \mathit{yves}, \mathit{inform}(\mathit{Info}', \mathit{Answer}'), D'), T_1'), T_1' < T' + 10 \\ \qquad \vee \mathbf{E}(\mathit{tell}(B', \mathit{yves}, \mathit{refuse}(\mathit{Info}'), D'), T_1'), T_1' < T' + 10 \end{array} \right.$$

After applying case analysis for each equality in the body, and the successive constraint solving step, we have only one non-failure node:

$$N_{10} = \langle \emptyset, \emptyset, PSIC_{10}, \emptyset, \mathbf{HAP}_{10}, \emptyset, \emptyset \rangle$$

$$\begin{aligned} PSIC_{10} = \{ & IC_1, IC_2, IC_3, \\ & \mathit{true} \rightarrow \mathbf{E}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \mathit{Answer}'), d_1), T_1'), \\ & \qquad T_1' < 1 + 10 \\ & \qquad \vee\ \ \mathbf{E}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{refuse}(\mathit{train\_info}), d_1), T_1'), T_1' < 1 + 10\} \\ \mathbf{HAP}_{10} = \{ & \mathbf{H}(\mathit{tell}(\mathit{yves}, \mathit{david}, \mathit{query\text{-}ref}(\mathit{train\_info}), d_1), 1)\} \end{aligned}$$

We apply *Logical Equivalence* to the implication with *true* antecedent. Then, since element R of the produced node (N_{11} not shown here) contains a disjunction, *splitting* can be applied, and its application generates two nodes. Let us consider the first node N'_{14}, having:

$$\begin{aligned} R'_{14} &= \emptyset \\ \mathbf{PEXP}'_{14} &= \{\mathbf{E}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \mathit{Answer}'), d_1), T'_1)\} \\ CS'_{14} &= \{T'_1 < 1 + 10\} \end{aligned}$$

The declarative reading of this node is:
$\exists \mathit{Answer}', \exists T'_1.\ \ T'_1 < 1 + 10$
$\wedge\ \mathbf{E}(\mathit{tell}(\mathit{david},\ \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \mathit{Answer}'), d_1), T'_1)$.

Suppose that now *happening* transition is applied with the second event in the external set of happened events[6].

$$\begin{aligned} \mathbf{HAP}_{15} = \{&\mathbf{H}(\mathit{tell}(\mathit{yves}, \mathit{david}, \mathit{query\text{-}ref}(\mathit{train\_info}), d_1), 1), \\ &\mathbf{H}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \text{“}\mathit{departs(sv,rm,10{:}15))}\text{”}, d_1), 2).\} \end{aligned}$$

We can now apply transition *fulfillment* **E** with the event $\mathbf{H}(\mathit{tell}(\mathit{david},\ \mathit{yves},\ \mathit{inform}\ldots))$ in the history. The transition opens two alternative nodes, N'_{16} and N''_{16}: either the event in the expectation unifies with the event in the history, and becomes fulfilled, or it does not unify and remains pending.

$$\begin{aligned} CS'_{16} &= \{\mathit{Answer}' = \text{“}\mathit{departs(sv,rm,10{:}15)}\text{”} \wedge T'_1 = 2 \\ &\quad \wedge T'_1 < 1 + 10\} \\ \mathbf{FULF}'_{16} &= \{\mathbf{E}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \mathit{Answer}'), d_1), T'_1)\} \\ \mathbf{PEXP}'_{16} &= \emptyset \\ CS''_{16} &= \{(\mathit{Answer}' \neq \text{“}\mathit{departs(sv,rm,10{:}15)}\text{”} \vee T'_1 \neq 2) \\ &\quad \wedge T'_1 < 1 + 10\} \\ \mathbf{FULF}''_{16} &= \emptyset \\ \mathbf{PEXP}''_{16} &= \{\mathbf{E}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}(\mathit{train\_info}, \mathit{Answer}'), d_1), T'_1)\} \end{aligned}$$

The second node can be fulfilled if the history is still open, as other events may happen matching the pending expectation. If the history gets closed, the pending expectation will become violated, so the second will be a violation node. This does not mean that the proof is in a global violation. As in SLD resolution a global failure is obtained only if all the leaves of the proof tree are failure nodes, in the same way in $\mathcal{S}$CIFF we have a global violation only if all the leaves contain violations (i.e., in all alternative branches, $\mathbf{VIOL} \neq \emptyset$). This is not the case in this example, since in the first node, N'_{16}, the expectations are fulfilled).

Other transitions are applicable to this node; we do not continue the example because their application is very similar to the ones already presented. For example, transition *Propagation* will be applied to IC_2 and to the event $\mathbf{H}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{inform}\ldots))$ in the history, thus providing a new expectation $\mathbf{EN}(\mathit{tell}(\mathit{david}, \mathit{yves}, \mathit{refuse}(\mathit{train\_info}), d_1), T'_1)$, $T'_1 > 2$.

[6] Of course, the *happening* transition was applicable also to the previous nodes. We are giving here a sample derivation, but others may be possible.

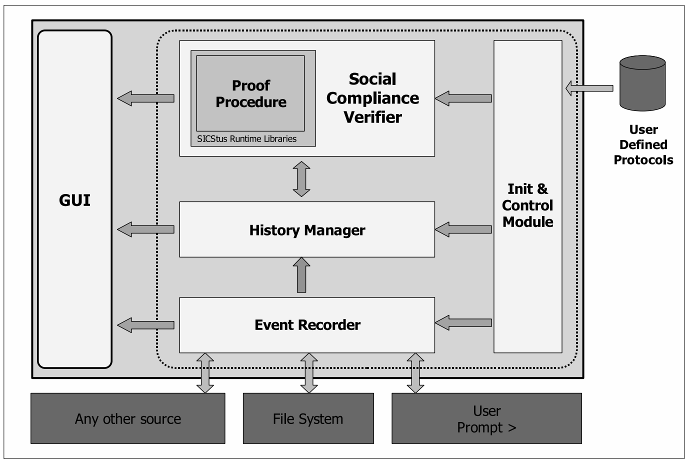


Fig. 1. Overview of the *SOCS-SI* architecture

4 Implementation

In this section, we describe the implementation of SOCS-SI, the tool for compliance verification of agent interaction. The tool is composed of an implementation of the $\mathcal{S}$CIFF proof-procedure specified in the previous section, interfaced to a graphical user interface and to a component for the observation of agent interaction.

The SOCS-SI software application is composed of a set of modules. All the components except one ($\mathcal{S}$CIFF) are implemented in the Java language.

The core of SOCS-SI is composed of three main modules (see Fig. 1), namely:

- *Event Recorder*: fetches events from different sources and stores them inside the *History Manager*.
- *History Manager*: receives events from the *Event Recorder* and composes them into an "event history".
- *Social Compliance Verifier*: fetches events from the *History Manager* and passes them to the proof-procedure in order to check the compliance of the history to the specification.

Computees communicate by exchanging messages, which are then translated into **H** events. The *Event Recorder* fetches events and records them into the *History Manager*, where they become available to the proof-procedure (see Sect. 4.1). As soon as the proof-procedure is ready to process a new event, it fetches one from the *History Manager*. The event is processed and the results of the computation are returned to the GUI. The proof-procedure then continues its computation by fetching another event if there is any available, otherwise it suspends, waiting for new events.

A fourth module, named *Init&Control Module* provides for initialization of all the components in the proper order. It receives as initial input a set of protocols defined by the user, which will be used by the proof-procedure in order to check the compliance of agents to the specification.

4.1 Implementation of $\mathcal{S}$CIFF

The $\mathcal{S}$CIFF proof procedure has been implemented in SICStus Prolog [23], exploiting its constraint libraries and, in particular, the *Constraint Handling Rules* (CHR) library [24].

The data structures representing the proof tree nodes are represented as follows:

- R is represented by the Prolog resolvent;
- CS is the CLP (CLPFD, CLPB) constraint store;
- $PSIC$, **EXP**, **HAP**, **FULF**, **VIOL** are represented as *CHR* constraints.

Attributes [25] are used to represent the quantification (existential or universal) of variables in expectations; an *ad-hoc CHR* constraint (`reif_unify/3`) implements reified unification (i.e., both the constraints $=$ and $\neq$) between variables and terms.

Thanks to the representation of most data structures as *CHR* constraints, the transitions (such as propagation, happening, fulfillment/violation) that modify those data structures have been implemented by exploiting the *CHR* computational model.

For instance, the following rule immplements the check for **E**-consistence:

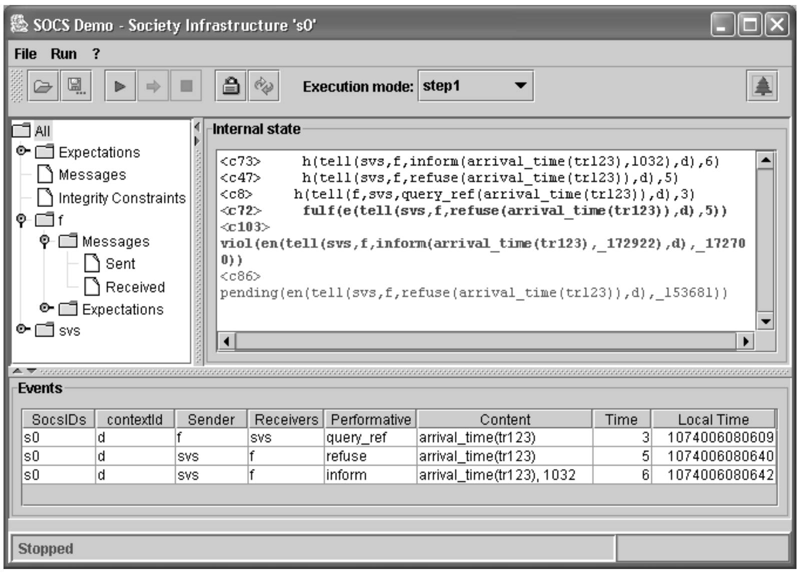
```
e_consistency @
        e(EEvent,ETime),
        en(ENEvent,ENTime)
        ==>
        reif_unify(p(EEvent,ETime),p(ENEvent,ENTime),0).
```

This is a *propagation* rule, i.e., a rule that adds a constraint to the *CHR* store whenever a combination of constraints is present in the store. The name of the rule is `e_consistency`. `e(EEvent,ETime)` and `en(ENEvent,ENTime)` are the two *CHR* constraints representing the expectations $\mathbf{E}(EEvent, ETime)$ and $\mathbf{EN}(ENEvent, ENTime)$, respectively. Whenever these two constraints are in the *CHR* store, the dis-unification constraint

```
reif_unify(p(EEvent,ETime),p(ENEvent,ENTime),0)
```

is added to the store to impose that the arguments of the positive and the negative expectations do not unify, as required by **E**-consistency (see Def. 3).

The CLP transitions, instead, are delegated to the CLP solvers available in SICStus Prolog: we have used CLPFD for finite domains and CLPB for binary domains variables, but in principle it would be possible to use any CLP library based on SICStus.



```
<c73>     h(tell(svs,f,inform(arrival_time(trl23),1032),d),6)
<c47>     h(tell(svs,f,refuse(arrival_time(trl23)),d),5)
<c8>     h(tell(f,svs,query_ref(arrival_time(trl23)),d),3)
<c72>     fulf(e(tell(svs,f,refuse(arrival_time(tr123)),d),5))
<c103>
viol(en(tell(svs,f,inform(arrival_time(tr123),_172922),d),_17270
0))
<c86>
pending(en(tell(svs,f,refuse(arrival_time(trl23)),d),_153681))
```

SocsIDs	contextId	Sender	Receivers	Performative	Content	Time	Local Time
s0	d	f	svs	query_ref	arrival_time(tr123)	3	1074006080609
s0	d	svs	f	refuse	arrival_time(tr123)	5	1074006080640
s0	d	svs	f	inform	arrival_time(tr123), 1032	6	1074006080642

Fig. 2. A screenshot of the application

The $\mathcal{S}$CIFF proof tree is searched with a depth-first strategy, so to exploit the Prolog stack for backtracking. The success of the proof procedure (see Def. 8) is mapped onto a successful Prolog derivation.

4.2 The Graphical User Interface

The Graphical User Interface is implemented by using the Swing graphic library, and implements the Model-View-Control programming pattern. The main window is composed of three areas (or sub-windows), and of a button bar that contains the controls (see Fig. 2).

The bottom area contains the list of all the messages received by SOCS-SI. The left pane contains the list of agents known by the society, i.e., agents that have performed at least one communicative action. The central pane contains the results of the computation, returned by the proof-procedure. These results are expressed in terms of society expectations about the future behavior of agents, and also in terms of fulfilled expectations and violations of social rules. By selecting an agent from the left pane, it is possible to restrict the information shown on the larger pane to be only that relevant to that particular agent. Among other features, it is possible to execute step-by-step the application, so that it elaborates one message at a time and then waits for a user acknowledge (similarly to the debug interface of modern compilers).

5 Related Work

The main result of the first two years of the SOCS project, with respect to *societies* od computees, is the definition of a social framework. In doing this, we have provided (*i*) a declarative representation of the social knowledge, (*ii*) a *logic formalism* based on social expectations and $\mathcal{IC}_S$, for *specifying* social rules and easily verifiable protocols, and for defining the semantics of communicative acts in an open system scenario, (*iii*) a *proof-procedure* proven correct with respect to the declarative representation of the social knowledge, and (*iv*) a prototypical *implementation* of the social framework which can be used to test the framework with a number of scenarios, protocols, and communication languages.

Our work relates to several aspects of Multi-Agent Systems research, in terms of social and interaction models, operational frameworks, and implemented systems, and to work done in Computational Logics, specifically in extensions of logic programming. Space limitations prevent us from thoroughly discussing here the SOCS social framework in relationship with other conspicuous work done on all these areas. We will only give an overview of some related work, and give a reference to the relevant project deliverables for further discussions.

$\mathcal{IC}_S$ represent in a way social norms. Several researchers have studied the concepts of norms, commitments and social relations in the context of Multi-Agent Systems [26]. Furthermore, a lot of research has been devoted in proposing architectures for developing agents with social awareness (see, for instance [27]). Our approach can be conceived as complementary to these efforts, since instead of proposing a specific architecture for designing computees, our work is mainly focused on the definition of a society infrastructure based on Computational Logic for regulating and improving robustness of interaction in an open environment, where the internal architecture of the computees might be unknown.

Our work is very related, as far as objectives and methodology, to the work on computational societies presented and developed in the context of the ALFEBIITE project [28]. We have in fact the same understanding of openness, as we pointed out in the introduction. In turn, our work is especially oriented to computational aspects, and it was developed with the purpose of providing a computational framework that can be directly used for automatic verification of properties such as compliance.

Most formal approaches to protocol modelling specify protocols as legal sequences of actions [29, pp.19–22]. In this way, protocols can be over-constrained, and this affects autonomy, heterogeneity, and ability to exploit opportunities and exceptions [30]. Moreover, the mentalistic approach to protocol definition has been much criticized mainly because its assumptions regarding agents' internals are not realistic in open societies of heterogeneous agents [31]. Therefore we advocated a *social* approach, where the semantics of interactions is defined in terms of the effects of the computees interactions on the society. Following this approach, even if the computees mental state cannot be accessed, it is possible to verify whether communicating computees in a society comply to some social laws and norms which regulate the interactions. Another expressive advantage of our framework is that it can express with the same formalism both protocols and social semantics of communicative acts.

In our social approach we drew inspiration from work done by Yolum and Singh [30] where a social semantics of agent interaction protocols is exemplified by using a commitment-based approach, and by Fornara and Colombetti [32,33,34], especially as it concerns the semantics of communication languages [29, pp. 37–41]. In particular, the latter approach is similar to ours in that it specifies the semantics of actions in terms of their social effects, and presupposes a social framework (which is called *institution* in [33]) for assigning agents with roles, verifying their social behavior and, possibly, recovering from violation conditions. There are, however, some significant differences with [33], mainly originating from the different paradigm we have chosen to express semantics (*logic-based* instead of *object-oriented*), as it is shown in [29, p. 71].

Yolum and Singh also propose an interesting way of linking together communicative acts and protocol specifications using the idea of social semantics in [30], where an agent can find a communication path leaving no pending commitments by exploiting its reasoning/planning capabilities. Our approach rather aims at ensuring protocol compliance regardless of computees' reasoning capabilities. In fact, $\mathcal{IC}_S$ are designed to explicit constraints between communicative acts. However, equipping the communication model of single computees with sufficient knowledge to reason about social expectations is certainly an interesting option. This topic is discussed in D4 [35].

Finally, there exist other approaches based on Deontic Logic to formally defining norms and dealing with their possible violations. An example of it is work by Dignum et al. [36], as discussed in [29, pp. 72]. Deontic operators have been used not only at the social level, but also at the agent level. Notably, in IMPACT [37,38], agent programs may be used to specify what an agent is obliged to do, what an agent may do, and what an agent cannot do on the basis of deontic operators of Permission, Obligation and Prohibition (whose semantics does not rely on a Deontic Logic semantics). In this respect, IMPACT and our work have similarities even if their purpose and expressivity are different. The main difference is that the goal of agent programs in IMPACT is to express and determine by its application the behavior of a single agent, whereas our goal is to express rules of interaction, that instead cannot really determine and constrain the behavior of the single computees participating to the interaction protocols, since computees are autonomous.

Our work is not only directly related to social aspects of MAS, but also to extensions of Logic Programming for MAS. In particular, the syntax of ic_S is strictly related to that of integrity constraints in the IFF proof-procedure [8]. In [29, pp. 92–94] work on $\mathcal{IC}_S$ is discussed with that done by Fung and Kowalski, with a focus on some syntactic aspects of the integrity constraints handled by the IFF proof-procedure. Briefly, the $\mathcal{S}$CIFF can be considered as an extension of the IFF proof procedure that also:

– abduces atoms with variables universally quantified;

– deals with CLP constraints, also imposed as quantifier restrictions on universally quantified variables;
– is more dynamic, in fact new events may arrive, and the proof procedure dynamically takes them into consideration in the knowledge base;
– has the new concepts, related to on-line verification, of *fulfillment* and *violation.*

The IFF is not the only abductive logic programming proof-procedure in literature. Various abductive proof procedures have been proposed in the past. In [39, p. 173] other procedures are discussed in relationship with our choice based on the IFF.

Other authors also proposed using abduction for verification. Noteworthily, Russo et al. [40] use an abductive proof procedure for analyzing event-based requirements specifications. In their approach, the system has a declarative specification given through the Event Calculus [41] axioms, and the goal is proving that some invariant I is true in all cases. This method uses abduction for analyzing the correctness of specifications, while our system is more focussed on the on-line check of compliance of a set of agents.

We will conclude this section by quickly mentioning two other implementations of social frameworks. The cited above ALFEBIITE project delivers a tool (*Society Visualiser*) to demonstrate animations of protocol runs in such systems [5]. The Society Visualiser's main purpose is to explicitly represent the institutional power of the members and the concept of valid action. As we stressed earlier on, our work is not based on any deontic infrastructure. For this reason, the SOCS social framework could be used in a different, possibly broader spectrum of application domains, ranging from intelligent agents to reactive systems.

ISLANDER [42] is a tool for the specification and verification of interaction in complex social infrastructures, such as electronic institutions. ISLANDER allows to analyse situations, called scenes, and visualise liveness or safeness properties in some specific settings. The kind of verification involved is static and is used to help designing institutions. Although our framework could also be used at design time, its main intended use is for on-the-fly verification of heterogeneous and open systems.

6 Conclusion and Future Work

In this work, we reported on a Computational Logic-based framework for modelling societies of computees and their interactions. We presented both published and original work done in the first two years of the SOCS project about modelling interactions among agents/computees. In [29, pp. 9–10] we give a list of pointers to publications where some of the results presented here can be seen in more detail.

One of the main objectives of SOCS was to deliver a formal logic-based framework to characterize the interactions between computees in a rule-based manner, either by relying on protocols shared and agreed upon by all computees in a given society, or by interaction patterns that are specific to individual computees and possibly different for different computees.

We defined a model which lent itself easily to a computational realisation, and which is precise and amenable to formal verification of properties pertaining to the interactions of the computees belonging to a particular societies.

The social model of interaction among computees that we propose follows a Computational Logic-based approach. In this model, Logic Programming, suitably extended with the concept of $\mathcal{IC}_S$ and expectations (interpreted as abducibles), acts as a uniform language for protocols, interaction policies and patterns.

A degree of openness, understood as the freedom of its members to join or leave the society, is given by the model presented in Section 2, which caters for new members joining a society and existing members leaving it [29, p. 27]. Another degree of openness, understood as the possibility to have a society of heterogeneous computees, is achieved by the fact that the model of the society, including the handling of expectations, the protocol conformance checking and the generation of violations, are only based on the socially observable behaviour of computees: no assumption is made on the internals of computees, but their social behaviour is constrained by the semantics of social actions and protocols. Non-conforming behaviour of computees is still possible, but it will be detected by the society infrastructure and it will have social consequences.

Violation handling and recovery is a matter of current and future work. The SOCS model of society caters for reasoning under incomplete information, in the sense that events that did not happen or that have not been “detected” are treated as unattended expectations, and it is possible to reason over both expectations and happened events.

The formalism for expressing society rules and protocols, together with the semantics of the individual communication utterances, is based on Abductive Logic Programming and constraints over abducible predicates, and its declarative semantics has been given in terms of logical entailment. The $\mathcal{S}$CIFF provides the operational support for the underlying infrastructure.

Time is explicit in the model. The “reasoning” at a social level is made over time, and it takes into account issues such as dealing with deadlines, that are important also from a practical viewpoint [29, pp. 35–36]. In this way, we propose a social framework which is suitable for modelling a dynamic setting and able to handle changes in a dynamic environment.

In [43] we present an evaluation of the society model in the context of the Global Computing programme. We believe that one of the strong points of our approach are to be found in its formality, not only at a syntactic level (definition of what is the format of the society knowledge, ic_S, protocols, CCL format and constraints), but also, and more interestingly, at the semantic level, which allows us to describe what are the desirable evolutions of a society and link these formally to the social structure and social behaviour of the computees

Most importantly, the formality of the framework is indeed backed-up by an existing and well-defined operational counterpart, the $\mathcal{S}$CIFF, a proof-procedure which has been proven correct with respect to the model, implemented, and integrated in the implementation of a tool, SOCS-SI, which can be used to run some tests.

We have interpreted the protocol conformance checks and the normative control performed by the society as abductive tasks, and defined an extension of the IFF abductive proof procedure to deal with this task. The extension is non-trivial, and deals with complex forms of variables quantification in abductive logic programs, as well as constraint predicates.

Finally, we believe that a further contribution of the work presented in this document is that the computational models devised within SOCS for the computee and for their societies are can be easily integrated with each other, since they are based on a similar formalism and on the same technology.

Future work will go in the direction of testing the system in different scenarios and studying properties of the model and of specific instances of societies. Among the scenarios that we are considering to evaluate the expressiveness of the designed interaction models are: dialogue-based interaction, with a special focus on resource reallocation, a combinatorial auction and an electronic payment network protocol.

Acknowledgements

This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 SOCS Project [3], and by the MIUR COFIN 2003 projects *Sviluppo e verifica di sistemi multi-agente basati sulla logica* [44], and *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici.*

References

1. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Toni, F., Terreni, G.: The KGP model of agency: Computational model and prototype implementation. In this volume.
2. Global Computing, Future and Emerging Technologies: Co-operation of Autonomous and Mobile Entities in Dynamic Environments. Home Page: `http://www.cordis.lu/ist/fetgc.htm`.
3. Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. Home Page: `http://lia.deis.unibo.it/research/socs/`.
4. Hewitt, C.: Open information systems semantics for distributed artificial intelligence. Artificial Intelligence **47** (1991) 79–106
5. Artikis, A., Pitt, J., Sergot, M.: Animated specifications of computational societies. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy, ACM Press (2002) 1053–1061
6. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Modeling interactions using *Social Integrity Constraints*: A resource sharing case study. In Leite,

J.A., Omicini, A., Sterling, L., Torroni, P., eds.: Declarative Agent Languages and Technologies. Lecture Notes in Artificial Intelligence **2990**. Springer-Verlag (2004) 243–262
7. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5., Oxford University Press (1998) 235–324
8. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
9. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
10. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of interaction protocols: a computational logic approach based on abduction. Technical Report CS-2003-03, Dipartimento di Ingegneria di Ferrara, Ferrara, Italy (2003) Available at `http://www.ing.unife.it/aree_ricerca/informazione/cs/technical_reports`.
11. 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
12. Stathis, K., Kakas, A.C., Lu, W., Demetriou, N., Endriss, U., Bracciali, A.: PROSOCS: a platform for programming software agents in computational logic. 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) 523–528
13. Lloyd, J.W.: Foundations of Logic Programming. 2nd extended edn. Springer-Verlag (1987)
14. Torroni, P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: A logic based approach to interaction design in open multi-agent systems. In: Proceedings of the 13th IEEE international Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), 2nd international workshop "Theory and Practice of Open Computational Systems (TAPOCS)", Modena, Italy, IEEE Press (2004) to appear.
15. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An Abductive Interpretation for Open Societies. In Cappelli, A., Turini, F., eds.: AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa. Lecture Notes in Artificial Intelligence **2829**. Springer-Verlag (2003) 287–299
16. Cox, P.T., Pietrzykowski, T.: Causes for events: Their computation and applications. In: Proceedings CADE-86. (1986) 608–621
17. Eshghi, K., Kowalski, R.A.: Abduction compared with negation by failure. In Levi, G., Martelli, M., eds.: Proceedings of the 6th International Conference on Logic Programming, MIT Press (1989) 234–255
18. 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
19. Poole, D.L.: A logical framework for default reasoning. Artificial Intelligence **36** (1988) 27–47

20. Kunen, K.: Negation in logic programming. In: Journal of Logic Programming. Volume 4. (1987) 289–308
21. Jaffar, J., Maher, M., Marriott, K., Stuckey, P.: The semantics of constraint logic programs. Journal of Logic Programming **37(1-3)** (1998) 1–46
22. FIPA Query Interaction Protocol (2001) Published on August 10th, 2001. Available for download from the FIPA web site, `http://www.fipa.org`.
23. SICStus prolog user manual, release 3.11.0 (2003) Available for download from the SICS web site, `http://www.sics.se/isl/sicstus/`.
24. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming **37** (1998) 95–138
25. Holzbaur, C.: Specification of constraint based inference mechanism through extended unification. Dissertation, Dept. of Medical Cybernetics & AI, University of Vienna (1990)
26. Conte, R., Falcone, R., Sartor, G.: Special issue on agents and norms. Artificial Intelligence and Law **1** (1999)
27. Castelfranchi, C., Dignum, F., Jonker, C., Treur, J.: Deliberative normative agents: Principles and architecture. In Jennings, N.R., Lespérance, Y., eds.: Intelligent Agents VI. Lecture Notes in Computer Science **1757**. Springer-Verlag (1999) 364–378
28. ALFEBIITE: A Logical Framework for Ethical Behaviour between Infohabitants in the Information Trading Economy of the universal information ecosystem. IST-1999-10298 (1999) Home Page: `http://www.iis.ee.ic.ac.uk/~alfebiite/ab-home.htm`.
29. Mello, P., Torroni, P., Gavanelli, M., Alberti, M., Ciampolini, A., Milano, M., Roli, A., Lamma, E., Riguzzi, F., Maudet, N.: A logic-based approach to model interaction amongst computees. Technical report, SOCS Consortium (2003) Deliverable D5. Available on request.
30. Yolum, P., Singh, M.: Flexible protocol specification and execution: applying event calculus planning using commitments. 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) 527–534
31. Singh, M.: Agent communication language: rethinking the principles. IEEE Computer (1998) 40–47
32. Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language. 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) 535–542
33. Colombetti, M., Fornara, N., Verdicchio, M.: The role of institutions in multiagent systems. In: Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002, Siena, Italy. (2002)
34. Colombetti, M., Fornara, N., Verdicchio, M.: A social approach to communication in multiagent systems. In Leite, J.A., Omicini, A., Sterling, L., Torroni, P., eds.: Declarative Agent Languages and Technologies. Lecture Notes in Artificial Intelligence **2990**. Springer-Verlag (2004) 193–222
35. Kakas, A., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: A logic-based approach to model computees. Technical report, SOCS Consortium (2003) Deliverable D4. Available on request.

36. Dignum, V., Meyer, J.J., Dignum, F., Weigand, H.: Formal specification of interaction in agent societies. In: Proceedings of the Second Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS), Maryland. (2002)
37. Arisha, K.A., Ozcan, F., Ross, R., Subrahmanian, V.S., Eiter, T., Kraus, S.: IMPACT: a Platform for Collaborating Agents. IEEE Intelligent Systems **14** (1999) 64–72
38. Eiter, T., Subrahmanian, V., Pick, G.: Heterogeneous active agents, I: Semantics. Artificial Intelligence **108** (1999) 179–255
39. Kakas, A.C., Lamma, E., Mancarella, P., Mello, P., Stathis, K., Toni, F.: Computational model for computees and societies of computees. Technical report, SOCS Consortium (2003) Deliverable D8. Available on request.
40. 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. Lecture Notes in Computer Science **2401**. Springer-Verlag (2002) 22–37
41. Kowalski, R.A., Sergot, M.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
42. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: an electronic institutions editor. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, Bologna, Italy, ACM Press (2002) 1045–1052
43. Bracciali, A., Kakas, A.C., Lamma, E., Mello, P., Stathis, K., Toni, F., Torroni, P.: D11: Evaluation and self assessment. Technical report, SOCS Consortium (2003) Deliverable D11. Available on request.
44. MASSiVE: sviluppo e verifica di sistemi multi-agente basati sulla logica. Home Page: `http://www.di.unito.it/massive/`.