A Generalized Commitment Machine for 2CL Protocols and its Implementation

Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati,
Elisa Marengo, and Viviana Patti

Università degli Studi di Torino
Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)
Email: name.surname@unito.it

Abstract. This work proposes an operational semantics for the commitment protocol language 2CL. This semantics relies on an extension of Singh's Generalized Commitment Machine, that we named 2CL-Generalized Commitment Machines. The 2CL-Generalized Commitment Machine was implemented in Prolog by extending Winikoff, Liu and Harland's implementation. The implementation is equipped with a graphical tool that allows the analyst to explore all the possible executions, showing both commitment and constraint violations, and thus helping the analyst as well as the protocol designer to identify the risks the interaction could encounter. The implementation is part of an Eclipse plug-in which supports 2CL-protocol design and analysis.

Keywords: Commitment protocols, constraints among commitments, commitment machine, commitment machine implementation

1 Introduction and Motivation

Agent interaction is generally specified by defining *interaction protocols* [21]. For communicating with one another, agents must follow the schema that the protocol shapes. Different protocol models can be found in the literature, this work concerns *commitment-based protocols* [19, 24]. This kind of protocols relies on the notion of commitment, which in turn encompasses the notions of debtor and creditor: when a commitment is not fulfilled, the debtor is liable for that violation but as long as agents reciprocally satisfy their commitments, any course of action is fine.

In many practical contexts where protocols model *business interactions* (e.g. trading, banking), designers must be able to regulate and constrain the possible interactions as specified by *conventions*, *regulations*, *preferences* or *habits* [2, 5]. Some proposals address the issue of introducing similar regulations inside commitment protocols [4, 11, 7, 17], but none of them developed tools for visualizing and analyzing how regulations or constraints impact on the interactions allowed by a commitment-based protocol. The availability of intuitive and

possibly graphical tools of this kind would support the identification of possible violations, thus enabling an *analysis of the risks* the interaction could encounter. As a consequence, it would be possible to raise alerts concerning possible violations before the protocol is enacted, and to reduce risks by defining proper operational strategies, like regimentation (aimed at preventing the occurrence of violations) or enforcement (introduction of warning mechanisms) [14].

The work presented in this paper aims at filling this gap. To this purpose, we started from the commitment protocol language 2CL described in [4], whose key characteristic is the extension of the regulative nature of commitments by featuring the definition of patterns of interaction as sets of *constraints*. Such constraints declaratively specify either conditions to be achieved or the order in which some of them should be achieved. The first contribution is, therefore, a formal, operational semantics for the proposal in [4], which relies on the Generalized Commitment Machine in [20]. We named our extension 2CL-Generalized Commitment Machines (2CL-GCM for short). On top of this, it was possible to realize the second contribution of this work: a Prolog implementation for 2CL-GCM, which extends the implementation in [22], and is equipped with a graphical tool to explore all the possible executions, showing both commitment and constraint violations. The implementation is part of a plug-in Eclipse which supports 2CL-protocol design and analysis.

The chief characteristic of our solution is that it performs a *state evaluation* of protocol constraints, rather than performing path evaluation (as, instead, done by model checking techniques). State evaluation allows considering each state only once, labeling it as a state of violation if some constraint is violated in it or as a legal state when no constraint is violated. This is a great difference with respect to path evaluation, where a state belonging to different paths can be classified as a state of violation or not depending on the path that is considered. The advantage is practical: state evaluation allows to easily supply the user an overall view of the possible alternatives of action, highlighting those which will bring to a violation and those that will not. State evaluation, however, is possible only by making some restriction on the proposal in [4]. Specifically, we assume that the domain is expressed in terms of positive facts only.

The paper is organized as follows. Section 2 briefly summarizes 2CL interaction protocol specification. Section 3 describes the formalization of 2CL-GCM. Section 4 presents a Prolog implementation of 2CL-GCM. Section 5 describes the 2CL Tools that supply features for supporting the protocol design and analysis. Section 6 discusses Related Work and Conclusions. Along the paper we use as a running example the well-known NetBill interaction protocol.

2 Background: 2CL Interaction Protocols

Let us briefly recall the main characteristics of commitment protocols, as defined in [4]. In this approach, commitment protocols are extended with a set of temporal constraints the interaction should respect. Constraints relate *commitments*. By $C(x, y, r, p)$ agent x commits to an agent y to bring about the consequent

Table 1. 2CL operators and their meaning.

	Relation	*Operator*	*Repr.*	*LTL formula*
Relation Operators	Correlation	*A correlate B*	$A \bullet\!\!- B$	$\Diamond A \supset \Diamond B$
		A not correlate B	$A \bullet\!\!\not- B$	$\Diamond A \supset \neg\Diamond B$
	Co-existence	*A co-exist B*	$A \bullet\!\!-\!\!\bullet B$	$A \bullet\!\!- B \wedge B \bullet\!\!- A$
		A not co-exist B	$A \bullet\!\!\not-\!\!\bullet B$	$A \bullet\!\!\not- B \wedge B \bullet\!\!\not- A$
Temporal Operators	Response	*A response B*	$A \bullet\!\!\rightarrow B$	$\Box(A \supset \Diamond B)$
		A not response B	$A \bullet\!\!\not\rightarrow B$	$\Box(A \supset \neg\Diamond B)$
	Before	*A before B*	$A \rightarrow\!\!\bullet B$	$\neg B \cup A$
		A not before B	$A \not\rightarrow\!\!\bullet B$	$\Box(\Diamond B \supset \neg A)$
	Cause	*A cause B*	$A \bullet\!\!\rightarrow\!\!\bullet B$	$A \bullet\!\!\rightarrow B \wedge A \rightarrow\!\!\bullet B$
		A not cause B	$A \bullet\!\!\not\rightarrow\!\!\bullet B$	$A \bullet\!\!\not\rightarrow B \wedge A \not\rightarrow\!\!\bullet B$

condition p when the antecedent condition r holds. When r equals $true$, we use the short notation $C(x, y, p)$. Commitments are used to define the *social effects* of the protocol actions.

Definition 1 (Interaction protocol). $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, \mathrm{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$ *An interaction protocol* $\mathcal{P}$ *is a tuple* $\langle \mathsf{Ro}, \mathsf{F}, \mathrm{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$*, where* Ro *is a set of roles, identifying the interacting parties,* F *is a set of facts and commitments that can occur in the social state,* s_0 *is the set of facts and commitments in the initial state of the interaction,* A *is a set of actions, and* Cst *is a set of constraints.*

The set of social actions A, defined on F and on Ro, forms the *constitutive specification* of the protocol. The social effects are introduced by the construct *means*, which amounts to a *counts-as* relation [18, 14]: by means of it, a physical event is given a social meaning. An *if* condition denotes the context in which a counts-as relation holds. For instance, consider the action *sendGoods* reported in Table 1. Its social meaning is that it makes the facts *goods* true (the goods were delivered to the customer) and creates the commitment $\mathsf{C}(m, c, pay, receipt)$ that corresponds to a promise by the merchant to send a receipt after the customer has paid. Further examples can be found in the first part of Table 1, which reports all the actions of the NetBill protocol. The formalization is inspired by those in [24, 22].

2CL constraints Cst, defined on F and on Ro as well. Constraints express what is mandatory and what is forbidden without the need of listing the possible executions extensionally. The syntax is "dnf_1 op dnf_2", where dnf_1 and dnf_2 are disjunctive normal forms of facts and commitments, and op is one of the 2CL operators, reported in Table 1 together with their Linear-time Temporal Logic [10] interpretation and with their graphical notation.

Constraints can either be *relational* or *temporal*. The former kind expresses constraints on the co-occurrence of conditions (if a condition is achieved then also another condition must be achieved, but the order of the two achievements does not matter). For instance, one may wish to express that both the payment

Action Definitions
(a1) *sendRequest* **means** *request* **if** $\neg quote \wedge \neg goods$
(a2) *sendQuote* **means** $quote \wedge create(\mathsf{C}(m, c, \mathsf{C}(c, m, goods, pay), goods))$
$\wedge\ create(\mathsf{C}(m, c, pay, receipt))$
(a3) *sendAccept* **means** $create(\mathsf{C}(c, m, goods, pay))$ **if** $\neg pay$
(a4) *sendGoods* **means** $goods \wedge create(\mathsf{C}(m, c, pay, receipt))$
(a5) *sendEPO* **means** pay
(a6) *sendReceipt* **means** $receipt$ **if** pay

Constraints
(c1) $quote \rightarrowtail\!\bullet\ \mathsf{C}(c, m, goods, pay) \vee \mathsf{C}(c, m, pay)$
(c2) $\mathsf{C}(m, c, pay, receipt) \wedge goods \rightarrowtail\!\bullet\ pay$
(c2) $pay \bullet\!\!\rightarrowtail receipt$

Fig. 1. Actions and constraints for the NetBill protocol: m stands for merchant while c stands for customer.

for some item and its delivery must occur without constraining the order of the two conditions: no matter which occurs first, when one is met, also the other must be achieved. Temporal constraints, instead, capture the relative order at which different conditions should be achieved. Fig. 1 reports the constraints imposed by the NetBill protocol: *(c1)* means that a quotation for a price must occur before a commitment to pay or a conditional commitment to pay given that some goods were delivered; *(c2)* that the conditional commitment to send a receipt after payment and the delivery of goods must occur before the payment is done; *(c3)* that after payment a receipt must be issued and if a receipt is issued a payment must have occurred before.

Only interactions which respect the constraints are *legal*. Violations amounting to the fact that a constraint is not respected can be detected *during the execution*.

3 2CL Generalized Commitment Machine

The semantics of 2CL commitment protocols is given based on the 2CL *generalized commitment machine* (2CL-GCM). In turn, 2CL-GCM relies on the notion of *generalized commitment machine* (GCM) (introduced in [20]), extending it with a proper account of 2CL constraints. Below we introduce the technical elements on top of which the definition of a 2CL-GCM will be given.

Propositions. Propositions are meant to capture conditions of interests (e.g. the fact that a payment has occurred or that a request for quote has been made) and social relationships among the interacting parties. We represent them in terms of facts and commitments, whose meaning is assumed to be known and agreed by all the agents. Let us assume true and false to be part of this set, representing respectively the *true* and the *false* values of propositional logic.

States. The evolution of an interaction is represented by means of states: each state captures a snapshot on a particular moment of the interaction. According to [20], a GCM features a set S of possible *states*, each of which is represented by a logical expression defined on a set of propositions.

Example 1. Considering NetBill, *goods* $\wedge$ $\mathsf{C}(c, m, pay)$ represents one possible configuration of the social state, i.e. it is a state in S. This expression means that the goods were shipped and that there is a commitment from c (customer) to m (merchant) to pay for them.

Initial state. Denoted by s_0, it is the state from which the interaction starts.

Example 2. In the NetBill example, if we assume the commitment $\mathsf{C}(m, c,$ *pay*, *goods*) to be part of the initial state it represents that, when accepting the role of merchant, the agent is also taking the engagement to send the goods when these are paid.

Good States. We identify a set $\mathsf{G} \subseteq \mathsf{S}$ as the set of *good states*. Intuitively, they capture desired possible endings of the interaction. For instance, they may be those that do not contain unsatisfied active commitments, or those satisfying a condition of interest (e.g. payment done and goods shipped).

Physical Events. The interaction evolves as the consequence of the occurrence of physical events. We denote by $\mathsf{L_A}$ their set.

Action Theory. Given the definition of an action a, and two states s and s', it is possible to determine whether a transition between the two can be inferred as a consequence of the occurrence of a physical event a. As in [20], in 2CL-GCM transitions between the states are logically inferred on the basis of an action theory, that contains a set of axioms of the kind $p \overset{a}{\hookrightarrow} q$, meaning that q is a consequence of performing action a in a state where p holds. When q is false the meaning is that a is impossible if p holds. Only transitions that find correspondence in an axiom of the action theory can be inferred. In the following E is the conjunction of E_{F}, which is a logical expression (possibly true) concerning facts only, and E_{op}, which is a logical expression (possibly true) concerning operations on commitments only.

Definition 2 (Action theory). *An action axiom* $s \overset{a}{\hookrightarrow} s'$ *belongs to the action theory* Δ *of a protocol* $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, \mathrm{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$ *iff there exists a definition "a* ***means*** *E* ***if*** *Cond" in* A *s.t.* $s \vdash Cond$ *and:*

(a) $\forall e_{\mathsf{op}}$ *s.t.* $E_{\mathsf{op}} \vdash e_{\mathsf{op}}$ *and given* $z \overset{e_{\mathsf{op}}}{\hookrightarrow} z'$ *according to commitment operations' axioms (see [20, Section 2.2]) then if* $s \vdash z$ *then* $s' \vdash z'$*; and*
(b) $\forall e_{\mathsf{F}}$ *s.t.* $E_{\mathsf{F}} \vdash e_{\mathsf{F}}$ *then* $s' \vdash e_{\mathsf{F}}$ *and:*
  (b.1) if $s \vdash C(x, y, e, e_{\mathsf{F}})$ *(with* e *possibly* true*) then* $s' \vdash \neg C(x, y, e, e_{\mathsf{F}})$ *and*
  (b.2) if $s \vdash C(x, y, e_{\mathsf{F}}, e')$ *then:*
    if $s' \nvdash e'$ *then* $s' \vdash \neg C(x, y, e_{\mathsf{F}}, e') \wedge C(x, y, e')$*; otherwise* $s' \vdash \neg C(x, y, e_{\mathsf{F}}, e') \wedge \neg C(x, y, e')$.

where $\vdash$ *and* $\equiv$ *represent respectively the logical consequence and the logical equivalence of propositional logic.*

Given two states, in order to determine whether the latter can be a consequence of the occurrence of a physical event in the former, it is necessary to consider the definition of the corresponding action 'a **means** E **if** $Cond$'. A transition labelled by 'a' can be inferred only if the condition $Cond$ can be derived in the starting state. In this case it is necessary to consider the effects E of the action and whether they can be derived on the target state. The target state should derive all the facts in E_{F} while for the operations on commitments E_{op} we apply the rules defined in [20, Section 2.2]. Finally, conditions *(b.1)* and *(b.2)* in Definition 2 check the *discharge* and the *detach* of commitments, due to facts derived from E_{F}.

Example 3. The action *sendAccept*, performed by the customer to accept a quote by the merchant, is defined as *sendAccept* **means** CREATE($\mathsf{C}(c, m, goods, pay)$) **if** $\neg pay$. The corresponding axiom is $\neg pay \overset{sendAccept}{\hookrightarrow} \mathsf{C}(c, m, goods, pay)$. Note that, given a state, in which $\neg pay \wedge quote$ holds, it is also possible to infer the axiom $\neg pay \wedge quote \overset{sendAccept}{\hookrightarrow} \mathsf{C}(c, m, goods, pay)$.

Constraints. A 2CL-GCM accounts for a set of constraints Cst that coincides with that defined in the corresponding protocol. These constraints will be taken into account for determining whether an interaction can be considered as a path of the machine.

We now have all the elements for defining a 2CL-GCM. The definition adopts the same notation in [20].

Definition 3 (2CL-GCM of a protocol). *A* 2CL-GCM *of a protocol* $\mathcal{P} = \langle \mathsf{Ro}, \mathsf{F}, \mathsf{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$ *is a tuple* $\mathbf{P} = \langle \mathsf{S}, \mathsf{L_A}, \mathsf{s}_0, \Delta, \mathsf{G}, \mathsf{Cst} \rangle$ *where:*

- S *is a set of states represented as logical expressions;*
- $\mathsf{L_A}$ *is a set of physical events s.t.* $a \in \mathsf{L_A}$ *iff* $\exists a$ ***means*** E ***if*** $Cond \in \mathsf{A}$ *;*
- $\mathsf{s}_0 \in S$ *and represents the initial state;*
- Δ *is an action theory s.t.* $\forall s, s' \in \mathsf{S}, s \overset{a}{\hookrightarrow} s' \in \Delta$ *iff there exists a* ***means*** E ***if*** $Cond \in \mathsf{A}$ *s.t.* $s \overset{a}{\hookrightarrow} s'$ *is an action axiom of 'a' according to Definition 2;*
- $\mathsf{G} \subseteq \mathsf{S}$ *is a set of good states;*
- Cst *is a set of* 2CL *constraints.*

Moreover:

- $\forall s, s' \in \mathsf{S}, s \not\equiv s'$*, i.e. members of* S *are logically distinct;*
- $\mathsf{false} \notin \mathsf{S}$*; and*
- $\forall s \in \mathsf{G}, s' \in \mathsf{S} : (s' \vdash s) \Rightarrow (s' \in \mathsf{G})$*, i.e. any state that logically derives a good state is also good.*

Notice that by varying the sets S and G different 2CL-GCMs associated to the same protocol can be obtained: when S contains all the states that can be reached from s_0, applying the protocol actions, the machine can infer all the possible interactions; when S is smaller, only a subset of the possible interactions is determined.

3.1 Path of a 2CL-GCM.

Interactions between agents can be seen as paths traversing states, the transitions among which are labeled by the physical events which caused them. We denote a path τ as the sequence $\langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2), \dots\rangle$. In order for a path to be part of a 2CL-GCM it must respect some conditions:

1. The path must be infinite;
2. All the transitions of the path must be inferable by the machine; and
3. All constraints must be satisfied in the path.

It is not restrictive to focus on infinite paths. Indeed, all finite paths can be transformed into infinite ones by adding a transition from the last state of the finite path towards an artificial new state with a self loop [20]. In 2CL-GCM we assume that the action axioms that allow inferring such transitions are part of Δ.

2CL constraints verification can be done by exploiting the LTL formula associated to each of them. In particular, a constraint is satisfied in a path when it is verified in the *transition system* corresponding to the path. Given a path, the corresponding transition system can be derived quite straightforwardly.

Definition 4 (Transition System). *A transition system $T(\tau)$ of a path $\tau = \langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2), \dots\rangle$ is a tuple $\langle S_\tau, \delta_\tau, L_\tau\rangle$ where:*

- *$S_\tau = \{\tau_i |\ \tau_i$ is a state in $\tau\}$;*
- *$\delta_\tau : S_\tau \to S_\tau$ is a transition function s.t. $\delta(\tau_j) = \tau_k$ iff (τ_j, a, τ_k) is in τ;*
- *$L : S_\tau \to 2^F$ is a labelling function, s.t. F is a set of facts and commitments and $L(\tau_i) = \{e | \tau_i \vdash e\}$.*

To define a 2CL-GCM path, we extend the definition of GCM path by additionally requiring the satisfaction of all the constraints of the 2CL-GCM.

Definition 5 (2CL-GCM path). *A path $\tau = \langle(\tau_0, a_0, \tau_1), (\tau_1, a_1, \tau_2)\dots\rangle$ is a path of a 2CL-GCM $\mathbf{P} = \langle \mathsf{S}, \mathsf{L_A}, \mathsf{s_0}, \Delta, \mathsf{G}, \mathsf{Cst}\rangle$ when:*

i. $\forall(\tau_i, a_i, \tau_{i+1})$ in τ then $\tau_i, \tau_{i+1} \in \mathsf{S}$, $a_i \in \mathsf{L_A}$, and $\tau_i \overset{a_i}{\hookrightarrow} \tau_{i+1} \in \Delta$; and
ii. being $inf(\tau)$ the set of states that occur infinitely often in τ, then $inf(\tau) \cap \mathsf{G} \neq \emptyset$; and
iii. being $T(\tau)$ the transition system of τ according to Definition 4, $\forall c \in \mathsf{Cst}$: $T(\tau), \tau_0 \models_{LTL} c$.

where the LTL satisfaction relation $\models_{LTL}$ is the one defined in [1].

In the above definition, *(i)* and *(ii)* are the conditions for a path to be generated. Condition *(i)* requires that each state in the path is a state of the 2CL-GCM, that the action that causes the transition from a state to the subsequent one in the path is an action of the 2CL-GCM, and that the transition is inferable according to the axioms in Δ. Condition *(ii)* requires that at least one *good state* occurs infinitely often in the path. Condition *(iii)* accounts for the evaluation of the constraints. According to the LTL semantics, $T(\tau), \tau_0 \models_{LTL} c$ amounts to checking if c is satisfied in all the paths of the transition system, corresponding to τ. By construction $T(\tau)$ is a transition system made only of *one* linear path, whose starting state is the starting state of τ.

4 Implementation of the 2CL Commitment Machine

This section describes a Prolog implementation that allows exploring all the possible executions of an interaction protocol, showing the *regulative violations* – i.e. both those states in which some constraint is violated and those that contain unsatisfied commitments. We also prove the soundness of the implementation w.r.t. the 2CL-GCM formalization presented in the previous sections.

We used *tuProlog*[1] in our implementation, starting from the *enhanced commitment machine* by Winikoff et al. [22]. By relying on it, we inherit the mechanisms for the computation of the possible interactions. Specifically, the enhanced commitment machine features the generation of the reachable states, the transitions among them and the management of commitments (like the operations of discharge, creation and so on). Our extension equips it with the possibility of evaluating 2CL constraints.

The main characteristic of our tool is that it provides an overall graphical view of the possible interactions, highlighting those that will bring to a violation and those that will not. To this aim, constraints are used as a means to classify the possible interactions, rather than to prune the search space. The interacting parties, indeed, are not prevented from entering in illegal paths (due to the agent's autonomy), but they are made aware of the risks they are encountering and that they may incur in penalties as a consequence of the violations they caused [5]. This is a difference compared with those proposals where only the set of legal paths is shown, or with other proposals that aim at properties verification. In these cases, the verification ends when a path that does not satisfy the property is found. Alternatively, only one path at a time is considered [6, 9]. Thus, none of these proposals provide an overview of possible interactions.

Starting from a protocol specification, our implementation determines the set of reachable states by applying a depth-first search (as in [22]). Specifically, given a state the program finds the set of applicable actions and computes the set of successors. A state is added to the graph only if it is new, otherwise only the transition is added. For what concerns the evaluation of protocol constraints, we implemented it as a *state evaluation*, that is to say that given a constraint its evaluation can be done on a state by considering its content only. In this way, each possible state (reachable given the starting state and the protocol actions) is considered only once and it is classified as a state of violation if some constraint is violated in it or as a legal state when no constraint is violated. This is a great difference with respect to path evaluation, where a state belonging to different paths can be classified as a state of violation or not depending on the path that is considered. The advantage is practical: given the set of reachable states, the user is able to immediately determine which of them are legal and which violate some constraints. Moreover, the overall representation results to be more compact because each state appears only once.

In order to perform the *state evaluation* we consider states whose content is given in terms of commitments and positive facts only. The characteristic of a

[1] `http://www.alice.unibo.it/xwiki/bin/view/Tuprolog/`

fact is that it is false until it becomes invariably true. In this setting, the evaluation of 2CL constraints can be made on single states. For instance, if in a state b holds but a does not, we can infer that the constraint '*a before b*' is violated. Moreover, besides facts asserted by the protocol actions, in our implementation we additionally consider a set of facts associated to the operations performed on commitments. Specifically, along the line of Mallya *et al.* [15], whenever an operation is performed on a commitment, a corresponding predicate is automatically asserted in the state. For instance, when a commitment $\mathsf{C}(x, y, r, p)$ is created, the predicate CREATED($\mathsf{C}(x, y, r, p)$) is added to the state; when it is discharged, the predicate DISCHARGED($\mathsf{C}(x, y, r, p)$) is added, and so forth for the other operations. Notice that these predicates are not meant to express whether a commitment is active or not. For instance, CREATED does not mean that the commitment is active in the state but simply that the corresponding operation has been performed on the commitment. 2CL constraints can be defined by considering these facts also.

In order to achieve the benefits of a state evaluation while guaranteeing the soundness of the verification with the theoretical framework presented in the previous section, we need to make some assumptions on the way protocols are specified:

1. Actions should be defined in such a way to do not retract facts;
2. The condition involved in constraints must involve conditions that persist (i.e. that involve DNFs of facts without negation);
3. Constraints expressed on commitments are to be opportunely transformed into constraints concerning operations preformed on commitments.

For the sake of clarity, we use the symbol $\mathbf{P_I}$ to refer to a protocol that respects these assumptions.

4.1 Generation of the Labeled Graph and its Soundness.

Let us consider a protocol $\mathbf{P_I}$. As reported in Listing 1.1, the exploration of the search space is made as a depth-first-search (as in [22]). Specifically, given a state, `explore` and `nextstate` find the set of possible successors, obtained by considering the set of actions in $\mathbf{P_I}$. For those actions whose preconditions are satisfied in the state, `nextstate` determines the resulting state by adding the facts which constitute their social meaning, executing the operations performed on commitments according to the commitments' life cycle and asserting the corresponding facts concerning such operations. Once the successor states are obtained they are added to the set of reachable states together with the corresponding transitions. The computation is rooted in the initial state. In our program, states are represented as predicates.

Definition 6 (State). *The predicate* `state(ID, Content, Label)` *represents a state in the implementation where:* `ID` *is a unique identifier associated to the state,* `Content` *is a list of facts and commitments and* `Label` $\subseteq$ {*final,not-final, violation, pending*} *is a list of labels that captures the absence (*final*) or the*

```
1  explore(StateNum,Free,NextFree) :- state(StateNum,State,_),
2    findall(t(StateNum,A,S2),nextstate(State,A,S2),Ts),
3    add_states(Ts,Free,NextFree), add_transitions(Ts).
4
5  nextstate(State,Action,Result) :- happens(Action,State),
6    findall(Add,initiates(Action,Add,State),AddS),
7    findall(Del,terminates(Action,Del,State),DelS),
8    merge_addList(AddS,State,NewState),
9    findall(StableProp,initiates_stable_prop(Action,
10             StableProp,State,NewState),StablePropS),
11   merge_addList(AddS,StablePropS,AddList),
12   compute_next_state(State,AddList,DelS,New),
13   remove_duplicates(New,Result).
14
15 add_states([],N,N).
16 add_states([t(_,_,S)|Ss],N,N1) :-
17   state(_,St,_), seteq(St,S), !, add_states(Ss,N,N1).
18 add_states([t(_,_,S)|Ss],N,N3) :-
19   labels(S,L), assert(state(N,S,L)),
20   N1 is N+1, explore(N,N1,N2), add_states(Ss,N2,N3).
21
22 add_transitions([]).
23 add_transitions([t(S1,A,S2)|Ss]) :- transition(S1,A,Ss2),
24   seteq(S2,Ss2), !, add_transitions(Ss).
25 add_transitions([t(S1,A,S2)|Ss]) :- state(N2,Ss2,_), seteq(Ss2,S2),
26   assert(transition(S1,A,N2)), add_transitions(Ss).
27
28 subsumes(P,P).
29 subsumes(P,c(_,_,PP)) :- subsumes(P,PP).
30 subsumes(P,cc(_,_,_,PP)) :- subsumes(P,PP).
31 subsumes(c(X,Y,P),cc(X,Y,_Q,PP)) :- subsumes(P,PP).
32
33 happens(E,T) :- isAction(E), precond(E,P), implied(P,T).
34
35 initiates(E,P,T) :- happens(E,T), isFluent(P), causes(E,P).
36 initiates(E,c(X,Y,P),T) :- causes(E,create(c(X,Y,P))), happens(E,T),
37         \+(implied(P,T)).
38 initiates(E,c(X,Y,P),T) :- causes(E,create(cc(X,Y,Q,P))),
39         happens(E,T), implied(Q,T), \+(implied(P,T)).
40 initiates(E,cc(X,Y,P,Q),T) :- causes(E,create(cc(X,Y,P,Q))),
41         happens(E,T), \+(implied(Q,T)), \+(implied(P,T)).
42 initiates(E,c(X,Y,Q),T) :- holdsAt(cc(X,Y,P,Q),T), happens(E,T),
43         subsumes(PP,P), initiates(E,PP,T).
44
45 terminates(E,c(X,Y,P),T) :- holdsAt(c(X,Y,P),T), happens(E,T),
46         subsumes(PP,P), initiates(E,PP,T).
47 terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T), happens(E,T),
48         subsumes(QP,Q), initiates(E,QP,T).
49 terminates(E,cc(X,Y,P,Q), T) :- holdsAt(cc(X,Y,P,Q),T), happens(E,T),
50         subsumes(PP,P),  initiates(E,PP,T).
```

Listing 1.1. Prolog clauses that compute the set of reachable states and that assert the corresponding transitions The complete program can be downloaded at the URL http://di.unito.it/2cl.

*presence (*not-final*) of unsatisfied active commitments, the presence of* pending *constraints or the* violation *of a constraint.*

Notice that according to the clause `add_state` reported in Listing 1.1 (line 16), a state is added only if it is new, that is to say: there are no existing states with the same content.

Relation	*State Condition*
Correlation	$\psi(A \bullet\!\!- B) = A \wedge B$
	$\psi(A \bullet\!\!\not- B) = \neg(A \wedge B)$
Co-existence	$\psi(A \bullet\!\!-\!\!\bullet B) = \psi(A \bullet\!\!- B) \ \wedge \psi(B \bullet\!\!- A)$
	$\psi(A \bullet\!\!\not-\!\!\bullet B) = \psi(A \bullet\!\!\not- B) \ \wedge \psi(B \bullet\!\!\not- A)$
Response	$\psi(A \bullet\!\!-\!\!\triangleright B) = A \wedge B$
	$\psi(A \bullet\!\!\not-\!\!\triangleright B) = \neg(A \wedge B)$
Before	$\psi(A \rightarrow\!\!\bullet B) = \neg(B \wedge \neg A)$
	$\psi(A \not\rightarrow\!\!\bullet B) = \neg(A \wedge B)$
Cause	$\psi(A \bullet\!\!-\!\!\triangleright\!\!\bullet B) = \psi(A \bullet\!\!-\!\!\triangleright B) \ \wedge \psi(A \rightarrow\!\!\bullet B)$
	$\psi(A \bullet\!\!\not-\!\!\triangleright\!\!\bullet B) = \psi(A \bullet\!\!\not-\!\!\triangleright B) \ \wedge \psi(A \not\rightarrow\!\!\bullet B)$

Table 2. State conditions corresponding to 2CL operators.

Also transitions are represented by means of predicates, expressing the starting and the target state and the physical events that caused them.

Definition 7 (Transition). *The predicate* `transition(ID1, A, ID2)` *represents a transition where* `ID1` *and* `ID2` *correspond to the identifiers of existing states, and* `A` *is the action responsible for the transition.*

Before adding a state, this is labeled according to the constraints it satisfies or violates and to the commitments holding in it. Thanks to the assumptions that constraints are defined in terms of positive facts that persist along the interaction, the LTL formulas associated to the operators can be simplified. The resulting formulas are reported in Table 2 (a proof of their soundness can be found in [16, Chapter 6]. Below we provide an intuition).

Given a constraint c, we denote by $\psi(c)$ the corresponding condition to be verified on one state at a time (*state condition*). Consider, for instance, the *before* operator ($\rightarrow\!\!\bullet$): it requires that A is met before or in the same state of B. So, given a run π, if in π there is a state j such that B holds while A does not, that is a state where a violation occurred, in formulas: $\pi_i \models_{LTL} A \rightarrow\!\!\bullet B \ \Leftrightarrow \ \neg\exists j \geq i \ s.t. \ \pi_j \models_{LTL} (B \wedge \neg A)$[2].

The other 2CL operators can be divided in two cases. *Correlation* ($\bullet\!\!-$) and *response* ($\bullet\!\!-\!\!\triangleright$) are tackled in a similar way. $A \bullet\!\!- B$ requires that if A is achieved in a run, then also B is achieved in the same run (before or after A is not relevant). If B is achieved before A it will remain true also after. Therefore, in those cases in which the constraint is satisfied, from a certain time onwards both conditions will hold. In formulas: $\pi_i \models_{LTL} A \bullet\!\!- B \ \Leftrightarrow \ \neg\exists j \geq i \ s.t. \ \pi_j \models_{LTL} A$ and $\forall j' \geq j, \ \pi_{j'} \models_{LTL} (A \wedge \neg B)$. The same equivalence holds for $\pi_i \models_{LTL} A \bullet\!\!-\!\!\triangleright B$. In 2CL $A \bullet\!\!-\!\!\triangleright B$ requires that when A is met, B is achieved at least once later (even if it already occurred in the past) but under our assumptions it can be checked in the same way of correlation. The state condition amounts to verifying whether

[2] Notice that since the second formula does not contain temporal operators it is verified in the current state. Thus it is verified in all the states of the path.

a state satisfies A but does not satisfy B. Notice that states that satisfy the test cannot be marked as states of violation because the constraint does not require B to hold *whenever* A holds. A state of violation is signaled when the interaction does not continue after it: we say that there is a *pending* condition.

Negated correlation, *response* and *before* correspond to the same formula: $\pi_i \models_{LTL} A \texttt{ op } B \Leftrightarrow \neg\exists j \geq i \; s.t. \; \pi_j \models_{LTL} (A \wedge B)$ where $\texttt{op} \in \{\bullet\not\rightarrow, \bullet\not\triangleright, \not\triangleright\bullet\}$. Intuitively, a constraint of the kind $A \bullet\not\rightarrow B$ (negative correlation) requires that if A holds, B is not achieved. Since facts persist, this amounts to check that the two conditions do not hold in the same state, otherwise a violation occurs. *Negative response* (*negative before*) adds a *temporal aspect* to not-correlation: if A holds, B cannot hold later (before, respectively). Since facts persist, the first achieved condition will remain true also after the other becomes true. Also in this case we only need to check that the two conditions do not hold together.

Derived operators are decomposed and the reasoning made for the operators, from which they derive, is applied. For instance, *cause* ($\bullet\!\!\rightarrowtriangle\!\!\bullet$) derives from *before* and *response*. If a state does not satisfy the response part of the cause, it is marked as "pending"; if it violates the before part, it is marked as a "violation". Both labels are applied when the state does not satisfy any of the two.

Summarizing, given a constraint formula and a state in which to verify it, we have three possible outcomes: *(i)* the state satisfies the formula; *(ii)* the state does not satisfy the formula and this leads to a violation; and *(iii)* the state does not satisfy the formula but the violation is potential, depending on future evolution. Considering all the constraints of a protocol, a state can both violate some constraint and have pending conditions. Moreover, states are also evaluated based on the presence of unsatisfied active commitments.

In our implementation, constraints are represented with predicates. For instance, `before(A,B,Id)` represents a constraint of kind *before* whose antecedent and consequent conditions are are respectively `A` and `B` and `Id` is a unique identifier for the constraint. The predicates for the other kinds of constraints are similar, where `before` is substituted with the operator name. Constraints verification is implemented as previously described. Listing 1.2 reports, as an example, the verification of a *response* and of a *before*. The clause *check_pending* that is reported verifies response constraints: it is satisfied if there is a constraint of kind response, whose antecedent condition can be derived in the state, while the consequent condition cannot. In this case, the label *pending* is added to the list of labels of the state. A similar clause checks the correlation constraint. Instead, the clause *check_violation*, checks constraints of kind before, which are violated if the consequent condition can be derived in the state while their antecedent cannot. Other similar clauses, checking different conditions, are defined for the other operators. Finally, the program checks the presence of unsatisfied commitments (*check_commitments*) and adds the label *final* or *not-final* accordingly. The result of running this program on a protocol specification is an annotated graph of the reachable states.

On the basis of the labels associated to a state, that are a consequence of constraints verification, we can define a legal path.

```
labels(State,Labels) :- find_labels(State,[],Labels).

find_labels(S,L1,R) :- check_violation(S,L1,L2),
  check_pending(S,L2,L3), check_commitments(S,L3,R).

check_pending(State,L,[pending(Constr)|L]) :- response(A,B,Constr),
  consequence(A,State), \+consequence(B,State).

check_violation(State,L,[violation(Constr)|L]) :- before(A,B,Constr),
  consequence(B,State), \+consequence(A,State).

check_commitments(State,L,[final|L]) :- \+member(c(_,_,_),State).
check_commitments(State,L,[non-final|L]) :- member(c(_,_,_),State).
```

Listing 1.2. Prolog clauses checking constraints and adding the corresponding labels to the states

Definition 8 (Legal path). *Let* $\mathbf{P_I} = \langle \mathsf{Ro}, \mathsf{F}, \mathrm{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$ *be a protocol. A legal path* π *for* $\mathbf{P_I}$ *is a sequence* $\langle (0, a_0, 1), \ldots, (n-1, a_{n-1}, n) \rangle$ *where* $\forall i\ 0 \leq i \leq n$, *i represents the identifier of a state,* a_i *is an action in* A *and* π *is such that:*

i. $\forall (i, a_i, i+1)$ *in* π, *there exist* `state(i,` π_i, `Label`$_i$`)` *and* `state(i+1,` π_{i+1}, `Label`$_{i+1}$`)` *and* `transition(i,` a_i, `i+1)`*; and*
ii. `state(n,` π_n, `Label`$_n$`)` *is such that final* $\in$ `Label`$_n$ *and* $\{$*violation,pending*$\}$ $\cap$ `Label`$_n = \emptyset$*; and*
iii. $\nexists i$ *in* π *s.t.* `state(i,` π_i, `Label`$_i$`)` *and violation* $\in$ `Label`$_i$.

In words, a sequence of states and transitions is a legal path for a program when *(i)* each state in the path can be reached from the initial state by applying the actions (and in the specified order) identified by the sequence; *(ii)* the last state of the path does not contain unsatisfied active commitments or pending constraints; and *(iii)* none of the states in the path violates constraints.

In order to prove the soundness of our implementation we have to show that a legal path for our implementation is also a legal path for the corresponding 2CL-GCM. This latter, however, works on infinite paths where states are represented as logical formulas rather then as sets of facts and commitments. Along the line of [20], we define an equivalent infinite path π^∞ for a path π.

Definition 9 (Equivalent infinite path). $\pi^\infty = \langle (\pi_0, a_0, \pi_1), \ldots) \rangle$ *is the equivalent infinite path corresponding to the finite path* $\pi = \langle (0, a_0, 1), \ldots, (n-1, a_{n-1}, n) \rangle$ *iff:*

i. $\forall i,\ 0 \leq i \leq n$, *given* `state(i,` π_i, `Label`$_i$`)` $\pi_i^\infty \vdash f$ *iff* $f \in \pi_i$*; and*
ii. $\forall i,\ 0 \leq i < n$ $(\pi_i^\infty, a_i, \pi_{i+1}^\infty)$ *is in* π^∞ *iff* $(i, a_i, i+1)$ *is in* π*; and*
iii. $\forall i \geq n$ $(\pi_i^\infty, a_i, \pi_{i+1}^\infty)$ *in* π^∞ *is such that* $\pi_i^\infty \equiv \pi_n^\infty$ *and* $\pi_{i+1}^\infty \equiv \pi_n^\infty$ *and* a_i *is the action 'act* ***means*** `true` *if* π_n^∞*'.*

Intuitively, the infinite path is obtained by adding a self loop on the last state of the finite path. Now we have all the elements for proving soundness.

Theorem 1 (Soundness). *Consider a protocol* $\mathbf{P_I} = \langle \mathsf{Ro}, \mathsf{F}, \mathrm{s}_0, \mathsf{A}, \mathsf{Cst} \rangle$. *Let* $\pi = \langle (0, a_0, 1), \ldots, (n-1, a_{n-1}, n) \rangle$ *be a path and let* π^∞ *be the corresponding infinite path. Let* $\mathbf{P} = \langle \mathsf{S}^\infty_\pi, \mathsf{L_A}, \mathtt{s}_0, \Delta, \mathsf{G}, \mathsf{Cst} \rangle$ *be a* 2CL-GCM *of* $\mathbf{P_I}$ *such that* $\mathsf{S}^\infty_\pi = \{\pi^\infty_i | \pi^\infty_i$ *is in* $\pi^\infty\}$ *and* $\mathsf{G} = \{\pi^\infty_i | \nexists \mathsf{C}(x,y,p)$ *s.t.* $\pi^\infty_i \vdash \mathsf{C}(x,y,p)\}$. *If* π *is a legal path for* $\mathbf{P_I}$, *then* π^∞ *is a path of* $\mathbf{P}$.

Given a protocol and the program representing it, if a path is legal according to this latter, then there exists a 2CL-GCM for which the corresponding infinite path is a path according to Definition 5. More precisely a 2CL-GCM of the protocol for which this condition holds is the one obtained by considering as set of states the states that are part of the path. As good states we consider those that do not contain unsatisfied active commitments.

Proof. In order for π^∞ to be a path of the 2CL-GCM $\mathbf{P} = \langle \mathsf{S}^\infty_\pi, \mathsf{L_A}, \mathtt{s}_0, \Delta, \mathsf{G}, \mathsf{Cst} \rangle$ it must satisfy the conditions *(i)–(iii)* of Definition 5:

i. $\forall (\pi^\infty_i, a_i, \pi^\infty_{i+1})$ in π^∞ then (i.1) $\pi^\infty_i, \pi^\infty_{i+1} \in \mathsf{S}^\infty_\pi$, (i.2) $a_i \in \mathsf{L_A}$, and (i.3) $\pi^\infty_i \overset{a_i}{\hookrightarrow} \pi^\infty_{i+1} \in \Delta$. Condition (i.1) holds by construction of $\mathbf{P_I}$. Condition (i.2) holds trivially by definition of $\mathbf{P}$ (see Definition 3). Condition (i.3). Let us assume, by absurd, that $\pi^\infty_i \overset{a_i}{\hookrightarrow} \pi^\infty_{i+1} \notin \Delta$. This is possible when one of the conditions in Definition 2 is not satisfied. For construction of π^∞ then $\exists (i, a_i, i+1) \in \pi$ and consequently $\pi^\infty_i \vdash$ *Cond* of a_i. Condition *(a)* holds because each commitment's axiom is translated into a corresponding clause (see [20, Section 2.3]). Condition *(b)* holds because of clause `initiates` at Line 35 in Listing 1.1. Conditions *(b.1)* and *(b.2)* are verified respectively by clauses at Lines 45 and 47-49 of Listing 1.1. Therefore, $\pi^\infty_i \overset{a_i}{\hookrightarrow} \pi^\infty_{i+1} \in \Delta$.
ii. $inf(\tau) \cap \mathsf{G} \neq \emptyset$. Being π a legal path for $\mathbf{P_I}$ then there exits `state(`n, π_n, `Label`$_n$`)` such that *final* $\in$ Label$_n$, thus there are no active commitments in π_n. For construction of π^∞, $\pi^\infty_n \in \mathsf{G}$ and $\pi^\infty_n \in inf(\pi^\infty)$.
iii. $\forall c \in \mathsf{Cst}: T(\tau), \tau_0 \models_{LTL} c$. Being π a legal path for $\mathbf{P_I}$ then $\nexists i \in \pi$ such that `state(`i, π_i, `Label`$_i$`)` and *violation* $\in$ Label$_i$. Moreover, *pending* $\notin$ `state(`n, π_n, `Label`$_n$`)`. Thus, for construction of π^∞, $\nexists c \in \mathsf{Cst}$ s.t. $T(\pi^\infty), \pi^\infty_0 \not\models_{LTL} c$.

□

5 2CL Tool for Protocol Design and Analysis

Based on the described technical framework, we developed a tool which supports the user in two different ways: *(i)* it features two graphical editors for specifying the protocol actions and the constraints; *(ii)* it generates different kinds of graphs for supporting the user in the analysis of the possible interactions and in understanding which of them are legal. The system is realized as an Eclipse plug-in, available at the URL `http://di.unito.it/2cl`. The functionalities that the system supports can be grouped into three components: *design*, *reasoning* and *visualization* (see Fig. 2).

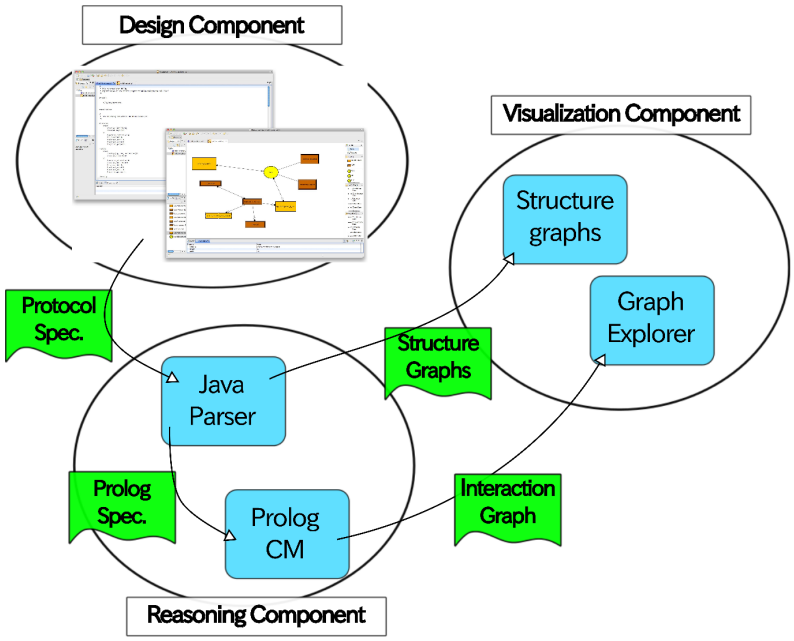

Fig. 2. Components and functionalities supplied by the system.

Design Component. The design component provides the tools that are necessary for defining the protocol. It supplies two editors: one for the definition of the actions and one for the definition of constraints (Fig. 3). The *action definition* editor is basically a text editor. The *regulative specification editor* allows the user to graphically define a set of constraints. Constraints are represented by drawing facts, connecting them with 2CL arrows (following the graphical representation of Table 1) or with logical connectives so as to design DNF formulas. The advantage of having a graphical editor is that it supplies a global view of constraints, thus giving the perception of the *flow* imposed by them, without actually specifying any rigid sequence (no-flow-in-flow principle [3]). Fig. 3 shows a snapshot of the constraint editor with a representation of the NetBill constraints. On the right the user can select the element to introduce in the graph. By editing the properties (bottom of the figure), instead, he/she can specify the name of facts and other graphical aspects.

Reasoning Component. The reasoning component consists of a Java Parser and of the Prolog implementation of the commitment machine described in Section 4. The former generates different kinds of graphs as well as the Prolog program corresponding to the protocol specification. The latter is the input of the Prolog implementation of the commitment machine for the generation of the labeled graph. As explained, the labeled graph represents all the possible interactions

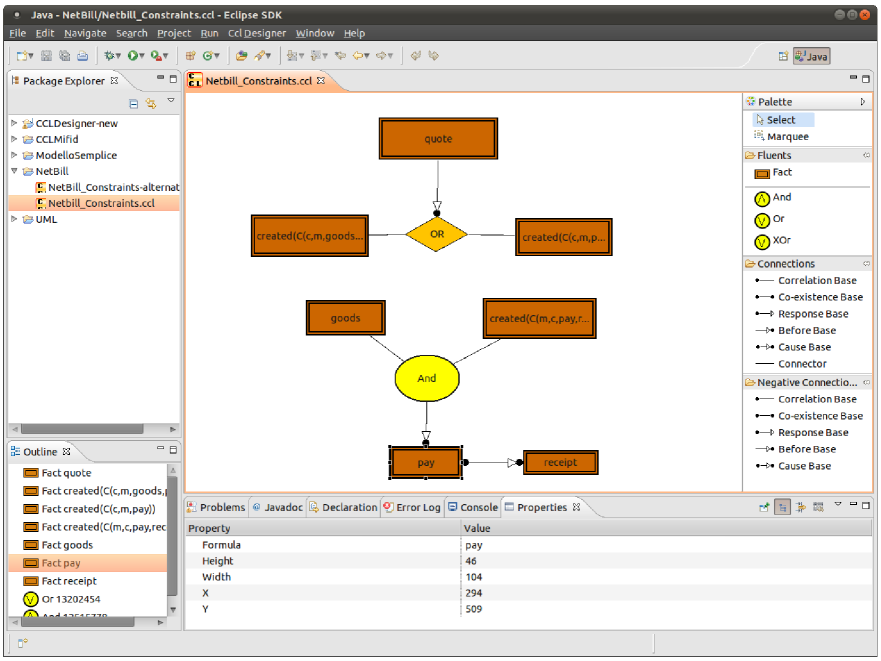

Fig. 3. Editor for constraint specification.

where each state is labeled according to the evaluation of the protocol constraints. The graphical conventions is: *(i)* a state of violation is represented as a red diamond, with an incoming red dashed arrow (e.g. states 54, 57, 108 in Fig. 4); *(ii)* a state in which there is a pending condition is yellow[3] (e.g. states 45, 53, 108); *(iii)* a state with a single outline, independently from the shape (e.g. 49, 57, 60), is a state that contains unsatisfied commitments; *(iv)* a state with a double outline, independently from the shape, does not contain active commitments (e.g. 41, 108). Graphical notations can be combined, e.g. a yellow diamond with single outline is a state where there are unsatisfied active commitments, where a constraint is violated and where there is a pending condition (e.g. 53, 57, 114).

Visualization Component. All the graphs produced by the reasoning component can be visualized as images. *Labelled graph*, however, can be explored by means of the tool *Graph Explorer*, which is implemented in Java and relies on iDot (Incremental Dot Viewer) – an open source project that uses the prefuse[4] visualization framework for Dot graph display. The Graph Explorer supplies different functionalities, like the visualization of the shortest path given a source

[3] Light gray states in black and white printing.
[4] `http://prefuse.org/`

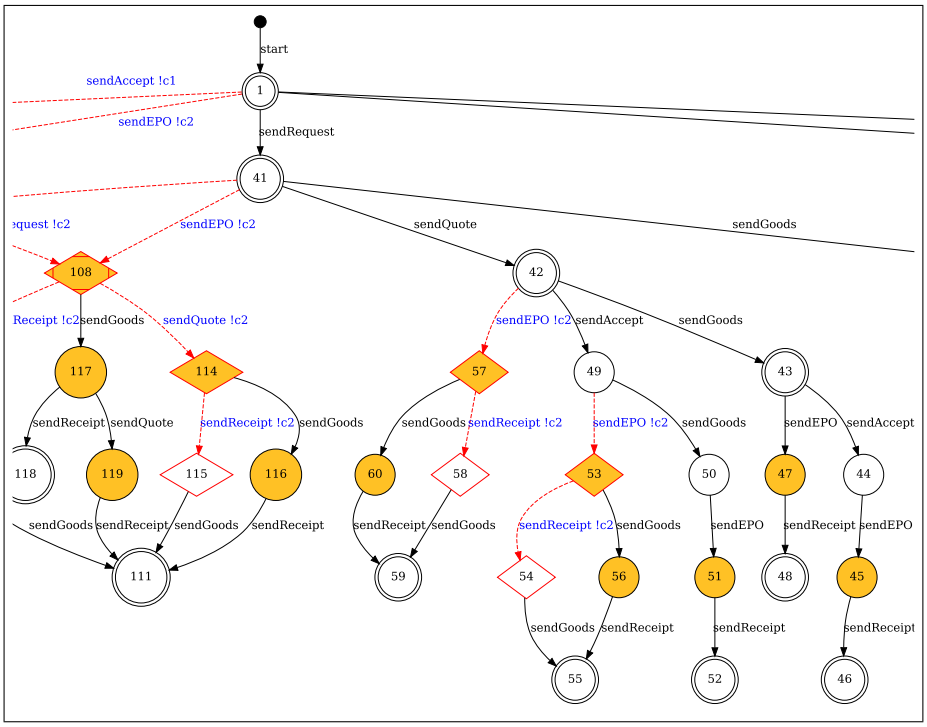

Fig. 4. Part of the labelled Graph for NetBill.

and a target state, and the visualization of legal (or illegal) paths only. The user can add or delete a node in a path; search a state starting from its label; and search all the states that contain a certain fact or commitment. Moreover, the tool allows the exploration of the graph one state at a time, by choosing which node to expand. Fig. 4 reports part of the labeled graph for NetBill.

Protocol Analysis. The tool can be used as a support in protocol analysis [5]. Particularly interesting is the possibility of exploring the labeled graph by means of the Graph Explorer, which can be used to predict whether performing a certain sequence of actions results in a violation and, in this case, if there is a way to return on a legal path. For what concerns the designer, it is not always easy, when specifying a protocol, to individuate which constraints to introduce but, with the help of the tool, it becomes easy to identify misbehaviors and revise the constraints so as to avoid them. Moreover, a designer can decide, by analyzing the graph, to modify the specification so as to regiment some of the patterns expressed as constraints, or to remove some of them. For instance, considering the running example, from Fig. 4 it is possible to infer that the protocol does not allow the customer to pay ($sendEPO$) before the merchant sends the goods. This is due to the constraint $\textsc{created}(\mathsf{C}(m, c, pay, receipt)) \wedge goods \multimap\!\bullet\ pay$. If

this behavior was not in the intention of the designer, he/she can discover it and, e.g., relax the *before* constraint ($\multimap\!\bullet$) transforming it into a *co-existence* ($\bullet\!\!-\!\!\bullet$). If, instead, that is exactly the desired behavior, one may decide to regiment *sendEPO* so as to enable the payment only after the goods have been sent.

The complete NetBill protocol encoding and the corresponding labeled graph together with further examples, like 2CL specifications of classical agent interaction protocols (CNet) and of real-life protocols (OECD guidelines and MiFID [5]) are available at `http://di.unito.it/2cl` (section Examples).

6 Related Work and Conclusions

This work provides an operational semantics of 2CL protocols [3, 4], based on an extension of the Generalized Commitment Machine [20], and describes a Prolog implementation of this formalization, where the constraint evaluation is performed thanks to state conditions rather than by considering paths. Our aim was to enrich commitment machines with a mechanism for constraint evaluation, in a way that is suitable to creating tools which are useful in application domains. The provided formalization allows the creation of compact and annotated graphs, which provide a global overview of the possible interactions, showing which are legal and which cause constraint (or commitment) violations. The aim was to support an implementation, which enables the verification of exposure to risk on the graph of the possible executions, and taking decisions concerning how to behave or to modify the protocol in order to avoid such a risk. Due to this aim, we decided to base our implementation on [22], rather than on formalizations which support, for instance, model checking. The reason is that this work already is along the same line of ours, the intent being to give a global view on desirable and undesirable states. Winikoff et al. [22], however, propose to cope with undesired paths or undesired final states by adding ad-hoc preconditions to the actions, or by adding active commitments to states that are desired not to be final. This, however, complicates the reuse and the adaptation of the specification to different domains. On the contrary, the proposal in [4] results to be easily adaptable and customizable so as to address different needs of different domains, and it also allows for the specification of more expressive patterns of interaction, given as 2CL constraints.

Concerning model checking, in [8] it is possible to find a proposal of a branching-time logic that extends CTL*, used to give a logical semantics to the operations on commitments. This approach was designed to perform verifications on commitment-protocol ruled interactions by exploiting symbolic model checking techniques. The properties that can be verified are those that are commonly checked in distributed systems: fairness, safety, liveness, and reachability. It would be interesting to integrate in this logical framework the 2CL constraints in order to combine the benefits of both approaches: on the one hand, the possibility to embed in the protocols expressive regulative specification, and, on the other hand, the possibility to exploit the logical framework to perform the listed verifications.

For what concerns the semantics of commitment protocols, the literature proposes different formalizations. Some approaches present an operational semantics that relies on commitment machines to specify and execute protocols [24, 23, 22]. Some others, like [12], use interaction diagrams, operationally specifying commitments as an abstract data type, and analyzing the commitment's life cycle as a trajectory in a suitable space. Further approaches rely on temporal logics to give a formal semantics to commitments and to the protocols defined upon them. Among these, [13] uses DLTL. All these approaches allow the inference of the possible executions of the protocol, but, differently than [4], all of them consider as the only regulative aspect of the protocol the regulative value of the commitments.

Acknowledgements

The authors would like to thank the reviewers for their valuable comments. This research was partially funded by "Regione Piemonte" through the project ICT4LAW.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
2. M. Baldoni and C. Baroglio. Some Thoughts about Commitment Protocols. In M. Baldoni, L. Dennis, V. Mascardi, and W. Vasconcelos, editors, *Post-Proc. of International Workshop on Declarative Agent Languages and Technologies, DALT 2012*, volume LNAI. Springer. In this volume.
3. M. Baldoni, C. Baroglio, and E. Marengo. Behavior-Oriented Commitment-based Protocols. In *Proc. of ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 137–142. IOS Press, 2010.
4. M. Baldoni, C. Baroglio, E. Marengo, and V . Patti. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Trans. on Int. Sys. and Tech., Spec. Iss. on Agent Communication*, 4(2), 2013.
5. M. Baldoni, C. Baroglio, E. Marengo, and V. Patti. Grafting Regulations into Business Protocols: Supporting the Analysis of Risks of Violation. In A. Antón, D. Baumer, T. Breaux, and D. Karagiannis, editors, *Forth International Workshop on Requirements Engineering and Law (RELAW 2011), held in conjunction with the 19th IEEE International Requirements Engineering Conference*, pages 50–59, Trento, Italy, August 30th 2011. IEEE Xplore.
6. F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment Tracking via the Reactive Event Calculus. In C. Boutilier, editor, *IJCAI*, pages 91–96, Pasadena, California, USA, July 2009.
7. A. K. Chopra and M. P. Singh. Constitutive Interoperability. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *Proc. of 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, volume 2, pages 797–804, Estoril, Portugal, May 2008. IFAAMAS.
8. M. El-Menshawy, J. Bentahar, and R. Dssouli. Verifiable Semantic Model for Agent Interactions Using Social Commitments. In M. Dastani, A. Seghrouchni El Fallah, J. Leite, and P. Torroni, editors, *LAnguages, methodologies and Development*

tools for multi-agent systemS (LADS 2009), volume 6039 of *LNCS*, pages 128–152, Torino, Italy, September 2010. Springer.
9. M. El-Menshawy, J. Bentahar, and R. Dssouli. Symbolic Model Checking Commitment Protocols using Reduction. In A. Omicini, S. Sardina, and W. Vasconcelos, editors, *Declarative Agent Languages and Technologies VIII*, Toronto, Canada, 2011. Springer.
10. E. A. Emerson. *Temporal and Modal Logic*, volume B. Elsevier, Amsterdam, The Netherlands, 1990.
11. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proc. of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*, pages 520–527, Melbourne, Australia, July 2003. ACM.
12. N. Fornara and M. Colombetti. A Commitment-Based Approach To Agent Communication. *Applied Artificial Intelligence*, 18(9-10):853–866, 2004.
13. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5(2):214–234, 2007.
14. A. J. I. Jones and M. Sergot. *On the Characterization of Law and Computer Systems: the Normative Systems Perspective*, pages 275–307. John Wiley & Sons, Inc., New York, NY, USA, 1994.
15. A. U. Mallya and M. P. Singh. Modeling Exceptions via Commitment Protocols. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 122–129, Utrecht, The Netherlands, July 2005. ACM.
16. E. Marengo. *2CL Protocols: Interaction Patterns Specification in Commitment Protocols*. PhD thesis, Università degli Studi di Torino, Research Doctorate in Science and High Technology, Specialization in Computer Science, October 2012. `http://www.di.unito.it/~emarengo/Thesis.pdf`.
17. E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and M. P. Singh. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *AAMAS*, volume 1–3, pages 467–474, Taipei, Taiwan, May 2011. IFAAMAS.
18. J.R. Searle. *The construction of social reality*. Free Press, New York, 1995.
19. M. P. Singh. An Ontology for Commitments in Multiagent Systems. *Artificial Intelligence and Law*, 7(1):97–113, 1999.
20. Munindar P. Singh. Formalizing Communication Protocols for Multiagent Systems. In M. M. Veloso, editor, *IJCAI*, pages 1519–1524, Hyderabad, India, January 2007. AAAI Press.
21. Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.
22. M. Winikoff, W. Liu, and J. Harland. Enhancing Commitment Machines. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proc. of the Second International Workshop on Declarative Agent Languages and Technologies II (DALT 2004)*, volume 3476 of *LNCS*, pages 198–220, New York, NY, USA, July 2005. Springer.
23. P. Yolum and M. P. Singh. Designing and Executing Protocols Using the Event Calculus. In *Agents*, pages 27–28, New York, NY, USA, 2001. ACM.
24. P. Yolum and M. P. Singh. Commitment Machines. In J.-J. Ch. Meyer and M. Tambe, editors, *Proc. of the 8th International Workshop on Intelligent Agents VIII (ATAL 2001)*, volume 2333 of *LNCS*, pages 235–247, Seattle, WA, USA, August 2002. Springer.