Monitoring Time-Aware Social Commitments with Reactive Event Calculus

Federico Chesani, Paola Mello, Marco Montali, Paolo Torroni
DEIS - University of Bologna*
V.le Risorgimento, 2
40136 Bologna - Italy
{federico.chesani, paola.mello, marco.montali, paolo.torroni}@unibo.it

Abstract

Despite their dynamic nature, social commitments have been rarely used for monitoring purposes, and few attention has been paid to the relationship between commitments and the temporal dimension and to the corresponding run-time verification. Building on previous work, we present a declarative axiomatization of time-aware social commitments, extending their basic life cycle with time-related transitions and with compensation mechanisms. The formalization is based on a reactive version of the Event Calculus, which supports the monitoring of the commitments evolution during a system's execution, checking if the interacting agents are honoring them or not.

1 Introduction

Social commitments have been increasingly applied to capture normative aspects and interaction protocols in open Multiagent Systems [Yolum and Singh, 2002] and, more recently, to provide declarative abstractions for modeling Business Protocols and Service Oriented Systems [Singh *et al.*, 2009]. The basic idea is to offer the abstraction of *commitment* to model, at the social level, the mutual obligations established by the interacting parties: during the interaction, an agent becomes *debtor* towards a *creditor* agent to bring about some *property*. Each execution of the system under study can be characterized in terms of how the involved commitments evolve over time due to the occurrence of events. Such events, generated by the interacting agents, implicitly lead to manipulate commitments, causing them to change state. The state machine, containing the states in which a commitment can be and the operations associated to state transitions, is called *commitment life cycle*.

Despite their dynamic nature, social commitments have been rarely used for run-time verification purposes, i.e., for monitoring the system's executions and track the evolution of mutual obligations, checking if the interacting agents are honoring them or not. We argue that this lack is mainly due to the absence of monitoring frameworks able to capture the commitment's life cycle and, at the same time, to provide formal guarantees about their operational functioning (such as soundness, completeness and termination).

In the last few years, we have developed a computational logic-based reactive form of Event Calculus[1], called $\mathcal{REC}$ [Chesani *et al.*, 2009], which supports the modeling of Event Calculus specifications and carries out run-time, dynamic reasoning, computing and reporting back to the user the evolution of fluents caused by the events occurred so far. A $\mathcal{REC}$ specification is obtained by composing a general specification formalizing the calculus with a user-specified knowledge base $\mathcal{KB}$, made up of a set of Horn clauses relating specific events and fluents (modeling e.g. that a fluent is initiated by a certain event).

In [Chesani *et al.*, 2009], we have discussed the formal properties of $\mathcal{REC}$, showing that it guarantees soundness, completeness and termination (for the last two properties, provided that $\mathcal{KB}$ is *acyclic* [K. R. Apt and M. Bezem, 1990]), and that it generates irrevocable answers when employed for monitoring. We have also described how $\mathcal{REC}$ can be exploited to perform run-time monitoring of commitment-based interactions, relying on the Event Calculus-based formalization of the commitment life cycle proposed in [Yolum and Singh, 2002].

Since commitments evolve over time, the temporal dimension plays a key role and can be further investigated to extend their expressiveness, e.g. to introduce the notion of a deadline by which some commitment must be satisfied. The addition of quantitative temporal aspects in commitments modeling has been first addressed in [Mallya and Huhns, 2003] and then in [Torroni *et al.*, 2009], where $\mathcal{REC}$ is applied for tracking commitments augmented with temporal constraints, handling their violation and compensation.

In this work, we further develop such a line of research, reconciling the treatment of the time-aware commitments proposed in [Mallya and Huhns, 2003; Torroni *et al.*, 2009] with the original commitment life

*This work has been partially supported by the FIRB Project *TOCAI.IT*.

[1]We assume the reader is familiar with the Event Calculus and its ontology [Kowalski and Sergot, 1986; Shanahan, 1999].

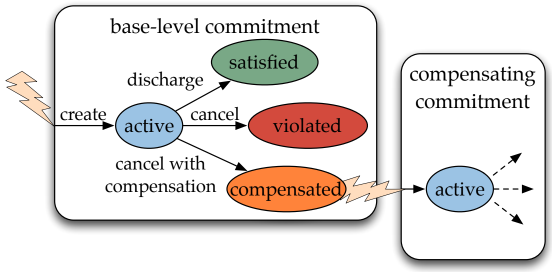

Figure 1: Base-level commitment life cycle extended with compensation.

cycle formalized in [Yolum and Singh, 2002], suitably extending it to handle the satisfaction and violation of time-aware commitments and to accommodate compensation mechanisms. The $\mathcal{REC}$ axiomatization of the extended life cycle brings two main advantages: on the one hand, all the formal properties proven for $\mathcal{REC}$ are inherited; on the other hand, time-aware commitment specifications can be directly monitored, relying on the operational counterpart of $\mathcal{REC}$.

The paper is organized as follows. In Section 2, we introduce time-aware commitments and discuss how the commitments life cycle can be extended to deal with them. In Section 3, we propose a $\mathcal{REC}$-based formalization of the extended life cycle. The potentialities and feasibility of our approach are shown in Section 4, by means of an effective example. Conclusion follows.

2 Extending the Commitment Life Cycle

A (base-level) social commitment relates three different entities: a *debtor* agent, which is committed towards a *creditor* agent to bring about a *property* desired by the creditor. By identifying these three entities with x, y and p respectively, this kind of commitment is denoted by $\mathsf{C}(x, y, prop(p))$, and will be called *basic commitment* throughout the paper. During the interaction, the events generated by the agents implicitly lead to execute *operations* on commitments, manipulating them by affecting their status. For the sake of space, we will focus only on the three fundamental operations of *create*, *discharge* and *cancel* applied to base-level commitments. The other operations, as well as the treatment of conditional commitments, can be seamlessly introduced in our framework.

2.1 Life Cycle and Compensation

The commitment life cycle targeted in this work, taking inspiration from [Singh *et al.*, 2009], is illustrated in Figure 1. At the beginning of execution, the commitment does not exist (or, alternatively, can be considered in a *null* state). The commitment starts to exist when a *create* operation is executed, causing a commitment's transition to the *active* state: from now on, the debtor agent becomes committed to bring about the involved property. An active commitment makes a further transition when it is manipulated by a *discharge* or *cancel* operation. In the first case, the commitment has been honored by the debtor, and the new state is therefore *satisfied*; in the latter case, a problem or exception occurred, leading to a *violation* of the commitment.

In addition to these "standard" transitions and states, we also support a further situation, in which the commitment is canceled but a new commitment (called *compensating commitment*) is created to handle (compensate) the violation, trying to recover it inside the interaction protocol. If the user defines that commitment c_2 represents a compensation for commitment c_1, the impact of canceling c_1 is twofold: instead of becoming violated, c_1 makes a transition to the *compensated* state while, at the same time, c_2 is created, becoming active[2].

Let us now focus on the semantics of operations in terms of events and commitments' status. Operations are partly specified at the domain-dependent level, and partly in a domain-independent fashion. In particular, the creation of a compensating commitment is always defined in terms of the cancelation of the compensated commitment, while the creation of a "normal" commitment is user-defined by means of a domain-specific event. A similar dichotomy exists for the discharge and cancel operation. On the one hand, the semantics of discharge is defined in a domain-independent manner, and states that a commitment is discharged by an event if such an event has the effect of bringing about the commitment's property; on the other hand, the cancelation of a commitment is caused by the generation of a specific domain-dependent event during the interaction.

2.2 Time-Aware Commitments

This analysis points out a limitation of the presented life cycle: an active commitment which is not explicitly canceled, and whose property is never made true by the debtor agent, will continue to persist indefinitely in the active status. It would be therefore desirable to provide suitable abstractions for modeling temporal constraints regulating *when* the commitment's property must be made true by the debtor agent. To realize this objective, the temporal dimension must be introduced inside the specification of commitments, making them time-aware.

By relying on [Mallya and Huhns, 2003; Torroni *et al.*, 2009], we propose two classes of time-aware commitments:

- $\mathsf{C}(x, y, prop(e(t_1, t_2), p)))$ represents an *existential commitment*, where x is committed to bring about p inside the time interval $[t_1, t_2]$[3];

[2]Other choices could be taken to model the active-compensated transition; for example, a violated commitment could be considered compensated only when the compensating commitment has been satisfied.

[3]A basic commitment can be therefore considered as a special case of existential commitment, where t_1 is the time at which the commitment is created, and $t_2 = \infty$.

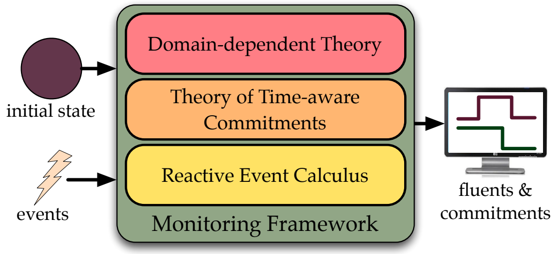

Figure 2: Monitoring Framework for Time-Aware Commitments.

- $\mathsf{C}(x, y, prop(u(t_1, t_2), p)))$ represents an *universal commitment*, where x is committed to maintain p valid along the whole time interval $[t_1, t_2]$.

Being the properties involved in such commitments time-dependent, not only their discharge, but also their cancelation, can be defined in a domain-independent manner.

In particular, $\mathsf{C}(x, y, prop(e(t_1, t_2), p)))$ becomes satisfied if x generates an event ev at a time $t \in [t_1, t_2]$, such that ev makes p true. Conversely, let us suppose that the existential commitment is still active after t_2; this attests that x has not made p true inside $[t_1, t_2]$, and therefore the commitment must be canceled due to a violation of the temporal constraints.

The case of an universal commitment is the opposite. If $\mathsf{C}(x, y, prop(u(t_1, t_2), p)))$ is active at a time $t \in [t_1, t_2]$, and p is not true at time t, then the validity of p along the whole interval $[t_1, t_2]$ has been "broken", and the commitment is violated. Conversely, let us suppose that the universal commitment is still active after t_2; this attests that the commitment has not been canceled before, i.e., that the validity of p has been maintained during $[t_1, t_2]$, and therefore the commitment is satisfied.

3 Formalizing the Life Cycle of Time-Aware Commitments in $\mathcal{REC}$

We now present how the commitment life cycle described in Section 2 can be formalized in $\mathcal{REC}$. As pointed out in the introduction, a $\mathcal{REC}$ theory is a knowledge base $\mathcal{KB}$ composed by a set of Horn clauses which bind together events and fluents. It is worth noting that such a $\mathcal{KB}$ relies on the "standard" Event Calculus ontology, making other Event Calculus reasoners seamlessly applicable as well.

In the case of time-aware commitments, the theory itself is composed by two different knowledge bases, as shown in Figure 2:

1. The domain-independent theory formalizing the life cycle of time-aware commitments. It relates the initiation/termination of the commitments' status with operations, and defines the domain-independent semantics of operations.
2. The theory representing a specific domain. It includes domain-dependent fluents and commitments and their relationship with specific events.

In this Section, we will focus on the general formalization of time-aware commitments. An example of domain-dependent theory will be instead presented in Section 4.

The $\mathcal{REC}$-based axiomatization of time-aware commitments is inspired by [Yolum and Singh, 2002], where Event Calculus is employed to provide a formalization of the commitment life cycle. In the Event Calculus setting, properties are represented by fluents, whose validity evolve over time as event occurs. Therefore, the central concept of "bringing about some property p" is translated as "*initiating* fluent p", while the validity/truth of p at a given time is expressed by stating that fluent p *holds at* that time.

There are two main differences between the formalization proposed by Yolum and Singh and ours. First of all, while in [Yolum and Singh, 2002] commitments are directly mapped onto fluents (initiated through the *create* operation and terminated by the *cancel/discharge* operations), we map each possible commitment's status to a separate fluent $status/2$, where $status(c, s)$ expresses that commitment c is in state s. In this way, commitment's states are reified and can be explicitly reported to the user by the monitoring framework, as well as involved in the domain-dependent theory (e.g. to state that a commitment c is created by event ev if another commitment c_2 is currently active). The second main difference is that our formalization also deal with time-aware commitments and their compensation.

Since the knowledge base expressing the extended life cycle is a general theory, all the involved events, agents and properties are variable: their grounding will be defined by the domain-dependent theory, together with the concrete events characterizing the monitored execution of the system under study. The first five axioms characterize the commitment's life cycle transitions depicted in Figure 1 in terms of the corresponding operations, while the remaining axioms capture the domain-independent semantics of operations, as informally described in Section 2.2.

Axiom 1 (Status query) *A commitment C is active/satisfied/violated/compensated at time T if the corresponding status fluent holds at time T. For example, for the* active *state we have:*

$$active(C, T) \leftarrow holds\_at(status(C, active), T).$$

Axiom 2 (Active state) *A commitment becomes* active *when it is created by the debtor agent through an event occurrence:*

$$initiates(E, status(\mathsf{C}(X, Y, P), active), T) \leftarrow create(E, X, \mathsf{C}(X, Y, P), T).$$

The active state is left when the commitment is discharged or canceled by another event occurrence:

$$terminates(E, status(\mathsf{C}(X, Y, P), active), T) \leftarrow discharge(E, X, \mathsf{C}(X, Y, P), T).$$
$$terminates(E, status(\mathsf{C}(X, Y, P), active), T) \leftarrow cancel(E, X, \mathsf{C}(X, Y, P), T).$$

Axiom 3 (Active-discharged transition) *A commitment makes a transition from the* active *status to the* satisfied *one when it is discharged by the debtor agent through an event occurrence:*

$$initiates(E, status(\mathsf{C}(X,Y,P), satisfied), T) \leftarrow discharge(E, X, \mathsf{C}(X,Y,P), T).$$

Axiom 4 (Active-canceled transition) *A commitment C makes a transition from the* active *status to the* violated *one if it is canceled by an event occurring at time T, and no compensating commitment has been defined for C at T:*

$$initiates(E, status(\mathsf{C}(X,Y,P), violated), T) \leftarrow \neg compens(\mathsf{C}(X,Y,P), \_, T) \wedge cancel(E, X, \mathsf{C}(X,Y,P), T).$$

It is worth noting that the definition of compensating commitments is done at the domain-dependent level through the *compens*/3 predicate, where $compens(C_1, C_2, T)$ states that commitment C_1 can be compensated by means of C_2 at time T.

Axiom 5 (Compensation) *A commitment C makes a transition from the* active *status to the* compensated *one if it is canceled by an event occurring at time T, and a compensating commitment has been defined for C at T:*

$$initiates(E, status(\mathsf{C}(X,Y,P), compensated), T) \leftarrow compens(\mathsf{C}(X,Y,P), \_, T) \wedge cancel(E, X, \mathsf{C}(X,Y,P), T).$$

At the same time, the compensating commitment becomes active:

$$initiates(E, status(\mathsf{C}(W,Z,P_2), active), T) \leftarrow compens(\mathsf{C}(X,Y,P), \mathsf{C}(W,Z,P_2), T) \wedge cancel(E, X, \mathsf{C}(X,Y,P), T).$$

Axiom 6 (Discharge) *An active* basic commitment *C is discharged by the occurrence of an event if the event brings about the C's property:*

$$discharge(E, X, \mathsf{C}(X,Y,prop(P)), T) \leftarrow active(\mathsf{C}(X,Y,prop(P)), T) \wedge initiates(E, P, T).$$

An active existential commitment *C is discharged by an event if the event occurs inside the time interval targeted by C and brings about the commitment's property:*

$$discharge(E, X, \mathsf{C}(X,Y,prop(e(T_1,T_2),P))), T) \leftarrow active(\mathsf{C}(X,Y,prop(e(T_1,T_2),P))), T) \wedge T \geq T_1 \wedge T \leq T_2 \wedge initiates(E, P, T).$$

A universal commitment *is automatically discharged after its targeted time interval if it is still active (this means that it has not been canceled in between, attesting that the property has been maintained valid throughout):*

$$discharge(E, X, \mathsf{C}(X,Y,prop(u(T_1,T_2),P))), T) \leftarrow active(\mathsf{C}(X,Y,prop(u(T_1,T_2),P))), T) \wedge T \geq T_2.$$

Axiom 7 (Cancel) *The cancelation of a basic commitment is user-defined. An* existential commitment *is automatically canceled after its targeted time interval if it is still active (this means that it has not been discharged before, i.e. the debtor agent has not brought about the property when expected):*

$$cancel(E, X, \mathsf{C}(X,Y,prop(e(T_1,T_2),P))), T) \leftarrow active(\mathsf{C}(X,Y,prop(e(T_1,T_2),P))), T) \wedge T \geq T_2.$$

An active universal commitment *is canceled during its targeted time interval as soon as it is detected that the commitment's property is not holding:*

$$cancel(E, X, \mathsf{C}(X,Y,prop(u(T_1,T_2),P))), T) \leftarrow active(\mathsf{C}(X,Y,prop(u(T_1,T_2),P))), T) \wedge T \geq T_1 \wedge T \leq T_2 \wedge \neg holds\_at(P, T).$$

4 A Car Rental Example

In this section, we introduce and discuss a simple but effective example, which shows the potentialities of time-aware commitments and of the underlying $\mathcal{REC}$ monitoring framework.

A contract formalizes the mutual obligations between a customer and an agency when a car is rented. In particular, the following statements are included in the contract:

(S1) the customer is committed of taking the car back to the car rental agency within the agreed number of days;

(S2) the agency, in turn, guarantees that the rented car will not break down for the first three days;

(S3) if the rented car breaks down before the third day has elapsed, the agency promises a "1-day" immediate replacement;

(S4) in case of a car replacement, the customer receives two more rental extra-days for free.

To formalize this contract in terms of (time-aware) commitments and enable monitoring, the following steps must be followed:

A. Identification of the events that can be extracted from the car rental agency's information system.

B. Elicitation of the fluents which characterize the states of affairs of the running system.

C. Binding between events and fluents (i.e., definition of how the events affect fluents through *initiates* and *terminates* predicates).

D. Elicitation of the commitments formalizing the statements included in the contract
   - using (some of the) fluents identified during step B to represent the "property part";
   - introducing existential/universal temporal constraints if needed;
   - defining their operations (*create*, *discharge*, *cancel*, *compensation*) in terms of the events identified during step A.

A. Events Identification We suppose that the information system of each car rental agency collects and stores the relevant events characterizing the evolution of each rental:

$rent(C, A, Car, N)$ - customer C rents a car Car at the agency A for N days;

$drive\_back(C, Car, A)$ - customer C drives Car back to the agency A;

$break\_down(Car)$ - Car breaks down;

$replace(A, C, Car_{old}, Car_{new})$ - agency A takes back Car_{old} from the customer C and substitutes it with Car_{new}.

Beside these domain-dependent events, we also suppose that three further events *start*, *complete* and *tick* are delivered to the monitoring framework. The first two events are used to respectively alert $\mathcal{REC}$ that the execution has begun/finished; the *tick* event, instead, is used to inform $\mathcal{REC}$ about the current time: $\mathcal{REC}$ itself has no explicit notion of the time flow - it reacts to each incoming event updating the status of fluents and commitments and then waiting until a new event occurs.

B. Fluents Elicitation The system is characterized by the status of the cars owned by the agency:

$in\_agency(A, Car)$ states that Car is parked inside agency A;

$great\_car(Car)$ states that Car is working;

$hired(C, Car, D)$ states that Car is being rented by customer C until date D;

$car\_replaced(Car)$ states that Car has been replaced.

C. Events-Fluents Binding Fluents are affected by the events in the following way. First of all, when a customer rents a car, the car is no more *in agency* and becomes *hired* until the date obtained by the current date plus the chosen number of days:

$$terminates(rent(C, A, Car, N), in\_agency(A, Car), T).$$
$$initiates(rent(C, A, Car, N), hired(C, Car, D), T) \leftarrow D \text{ is } T + N.$$

When the customer drives back to the agency, then the car is no more *hired* and it starts to be *in agency* again:

$$terminates(drive\_back(C, Car, A), hired\_until(C, Car, D), T).$$
$$initiates(drive\_back(C, Car, A), in\_agency(A, C), T).$$

When the car breaks down, it is no more a *great car*:

$$terminates(break\_down(Car), great\_car(Car), T).$$

When the agency replaces a car, it becomes *replaced*:

$$initiates(replace(A, C, Car_1, \_), car\_replaced(Car_1), T).$$

Furthermore, the replaced car is brought back to the agency, while the new one is carried out from the agency and given to the customer:

$$initiates(replace(A, C, Car_1, \_), in\_agency(A, Car_1), T).$$
$$terminates(replace(A, C, \_, Car_2), in\_agency(A, Car_2), T).$$

Finally, car's replacement ceases the hiring of the old car, and causes the new car to be hired. Following the prescription of the contract Statement S4, the new car is hired until the date fixed for the old one plus two extra-days:

$$terminates(replace(A, C, Car_1, \_), hired(C, Car_1, D), T).$$
$$initiates(replace(A, C, Car_1, Car_2), hired(C, Car_2, D), T) \leftarrow holds\_at(hired(C, Car_1, D_{old}), T), D \text{ is } D_{old} + 2.$$

D. Commitments Elicitation We now rephrase Statements S1, S2 and S3 in terms of time-aware commitments. Statement S1 is a commitment which is created when the customer rents a car, and is associated to a *deadline*. The deadline can be expressed by means of an existential temporal constraint imposing that the commitment's property – "bringing the car back" – must be initiated by the customer between the time at which the commitment is created and the agreed number of days. The property corresponds to the *in_agency* fluent, while the value of the deadline can be obtained as done for the *hired* fluent. We therefore have:

$$create(rent(C, A, Car, N), C, \mathsf{C}(C, A, prop(e(T, T_e), in\_agency(A, Car)))), T) \leftarrow T_e \text{ is } T + N, holds\_at(in\_agency(A, C), T).$$

Statement S2 can be represented by an universal commitment, also created when the customer rents a car. Indeed, guaranteeing that the rented car will not break down for three days can be formalized by stating that the *great_car* fluent related to the car should continuously hold for such three days:

$$create(rent(C, A, Car, N), A, \mathsf{C}(A, C, prop(u(T, T_e), great\_car(Car)))), T) \leftarrow T_e \text{ is } T + 3.$$

Finally, Statement S3 refers to a situation in which the commitment introduced by Statement S2 has been violated, and can be therefore formalized as a compensating commitment. Such a commitment is existential, and states that the agency is committed to bring about the *car_replaced* fluent within one day from the cancelation of the compensated commitment. The compensation can then be expressed by relating these two commitments with the *compens*/2 predicate:

$$compens(\mathsf{C}(A, C, prop(u(T_s, T_e), great\_car(Car)))), \mathsf{C}(A, C, prop(e(T, T_r), car\_replaced(Car)))), T) \leftarrow T_r \text{ is } T + 1.$$

4.1 Monitoring Instance

Figure 3 depicts the result computed by $\mathcal{REC}$ when reasoning upon the formalization of the presented example in a specific case, which captures the interaction between a car rental agency *ag* and customer *ian*. A monitoring instance is characterized by a (growing) execution trace collecting all the events occurred so far, and by an initial state, describing which fluents initially hold. In our case, *ag* has initially two cars in the agency, the initial state is then described by:

$$initially\_holds(in\_agency(ag, bo123)).$$
$$initially\_holds(in\_agency(ag, bo124)).$$

As reported in the bottom part of Figure 3 (considering a day as the time unit), the execution under study models a situation in which *ian* rents car *bo*123 from *ag*, but the car breaks out during the guarantee; a compensation must be therefore handled by *ag*, which however misses the deadline, replacing the car only after 4 days and causing a violation of the compensating

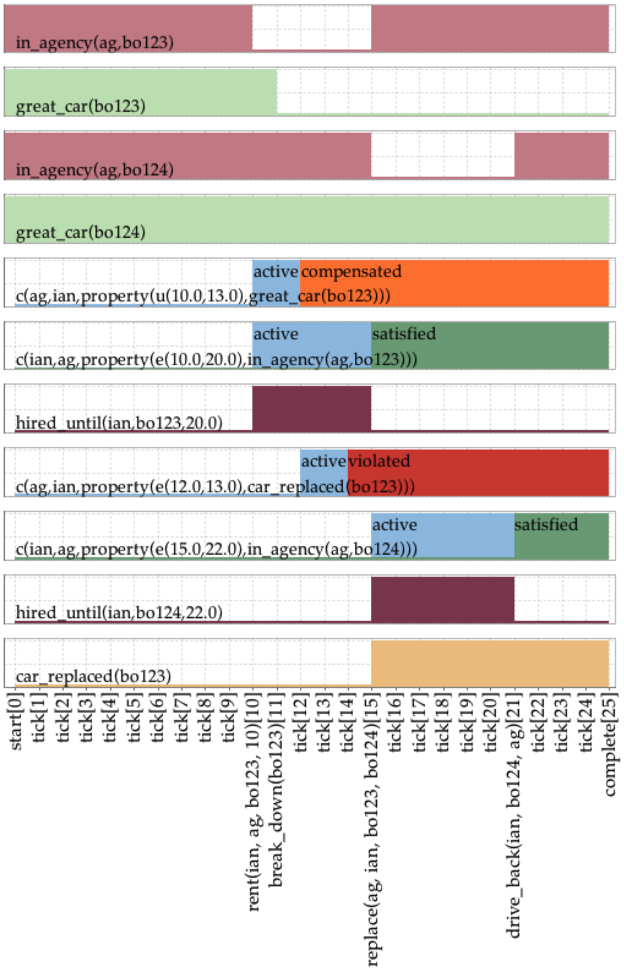

Figure 3: Sample outcome shown by jREC.

commitment. The commitments having *ian* as debtor are instead both satisfied: the first due to the replacement of car *bo*123, the second because *ian* drives car *bo*124 back to *ag* one day before the expected date.

$\mathcal{REC}$ took a total time of 2.85 seconds to reason upon the entire execution trace on a MacBook Pro Intel CoreDuo 2.66 GHz machine.

5 Conclusion

We have proposed an extended commitment life cycle accommodating time-aware social commitments (commitments whose involved property is associated to temporal constraints) and dealing with their compensation. We have formalized such an extended life cycle as an Event Calculus theory, using a reactive version of the Event Calculus, called $\mathcal{REC}$, for monitoring the executions of the system under study, tracking how the status of commitments evolve as events occur. Since the presented theory is acyclic, the fact that also the domain-dependent theory is acyclic is a necessary and sufficient condition for ensuring that the operational counterpart of $\mathcal{REC}$, used to effectively carry out the monitoring task, is sound and complete and guarantees termination [Chesani *et al.*, 2009].

A Java-based tool called jREC is currently being implemented to wrap $\mathcal{REC}$, providing a generic event acquisition module for delivering the occurring events to the monitoring framework, and equipping it with a GUI able to give a constantly updated snapshot about the status of fluents and commitments. Figure 3 shows how the outcome produced by $\mathcal{REC}$ when reasoning upon the example described in Section 4 is graphically reported to the user inside jREC.

Other ongoing work is concerned with an extensive experimental evaluation of the proposed framework, to compare its performance with other state of the art Event Calculus reasoners. We are also investigating how commitment-based interaction models could be integrated with declarative, constraint-based Business Process specifications. A first investigation can be found in [Chesani *et al.*, 2010].

References

[Chesani *et al.*, 2009] F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment tracking via the reactive event calculus. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 91–96, 2009.

[Chesani *et al.*, 2010] F. Chesani, P. Mello, M. Montali, S. Storari, and P. Torroni. On the Integration of Declarative Choreographies and Commitment-based Agent Societies into the SCIFF Logic Programming Framework. *Special Issue of Multiagent and Grid Systems*, 6(2), 2010.

[K. R. Apt and M. Bezem, 1990] K. R. Apt and M. Bezem. Acyclic Programs. In *Logic Programming*, pages 617–633. MIT Press, 1990.

[Kowalski and Sergot, 1986] R. A. Kowalski and M. Sergot. A Logic-Based Calculus of Events. *New Generation Computing*, 4(1):67–95, 1986.

[Mallya and Huhns, 2003] A. U. Mallya and M. N. Huhns. Commitments Among Agents. *IEEE Internet Computing*, 7(4), 2003.

[Shanahan, 1999] M. Shanahan. The Event Calculus Explained. In M. Wooldridge and M. M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, volume 1600 of *LNCS*, pages 409–430. Springer Verlag, 1999.

[Singh *et al.*, 2009] M. P. Singh, A. K. Chopra, and N. Desai. Commitment-Based Service-Oriented Architecture. *IEEE Computer*, 42(11), 2009.

[Torroni *et al.*, 2009] P. Torroni, F. Chesani, P. Mello, and M. Montali. Social Commitments in Time: Satisfied or Compensated. In *Post-proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT 2009)*, 2009.

[Yolum and Singh, 2002] P. Yolum and M. P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning Using Commitments. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, pages 527–534. ACM Press, 2002.