Cybernetics and Systems: An International Journal

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ucbs20

MONITORING TIME-AWARE COMMITMENTS WITHIN AGENT-BASED SIMULATION ENVIRONMENTS

Federico Chesani [a], Paola Mello [a], Marco Montali [b] & Paolo Torroni [a]

[a] DEIS, University of Bologna, Bologna, Italy

[b] KRDB Research Centre, Free University of Bozen-Bolzano, Bolzano, Italy

Available online: 03 Oct 2011

To cite this article: Federico Chesani, Paola Mello, Marco Montali & Paolo Torroni (2011): MONITORING TIME-AWARE COMMITMENTS WITHIN AGENT-BASED SIMULATION ENVIRONMENTS, Cybernetics and Systems: An International Journal, 42:7, 546-566

To link to this article: http://dx.doi.org/10.1080/01969722.2011.610711

Cybernetics and Systems: An International Journal, 42:546–566

ISSN: 0196-9722 print/1087-6553 online
DOI: 10.1080/01969722.2011.610711

Monitoring Time-Aware Commitments Within Agent-Based Simulation Environments

FEDERICO CHESANI[1], PAOLA MELLO[1], MARCO MONTALI[2], and PAOLO TORRONI[1]

[1]*DEIS, University of Bologna, Bologna, Italy*
[2]*KRDB Research Centre, Free University of Bozen-Bolzano, Bolzano, Italy*

Despite their dynamic nature, social commitments have rarely been used for monitoring purposes. Little 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 compensation mechanisms. The formalization is based on a reactive version of the event calculus, able to monitor the commitment's evolution during a system's execution, to check whether the interacting agents are honoring them or not. The resulting monitoring framework can be used in the context of agent-based simulation, either to dynamically evaluate whether a running simulation is compliant with a commitment-based contract or to provide useful information to the interacting agents, helping them to behave in a compliant manner.

KEYWORDS agent-based simulation, commitments, monitoring, multi agent systems, temporal reasoning

INTRODUCTION

Social commitments have been increasingly applied to capture normative aspects and interaction protocols in open multi-agent systems (Yolum and Singh 2002) and, more recently, to provide declarative abstractions for modeling business protocols and service-oriented systems. The basic idea is to

The authors thank Alessandro Zorzi for the experimental evaluation of REC, carried out as part of his M.Eng. thesis.

Address correspondence to Federico Chesani, DEIS, University of Bologna, Ville Risorgimento, 2, Bologna 40136, Italy. E-mail: federico.chesani@unibo.it

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 toward 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 manipulating commitments, causing them to change state. The state machine of the possible commitment's states and of the operations associated with state transitions is called a *commitment life cycle*.

Despite their dynamic nature, social commitments have been rarely used for run-time verification purposes; that is, for monitoring the system's executions and tracking the evolution of mutual obligations to check whether 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 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 (EC; Kowalski and Sergot 1986) called REC (Reactive Event Calculus) (Chesani et al. 2009, 2010), which supports the modeling of EC specifications and carries out run-time, dynamic reasoning, computing, and reporting back to the user the evolution of fluents caused by the events that have occurred so far. REC is inspired by cached EC (Chittaro and Montanari 1996), for which a theoretical investigation concerning the spatial and temporal complexity has been carried out, proving that it guarantees better performance than classical EC when reasoning upon a growing execution trace. An REC specification is obtained by composing a general specification formalizing the calculus with a user-specified knowledge base (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. (2010), we discussed the formal properties of REC, showing that it guarantees soundness, completeness, and termination (for the last two properties, provided that KB is acyclic and bounded; Apt and Bezem [1990]) and that it generates irrevocable answers when employed for monitoring. We have also described how REC can be exploited to perform run-time monitoring of commitment-based interactions, relying on the EC-based formalization of the life cycle proposed by Yolum and Singh (2002).

Because commitments evolve over time, the temporal dimension plays a key role and can be further investigated to extend their expressiveness; for example, to introduce the notion of a deadline by which some commitment must be satisfied. The addition of quantitative temporal aspects in commitments modeling was first addressed by Mallya and Huhns (2003) with a branching time logic with quantitative time constraints and then by Torroni et al. (2009), in which REC was 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) and Torroni et al. (2009) with the original commitment life cycle formalized by Yolum and Singh (2002), suitably extending it to handle the satisfaction and violation of time-aware commitments and to accommodate compensation mechanisms. The REC axiomatization of the extended life cycle has two main advantages: on the one hand, all of the formal properties proven for REC are inherited; on the other hand, time-aware commitment specifications can be directly monitored, relying on the operational counterpart of REC.

We then show how the resulting monitoring framework can be exploited in the context of agent-based simulation. On the one hand, the REC monitoring framework can rely on the underlying simulation environment to obtain all of the relevant occurring events and consequently infer the status of commitments. On the other hand, the simulation framework can exploit REC to verify whether the simulated agent interactions are compliant with a commitment specification. Finally, the outcome of the commitment monitoring framework can be used by the agents themselves: They can query REC and obtain valuable information that help them to behave in a compliant manner.

The article is organized as follows. In the following section, we introduce time-aware commitments and discuss how their life cycle can be extended to deal with them. In the next section, we propose an REC-based formalization of the extended life cycle. The potentialities and feasibility of such an extended life cycle are then shown, by means of an effective example. In the following section the time-aware commitment monitoring framework is positioned in the context of agent-based simulation. Finally, we show how an agent-based framework simulating the example presented in the article can take advantage of our approach.

EXTENDING THE COMMITMENT LIFE CYCLE

A (base-level) social commitment relates three different entities: a debtor agent, which is committed to 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 $C(x, y, prop(p))$ and will be called *basic commitment* throughout the article. During the interaction, the events generated by the agents implicitly lead to executing operations on commitments, manipulating them by affecting their status. In the following we will focus on the three fundamental operations *create*, *discharge*, and *cancel* applied to base-level commitments. The other operations (such as *release, delegate*, and *assign*) as well as the treatment of conditional commitments can be seamlessly introduced in our framework.

Life Cycle and Compensation

The commitment life cycle targeted in this work is illustrated in Figure 1. At the beginning of execution, the commitment does not exist (or, alternatively, can be considered to be in a null state). The commitment starts to exist when a create operation is executed, causing a commitment's transition toward the active state: then, 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 debtor has honored the commitment, 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 support a further situation, in which the commitment is canceled but a new commitment (called a *compensating commitment*) is created to handle (compensate for) the violation, trying to recover it within the boundaries of 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 toward the compensated state and, at the same time, c_2 is created, becoming active. It is worth noting that other choices could be taken to model the active-compensated transition; for example, a violated commitment could be considered compensated for only when the compensating commitment has been satisfied.

Let us now focus on the semantics of operations in terms of events and commitment status. Operations are always applied in response to the occurrence of a corresponding event. They 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 cancellation of the compensated commitment, whereas 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 operations. On the one hand, the semantics of discharge are defined in a domain-independent

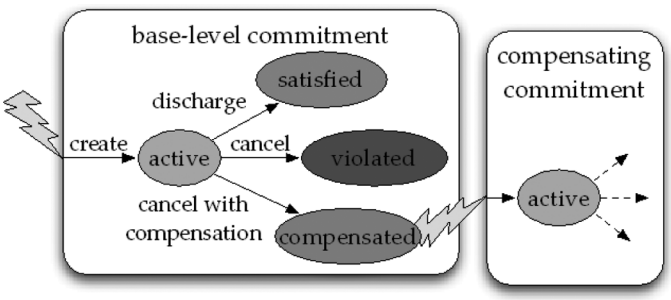

FIGURE 1 Base-level commitment life cycle extended with compensation.

manner and state 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 cancellation of a commitment is caused by the generation of a specific domain-dependent event during the interaction.

Time-Aware Commitments

This analysis points out a limitation in the basic life cycle: an active commitment that is not explicitly canceled, and whose property is never made true, will continue to persist indefinitely in the active status. It would be then desirable to introduce temporal constraints regulating when the commitment's property must be made true. To do so, the temporal dimension must be introduced inside the specification of commitments, making them time aware.

By relying on Mallya and Huhns (2003) and Torroni et al. (2009), we propose two classes of time-aware commitments, respectively focused on the achievement and maintenance of a certain property:

- $C(x,y,prop(e(t_1,t_2), p))$ represents an existential commitment, where x is committed to bring about p inside the time interval $[t_1,t_2]$ (a basic commitment can therefore be considered a special case of existential commitment, where t_1 is the time at which the commitment is created, and $t_2 = \infty$);
- $C(x, y,prop(u(t_1,t_2),p))$ represents an universal commitment, where x is committed to maintain$p$ valid along the whole time interval $[t_1,t_2]$.

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

In particular, $C(x, y, prop(e(t_1, t_2), p))$ becomes satisfied and event ev is generated at a time $t \in [t_1,t_2]$, such that ev makes p true. Conversely, if the existential commitment is still active after t_2, then p has not become true inside $[t_1,t_2]$, and therefore the commitment must be canceled due to a violation of the temporal constraints. It is worth noting that t_2 acts as a deadline by which p must be brought about to satisfy the commitment.

The case of universal commitments is the opposite. If $C(x, y, prop(u(t_1, t_2), p))$ is active at 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, if the universal commitment is still active after t_2, then the commitment has not been canceled before; that is, the validity of p has been maintained during $[t_1,t_2]$, and therefore the commitment is satisfied.

As we will see later, because all of the commitment's operations are applied when a corresponding event occurs, the actual evaluation of an existential (universal) commitment's cancellation (discharge) is performed when the first event after t_2 occurs.

FORMALIZING THE LIFE CYCLE OF TIME-AWARE COMMITMENTS IN REC

We now present how the commitment life cycle described in the previous section can be formalized in REC. As pointed out in the introduction, an REC theory is a KB composed by a set of Horn clauses, which bind together events and fluents. It is worth noting that such a KB relies on the standard EC ontology, making other EC reasoners seamlessly applicable as well. In the case of time-aware commitments, the theory itself is composed oftwo different knowledge bases, as shown in Figure 2. The first is a domain-independent theory formalizing the life cycle of time-aware commitments; it relates the initiation/termination of the commitment's status with operations and defines the domain-independent semantics of operations. The second is a theory representing the specific domain under study; it includes domain-dependent fluents and commitments as well as their relationship with specific events. In this section we focus on the general formalization of time-aware commitments. An example of domain-dependent theory will be presented in the car rental example.

The REC-based axiomatization of time-aware commitments was inspired by Yolum and Singh (2002), where EC was employed to provide a formalization of the commitment life cycle. In the EC setting, properties are represented by fluents, whose validity evolves over time as event occurs. Therefore, the concept of "bringing about some property p" is translated as "initiating fluent p" and the validity/truth of p at a given time is expressed by stating that fluent p holds at that time.

In addition tothe introduction of time-aware commitments and their compensation, there is a further difference between the formalization proposed by Yolum and Singh (2002) and ours. Whereas in Yolum and Singh (2002) commitments were directly mapped onto fluents (initiated through the create operation and terminated by the cancel/discharge operations), we map each commitment's status to a separate fluent *status*/2, where

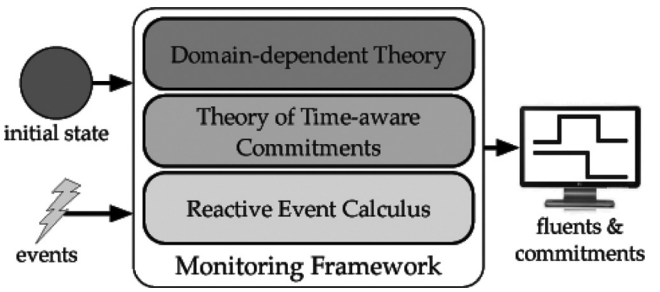

FIGURE 2 Monitoring framework for time-aware commitments.

status(*c, s*) expresses that commitment *c*is in state *s*. In this way, commitments' states are reified and can be reported to the user by the monitoring framework as well as be 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).

Because the knowledge base expressing the extended life cycle is a general theory, all of 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, and the remaining axioms capture the domain-independent semantics of operations, as informally described in the section on time-aware commitments.

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) ←holds_at(status(C, active), T).

Axiom 2 (Active State)

A commitment becomes active when it is created by the occurrence of an event E:

initiates(E, status(C(X, Y, P), active), T) ← create(E, C(X, Y, P), T).

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

terminates(E, status(C(X, Y, P), active), T) ←discharge(E, C(X, Y, P), T).

terminates(E, status(C(X, Y, P), active), T) ← cancel(E, C(X, Y, P), T).

Note that, usually, only the debtor agent can create a commitment. For the sake of generality, in Axiom 2 such a constraint is relaxed, but it can be seamlessly enforced by deciding a format for events (e.g., *do*(*X, E*) to represent that agent *X* generates event *E*) and imposing that *X* is the commitment's debtor.

Axiom 3 (Active-Discharged Transition)

A commitment makes a transition from the active status to the satisfied one when it is discharged by an event occurrence:

initiates(E),status(C(X,Y,P),satisfied),T) ← discharge(E,C(X, Y, P), T).

Axiom 4 (Active-Canceled Transition)

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(C(X, Y, P), violated), T) ←compens(C(X, Y, P), _,T) ∧cancel(E,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 for by means of C_2 at time T (i.e., that if a violation of C_1 happens at time T, then C_2 is created as a compensating commitment).

Axiom 5 (Compensation)

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 compensation has been defined for C at T:

$initiates(E, status(C(X, Y, P), compensated), T) \leftarrow compens(C(X,Y, P), \_, T)$

$\wedge cancel(E,X,C(X,Y,P),T).$

At the same time, the compensating commitment becomes active:

$initiates(E, status(C(W, Z, P_2), active), T) \leftarrow compens(C(X, Y, P), C(W, Z, P_2), T)$

$\wedge cancel(E, X, C(X, Y, P), T).$

Axiom 6 (Discharge)

An active basic commitment C is discharged by the occurrence of an event if the event brings about C's property:

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

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

$discharge(E, X, C(X, Y, prop(e(T_1, T_2), P))), T) \leftarrow active(C(X,Y, prop(e(T_1,T_2), P))),T)$

$\wedge T \geq T_1 \wedge T \leq T_2 \wedge initiates(E, P, T).$

A universal commitment is automatically discharged after its targeted time interval if it is still active (i.e., it has not been canceled before):

$discharge(E, X, C(X, Y, prop(u(T_1, T_2), P))), T) \leftarrow active(C(X,Y, prop(u(T_1,T_2),P))),T)$

$\wedge T \geq T_2.$

Axiom 7 (Cancel)

The cancellation 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):

$cancel(E, X, C(X, Y, prop(e(T_1, T_2), P))), T) \leftarrow active(C(X,Y, prop(e(T_1,T_2),P))),T)$

$\wedge T \geq T_2.$

An active universal commitment is canceled during its targeted time interval as soon as it is detected that the commitment's property has become false:

$cancel(E, X, C(X, Y, prop(u(T_1, T_2), P))), T) \leftarrow active(C(X,Y, prop(u(T_1,T_2),P))),T)$

$\wedge\ T \geq T_1 \wedge T \leq T_2 \wedge holds\_at(P,T).$

A CAR RENTAL EXAMPLE

We now discuss a simple but effective example, which shows the potentialities of time-aware commitments and their event calculus–based modeling. In on applying REC to the example presented in this section, we will illustrate the corresponding reasoning capabilities of REC and how an agent-based simulation framework can exploit them.

The example focuses on a contract that formalizes the mutual obligations between a customer and an agency when a car is rented; the following statements are included in the contract:

S1. The customer is committed to taking the car back to the car rental agency within the agreed-upon 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 "one-day" immediate replacement.
S4. In case of a car replacement, the customer receives two more rental 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. We model each event using a *do (Agent, Event)* notation, where *Agent* is the entity that generates *Event*.
B. Elicitation of the fluents that 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.

i. Using (some of the) fluents identified during step B to represent the "property part."
ii. Introducing existential/universal temporal constraints if needed.
iii. Defining their operations (create, discharge, cancel, compensation) in terms of the events identified during step A.

A. Events Identification

We suppose that the agency information system collects and stores the events characterizing the evolution of each rental:

- *do(C, rent(Ag, Car, N))*: customer *C* rents a car *Car* at the agency *Ag* for *N* days;

- *do(C, drive_back(Car, Ag))*: customer *C* drives *Car* back to the agency *Ag*;
- *do(Car, break_down):Car* breaks down;
- *do(Ag, replace(C,Car_{old},Car_{new}))*: agency *Ag* takes Car_{old} back from customer *C* and substitutes it with Car_{new}.

In addition to these domain-dependent events, we suppose that three further events, *start, complete*, and *tick*, are delivered to the monitoring framework. The first two events are used to respectively alert REC that the execution has begun/finished; the tick event, instead, is used to inform REC about the current time: REC itself has no explicit notion of the time flow—it reacts to each incoming event by updating the status of fluents and commitments and then waiting until a new event occurs. The delivery of tick events (e.g., by employing an external clock) is thus useful to enable the prompt evaluation of temporal constraints by REC (see Axiom 7, first rule, and Axiom 6, third rule), which in turn permits determining whether an existential (universal respectively) commitment must be canceled (discharged respectively). If tick events are not employed, such an evaluation is carried out as soon as the first suitable domain-dependent event occurs.

B. Fluents Elicitation

The system is characterized by the status of the cars owned by the agency:

- *in_agency(Ag,Car):Car* is parked in agency *A*;
- *great_car(Car):Car* is working;
- *hired(C,Car,D):Car* is being rented by customer *C* until date *D*;
- *car_replaced(Car):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 longer in agency and becomes hired until the date obtained by the current date plus the chosen number of days:

$$terminates(do(C,\ rent(Ag,Car,N)),in\_agency(Ag,Car),T).$$
$$initiates(do(C,\ rent(Ag,Car,N)),hired(C,Car,D),T) \leftarrow D = T + N.$$

When the customer drives back to the agency, the caris no longer hired and starts to be in agency again:

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

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

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

When the agency replaces a car, it becomes replaced:

$$initiates(do(Ag,\ replace(C,Car_{old},\ Car_{new})),\ car\_replaced(Car_{old}),T).$$

Furthermore, the replaced car is brought back to the agency, andthe new one is taken from the agency and delivered to the customer:

$$initiates(do(Ag,\ replace(C,Car_{old},\ Car_{new})),\ in\_agency(Ag,Car_{old}),T).$$
$$terminates(do(Ag,\ replace(C,\ Car_{old},\ Car_{new})),in\_agency(Ag,\ Car_{new}),T).$$

The 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 extradays:

$$terminates(do(Ag,\ replace(C,Car_{old},\ Car_{new})),hired(C,Car_{old},D),T).$$
$$initiates(do(Ag,\ replace(C,\ Car_{old},\ Car_{new})),\ hired(C,\ Car_{new},\ D),\ T) \leftarrow$$
$$holds\_at(hired(C,Car_{old},D_{old}),T) \wedge D = D_{old} + 2.$$

D. Commitments Elicitation

We now rephrase statements (S1), (S2), and (S3) in terms of time-aware commitments. Statement (S1) is a commitment that is created when the customer rents a car and is associated with 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-upon number of days. The property corresponds to the *in_agency* fluent, and the value of the deadline can be obtained as done for the *hired* fluent:

$$create(do(C,\ rent(Ag,\ Car,\ N)),\ C,\ C(C,\ Ag,\ prop(e(T,\ T_e),\ in\_agency(Ag,\ Car)))),\ T) \leftarrow$$
$$T_e = T+N \wedge holds\_at(in\_agency(Ag,C),T).$$

Statement (S2) can be represented by a 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:

$$create(do(C,\ rent(Ag,\ Car,\ N)),\ Ag,\ C(Ag,\ C,\ prop(u(T,\ T_e),\ great\_car(Car)))),\ T) \leftarrow T_e = T+3.$$

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 using the *compens*/2 predicate. The compensating commitment is existential and states that the agency is committed to bring about the *car_replaced* fluent within one day of the cancellation of the compensated commitment:

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

EXPLOITING REC IN THE CONTEXT OF AGENT-BASED SIMULATION

Given the formalization of the commitments life cycle presented in the section on formalizing the life cycle of time-aware commitments in REC and a domain-dependent commitment specification, REC is able to monitor a running execution of the system under study and to dynamically update the status of commitments and their corresponding properties. Such a run-time capability can be fruitfully exploited by an agent-based simulation framework in which commitments are used to model the normative aspects or the interaction protocols, as the following discussion shows.

We suppose that the simulation framework has complete visibility of all of the events generated by the interacting agents. In this way, it can dynamically deliver to REC all of the events that occurred during a simulation, providing all of the relevant information needed by REC to infer the status of commitments and properties. Depending on the agents' capabilities and on the purpose of the simulation, such information can be accessed and used during the execution either by the simulation environment, or by the interacting agents.

More specifically, the commitment monitoring framework can be exploited in two ways:

1. As a *run-time compliance checker* (Figure 3a), which can be queried by the simulation framework to obtain feedback about the impact of the partial course of interaction simulated so far on the commitment specification.
2. As a *compliance supporting facility* (Figure 3b), able to provide useful advice on how the interacting agents should behave in order to comply with the commitment specification.

REC as a Run-Time Compliance Checker

The simulation environment can interact with REC to know whether a running simulation is currently compliant with a commitment specification or not. For example, the simulation framework could query REC to know whether some commitment has been violated, consequently halting the

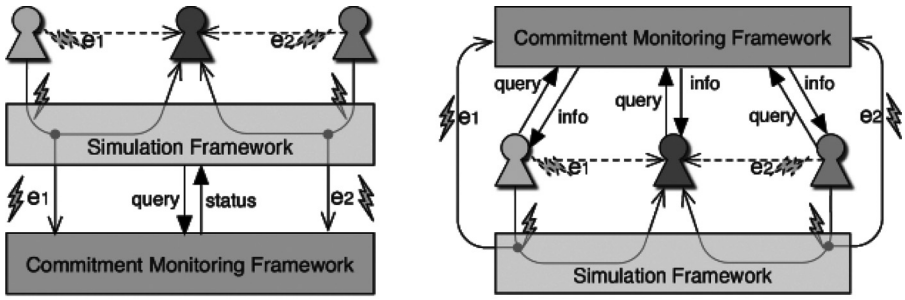

FIGURE 3 (a) Using the commitment monitoring framework as a runtime compliance checker. (b) Using the commitment monitoring framework as a compliance supporting facility.

simulation or taking proper countermeasures (i.e., affecting the simulation by, e.g., sanctioning or expelling the debtor agent).

In addition to the REC's ability toreturnacomplete snapshot of commitments and properties (i.e., the "shape" of all fluents representing the validity of properties and the status of commitments), it is possible to equip it with further predicates aimed at providing run-time support to the simulation framework. Due to the declarativeness or our approach, the definition of such predicates must be simply added to the other clauses.

All of these additional predicates have a time variable as parameter, because they characterize a specific state of affairs, reached at a particular time point.

First of all, the simulation framework could be interested in the detection of agents' misbehaviors with respect to the commitment specification. In this respect, the *misbehaving(T, Agents)* predicate returns the list of all agents that are debtors of a commitment that is violated at time *T*, thus providing support for identifying a noncompliant state of affairs and the involved agents:

misbehaving(T, Agents) ← *findall(Debtor, violated(C(Debtor,_,_), T), Agents).*

The definition of *misbehaving*/2 relies on the *findall*/3 Prolog predicate, where *findall(Objects, Goal, List)* produces the *List* of all *Objects* that satisfy the given *Goal*.

In addition to the detection of noncompliant states of affairs and the identification of misbehaving agents, REC can be queried to know whether the simulation has reached a quiescent state of affairs; that is, a state of affairs in which no commitment is active:

quiescent(T) ← *findall(_, active(C(_, _, _), T), []).*

Finally, compliant states of affairs can be identified by combining the *misbehaving*/2 and *quiescent*/1 predicates. In particular, a simulation is compliant with the monitored commitment specification at time *T* if it is quiescent at that time and there does not exist a misbehaving agent:

compliant(T) ← *quiescent(T), misbehaving(T, []).*

The simulation framework could exploit the *compliant*/1 predicate to know whether a simulated interaction can be currently terminated in a successful manner. Indeed, the absence of misbehaving agents attests that no commitment has been violated, and quiescence consequently guarantees that all commitments have been satisfied or compensated.

REC as a Compliance Supporting Facility

The interacting agents could behave in a completely autonomous manner, but they could also be aware that a normative layer actually exists. In this case, not only the underlying simulation environment but also the single agents themselves could be interested in obtaining information from the commitment monitoring framework. In particular, each agent can receive from REC all of the basic pieces of information needed to analyze the current state of affairs and make strategic decisions on how to behave next. The most important information is related to the active commitments for which the agent is debtor, because their correct discharging depends on the agent's behavior.

In order to provide run-time support to the interacting agents, another set of dedicated predicates is introduced. Such predicates are used by agents to extract information in a particular state of affairs, and therefore they always involve a time variable as well as a parameter representing the agent identifier.

When a commitment having the agent as debtor exists, a compliant behavior is exhibited by the agent if:

- it brings about the involved property inside the expected time span, if the property is existential and does currently not hold;
- it maintains the involved property valid inside the whole expected time span, if the property is universal.

Thus, a first set of supporting predicates is devoted to extracting all of the properties pending at the time. For existential properties, we state that a certain property *P* referring to agent *Ag* is pending at time *T* if there exists an active existential commitment whose debtor is *Ag* and whose property is associated with a time span containing *T*. In addition tothe property itself, the

predicate can be exploited to know the deadline by which P must be brought about in order to correctly discharge the commitment.

pending_e_prop(Ag, T, [P, By]) ← active(C(Ag, _, prop(e(From, By), P)), T) ∧ T ≥ From
∧holds_at(P, T).
pending_e_prop(Ag, T, [P, ∞]) ← active(C(Ag, _, prop(P)), T)∧holds_at(P, T).

The second clause is used to deal with nontimed properties; indeed, remember that a basic commitment can be interpreted as an existential commitment whose property is associated with the time interval $[t_{\text{create}}, \infty]$, where t_{create} is the time at which the commitment has been created.

The predicate

pending_e_props(Ag, T, L) ← findall(AP, pending_e_prop(Ag, T, AP), L).

can be then used by an agent Ag to obtain the list L of all existential properties under its responsibility that are pending at time T, together with their corresponding deadlines.

Similarly, for pending universal properties we have the following:

pending_u_prop(Ag, T, [P, Until]) ← active(C(_, Ag, prop(u(From, Until), P)), T) ∧ T ≥ From.
pending_u_props(Ag, T, L) ← findall(AP, pending_u_prop(Ag, T, AP), L).

Predicate *pending_u_props*/3 returns all of the universal properties that are pending at time T and are under the responsibility of Ag, together with the time points until which they must be maintained valid.

It is worth noting that the commitment specification not only contains the relationship between events and commitments but also between events and properties. Hence, REC could be used not only to obtain the set of currently pending universal and existential properties but also to know which actions could be performed to discharge an existential commitment or must be avoided because their execution would lead to violate a universal commitment.

Let us first discuss the case of existential commitments. For each pending existential property *Prop* under its responsibility, an agent Ag could exploit the following predicate:

declipping_actions(Ag, T, Prop, Acts) ← findall(Ev, initiates(do(Ag, Ev), Prop, T), Acts).

to obtain the set *Acts* of actions that are able to bring about *Prop* at time *T*, leading in turn to discharge the existential commitment associated to *Prop*.

Each event contained in *Acts* represents a possible alternative for discharging the commitment.

The case of universal commitments is the opposite. Given a universal property *Prop* under the responsibility of *Ag* and pending at time *T*, the following predicate

clipping_actions(Ag, T, Prop, Acts) ← *findall(Ev, terminates(do(Ag, Ev), Prop, T), Acts).*

can be used by *Ag* to obtain a set of activities that cannot be currently executed without leading to a misbehavior: the execution of one among the actions contained in the *Acts* set would terminate *Prop*, consequently leading to violate the universal commitment associated to *Prop*.

Ag can simply obtain the whole set of activities forbidden at time *T* by combining the clipping actions of each pending universal property:

forbidden(Ag, T, Acts) ← *pending_u_props(Ag, T, L), forbidden(Ag, T, L, Acts).*
forbidden(Ag, T, [], []).
forbidden(Ag, T, [[Prop,Within] | L], Acts) ← *forbidden(Ag, T, L, OtherActs)*
∧*clipping_actions(Ag, T, Prop, PropActs)*
∧*append(OtherActs, PropActs, Acts).*

At a given time, an agent can therefore query REC to know which actions are forbidden and which alternative actions could be performed to discharge an existential commitment. The agent could then take into account the actions contained in these two sets when planning the next actions.

APPLYING REC TO THE CAR RENTAL EXAMPLE

In this section, we show how REC can be concretely exploited to monitor a simulation of the car rental example presented in this article, providing useful information to the interacting agents and the simulation environment. In the example, three different agent types are mentioned: cars, customers, and car rental agencies.

We consider a simulation involving four agents: a car rental agency *ag* managing two cars (*bo123* and *bo124*), and a customer agent *ian*.

We suppose that the simulation environment forwards all of the occurring events to REC and that the running simulation has currently reached time 18 (we rely on a time granularity of days).

The running simulation is characterized by a (growing) execution trace collecting all of the events that occurred so far (i.e., before time 18) and by an

initial state, describing which fluents initially hold. In our case, *ag* initially has both cars in the agency, and the initial state is therefore described by:

```
initially_holds(in_agency(ag, bo123)).
initially_holds(in_agency(ag, bo124)).
```

Figure 4 illustrates the events that occurred so far during the simulation, together with a graphical representation of the reasoning outcome produced by REC, showing the evolution of properties as well as the state transitions of each created commitment. At time 10, *ian* has rented car *bo123* from *ag*, but the car has broken down during the guarantee; car *bo123* has then been replaced by the agency.

In particular, the figure shows that when car *bo123* has broken down, a compensation has been triggered; however, *ag* has missed the deadline associated with the compensating commitment, replacing the broken car only after four days and consequently causing a violation. The car's replacement has instead had the effect of manipulating the commitments associated with *ian*: *ian* is not entitled to bring back car *bo123* anymore but is instead in charge of bringing back the new car *bo124* within time 22 (i.e., time 20 plus 2 extra days).

At time 18, either the simulation environment or one of the interacting agents can query REC using the predicates introduced in the previous section respectively. For example, the simulation environment can ask REC whether the simulation is currently quiescent by invoking the goal *?-quiescent(18)*.

The answer is negative, because there still exists an active commitment having *ian* as debtor. The simulation is also noncompliant, because one commitment is violated; by asking the query *?-misbehaving(18,List)* the simulation environment can also know that the misbehavior has been caused by the car rental agency *ag* (*List/[ag]*).

Let us now discuss how *ian* could exploit REC to plan what to do next.

First of all, *ian* can query REC to know which pending existential properties are under its responsibility: *?-pending_e_props(ian, 18, Props)*. The answer produced by REC states that *ian* is in charge of bringing about property *in_agency(ag, bo124)* within time 22 (*Props/[[in_agency(ag,bo124), 22]]*).

Ian can then query REC to know which actions can be undertaken to bring about such a property: *?-declipping_actions(ian, 18, in_agency(ag, bo124), Acts)*. The computed answer states that *ian* can drive the car back to bring about the property: *Acts/[do(ian,drive_back(bo124,ag)]*.

Finally, *ian* checks whether such an action belongs to the list of currently forbidden events: the query *?-forbidden(ian, 18, FActs)* returns an empty list (*FActs/[]*), because no universal commitment with *ian* as debtor is active at time 18.

Now *ian* knows that it is responsible for an active commitment that-should be discharged within time 22 and that driving car *bo124* back to

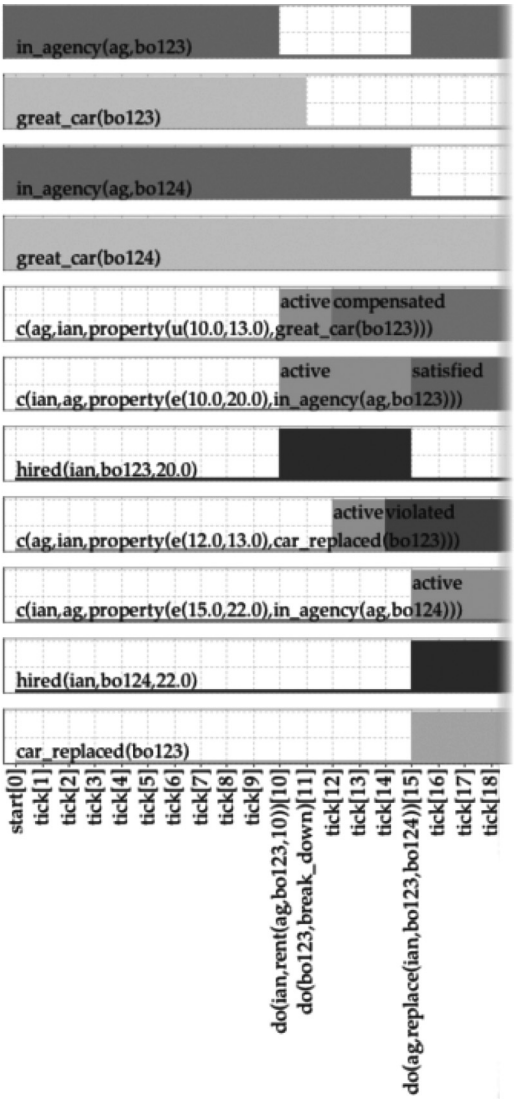

FIGURE 4 Outcome shown by REC when reasoning upon the example described in this article. REC took a total time of 2.02 s to reason upon the partial simulation on a MacBook Pro Intel CoreDue 2.66-GHz machine.

agency *ag* is an executable action that would lead to discharge this active commitment.

DISCUSSION AND CONCLUSION

In the first part of this contribution, we proposed an extended commitment life cycle accommodating time-aware social commitments and their

compensation. We formalized the life cycle as an EC theory, using a reactive version of EC called REC to monitor the execution of the system under study, tracking the commitment evolution as events occur. The acyclicity of the domain-dependent theory is a necessary and sufficient condition for ensuring that the monitor is sound, complete, and guarantees termination (Chesani et al., 2009).

Alternative formalizations of commitments rely on temporal logics (CTL [Computation Tree Logic] in particular). REC has two main advantages if compared with such approaches: it supports the modeling of data (and conditions) and quantitative time constraints used to handle time-aware commitments; support is provided at the language and operational levels. The introduction of deadlines inside a logic combining CTL and deontic aspects has been studied in Broersen et al. (2004), but the approach is not grounded in a reasoning framework, and deadlines do not refer to time values but to the (unknown) time at which a certain property becomes true. To accommodate quantitative time, metric temporal logics should be used, as done in Mallya and Huhns (2003). Our time-aware commitments cover all of the cases described there, and can be effectively computed by REC. A more detailed comparison can be found in Torroni et al. (2009).

We developed two different Java-based implementations for providing EC-based monitoring facilities. They rely on a two-way integration between a reactive reasoning engine for the EC and a Java-based generic event acquisition module. The first implementation exploits REC (Chesani et al. 2010) as the reasoning engine, whereas the second one embeds a pure-Prolog lightweight implementation of the cached EC (see http://www.inf.unibz.it/~montali/tools.html#ComMon). Because both reasoners use the same EC ontology, the EC-based formalization of time-aware commitments we have presented seamlessly works with both of them.

In addition to the acquisition and delivery of events, Java is used to graphically report the outcome dynamically produced by the reasoner, giving a constantly updated snapshot of the status of fluents and commitments.

Although an extensive evaluation of the framework's performance is still the subject of ongoing work, the initial experiments are encouraging. For example, we tested REC using the car rental case study, setting up an instance with 14 agents, eightcars, and more than 30 event occurrences. The total time required by REC to reason upon the entire trace was 80 s, but only 6 s was needed under the assumption that events are received by REC in ascending order (which enables the possibility of exploiting an optimized version of the reasoner).

The second part of the contribution was devoted to showing how the REC-based commitment monitoring framework can be fruitfully exploited in the context of agent-based simulation. In this respect, our work goes in the direction foreseen by Fasli (2004), who claimed that "there is a lot of scope for cooperation between formal systems and agent-based social

simulation." In particular, we extended the commitment formalization with further supporting predicates, which can be exploited by the simulation framework to obtain feedback about the current normative state of affairs or by the interacting agents to obtain useful information that can be taken into account when planning the next actions. As far as we are concerned, Hägg (1998) was the first author to combine commitments with agent-based simulation. In his work, commitments were managed by taking inspiration from the two-phase commitment scheme used in active databases. In order to simulate commitment-regulated interactions, Hägg (1998) introduced an acceptance function, used by an agent to take a commitment proposal and emit an acceptance, rejection, or modified counter proposal.

The approach presented in this work could be integrated with Hägg's (1998) approach by developing the internals of the simulated agents: deliberative agents (such as customers and car rental agencies in our running example) could be simulated by means of an acceptance function that interacts with REC in order to obtain the set of forbidden and expected actions and then determines the next action.

Broadly speaking, the presented framework can be easily embedded into any simulation environment, where commitments can be fruitfully adopted as a mean of capturing the normative states of affairs. In addition to open multi-agent systems, commitments have been proven effective in many other settings, such as for tackling interaction in service-oriented computing (Singh et al. 2009), as well as business contracts and protocols in business process management (Telang and Singh 2011). In this respect, REC can be used to either monitor simulations or assess their compliance with respect to the modeled commitments. The latter case is simply a specific monitoring setting in which the monitored simulation is considered noncompliant as soon as some commitment enters the violated state.

From a software engineering perspective, the seamless integration of REC into an agent-based simulation platform is mainly due to the fact that its only requirement is to have access to all of the event occurrences characterizing a running simulation. The rest of the system is treated by REC as a blackbox, and the system can consider REC as a blackbox whose interface is determined by the predicate for receiving events and by the supporting predicates discussed in the section on exploiting REC in the context of agent-based simulation.

As ongoing work, we intend to relax the hypothesis of full access to such event occurrences, investigating the possibility of splitting the commitment specification and substituting the global REC-based monitor with a set of local monitors, each one dedicated to monitoring the behavior of a single agent. In this respect, two challenging aspects arise: how to split the overall commitment specification into a set of local specifications and how to combine the results produced by the local monitors into a single, aggregated outcome.

REFERENCES

Apt, K. R. and Bezem, M. "Acyclic Programs." In *Proceedings of the 7th International Conference on Logic Programming* (ICLP1990), edited by David H. D. Warren and Péter Szeredi. Jerusalem, Israel: MIT Press, 1990.

Broersen, J., Dignum, F., Dignum, V., and Meyer, J. Ch. "Designing a Deontic Logic of Deadlines." In *7th International Workshop on Deontic Logic in Computer Science*, edited by Alessio Comuscio and Donald Nute, 48–56. Madeira, Portugal: Springer, 2004.

Chesani, F., Mello, P., Montali, M. and Torroni, P. "Commitment Tracking via the Reactive Event Calculus." In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, edited by C. Boutilier. Pasadena, CA: AAAI Press, 2009.

Chesani, F., Mello, P., Montali, M., and Torroni, P. "A Logic-based, Reactive Calculus of Events." *Fundamenta Informaticae* 105, nos. 1–2 (2010): 135–61.

Chittaro, L. and Montanari, A. "Efficient Temporal Reasoning in the Cached Event Calculus." *Computational Intelligence* 12 (1996): 359–82.

Fasli, M. "Formal Systems ∧ Agent-based Social Simulation = ⊥?" *Journal of Artificial Societies and Social Simulation* 7, no. 4 (2004).

Hägg, S. "Commitment in Agent Cooperation, Applied to Agent-based Simulation." In *Proceedings of the 5th Conference on Intelligent Autonomous Systems*, edited by Y. Kakazu, M. Wada, et al. Amsterdam: IOS Press, 1998.

Kowalski, R. A. and Sergot, M. "A Logic-based Calculus of Events." *New Generation Computing* 4, no. 1 (1986): 67–95.

Mallya, A. U. and Huhns, M. N. "Commitments Among Agents." *IEEE Internet Computing* 7, no. 4 (2003): 90–3.

Singh, M. P., Chopra, A. K., and Desai, N. "Commitment-based Service-oriented Architecture." *IEEE Computer* 42, no. 11 (2009): 72–9.

Telang, P. R. and Singh, M. P. "Specifying and Verifying Cross-organizational Business Models: An Agent-oriented Approach." *IEEE Transactions on Services Computing* 4 (2011): 1.

Torroni, P., Chesani, F., Mello, P., and Montali, M. "Social Commitments in Time: Satisfied or Compensated." In *Post-proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies*, edited by M. Baldoni, J. Bentahar, M. B. van Riemsdijk, and J. Lloyd. Budapest, Hungary, 2009.

Yolum, P. and Singh, M. P. "Flexible Protocol Specification and Execution: Applying Event Calculus Planning Using Commitments." In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems*. Bologna, Italy: ACM Press, 2002.