Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-012-9202-0

Representing and monitoring social commitments using the event calculus

Federico Chesani · Paola Mello · Marco Montali · Paolo Torroni

Abstract Multiagent social commitments provide a principled basis for agent interactions, and serve as a natural tool to resolve design ambiguities. Indeed, they have been the subject of considerable research for more than a decade. However, the take-up of the social commitments paradigm is yet to come. To explain this negative result, we pinpoint a number of shortcomings, which this article aims to address. We extend current commitment modelling languages, thus leveraging expressive possibilities that were precluded by previous formalizations. We propose a novel axiomatization of commitment operations in a first order Event Calculus framework, that accommodates reasoning with data and metric time. Finally, we illustrate how publicly available $\mathcal{REC}$ implementations can be exploited for commitment monitoring purposes.

Keywords Social commitments · On-line monitoring · Multiagent systems · Event calculus

1 Introduction

Social commitments are made from one agent to another agent to bring about a certain property. They have been originally proposed to model and reason upon agent social semantics [12,56]. Representing the commitments that the agents have to one another and specifying their expected interactions in terms of commitments provides a principled basis for

F. Chesani (✉) · P. Mello · P. Torroni
University of Bologna, Bologna, Italy
e-mail: federico.chesani@unibo.it

P. Mello
e-mail: paola.mello@unibo.it

P. Torroni
e-mail: paolo.torroni@unibo.it

M. Montali
Free University of Bozen-Bolzano, Bolzano, Italy
e-mail: montali@inf.unibz.it

Published online: 22 June 2012

agent interactions. Since their introduction in multiagent research, social commitments have become a well-accepted modelling abstraction and engineering tool [22,33,57,58,66], and they have been successfully applied to other research fields such as, e.g., (cross-organizational) business processes [24] and protocols [25,27], business contracts [26] and service-oriented architectures [60].

The success of social commitments in these domains, characterized by heterogeneity and distribution of active components, is due to a variety of reasons. Commitments are about agent interaction, and abstract away from agent internals. Thus commitments support *openness* and *heterogeneity*, enabling the seamless integration of different agent models and implementations in the same application. Besides, commitments are *declarative*, i.e., they focus on *what* social interaction, protocols and communication actions are about, without specifying *how* interaction should be carried out. For this reason, they are *flexible*, since they can capture many different interaction models under a single specification [66]. Commitments are not linked to any specific framework, implementation or underlying formalism: they can be used to model and reason upon any system with a focus on the interaction among its components. Finally, social commitments offer solid ground for *verification* tasks, which is also one of the main motivation of social semantics of agent interaction [57]. At design time, they enable representing the consequences of agent actions on the system, and possibly identifying otherwise unforeseen outcomes. At run time, they provide a reference model for monitoring, thus determining if agents are behaving as prescribed, and what is expected of them.

However, in spite of their popularity in multiagent research, commitments are yet to be seen at work in concrete applications. We ascribe this to three reasons. The first one is that current commitment models have strong limitations in terms of *expressive possibilities*: existing proposals adopt propositional representations of commitments, thus precluding a number of models where dynamic aspects influence commitments evolution, and data play a fundamental role in the commitment-based modelling.

A second reason is that *temporal aspects* (such as, e.g., deadlines) have not yet been explored enough. In [33] Fornara and Colombetti recognize the need for distinguishing between *achievement properties* and *maintenance properties*. In [49,50] Mallya et al. propose ways to express and verify complex temporal properties by means of CTL and Allen's interval logic. Indeed, Temporal Logics such as LTL, CTL and their many variants, are particularly suited for expressing temporal properties. However, these logics' main application is static verification, i.e., to determine if the system will exhibit some properties. Dealing with agent-based open environments, instead, also requires reasoning at run time, in a highly dynamic environment, where a priori assumptions and models can represent only the static part of the domain, but hardly accommodate the dynamic part.

The third reason is the lack of an effective *framework* for commitments and their implementation. In particular, *monitoring tools*, faithful to the commitment models and therefore able to guarantee a sound and complete output, are still missing.

We believe then that the key to success for social commitments is in an operational framework, which should be expressive enough to accommodate scenarios of practical utility. To illustrate our desiderata for such a framework, let us consider a car rental scenario, where a customer signs a rental contract with a car rental agency. Let us examine some key sentences of a possible contract.

(S1) The Customer must return the car to the Agency on the agreed day. Should the car be returned before the last rental day, no reimbursement is due.

By signing the contract, the customer commits herself to returning the car. It is reasonable to assume, however, that such obligation is established only if the customer picks up the car. It would be nonsense to oblige a customer to return a car which she did not pick up. Hence, the commitment depends on a dynamic *context*. Note also that there is a temporal requirement: the customer should bring back the car within a certain *time frame*, established at the moment of signing the contract. Such a time frame could be defined in relative terms (for instance, within 10 days, as of the day the contract is signed), or in absolute terms, directly specified in the contract ("the car should be returned by September 25th, 2011").

Existing commitment frameworks generally abstract away from the property commitments are about, which is usually defined as a proposition. A sentence such as "the car should be returned by September 25th, 2010" could be represented by a proposition p, although it might be difficult (if not impossible) to include into p a sensible temporal deadline, since that is determined only at runtime.

(S2) The Customer is required to pay for the whole rental period. The payment can be done by the Customer, or by anyone else on her behalf. Payment must be made at the latest on the first rental day. The rental fee is calculated as the fee currently assigned to the category the car belongs to, times the number of days.

Sentence (S2) describes an obligation, that can be naturally modelled as the commitment that arises, upon contract signing, between the customer and the agency. Again, the commitment must get satisfied within a temporal deadline. However, the payment *can be made by anyone* (and not only by the customer). For example, a manager renting a car could ask her secretary to pay, using the firm's credit card. More importantly, the commitment is about paying a rental fee that is not statically defined, but *calculated*, based on two different parameters which will be instantiated only when the contract is signed.

Frameworks based on propositional approaches cannot provide a faithful and precise account of such commitments. Besides the obvious problem of representing the (dynamically calculated) fee, it is also difficult to specify that the property "payment" should be brought about by the debtor *"or by anyone else on her behalf."* One might be tempted to represent (S2) as a commitment where the debtor is the customer, and the same customer is allowed to delegate such commitment. Within the delegation, the debtor could keep the responsibility or not. In any case, the delegation would establish another debtor, while the business sentence calls for *anyone*. In real scenarios many different situations are possible, where a commitment can be satisfied by the debtor (no delegation), by anyone else (unknown until the commitment is satisfied), by a restrict group of agents (by a set identified by static or dynamic conditions), or by a specific agent (delegation, with or without responsibility).

(S3) The Agency guarantees that the car will work properly for the whole rental period. In the event of a breakdown:

(S3a) If the Customer informs the rental company of the breakdown in the last two days of the rental period, the Agency will not substitute the broken car, but it will withdraw it free of charge. All moneys paid for the remaining days will reimbursed at the end of the rental period;

(S3b) In all other cases, the Agency will replace the broken car within two days of the notification;

In all cases, as soon as the Customer informs the rental company of the breakdown, 200$ will be transferred to the Customer's account for compensation.

Several commitments can arise, as consequence of the same events (car breakdown and user notification). However, depending on *when* such events occur, the outcomes can be completely different: withdrawal (and reimbursement) versus substitution of the car. Moreover, while (S1) and (S2) are about seeing to it that a property is brought about *at a point in time*, (S3) is about seeing to it that a property *holds* during a certain time interval ("…the car will work properly for the whole rental period").

Although recent research on commitments addresses the temporal dimension of commitment properties, both for time points and for time intervals (see [50], by which this example was inspired), they do not enable on-line verification (monitoring) tasks. On the other hand, existing works on run-time verification of interaction protocols (see [1]) do model temporal aspects, but only enable point-time event verifications.

(S4) If, as consequence of a breakdown, a substitution car is needed, and if the substitution is made more than two days after the Customer's notification, the Agency will reimburse 20$ for each day of delay with respect to the deadline written in the contract. The Customer will be reimbursed within 30 days of the end of the rental period.

Sentence (S4) establishes a reimbursement policy. The amount of the reimbursement depends on the number of days between an expected deadline (two days after the breakdown) and the day the car is actually substituted. Hence, until the car is substituted, it is not possible to tell the amount of the reimbursement. Here a commitment *is only partially specified*. Existing commitment models do not accommodate partially specified commitments.

To summarize, real-life contracts establish mutual obligations among several parties. The object of such obligations (e.g., the amount of a fee), are often determined at runtime, depending on the context. Moreover, obligations are often framed within temporal boundaries, and can be about properties being *brought about*, or being *maintained*. Current commitment frameworks based on propositional logics show many limits when dealing with such obligations, norms and situations, which demands more expressiveness.

If we formalize such contracts in a machine-readable format, we could also automatically process relevant events, as they occur, and obtain an up-to-date picture of the situation. In other words, we could monitor the state of commitments at runtime. Concrete applications call for effective and sound monitoring tools.

However, existing monitoring tools are either inefficient, or not expressive enough for practical use (e.g., they do not accommodate data variables and metric time), or they implement procedural monitoring routines that hardly match the language logic and the commitment semantics, which may hinder the soundness of such implementations with respect to the higher level reference framework. In order to support commitment monitoring in concrete applications, monitoring tools must be expressive, efficient, and sound with respect to the language used to define the monitored properties.

This article describes a framework for the definition of commitment-based interactions patterns. It is based on the Event Calculus ($\mathcal{EC}$, [45]) and is rich enough to model contracts such as in the car rental example. Most significantly, it supports concepts of variables, data, and metric temporal reasoning. Although some of these concepts are not new in the related literature (e.g., achievement and maintenance properties are discussed in [33,50]), for the first time all these different concepts are integrated within a single framework.

Besides giving a semantics to commitments in terms of $\mathcal{EC}$ fluents, we provide an axiomatization of *commitment operations* in terms of $\mathcal{EC}$ primitives. Such an axiomatization thus serves two purposes: it provides both a formal semantics to commitments operations, and an executable specification, which concretely enables commitment monitoring.

To effectively perform the monitor task, we exploit the Reactive Event Calculus ($\mathcal{REC}$ [15,11]), an implementation of $\mathcal{EC}$ that overcomes some limitations of previous proposals, whose "reactive" behaviour makes it especially suitable for monitoring. The $\mathcal{REC}$-based commitment monitor is publicly available, packaged in the j$\mathcal{REC}$ Java archive [14] together with a graphical user interface, or as a stand-alone pure-Prolog, YAP-based tool.

This article builds on and extends some of our previous work [13,16,17,62] on this topic. In particular, in [62] we investigated for the first time the problem of time-aware properties within commitments, and we proposed the extensions for dealing with temporal aspects, although we did not discuss how to formally represent time-aware commitments. In [13] we first introduced $\mathcal{REC}$ in relation with commitments. We presented a simple framework, less expressive than the present one in terms of the treatment of data, variables and time. In [17], we exploited $\mathcal{REC}$ for the first time for commitment monitoring, although conditional commitments were not supported, and the focus was more on compensation mechanisms. Finally, in [16], we used the results discussed in [17] to enhance agent-based simulation, but with no support for conditional commitment. A retrospective on our research done on the $\mathcal{REC}$ in relation with social commitments can be found in [63]. Here we present a comprehensive axiomatization of the commitments, where significant extensions (some of them proposed in previous works, while other newly introduced in this paper) have been taken into account, and supported within the framework. Some extensions are about the expressiveness of the language for defining commitments and properties; others directly address the commitment life-cycle.

This work continues as follows. The next section gives the necessary background on social commitments, and defines our extensions to the commitment model, operations, and life-cycle. Section 3 introduces the $\mathcal{EC}$ and the $\mathcal{REC}$, formalizes the theory of commitments in the $\mathcal{REC}$, and elaborates on some key technical aspects of such a formalization. Section 4 illustrates the approach using the car rental example and offers an experimental evaluation of $\mathcal{REC}$'s efficient YAP-based implementation. Section 5 discusses related work, and Sect. 6 concludes.

2 Social commitments

Social commitments are used in multiagent systems to model, at the social level, the mutual obligations established by interacting agents. Messages exchanged, actions taken, or other types of events related to agent interaction, could be reasons for an agent to become *debtor* towards a *creditor* agent to bring about some *property*. A commitment is written as:

$$\mathsf{C}(X, Y, P),$$

where X is the debtor, Y the creditor, and P the property. Each execution of the system under study can be characterized in terms of how the relevant commitments evolve over time, due to the occurrence of events. Such events, generated by the interacting agents, (implicitly) manipulate commitments, causing them to change *state*. The state machine, i.e. the states in which a commitment can be, and the possible state transitions, is called *commitment life cycle*. In Fig. 1 we report three different commitment life cycles from literature.

A commitment can be in one state at a time. Typically, each commitment model proposes a different set of states and a different set of state transitions, also referred to as *commitment operations*. All the approaches we are aware of assume that such operations are triggered by events executed by the interacting agents. The domain modeller usually defines which events trigger which commitment operations, under which conditions. For instance, suppose

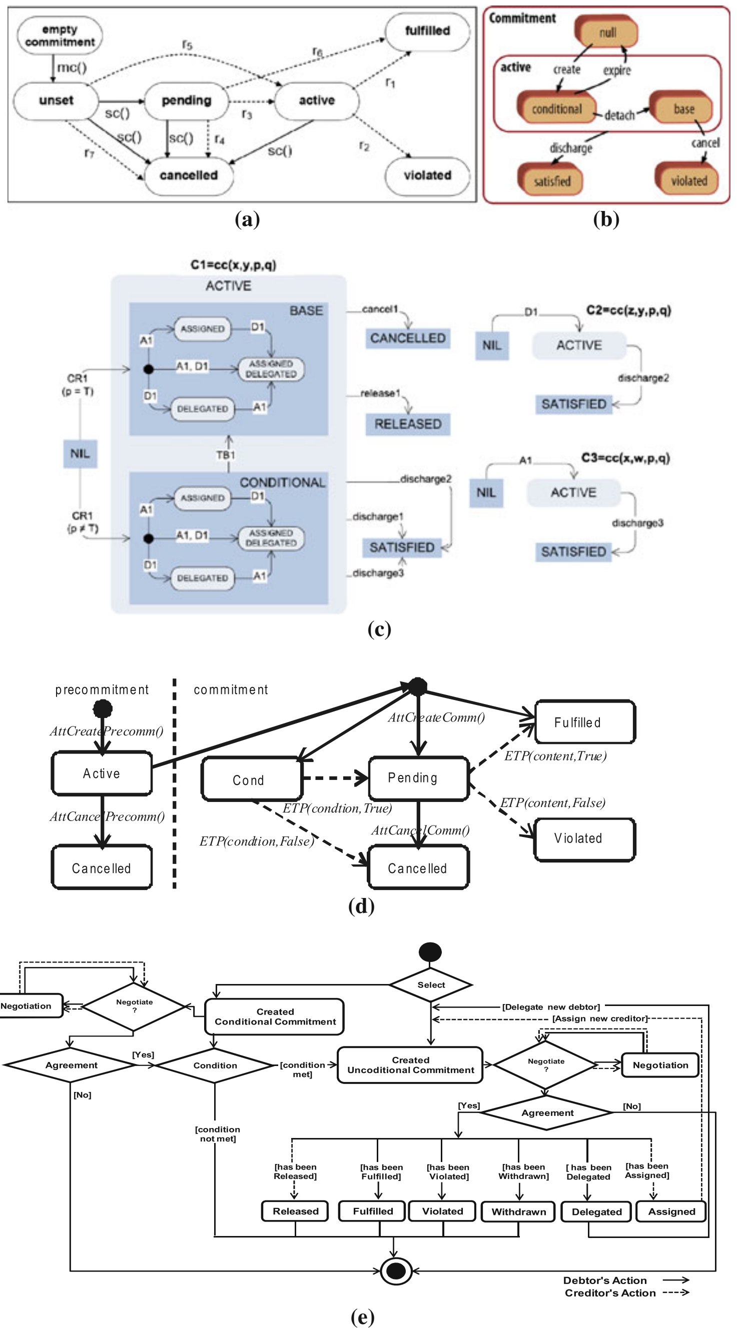

Fig. 1 Some commitment life cycles from literature. **a** Fornara and Colombetti's model, 2004 [33], **b** Singh et al.'s model for SOA, 2009 [60], **c** Desai et al's model for business processes, 2007 [24], **d** Fornara and Colombetti's model, 2009 [35], **e** El-Menshawy et al's commitment model, 2009 [29] (Color figure online)

that, in a given domain, if an agent a promises an agent b to complete a certain task, then a has an "obligation" to complete that task. Such a situation can be modelled by means of commitments, by saying that the event "a promises b" creates the commitment from a to b to complete the task, by way of a commitment operation named *create*. With respect to the models in Fig. 1, the commitment could enter the state "active (base)" of Fig. 1b, or the state "active" of Fig. 1a. A domain modeller could specify the conditions for the creation of a commitment as follows

$$create(promiseTaskCompletion(a, b),\ \mathsf{C}(a, b, completeTask)).$$

where the first parameter of the term $create$ is the event triggering the operation, and the second one is the commitment that will possibly become active. Sometimes, instead of agent names such as a and b, we will use variables to generalize the conditions, independently of the agents involved.[1] This is especially needed in an open setting, where the participating agents as well as the actual content of the occurring events are not known at design time.

Traditionally, an agent becomes committed to another agent only as a consequence of its own actions. This corresponds to the idea that an agent cannot become the debtor of a commitment, if not by its own choice. Other commitment operations are also subject to implicit restrictions on who should act: for example [67] clearly states that a commitment can be "assigned" to a new creditor only by the current creditor. The rationale behind such assumptions is to preserve the agents' freedom and autonomy, in particular when considering open MAS. However, in more closed multi-agent based systems, such as for example those where power structures and relations hold between the participants [28], such autonomy is not always a requirement. Thus, a commitment model that obliges to the assumptions cited above, might result to be of limited usage.

Besides commitments, many authors have introduced *conditional commitments*, in the following form:

$$\mathsf{CC}(X, Y, Q, P)$$

meaning that if a property Q (also called "antecedent" as opposed to the "consequent" P) is brought about, then a commitment $\mathsf{C}(X, Y, P)$ is established (becomes *active*). Here Q is intended as the *condition* that must hold for the commitment to become active. For instance, suppose to model a business contract where if a customer pays for an item, then a seller will provide certain items. Such a contract can easily be represented by the conditional commitment:

$$\mathsf{CC}(seller, customer, payment, deliver(item))$$

Of course, $seller$ becomes committed towards to $customer$ only if the latter pays for the item; no commitment is established otherwise. Some earlier works [33,58,67] distinguished between conditional commitments and base-level commitments. More recent work considered only conditional commitments, treating base-level commitments as their special case, where the condition Q is *true* [6,60] from the beginning. We agree with such a view. Nevertheless, for the sake of clarity, we will distinguish between base-level and conditional commitments by using two different notations: $\mathsf{C}(\ldots)$ and $\mathsf{CC}(\ldots)$.

[1] We adopt a Prolog-like notation, where terms starting with lower case indicate constants, and terms starting with a capital letter indicate variables. For instance, x is the name of a specific agent, while X is a variable, which may stand for a generic agent.

Our framework extends the state of the art in several directions:

(a) it enables modelling the *temporal extension* of commitment properties, quantitatively and with support for existential and universal quantification of temporal properties;
(b) it increases the expressive possibilities of the domain modeller, by accommodating data variables, which enables defining custom commitment structures and context-dependent commitment properties;
(c) it accommodates in a single framework different commitment states proposed in different approaches, such as for example in [33], in particular for what regards the idea of *violation* strictly related to some temporal property.

The motivations for such improvements stem from a need to apply commitment models to real-life business contracts, rather than to overly simplified case studies. Essential improvements to the state of the art should be such as to accommodate: support for data variables, quantitative (metric) temporal deadlines, run-time monitoring, static specification of properties and their verification. The following sections illustrate the proposed extensions and our commitment model.

2.1 Time-aware commitments

Let us consider the notation $\mathsf{C}(X, Y, P)$ and $\mathsf{CC}(X, Y, Q, P)$ we introduced above. In the traditional understanding of $\mathsf{C}(X, Y, P)$, X is committed to bringing about P, where P describes a certain situation, desired state or property. Few approaches consider P to be some temporal logic formula, like in [50], where P is a CTL^* formula. More frequently, P is considered to be a propositional formula. However, in real life situations, metric temporal aspects are frequently taken into account. For instance, a tax-payer is expected (is committed) to pay the taxes within a certain date; a fast delivery service is expected to deliver the items within three working days from the expedition; a consumer electronics producer guarantees that the device will work properly for 1 year, starting from the day the device was acquired.

We identify two different types of properties with respect to the temporal dimension: (i) properties about something that should be brought about within an interval (*achieving properties*); and (ii) properties about something that should hold during an interval (*maintenance properties*). For instance, the commitment of a researcher to submit a paper within a conference deadline belongs to the first type. A commitment for a poster author to stay near its poster during the whole poster presentation session belongs to the second type of commitment.

In [33] the need for taking into account property deadlines is acknowledged, and also existential and universal quantifications over time interval is suggested. However, such aspects are not investigated, and they are left to future work. A more closely related work is the one by Mallya et al. [49,50], where the concepts of properties that should be brought about and/or properties that should be maintained is explored in a structured way. However, as pointed out in [62], the model proposed has limited expressivity, due to the simplified domain variables model adopted for the temporal dimension.

With respect to the temporal dimension of the properties that commitments are about, we adopt the following notation:

– $\mathsf{C}(X, Y, property(P))$ represents a commitment about a *simple property*, where X is committed to bring about P at some time. Since there is no temporal deadline for achieving the commitment property P, ideally there is *infinite* available time for satisfying the commitment.

- $\mathsf{C}(X, Y, property(e(T_1, T_2), P))$ represents a commitment about an *existentially-quantified property*, where X is committed to *bring about* P *within* the time interval $[T_1, T_2]$.[2] The commitment is satisfied when the property P is brought about in any time instant within the interval $[T_1, \ldots, T_2]$.
- $\mathsf{C}(X, Y, property(u(T_1, T_2), P))$ represents a *universally-quantified property*, where X is committed to *maintaining* P valid *during* the whole time interval $[T_1, T_2]$.

We call existentially-quantified and universally-quantified properties "temporal properties," to distinguish them from the "simple properties."

We introduce temporal aspects also in the commitment operations. Traditionally, the creation of a commitment only depends on specific events; thus *create* operations are defined as:

$$create(Event,\ \mathsf{C}(X, Y, P)).$$

In our framework, commitment operations also specify the time T at which the event occurs:

$$create(Event,\ \mathsf{C}(X, Y, P),\ T).$$

T is thus reified, and as such it can also be used within the definition of P. For example, it is possible to define existentially-related properties whose deadline depends on *when* the commitment is created. Suppose the following situation: if an agent X accepts to do a task, then X has 5 time units to accomplish that task. This can easily be modelled by a *creation* operation[3]:

$$create(accept,\ \mathsf{C}(X, Y, property(e(T_1, T_2), taskDone)),\ T) \leftarrow \\ T_1 = T + 1,\ T_2 = T + 5.$$

Conditional commitments too can be about simple or temporal properties. In particular, both the property and the condition can be temporal properties. Hence, we can have commitments such as:

$$\mathsf{CC}(X,\ Y,\ property(e(T_1, T_2), Q),\ property(e(T_3, T_4), P)).$$

Finally, we adopt a time model where time is *discrete*. In order to provide a correct axiomatization of commitments, we further assume that no two events can happen at the same time instant.

2.2 Commitment fulfillment, cancelation, and violation

A commitment may fails to get satisfied. Literature work addresses this issue in various ways. In [66,67], the debtor can cancel the commitment, which may lead to the creation of a compensation commitment. The commitment then becomes $\mathsf{violated}$ (see also [60]). In [33], commitments are about properties whose truth value is initially unknown: then a property can be brought about (hence the commitment gets $\mathsf{satisfied}$) or its holding can be interrupted (hence the commitment gets $\mathsf{violated}$). Moreover, a commitment can also get

[2] A commitment about a simple property can be therefore considered as a special case of a commitment about an existentially quantified property where, given T as the time instant the commitment is created, then it holds $T_1 = T + 1$, and $T_2 = \infty$.

[3] Here, as well in the rest of the paper, we use a Prolog-like notation to specify rules, fact and conditions.

canceled. Note that cancelling is also the result of failing to bring about the commitment's property before a deadline.[4]

In our model, unless stated otherwise, all the properties are initially false (as we adopt an underlying two-valued logic). Moreover, we distinguish between the *cancelation* of a commitment, and its *violation*. The cancellation of a commitment is the result of an action (usually performed by the debtor), while a "violation" is a consequence of a failure to get a property verified. In the former case an agent takes an active role in the process, by performing an action that leads to cancelling the commitment. In the latter case, an agent responsible to bring about the property (usually the debtor), does not act in that sense, somehow playing a "passive" role. The expiration of a deadline in our model brings the commitment in a state of *violation*, if meanwhile the property has not been brought about.

Note that commitments about simple properties cannot get violated at runtime: since there is no deadline associated to the property, the debtor has infinite time to satisfy the commitment. Therefore, we cannot assert that the commitment got violated at a certain point in time, because there is still time to satisfy it. On the other hand, commitments about simple properties can be satisfied at runtime, that is, as soon as the property is brought about. Thus this type of commitments may be of interest if we wish to track which parts of a contract have been successfully executed, and which ones are still pending. Note however, that typically, interactions do have an (implicit or explicit) timeout, and however long lived they may be, they will terminate at some point. Thus we can foresee that the practical need for commitments about simple properties will be of limited extent. They are a part of our framework, mainly for compatibility with other proposals in the literature, which do not deal with time.

Commitments about *existentially-related properties* are *violated* as soon as the deadline expires. Let us suppose that $\mathsf{C}(x, y, property(e(t_1, t_2), p))$ is still active at a time $t_3 > t_2$. This implies that p has not been brought about inside $[t_1, t_2]$, and hence the commitment is violated. More specifically, we can detect a violation without $[t_1, t_2]$, i.e., as soon as we become aware that t_2 has passed, and p has remained false all along.

Commitments about *universally-quantified properties* instead get violated within the time interval they refer to, as soon as we become aware that p is false. Let us suppose that $\mathsf{C}(x, y, property(u(t_1, t_2), p))$ is active at time t_3, being $t_3 \in [t_1, t_2]$ the first time a new event occurs such that p does not hold. This implies that the validity of p during $[t_1, t_2]$ has been "broken", and therefore the commitment gets violated at time t_3.

Dually, a commitment $\mathsf{C}(x, y, property(e(t_1, t_2), p))$ is *fulfilled* as soon as an event ev occurs at a time $t \in [t_1, t_2]$, such that ev brings about p, and a commitment $\mathsf{C}(x, y, property(u(t_1, t_2), p))$ is fulfilled if the commitment is still active right after t_2; which implies that the commitment has not been cancelled before, i.e., that the validity of p has been maintained during $[t_1, t_2]$.

2.3 Which commitment operations is an agent entitled to?

Commitment operations are traditionally associated with either the debtor or the creditor of the commitment. For example, in [67] a commitment can be created and discharged only by the debtor. Such association has been formalized in terms of the conditions about occurring events that trigger the commitment operation. For example, a commitment gets discharged if the discharging event is performed (or generated) by the debtor agent. To enforce such

[4] More recent work by Fornara and Colombetti also discusses sanctions as a tool for norm enforcement [34]. Commitment monitoring in our view is necessary for enforcement, however, the scope of this paper does not include uses of monitoring tools but only their realization.

constraint, previous definitions of the commitment operations usually included both the event and the originator of such event.

We deem such approach too restrictive in the general case. There might be situations where restrictions about which agent is entailed to perform which operations are unjustified and even counterproductive. Consider the semantics of the *discharge* operation. There are cases where the commitment is about a property that must get satisfied by none other than the debtor of the commitment. There are other cases where the commitment should be considered to be satisfied independently of who brought about its property. In the latter case, the intended meaning of commitments is slightly different: a debtor is responsible for seeing to it that the property gets satisfied, but it is not strictly necessary that the property is brought about by the debtor. The domain modeller, when specifying the interaction patterns by means of commitments, should be allowed to distinguish between the two above cases. Note that such requirement has been recognized, discussed and partially supported by Mallya, Yolum and Singh in [50].

A second operation that demands for some flexibility is *create*. The state of the art recognizes that an agent becomes debtor of a commitment only as a consequence of its own actions. This design constraint is motivated by the idea of using commitments for modelling open multiagent systems. We fully agree with this vision: in an open system it should not be possible that an agent becomes debtor of a commitment against its will. However, things might be different when considering more structured MAS. For example, MAS organized on power relations, institutional norms and other organizational constraints [28], define a-priori duties and rights of their members. Indeed, in such systems, agents may become debtors because of the role they are enacting, and without their express will.

Conditional commitments have been used in the past to models situations of this type: e.g., the fact of enacting a certain role has been used as a condition for a conditional commitment, to automatically endorse an agent of the responsibility of the commitment. Another solution is to have the agent vowing to always accept any commitment it will be asked for (it commits to become committed). Then, such vow can be exploited as a condition of a conditional commitment. Both solutions works from a practical viewpoint, although they do not faithfully model the real situation. Indeed, these solutions preserve the principle of an agent becoming debtor on its own will: such principle however is not always accepted in hierarchically organized multiagent systems. Moreover, in our understanding, conditions of conditional commitments capture “conditions”, i.e., dynamically changing situations, upon which the commitments become established. Using conditional commitments to represents also structural organizations, norms and duties is not, in our view, the best way to address the problem.

Let us make all these consideration more practical by means of a simple example. Consider a father ordering his son to do the homework. In our view, the child becomes committed as the consequence of the order, and because of the power relation existing between him and his father. In this case, the commitment’s creation is not due to the debtor, but to the creditor. Moreover, the power relation existing between the father and the son does not depend on the son’s will. Of course, the child still maintains its independence, and he is free to disobey the order, thus *violating* the commitment.

As an alternative to our approach, it is possible to imagine the son promising to the father he will always obey, and use such event within a conditional commitment. Such solution would preserve the agent freedom, but also means that there is no way to ensure the child will commit to anything.

Summing up, both the aspects are equally important: on one side the freedom and autonomy of the agents are a fundamental requirements in open multi-agent systems. On the other side, less open systems would need to impose commitments: in this case commitments would resemble obligations.

Being aware that more flexibility would allow more risks, we opted to leave the domain modeller to choose the preferred approach. Hence our framework allows to model both cases where commitment operations are permitted only to certain agent roles, and those where operations are permitted to all roles. To this end, we explicitly model occurring events with the couple:

$$EventDescription, T$$

where the first term describes the event, and the second term is a variable specifying the time instant the event occurs. When needed, we extend the $EventDescription$ term with all the needed parameters: among them the originator of the event. For instance, a father issuing a "do your homework" order to his son at time instant 5 could be:

$$order(father, son, doHomeWork), 5.$$

Then we specify the commitment operations by defining conditions on the variables. For example, the *creation* operation takes three inputs: the event that generates the commitment, the commitment to be generated, and the time the event occurs. Besides, we can define a *context* (e.g., $father(Father,Son)$). The following definition:

$$\begin{aligned}&create(order(Father, Son, doHomeWork),\\&\qquad \mathsf{C}(Son, Father, property(doHomeWork)),\\&\qquad T) \leftarrow\\&\qquad\quad father(Father, Son).\end{aligned}$$

means that a commitment for agent Son is established if agent $Father$ orders it (the triggering event), and if we are in the context of a father-son relation.

The former behavior, where agents become committed only as consequence of their own actions, can be simply achieved by using the same variable for the originator of the event, and the debtor of the commitment. In this way, the domain modeller can explicitly impose that the commitment is created only when the debtor perform a certain action, thus preserving agents autonomy.

2.4 Commitment operations and life cycle

Our commitment model uses eight distinct operations:

Create. A (base-level or conditional) commitment is created when certain events occur. The domain modeller can specify those events using statements in the form $create(E, \mathsf{C}(X, Y,)P, T)$, where E defines the event triggering the creation of the commitment, $\mathsf{C}(X, Y, P)$ is the commitment to be created, and T is the time when E occurs.

Discharge. A (base-level) commitment is discharged (and gets satisfied) when the property it refers to has been brought about. The domain modeller defines the events that bring about the property. The commitment discharge is an immediate consequence of bringing about such a property. The domain modeller decides if the debtor is the only agent entitled to bringing about the property, or if the property can be brought about by any agent, in order for the commitment to become satisfied.

Cancel. A (base-level) commitment is canceled as a consequence of an act performed by some agent. The user defines which events cause $\mathsf{C}(X, Y, P)$ to get canceled,

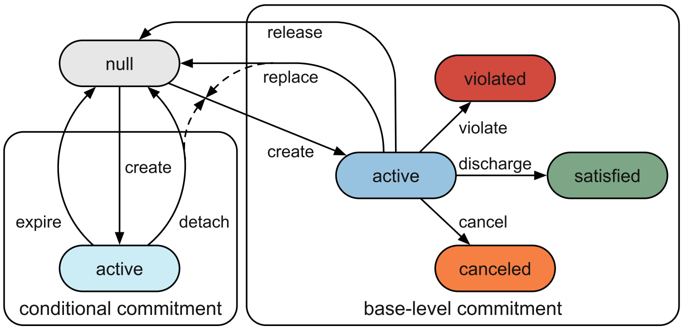

Fig. 2 Commitment life cycle (Color figure online)

via $cancel(E, \mathsf{C}(X, Y, P), T)$ statements, where E defines the event that causes $\mathsf{C}(X, Y, P)$'s cancelation, and T the time when E occurs.

Violate. A (base-level) commitment about a temporal property is violated when the property is not brought about (or not maintained) within the specified time interval. Note that only commitments about temporal properties can be violated. Commitments about simple properties could remain active indefinitely.

Release. Any agent can release a debtor agent from its own (base-level) commitments. This is done via $release(E, \mathsf{C}(X, Y, P), T)$ statements, where E is the event releasing $\mathsf{C}(X, Y, P)$ and T is the time then E occurs.

Replace. The *replace* operation corresponds to traditional (base-level) commitment operations *assign* and *delegate*. This is specified via statements in the form $replace(E, C_{old}, C_{new}, T)$, where C_{old} is the commitment that has to be substituted by C_{new}, if events E occurs at time T. If the debtor changes from the old to the new commitment, we have a delegation; otherwise, we have an assignment.

Detach. A (conditional) commitment is detached when its condition is brought about, and a base level commitment is created as a consequence.

Expire. A (conditional) commitment expires when the deadline for bringing about the condition expires. In that case, no base-level commitment is created.

We thus have five possible states in a commitment's life cycle: one for conditional commitments, which can only be active, and four for base-level commitments, which can be active, violated, satisfied, or canceled. Moreover, null indicates the state of a (base-level or conditional) commitment before or after its life cycle. The operations listed above cause commitments to change state.

The commitment life cycle is illustrated by Fig. 2 as a labelled graph, where the nodes are possible commitment states, solid arcs represent the transitions of a commitment from a state to another one, and dashed arcs represent the creation of a new commitment caused by an operation on an existing commitment. It is worthwhile noticing that the commitment life cycle proposed in Fig. 2 inherits its fundamental structure from previous works, such as those shown in Fig. 1. The *replace* operation is instead new.

Initially, any possible commitment is in the null state. Upon creation, a (conditional or base-level) commitment becomes active. An active base-level commitment can successfully get discharged (hence it becomes satisfied) or it can get canceled/violated; cancel and violation are captured by two different states. Usual operations like release and replace move the commitment to the null state (solid arc), and establish the creation of a new, active base-level commitment (dashed arc).

An active conditional commitment can instead expire, or get detached, terminating in either case its life cycle and entering the null state (solid arc). A detached commitment will establish the creation of a new, active base-level commitment (dashed arc).

3 Social commitments modelling in the event calculus

We formalize our commitment model using the Event Calculus ($\mathcal{EC}$) [45]: a logic formalism for representing and reasoning upon the effects of events in time. In Sect. 3.1 we briefly recall the $\mathcal{EC}$, while in Sects. 3.2 and 3.3 we present the formalization of base-level and conditional commitments, respectively. Finally, in Sect. 3.4 we discuss some effects of unbound variables and in Sect. 3.5 we show how to express complex conditions about temporal constraints.

3.1 The Event Calculus

The $\mathcal{EC}$ was proposed by Kowalski and Sergot [45] as a logic programming framework for representing and reasoning about the effects events have on properties of the world ("fluents") [45]. Events occur at points in time, while fluents hold during intervals of time, and are initiated/terminated by events. Given a set of events ("narrative"), the $\mathcal{EC}$ theory and domain-specific axioms together ("$\mathcal{EC}$ axioms") define which fluents hold at each time. There are many different formulations of these axioms [20]. One possibility is given by axioms ec_1 and ec_2, where P stands for *Fluent*, E for *Event*, and T represents time instants. Predicates $holds\_at$ and $clipped$ capture, respectively, the notion of a fluent that holds at time T, and a fluent that switched from true to false during an interval $[T_1, T_3]$.

$$holds\_at(P,T) \leftarrow initiates(E,P,T_{Start}) \wedge T_{Start} < T \wedge \neg clipped(T_{Start},P,T). \tag{ec_1}$$

$$clipped(T_1,P,T_3) \leftarrow terminates(E,P,T_2) \wedge T_1 < T_2 \wedge T_2 < T_3. \tag{ec_2}$$

$$initiates(E,P,T) \leftarrow happens(E,T) \wedge holds\_at(P_1,T) \wedge \ldots \wedge holds\_at(P_M,T). \tag{ec_3}$$

$$terminates(E,P,T) \leftarrow happens(E,T) \wedge holds\_at(P_1,T) \wedge \ldots \wedge holds\_at(P_N,T). \tag{ec_4}$$

Axioms ((ec_3), (ec_4)) are instead schemas for defining the domain-specific axioms: a certain fluent P is initiated/terminated at time T if an event E occurred at the same time, while some other fluents P_i hold. In other words, a certain event E initiates/terminates a fluent when the event happens and a certain context holds. Within EC, contexts are naturally represented by means of (conjunction of) fluents: since their truthiness values change along time, they are suitable for representing dynamically changing contexts. Summing up, fluents $P_1, \ldots, P_n$ represent particular conditions/situations, and their conjunction represent a specific context. By means of predicates $holds\_at$, it is possible to check if a particular context holds. In general, the context can be any conjunction of literals (defined within the Knowledge Base) and $\mathcal{EC}$ related predicates (such as $holds\_at$).

Table 1 The $\mathcal{EC}$ ontology

$happens(Ev, T)$	Event Ev occurs at time T
$holds\_at(F, T)$	Fluent F holds at time T
$holds\_for(F, [T_1, T_2])$	Fluent F holds during the interval $[T_1, T_2]$
$initially(F)$	Fluent F holds at the beginning of time
$initiates(Ev, F, T)$	Event Ev initiates fluent F at time T
$terminates(Ev, F, T)$	Event Ev terminates fluent F at time T

To say that a fluent holds at the beginning of time we can use the shorthand $initially(P)$. Dual axioms and predicates can be added to define when fluents *do*/*do not* hold [55]: e.g., an axiom can be added to define $declipped/3$ (an event caused a certain fluent to switch from false to true). The $\mathcal{EC}$ formalization above is called *simple* $\mathcal{EC}$ and uses the Horn fragment of first order logic, augmented with negation as failure.

An $\mathcal{EC}$ *theory* is *a knowledge base* $\mathcal{KB}$ composed by a set of clauses (*initiates*, *terminates*, …) that relate events and fluents. The set of all $\mathcal{EC}$ predicates that will be used throughout the paper is listed in Table 1.

Besides $\mathcal{EC}$ predicates, $\mathcal{KB}$ can include any number of logic programming clauses defining any number of predicates. Such clauses enable to model the context, at the desired degree of detail. For example, $\mathcal{KB}$ is the place to define organizational roles and relations and domain-specific knowledge.

Let us consider the following illustration, taken from literature. A fluent *light_on* toggles between true and false at every touch of the light switch. To tell whether the light is on or off, we need to know how the light is at the beginning of time, and what is the effect of using the switch. That depends on the current state: if the light is on, then the effect is to switch it off, and vice versa. A possible $\mathcal{EC}$ theory describing the light switch world is the one below:

$$\begin{aligned}
&initially(light\_on).\\
initiates(switch\_pressed, light\_on, T) &\leftarrow happens(switch\_pressed, T)\\
&\quad \wedge \neg holds\_at(light\_on, T).\\
terminates(switch\_pressed, light\_on, T) &\leftarrow happens(switch\_pressed, T)\\
&\quad \wedge holds\_at(light\_on, T).
\end{aligned}$$

3.2 Base-level commitments

The theory of commitments consists of two different theories, as shown in Fig. 3: a domain-independent theory, that formalizes the life cycle of time-aware commitments, and a theory that describes a specific domain. The domain-independent part relates the initiation/termination of commitment-related fluents by commitments operations, and by doing so it gives a semantics to such operations. The domain-dependent part instead defines the links between the specific events and fluents/commitments. In the following, we present the domain-independent axioms, and provide some illustrations of them.

Our starting point is Yolum and Singh's axiomatization of commitment operations [66,67], also based on the $\mathcal{EC}$. The authors set a correspondence between "bringing about some property p" and "*initiating* fluent p," and model commitments by way of fluents (hence a commitment either *holds* or *does not hold*).

Our formalization differs form the one proposed by Yolum and Singh in two key aspects. First, we map each possible commitment state to a fluent $state/2$, where $state(c, s)$ expresses

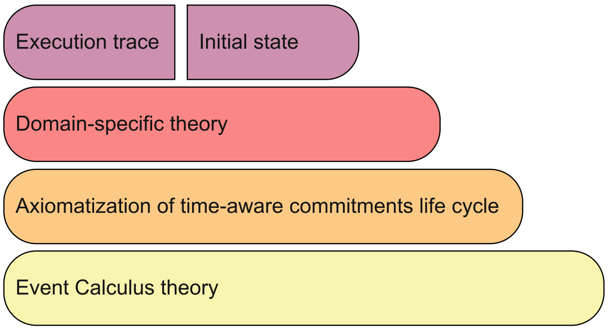

Fig. 3 Layered architecture of the monitoring framework (Color figure online)

that commitment c is in state s. By reifying the commitment state, we can explicitly refer to the state in the domain dependent theory. For instance, we can state that a commitment c is created by event ev if another commitment c_2 is currently active. Second, we explicitly deal with data, variables, and temporal aspects.

Axiom 1 (*State of a commitment*) A commitment C is active (respectively satisfied, violated, ...) at time T if the corresponding state fluent holds at time T.

$$\begin{aligned} active(C,T) &\leftarrow holds\_at(state(C, active), T).\\ canceled(C,T) &\leftarrow holds\_at(state(C, canceled), T).\\ satisfied(C,T) &\leftarrow holds\_at(state(C, satisfied), T).\\ &\dots \end{aligned} \tag{1}$$

Axiom (1) can be trivially extended to accommodate as many other states as needed.

3.2.1 Creating base-level commitments

A commitment is created as a consequence of the occurrence of events. The user specifies such events in the domain-dependent $\mathcal{KB}$, by means of facts like $create(E, \mathsf{C}(X, Y, P), T)$, where E stands for a particular event, T stands for the time when E occurs, and $\mathsf{C}(X, Y, P)$ has the usual meaning. The commitment enters the active state, and a proper *state* fluent is initiated.

Axiom 2 *(Create/Base-level Commitment)*

$$\begin{aligned} initiates(E, state(\mathsf{C}(X, Y, property(P)), active), T) \leftarrow&\\ create(E, \mathsf{C}(X, Y, property(P)), T).& \end{aligned} \tag{2.1}$$

$$\begin{aligned} initiates(E, state(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), active), T) \leftarrow&\\ create(E, \mathsf{C}(X, Y, property(e(T_1, T_2), P))), T),&\\ T < T_1, T_1 < T_2.& \end{aligned} \tag{2.2}$$

$$\begin{aligned} initiates(E, state(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), active), T) \leftarrow&\\ create(E, \mathsf{C}(X, Y, property(u(T1, T2), P))), T),&\\ T < T_1, T_1 < T_2.& \end{aligned} \tag{2.3}$$

Axioms (2.1)–(2.3) define the conditions for initiating a *state* fluent for the specified commitment. These axioms, as well all the axioms defined in the paper, are Prolog-like rules. For example, Axiom (2.1) should be read in the following way: a fluent *state(C(…), active)* should be initiated at the happening of event E at time instant T if the evaluation of a predicate *create(…)* is successful. The predicate *create(…)* is defined by the domain modeller: in this way, she/he can decide which are the conditions for the commitment to be established. In Example 1 it is shown a *create(…)* predicate defined by the user/domain modeller.

Note that while for a simple property there are no time constraints (Axiom (2.1)), things change if we consider temporal properties. In particular, we explicitly require for both existentially- and universally-quantified properties that the time intervals attached to the property are both about time instants later than the time of the event (Axiom (2.2) and (2.3), respectively). An event occurring at a time T triggers only those commitments about existential (universal) properties that are expected to be brought about (hold for an interval of time, respectively) in the future. It is not possible that an event occurring at time T generates a commitment that could be satisfied only by past events.

Example 1 Let us consider a home rule. Whomever promises to prepare the dinner, becomes committed to preparing it in no longer than 2 hours:

$$\begin{aligned}&create(promise(P_1, P_2, prepareDinner),\\ &\qquad \mathsf{C}(P_1, P_2, property(e(T_1, T_2), dinnerReady)),\\ &\qquad T) \leftarrow\\ &\qquad\quad T_1\ is\ T+1, T_2\ is\ T+2.\end{aligned}$$

Now suppose that we observe the following event:

$$promise(john, mike, prepareDinner) \quad 32$$

The happened event matches with the description given in the *create(…)*, and in particular P_1 unifies with *john*, P_2 with *mike*, and the time instant 32 unifies with T. As a consequence of such unification, John becomes committed to prepare the dinner between time 33 and time 34:

$$\mathsf{C}(john, mike, property(e(33, 34), dinnerReady)).$$

Hence, John is expected to take an action at a time point in the interval [33, 34], whose consequence is to bring about the property *dinnerReady*. □

3.2.2 Discharging commitments

Traditionally, a commitment about a generic property is discharged when *the debtor* brings about such a property [66,67]. The effect of a discharge operation on a commitment is that the commitment state changes from active to satisfied. In our formalization this means that the fluent representing the active state is terminated, and a new fluent representing the satisfied state is initiated. Axioms (3.1) and (3.2) define such state transition.

Axiom 3 *(Discharge)*

$$\begin{aligned}terminates(E, state(Commitment, active), T) \leftarrow&\\ discharge(E, Commitment, T).&\end{aligned} \tag{3.1}$$

$$\begin{aligned} initiates(E, state(Commitment, satisfied), T) \leftarrow \\ discharge(E, Commitment, T). \end{aligned} \tag{3.2}$$

$$\begin{aligned} discharge(E, \mathsf{C}(X, Y, property(P)), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(P)), T), \\ initiates(E, P, T). \end{aligned} \tag{3.3}$$

$$\begin{aligned} discharge(E, \mathsf{C}(X, Y, property(e(T_1, T_2), P))), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), T), \\ T \geq T_1, T \leq T_2, \\ initiates(E, P, T). \end{aligned} \tag{3.4}$$

$$\begin{aligned} discharge(\_, \mathsf{C}(X, Y, property(u(T_1, T_2), P))), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), T), \\ holds\_for(P, [T_1, T_2]), \\ T \geq T_2. \end{aligned} \tag{3.5}$$

Axiom (3.3) defines the condition for discharging a commitment about a simple property, i.e., an event occurs, which initiates P. The only requirement is that when the event occurs, the commitment should be active (i.e., already established). Axiom (3.4) instead focuses on commitments about existential properties. Such commitments are discharged if an event that brings about P occurs at a time instant within the time interval specified in the commitment.

A commitment about a universal property (see Axiom (3.5)) can get discharged only when it is possible to determine that the property P has held for the entire specified time interval. If such interval is $[T_1, T_2]$, we need to wait until T_2 before saying that P has held during the whole interval. Axiom (3.5) states that the commitment can get discharged (hence its state becomes satisfied) if, when any event[5] occurs at time T, the following conditions hold: (i) the commitment is still active at time T; (ii) the property P held for the entire interval; and (iii) T is equal or greater than the end of the interval, i.e., T_2.

A interesting possible extension arises if we consider in detail Axioms (3.3) and (3.4). To discharge the commitment, they require that something occurs that brings about the property. What if, instead, the property already holds? Governatori and Rotolo, in [39], when explaining the different types of obligations in deontic logic, distinguish this case by referring to non-preemptive vs. preemptive obligations. If we borrow their definition, we can say that commitments whose property must be brought about *after* the commitment is established, are *non-preemptive*. Otherwise, commitments whose property is considered satisfied also by events occurred *before* the commitment has been established, are *preemptive*. These two situations correspond to two different notions of *commitment satisfaction*. Interestingly, we are not aware of previous work that makes distinction about these two different notions. Our model and framework, in their current version, support only the former notion, i.e. non-preemptive commitments. However, thanks to the declarative nature of our approach, the commitment model can be easily extended to support preemptive commitments. E.g., it would be enough to modify Axiom (2.1), and to add a new one. The new semantics would be the following: if a property is already satisfied when generating a commitment about a

[5] Note the use of the underscore in the position reserved to the event description, meaning that *any* event can trigger the axiom.

simple property, the commitment would not be created, but rather immediately get satisfied (Axiom (2.1)). A further Axiom would be needed to establish that such commitments has been satisfied (as if it would have been discharged).

3.2.3 Canceling and violating commitments

As we discussed in Sect. 2.2, the case in which the debtor (or another agent) decides to cancel the commitment differs from the case in which the debtor does not satisfy the commitment by not bringing about/maintaining the property. We say in the former case that the commitment got *canceled*, in the latter case that it got *violated*. Axioms (4.1)–(4.6) model cancellation, while Axioms (5.1)–(5.4) model violation.

Axiom 4 *(Cancel)*

$$\begin{aligned} terminates(E, state(\mathsf{C}(X, Y, P), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, P), T), \\ cancel(E, \mathsf{C}(X, Y, P), T), \\ on\_time(T, P). \end{aligned} \tag{4.1}$$

$$\begin{aligned} initiates(E, \mathsf{C}(X, Y, P), canceled), T) \leftarrow \\ active(\mathsf{C}(X, Y, P), T), \\ cancel(E, \mathsf{C}(X, Y, P), T), \\ on\_time(T, P). \end{aligned} \tag{4.2}$$

$$on\_time(\_, property(\_)). \tag{4.3}$$

$$on\_time(T, property(e(\_, T_2), \_)) \leftarrow T \leq T_2. \tag{4.4}$$

$$on\_time(T, property(u(T_1, T_2), \_)) \leftarrow T \leq T_1. \tag{4.5}$$

$$on\_time(T, property(u(T_1, T_2), P)) \leftarrow T \leq T_2, T \geq T_1, holds\_for(P, T_1, T). \tag{4.6}$$

Axioms (4.3)–(4.5) address all types of properties defined earlier. A commitment about a simple property can be canceled at any time. Instead, a commitment about a temporal property can get cancelled only if it is active, and in any case before the deadline expires. After that, the commitment gets violated and as such it cannot get cancelled any more. Note that, in case of universally quantified properties, cancellation can occur at time T if T is before the time interval $[T_1, T_2]$ (Axiom (4.5)), or if the property hold up to time T (Axiom (4.6)).

Axiom 5 *(Violate)*

$$\begin{aligned} terminates(\_, state(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), T), \\ T > T_2. \end{aligned} \tag{5.1}$$

$$\begin{aligned} initiates(\_, state(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), violated), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), T), \\ T > T_2. \end{aligned} \tag{5.2}$$

$$\begin{aligned} terminates(\_, state(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), T), \\ T \geq T_1, T \leq T_2, \\ not\_holdsat(P, T). \end{aligned} \tag{5.3}$$

$$\begin{aligned} initiates(\_, state(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), violated), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), T), \\ T \geq T_1, T \leq T_2, \\ not\_holdsat(P, T). \end{aligned} \tag{5.4}$$

Axioms (5.1) and (5.2) specify when a commitment about an existential property gets violated. Since the property is expected to be brought about at some point within an interval $[T_1, T_2]$, only after T_2 can we tell that the commitment has been violated. After T_2, by the time (T) any event occurs, if the commitment is still active then the property has not been brought about (otherwise the commitment would be in the satisfied state). Hence, the active state is terminated (5.1) and the violated state is initiated (5.2).[6]

Axioms (5.3) and (5.4) specify when a commitment about a universal property is violated: whenever any event occurs at a time instant T, such that $T \in [T_1, T_2]$, and the property does not hold in T, then the commitment is violated. Note that Axioms (5.3) and (5.4) are quite inefficient, since they check for the validity of property P every time any event happens at a time instant $T \in [T_1, T_2]$. The rationale behind these Axioms is to detect the violation as soon as possible. Better performance could be obtained by releasing such desiderata.

Example 2 Let us consider a Traffic Agent (TA), a car identified by its Plate number, and the car's Owner. TA presents Owner with a traffic ticket, thus Owner is now expected to pay the fine, by a deadline, or, by the same deadline, to refute the traffic violation:

$$\begin{aligned} & create(issueFine(TA, Owner, fine(Plate, Amount)), \\ & \qquad \mathsf{C}(Owner, city, property(e(T_1, T_2), finePaid(Plate, Amount))), \\ & \qquad T) \leftarrow \\ & \qquad\; T_1 \; is \; T + 1, \\ & \qquad\; T_2 \; is \; T + 30. \end{aligned}$$

$$\begin{aligned} & initiates(payment(Owner, city, fine(Plate, Amount)), \\ & \qquad finePaid(Plate, Amount)), \\ & \qquad T). \\ & cancel(refutation(Owner, city, fine(Plate, Amount)), \\ & \qquad \mathsf{C}(Owner, city, property(e(T_1, T_2), finePaid(Plate, Amount))), \\ & \qquad T). \end{aligned}$$

When TA issues a fine, Owner becomes committed to paying the fine. Note that the creditor of such a commitment is the City Council (just "city" in the axioms above), and not TA, who has in fact generated the commitment. The commitment gets discharged when the owner

[6] This in fact depends on our ability to observe a system's evolution. We can ensure that a state of violation is flagged in the moment we become aware of the time, by either registering events at the time they occur, or for example by setting a special "alarm" event to fire at exactly one instant after T_2 (in which case, $T = T_2 + 1$).

pays the Amount indicated in the fine ticket. However, Owner can decide not to pay the fine. The cancelation of the commitment is a consequence of a refutation event, performed by Owner, provided that the refutation is made before the payment deadline.

Some authors have stressed the need for establishing "compensation" commitments in the case of cancelation [60]. In our example, the City Council requires the citizen to provide evidence for refuting the fine, within the next 5 days, as of the refutation. The following formula defines how a new commitment is created as a consequence of the refutation:

$$\begin{aligned}&create(refutation(Owner, city, fine(Plate, Amount)),\\ &\qquad \mathsf{C}(Owner, city, property(e(T_1, T_2), provideEvidence(Plate, Amount))),\\ &\qquad T) \leftarrow\\ &\qquad\quad T_1\ is\ T+1,\\ &\qquad\quad T_2\ is\ T+5.\end{aligned}$$

Now suppose that we observe the following events, involving agent john (the Owner) and agent ta51 (the TA), and happening at time instants 31 and 40, respectively:

$$\begin{array}{ll} issueFine(ta51, john, fine(ac123, 100\$)) & 31\\ refutation(john, city, fine(ac123, 100\$)) & 40\end{array}$$

When TA fines John, at time 31, the following commitment is established:

$$\mathsf{C}(john, city, property(e(32, 61), finePaid(ac123, 100\$)))$$

However, John refutes the fine at time 40. As a consequence, the previous commitment is canceled, and at the same time, a new commitment is created for John:

$$\mathsf{C}(john, city, property(e(41, 45), provideEvidence(ac123, 100\$)))$$

If John fails to provide evidence against the fine, this commitment's violation may be detected at any time after 45. □

3.2.4 Releasing commitments

The creditor of a commitment (or any other agent, if allowed) can release the debtor from such a commitment. As a consequence, the commitment is removed. The *release* operation can be performed only when a commitment is active. Moreover, we add a further constraint for commitments about temporal properties: if a property is defined over a time interval $[T_1, T_2]$, the corresponding *release* operation can be performed only before or at T_2. Hence we rely on the *on_time* predicate defined in Axioms (4.3)–(4.6).

Axiom 6 (Release)

$$\begin{aligned}terminates(E, state(\mathsf{C}(X, Y, P)), active), T) \leftarrow&\\ active(\mathsf{C}(X, Y, P), T),&\\ release(E, \mathsf{C}(X, Y, P), T),&\\ on\_time(T, P).&\end{aligned}\tag{6.1}$$

Axiom (6.1) defines the conditions for a commitment to be released: (i) the commitment must be still in the active state; (ii) a proper $release(\ldots)$ fact be in the $\mathcal{KB}$; and (iii) if all possible time constraints are satisfied.

3.2.5 Replacing commitments: assignment and delegation

As we discussed in Sect. 2.3, our approach extends traditional commitments by relaxing the constraint on who is allowed to perform certain operations. Such an improvement has a great impact on the traditional operations of *assign* and *delegate*: from the formal viewpoint, they differ only because the first operation is about changing the creditor, while the second one is about changing the debtor of a commitment. Our approach lets the domain modeller define which one of the involved roles has to be substituted, hence a single *replace* operation suffices to accommodate both assignment and delegation. Depending on how the user defines it, it can assume the behaviour of assignment, delegation, or both.

Note also that since assignment, delegation, and replacement operations lead to the creation of a new commitment, all the temporal constraints, if any, will have to be propagated to the new commitment.

A *replace* specification defined by the user is in the form:

$$replace(E, C_{old}, C_{new}, T),$$

where E is the event, occurring at time T, that fires the replacement of commitment C_{old}, by which a new commitment C_{new} will be created. Axioms 7.1 and 7.2 define the behaviour of the replace operation in the case of commitments about simple properties; Axioms 7.3 and 7.4 refer to commitments about existential properties, while 7.5 and 7.6 refer to commitments about universal properties.

Axiom 7 *(Replace)*

$$\begin{aligned} terminates(E, state(\mathsf{C}(X, Y, property(P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(P)), T), \\ replace(E, \mathsf{C}(X, Y, property(P)), \mathsf{C}(\_, \_, property(P)), T). \end{aligned} \tag{7.1}$$

$$\begin{aligned} initiates(E, state(\mathsf{C}(X_1, Y_1, property(P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(P)), T), \\ replace(E, \mathsf{C}(X, Y, property(P)), \mathsf{C}(X_1, Y_1, property(P)), T). \end{aligned} \tag{7.2}$$

$$\begin{aligned} terminates(E, state(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), T), \\ T \leq T_2, \\ replace(E, \mathsf{C}(X, Y, property(e(T_1, T_2), P))), \mathsf{C}(\_, \_, property(e(\_, \_), P))), T). \end{aligned} \tag{7.3}$$

$$\begin{aligned} initiates(E, state(\mathsf{C}(X_1, Y_1, property(e(T_3, T_4), P))), active), T) \leftarrow \\ active(\mathsf{C}(X, Y, property(e(T_1, T_2), P))), T), \\ T \leq T_2, \\ replace(E, \mathsf{C}(X, Y, property(e(T_1, T_2), P))), \mathsf{C}(X_1, Y_1, property(e(T_3, T_4), P))), T), \\ T < T_3, T_3 < T_4. \end{aligned} \tag{7.4}$$

$$\begin{aligned}
&terminates(E, state(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), active), T) \leftarrow \\
&\quad active(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), T), \\
&\quad T \leq T_2, \\
&\quad replace(E, \mathsf{C}(X, Y, property(u(T_1, T_2), P))), \mathsf{C}(\_, \_, property(u(\_, \_), P))), T),
\end{aligned} \tag{7.5}$$

$$\begin{aligned}
&initiates(E, state(\mathsf{C}(X_1, Y_1, property(u(T_3, T_4), P))), active), T) \leftarrow \\
&\quad active(\mathsf{C}(X, Y, property(u(T_1, T_2), P))), T), \\
&\quad T \leq T_2, \\
&\quad replace(E, \mathsf{C}(X, Y, property(u(T_1, T_2), P))), \mathsf{C}(X_1, Y_1, property(u(T_3, T_4), P))), T), \\
&\quad T < T_3, T_3 < T_4.
\end{aligned} \tag{7.6}$$

We impose that P is the same in the new commitment as it used to be in the old one, while debtor, creditor and the time intervals can change on the needs of the user. An assignment is a special case of replacement, in which the debtor does not change ($X = X_1$), whereas a delegation is a special case of replacement, in which the creditor does not change ($Y = Y_1$).

Note that replacement is a valid option only if the commitment is still active. Therefore, we impose that, in case of commitments about temporal properties, a replacement can be done only if the firing event occurs before the end of the time interval associated with P (condition $T \leq T_2$ in Axioms 7.3–7.6). Moreover, the newly created commitment must always be about temporal properties over a time interval that follows T (condition $T < T_3, T_3 < T_4$ in Axioms 7.4 and 7.6).

Example 3 Let us now consider a conference proceedings preparation scenario. The conference has strict deadlines for paper submission. All accepted papers' corresponding authors are committed to sending the camera ready copy of their paper to the Editor, within a certain deadline:

$$\begin{aligned}
&create(paper\_accepted(Editor, Auth, PaperID), \\
&\qquad \mathsf{C}(Auth, Editor, property(e(T_1, 39), submitCRC(PaperID)), \\
&\qquad T) \leftarrow \\
&\qquad\quad T < 30, T_1 \; is \; T + 1.
\end{aligned}$$

The intended meaning is that if the Editor communicates within a fixed deadline (before time instant 30) that the paper has been accepted, then the author is committed to providing the camera-ready copy any time after the notification, but before time 39 (a fixed temporal deadline). Now suppose that we observe the following event happening at time 24, in which agent John is Editor:

$$paper\_accepted(john, david, paper51) \quad 24$$

As a consequence of John's communication to David that paper51 has been accepted, david is committed to providing the paper's CRC any time before the deadline:

$$\mathsf{C}(david, john, property(e(25, 39), submitCRC(paper51))$$

Now suppose that John decides to resign from the role of Editor. We can accommodate such a situation, using a $replace(\ldots)$ statement. The change of editor does not have any impact on the deadline, therefore the authors are still committed to providing the work within the same deadline as before.

$$\begin{aligned}&replace(editorChanged(OldChair, NewChair),\\&\quad \mathsf{C}(Auth, OldChair, property(e(T_1, T_2), submitCRC(PaperID)),\\&\quad \mathsf{C}(Auth, NewChair, property(e(T_3, T_2), submitCRC(PaperID)),\\&\quad T) \leftarrow\\&\qquad T_3 = T + 1, T_3 < T_2.\end{aligned}$$

The intended meaning is that upon the notification that the Editor has changed, the old commitment is substituted with a new one with a new creditor, and the time interval to submit the paper has been slightly changed: the deadline is always the same, but now the lower limit is after the occurrence of the event. Suppose to observe the fact:

$$editor\_changed(john, marc) \quad 31$$

If, by now, David has already sent his CRC to John, David is not affected by the change of editor; otherwise, he will be committed to sending the CRC to Marc:

$$\mathsf{C}(david, marc, property(e(32, 39), submitCRC(paper51))$$

□

Example 3 has shown an assignment, in which the deadline for bringing about a property remains unchanged. In similar ways we can model delegations and deadline extensions.

Note that in [21] the authors refer to the notion of *delegation with/without retaining responsibility*. In the presented formalization we opt for delegation without retaining responsibility. In order to accommodate the delegation with retaining responsibility, we may either remove Axioms (7.3) and (7.5), or else we may extend the $replace(\dots)$ term, by including conditions that distinguish among different types of delegation.

3.3 Conditional commitments

Conditional Commitments are created as a consequence of a certain event, and similarly to commitments, they are *active* upon creation. In a conditional commitment

$$\mathsf{CC}(X, Y, Q, P)$$

Q and P can be simple or temporal properties. When Q holds, the following base-level commitment enters the *active* state:

$$\mathsf{C}(X, Y, P)$$

If P is a temporal property, it should always refer to time intervals that start after the base-level commitment is created.

Axiom 8 *(Create/Conditional Commitment)*

$$\begin{aligned}initiates(E, state(\mathsf{CC}(X, Y, Q, P), active), T) \leftarrow&\\ create(E, \mathsf{CC}(X, Y, Q, P), T),&\\ impose\_time\_constraint(T, Q).&\end{aligned} \tag{8.1}$$

$$\begin{aligned}&impose\_time\_constraint(T, property(\_)).\\&impose\_time\_constraint(T, property(e(T_s, T_e), \_)) \leftarrow T < T_s, T_s < T_e.\\&impose\_time\_constraint(T, property(u(T_s, T_e), \_)) \leftarrow T < T_s, T_s < T_e.\end{aligned} \tag{8.2}$$

In order to simplify the definition of Axiom (8.1), we introduced another predicate, *impose_time_constraint/2* (Axiom (8.2)), which checks if Q is a temporal property and, if it is the case, imposes the constraint that Q should be related to a time interval *following* the time the conditional commitment is created.

The detachment of a conditional commitment $\mathsf{CC}(X, Y, Q, P)$ to a base-level commitment happens when Q holds. Similarly to what happens for the discharging of base-level commitments (see Sect. 3.2.2), *a*) a conditional commitment is detached as soon as Q holds, if Q is a simple or an existentially-quantified property; or *b*) it is detached as soon as it is possible to verify that Q held for the whole specified time interval, if Q is a universally-quantified property.

Axiom 9 *(Detach)*

$$\begin{aligned} terminates(E, state(\mathsf{CC}(X, Y, Q, P), active), T) \leftarrow \\ detach(E, \mathsf{CC}(X, Y, Q, P), T). \end{aligned} \tag{9.1}$$

$$\begin{aligned} initiates(E, state(\mathsf{C}(X, Y, P), active), T) \leftarrow \\ detach(E, \mathsf{CC}(X, Y, Q, P), T). \end{aligned} \tag{9.2}$$

$$\begin{aligned} detach(E, \mathsf{CC}(X, Y, property(Q), P), T) \leftarrow \\ active(\mathsf{CC}(X, Y, property(Q), P), T), \\ initiates(E, Q, T), \\ impose\_time\_constraint(T, P). \end{aligned} \tag{9.3}$$

$$\begin{aligned} detach(E, \mathsf{CC}(X, Y, property(e(T_1, T_2), Q), P), T) \leftarrow \\ active(\mathsf{CC}(X, Y, property(e(T_1, T_2), Q), P), T), \\ initiates(E, Q, T), \\ T_1 \leq T, T \leq T_2, \\ impose\_time\_constraint(T, P). \end{aligned} \tag{9.4}$$

$$\begin{aligned} detach(E, \mathsf{CC}(X, Y, property(u(T_1, T_2), Q), P), T) \leftarrow \\ active(\mathsf{CC}(X, Y, property(u(T_1, T_2), Q), P), T), \\ holds\_for(Q, [T_1, T_2]), T > T_2, \\ impose\_time\_constraint(T, P). \end{aligned} \tag{9.5}$$

Axioms (9.1) and (9.2) define the effects of the detachment operation: the fluent representing the active state of the conditional commitment is terminated, while a new (base-level) commitment enters the active state. Axioms (9.3), (9.4), and (9.5) instead define the conditions for the detachment to happen. In particular, depending on the type of property Q, we can have the following cases:

- Q is a simple property: the conditional commitment is detached if E initiates Q, and if it is possible to impose temporal constraints to the property P, as specified in Axiom (9.3) (the latter operation is done by the predicate *impose_time_constraint/2*, that imposes constraints only if P is effectively a temporal constraint);

- Q is an existentially quantified property: the commitment is detached if an event E occurs such that E initiates Q. E must occur within the time interval linked to Q, and a proper temporal constraint is added between the time instant the detachment happens, and the property P, by means of the predicate *impose_time_constraint/2* (see Axiom (9.4));
- Q is a universally quantified property: the commitment is detached as soon as it is possible to verify that the property Q held during the whole time interval associated with Q; again, a proper temporal constraint is added between the time instant the detachment happens, and the property P, by means of the predicate *impose_time_constraint/2*(see Axiom (9.5)).

Axiom 10 *(Expire)*

Not unlike the violation of a commitment about temporal properties, expiration too regards only commitments with temporal conditions. The axioms for deciding the expirations are similar to those for detecting a violation: when the deadline for bringing about the condition has passed, then the conditional commitment cannot be detached any more. However, this situation does not cause a violation, but the mere elimination of the conditional commitment.

We do not report here the axioms concerning the expiration, because they are very similar to the ones of violation in Sect. 3.2.3. They are, however, in the implementation (see Sect. 4).

3.4 Commitment monitoring and irrevocability

Our formalization of the commitment models is based on a first order $\mathcal{EC}$ framework. Thanks to first order logic, we could use variables to define axioms in a very general form. However, in the context of online monitoring, the use of variables could lead to undesired effects. Such effects become an issue when using logics for modelling real-world situations. In the real world, we cannot "retract" or "backtrack" from the effects of actions. Logic-based models instead may very well accommodate such a behaviour.

Example 4 Suppose a husband is committed towards his wife to having some (otherwise unspecified) food ready for dinner:

$$\mathsf{C}(husband, wife, haveFood(X)).$$

Note that the property is only partially specified: we do not know in advance which food is to be obtained (X is unbound). Although this is a "strange" situation, where an agent is committed to something that is (partially) unknown, in the $\mathcal{EC}$ framework (in first order logic) this is perfectly well allowed, thanks to the variables.

The husband buys some *pizzas*, but then he remembers his wife does not really like it, and so he decides to buy also some *chicken*. We could observe the following facts happening at time 20 and 30, respectively:

$$buy(husband, food(pizza)) \quad 20$$
$$buy(husband, food(chicken)) \quad 30$$

Suppose that the $\mathcal{KB}$ also defines that the act of buying food satisfies the property *have-Food(X)*:

$$initiates(buy(husband, food(X)), haveFood(X), \_).$$

From the viewpoint of logic and $\mathcal{EC}$, this situation has two immediate solutions: either the husband was committed to buying some pizza, and he fulfilled such obligation at time

20 ($X = pizza$); or the husband was committed to buying some chicken, and he achieved the task at time 30 ($X = chicken$). There is also a third option, by which the husband was committed to the property $haveFood(X) \wedge (X \neq pizza) \wedge (X \neq chicken)$. In that case, the husband did not fulfilled his commitment. □

A sound and complete implementation of the $\mathcal{EC}$, such as $\mathcal{REC}$ [15], computes all the three answers above. However, in a run-time monitoring scenario, having so many answers could represent an issue. Suppose that the wife is using our monitoring facility to check her husband's behaviour. At time 20 the husband buys the pizza, and the monitor facility signals that the commitment got satisfied (discharged). Unfortunately, the wife does not like the pizza, and immediately starts complaining to her husband. Then the monitor facility, at time 30, detects that the husband has also bought some chicken, and then it provides a second analysis (together with the previous one): the commitment got satisfied, in one case with pizza, in the other with chicken. It is clear that any action taken by the wife between time 20 and 30 must be rethought in the light of the new events.

In an on-line monitoring setting, outputs must always be *irrevocable*, i.e., come what may, the answer computed so far by the monitoring facility (by our implementation of $\mathcal{EC}$) will not change [15]. The irrevocability issue is intrinsic in the $\mathcal{EC}$ based on first order logic. Many previous implementations of $\mathcal{EC}$ could ignore this issue, because they were restricted to propositional fluents. In $\mathcal{REC}$ we accommodate variables in fluents, and we leave it to the user to decide whether an irrevocable behaviour is required or not. In [15], we discuss the notion of *irrevocability* in detail, and provide some conditions to ensure it.

3.5 Specifying complex conditions involving temporal information

In Sects. 3.2 and 3.3 we introduced a possible formalization of the commitment operations seen in Sect. 2.4, with all the extensions proposed in our model. However, the use of the $\mathcal{EC}$ as underlying framework enables representing and reasoning upon more complex situations, in particular with respect to temporal aspects.

Example 5 Suppose that, because of a "sanction" event, a car driver has to pay a fine, whose amount depends on the day the fine is paid, e.g., if the payment occurs at day 5, the amount is 5$. A commitment for the driver should be created, as the "sanction" event occurs:

$$\begin{aligned} create(&sanction(Driver),\\ &\mathsf{C}(Driver,\ cityCouncil,\ property(payFine)),\\ &T). \end{aligned}$$

However, at the time of creation, we cannot express the amount to be paid, because that depends on the time of the "payment" event. Thanks to the $\mathcal{EC}$, one can define such a constraint via the conditions on initiation/termination of property *payFine*:

$$initiates(payment(X),\ payFine,\ X).$$

□

Example 6 A tax payer did not pay the taxes within the deadline. The tax payer can still pay the taxes within 30 days. In that case, she must also pay a surcharge equal to the delay. Suppose that the deadline for paying the taxes was day 60:

$$\mathsf{C}(taxPayer, country, property(e(61, 90), payTaxesWithDelay)).$$

In this example the amount that must be paid depends on two values: the day of the payment, and the first day of the period in which the payment is expected. We can model this situation, by defining a special discharge operation:

$$\begin{aligned}
&discharge(pay(TaxPayer,\ X),\\
&\qquad \mathsf{C}(TaxPayer, taxOffice, property(e(T_1, T_2), payTaxesWithDelay)),\\
&\qquad T) \leftarrow\\
&\qquad\quad active(\mathsf{C}(TaxPayer,\\
&\qquad\qquad taxOffice,\\
&\qquad\qquad property(e(T_1, T_2), payTaxesWithDelay), T),\\
&\qquad\quad T \geq T_1, T \leq T_2,\\
&\qquad\quad initiates(pay(TaxPayer,\ X), payTaxesWithDelay, T),\\
&\qquad\quad X = T - T_1.
\end{aligned}$$

This means that the event of paying X (when we observe the event, X will be ground) discharge the commitment if the following conditions hold: (i) the commitment is currently in the active state; (ii) the payment occurs within the specified temporal interval; (iii) the payment brings about the property $payTaxesWithDelay$; and (iv) the amount paid X is exactly the difference between the time instant T (when the payment occurs) and the lower bound T_1 of the interval. The difference with respect to Axiom (3.4), is that here we added the link (the equality constraint) between the variable X and the time of the event. □

The examples discussed so far show that commitments are about properties satisfied at certain time instants. It might be necessary to refer to such time instants when defining the properties. This is particularly true for conditional commitments, where beside the property P interested by the commitment, there is also the condition Q that triggers the commitment detachment.

4 An illustration

In this section we show how to use our monitor framework to represent and monitor the car rental example discussed in the introduction. All the examples discussed in this paper, as well as the application illustrated in this section, have been executed using our reactive $\mathcal{EC}$ implementations: j$\mathcal{REC}$ [14,15] and a pure Prolog $\mathcal{REC}$ version. In both versions, reactivity is what really enables monitoring applications.

The j$\mathcal{REC}$ tool, presented in [15], is based on the formal framework SCIFF [1,2,64]. Being this implementation based on SCIFF, it inherits all the SCIFF formal properties. Moreover, in [15] we also proved some important formal properties, such as the *irrevocability* property and the formal conditions to be fulfilled in order to have such property guaranteed. On a more down-to-earth note, j$\mathcal{REC}$ also features a graphical interface, which displays the evolution of commitments during the execution of the modelled system. Moreover, thanks to its embedding in a Java archive, j$\mathcal{REC}$ could be seamlessly integrated with other applications that need monitoring the state of commitments or other properties of the domain.

The pure Prolog $\mathcal{REC}$ implementation exploits ideas introduced by Chittaro and Montanari [18]. It can be proven that this second version is equivalent to the first version, i.e., given the same input, the outputs of the two implementations are the same.

We evaluated both implementations empirically. By our initial experiments, it is apparent that the second one outperforms the first one, in terms of runtime. Hence this second version may be preferable to the first one for event-intensive monitoring tasks. All the runtimes presented in this article are based on this second version. A distribution of j$\mathcal{REC}$ is publicly available, which is equipped with the sources and the example trace illustrated in this section.[7]

4.1 Modelling the car rental example

We start by defining the *context* that must hold to consider the contract sentences as applicable. We can easily represent such context by means of a fluent *contract(…)*, that will report all the interesting information for the contract.[8] The contract will be established when both the involved parties sign the contract, and will be terminated (by any event) as soon as the rental period is passed.

$$\begin{aligned}
&initiates(sign(X, contract(C,\ A,\ CarPlate,\ Start,\ End)),\\
&\qquad contract(C,\ A,\ CarPlate,\ Start,\ End),\\
&\qquad T) \leftarrow\\
&\qquad all\_signatures\_done(sign(X,\ contract(C,\ A,\ CarPlate,\ Start,\ End)),\ T).
\end{aligned} \tag{ex1}$$

$$\begin{aligned}
&all\_signatures\_done(sign(\ X,\ contract(\ C,\ S,\ CarPlate,\ Start,\ End)), T) \leftarrow\\
&\qquad (X = C\ ;\ X = S),\\
&\qquad happens(sign(\ C,\ contract(\ C,\ S,\ CarPlate,\ Start,\ End)), T_1),\ T_1 \le T,\\
&\qquad happens(sign(\ S,\ contract(\ C,\ S,\ CarPlate,\ Start,\ End)),\ T_2),\ T_2 \le T.
\end{aligned}$$

$$\begin{aligned}
&terminates(\_,\\
&\qquad contract(C,\ A,\ CarPlate,\ Start,\ End),\\
&\qquad T) \leftarrow\ T > End.
\end{aligned} \tag{ex2}$$

In Axioms (ex1)–(ex2) the intended meaning of the parameters is the following: C is the customer, A is the car rental agency; and $CarPlate$ is the plate of car that will be rented, from the day $Start$ to the day End (the rental period is given in absolute terms). In particular, (ex1) specify that upon the happening of an event of signing a contract, the fluent *contract(…)* is initiated if all the signatures have been done. This condition is checked by means of the auxiliary predicate *all_signatures_done*: this predicate checks only if both the parties have signed the contract, independently of the order they signed. It would be easy to add further conditions, such as for example assuring the customer has a valid driving license, by simply extending this predicate.

The first sentence (S1) can be easily represented as a base-level commitment about a temporal related property. In particular, the property is about to bring back the car within a certain deadline, thus it is an existentially-related property. Axiom (ex3) specify the conditions for creating the commitment: the contract should hold, and the customer should retire

[7] j$\mathcal{REC}$ with the commitment model, the example model and the trace example can be obtained from http://ai.unibo.it/projects/comMon.

[8] To keep the example easy to understand, we consider as important information the peers signing the contract (the Customer C and the agency A), the plate of the car being rented, and the starting/ending days of the rental. Other information, such as a contract's ID, can be easily added if needed.

the car any day within the rental period. Moreover, the act of returning a car initiates the fluent *return(…)*, that is the property specified within the commitment.

$$\begin{aligned}&create(withdraw(C,\ CarPlate),\\ &\qquad \mathsf{C}(C,\ A,\ property(e(T_1, T_2),\ return(CarPlate))),\\ &\qquad T) \leftarrow\\ &\qquad\quad holds\_at(contract(C,\ A,\ CarPlate,\ Start,\ End), T),\\ &\qquad\quad T \geq Start,\ T < End,\\ &\qquad\quad T_1 = T + 1,\ T_2 = End.\\ &initiates(return(CarPlate), return(CarPlate), \_).\end{aligned} \tag{ex3}$$

Also sentence (S2) can be represented by a base-level commitment. However, in this case we need to model more information. First of all, we need to represent the categories of the cars that can be rented at the agency, with the price. For example, we could say that the agency rents cars of category *medium* or category *small*, together with the daily rental price for each category (price in$, Axiom (ex4)). Secondly, we should represent some cars available at the rental agency, together with the category they belongs to. In Axiom (ex5) we represented three different cars, identified by the plate (the first parameter).

$$\begin{aligned}category(small, 35).\\ category(medium, 50).\end{aligned} \tag{ex4}$$

$$\begin{aligned}car(ab123cd, medium).\\ car(ef555ef, medium).\\ car(ga999tt, small).\end{aligned} \tag{ex5}$$

And then we can represent the obligation to pay by means of a base-level commitment, about a temporal-related property: the payment must be made no later than the first rental day (Axiom (ex6)). The commitment is created as the consequence of an event of signing. Note that this commitment is created independently that the customer will ever withdraw the car. Instead, the commitment is established only when both the parties have signed

$$\begin{aligned}&create(sign(X,\ contract(C,\ A,\ CarPlate,\ Start,\ End)),\\ &\qquad \mathsf{C}(C,\ A,\ property(e(T_1,\ T_2),\ pay(Amount))),\\ &\qquad T) \leftarrow\\ &\qquad\quad all\_signatures\_done(sign(X,\ contract(C,\ A,\ CarPlate,\ Start,\ End)),\ T),\\ &\qquad\quad car(CarPlate,\ Type),\\ &\qquad\quad category(Type,\ DailyFee),\\ &\qquad\quad RentTime = End - Start,\\ &\qquad\quad Amount = DailyFee * RentTime,\\ &\qquad\quad T_1 = T + 1,\ T_2 = Start.\end{aligned} \tag{ex6}$$

The commitment to pay is satisfied if any agent will pay (*payment* event) the same amount specified by the commitment, as in Axiom (ex7):

$$initiates(payment(\_, X), pay(X), \_). \tag{ex7}$$

Note that Axiom (ex7) defines the conditions for which the property *pay(…)* is initiated (fulfilled). Our commitment axiomatization then takes care of discharging the commitment (the commitment becomes satisfied).

Sentences (S3) is about maintaining a certain property, i.e. to guarantee that the car works. We can represent this condition using a base-level commitment, but this time the property is temporally related universally with respect to the time interval. Again, the commitment is established both the parties have signed, as stated in Axiom (ex8). Moreover, we must state that the cars initially work properly: we will use the fluent *works(CarPlate)* to state this initial condition (Axiom (ex9)).

$$\begin{aligned}
&create(sign(X,\ contract(A,\ CarPlate,\ Start,\ End)),\\
&\quad \mathsf{C}(C,\ A,\ property(u(T_1, T_2),\ works(CarPlate))),\\
&\quad T) \leftarrow\\
&\quad all\_signatures\_done(sign(X,\ contract(C,\ A,\ CarPlate,\ Start,\ End)),\ T),\\
&\quad T_1 = Start,\ T_2 = End.
\end{aligned} \tag{ex8}$$

$$\begin{aligned}
&initially(works(ab123cd)).\\
&initially(works(ef555ef)).\\
&initially(works(ga999tt)).
\end{aligned} \tag{ex9}$$

The notification of a car breakdown terminates the property *works(…)*, as stated in Axiom (ex10):

$$terminates(notify(C,\ A,\ broken(CarPlate)),\ works(CarPlate),\ \_). \tag{ex10}$$

In the case of breakdown, a first compensation (a reimbursement of 200$) is established:

$$\begin{aligned}
&create(notify(C,\ A,\ broken(CarPlate)),\\
&\quad \mathsf{C}(A,\ C,\ property(e(T_1,\ T_1),\ reimbursement(A,\ C,\ 200.0))),\\
&\quad T) \leftarrow\\
&\quad\ holds\_at(contract(C,\ A,\ CarPlate,\ Start,\ End),\ T),\\
&\quad\ active(c(C,\ A,\ property(u(Tc1,\ Tc2), works(CarPlate))),\ T),\\
&\quad\ terminates(notify(C,\ A,\ broken(CarPlate)),\ works(CarPlate),\ T),\\
&\quad\ T >= Start,\ T <= End,\\
&\quad\ T_1 = T + 1.
\end{aligned} \tag{ex11}$$

$$initiates(payment(A,\ C,\ X),\ reimbursement(A,\ C,\ X),\ \_). \tag{ex12}$$

Axiom (ex11) is a little bit more complex: the rental agency wants to be really sure that it owes a reimbursement to the customer. In fact, the following conditions are checked before creating a commitment for the reimbursement: (i) the contract holds; (ii) the commitment for maintaining the car working properly is active; (iii) the event *notify(…)* terminates exactly the property that the commitment of condition (ii) is about; and finally (iv) the breakdown occurs within the rental interval. Axiom (ex12) simply asserts that the act of payment X\$ makes the property *reimbursement(…)* to hold.

Sentence (S3)-a amount to establish two commitments, if the conditions are met: a commitment to retire the broken car (Axioms (ex13) and (ex14)) and a commitment to reimburse the client for the remaining days (Axiom (ex15)).

$$\begin{aligned}&create(notify(C,\ A,\ broken(CarPlate)),\\&\quad \mathsf{C}(A,\ C,\ property(e(T_1,\ T_1),\ retire(CarPlate))),\\&\quad T) \leftarrow\\&\qquad holds\_at(contract(C,\ A,\ CarPlate,\ Start,\ End),\ T),\\&\qquad T \geq End - 1,\ T \leq End,\quad T_1 = T + 1.\end{aligned} \tag{ex13}$$

$$initiates(retire(CarPlate), retire(CarPlate), \_). \tag{ex14}$$

$$\begin{aligned}&create(notify(C,\ A,\ broken(CarPlate)),\\&\quad \mathsf{C}(A,\ C,\ property(e(T_1,\ T_1),\ reimbursement(A,\ C,\ Amount))),\\&\quad T) \leftarrow\\&\qquad holds\_at(contract(C,\ A,\ CarPlate,\ Start,\ End),\ T),\\&\qquad T \geq End - 1,\ T \leq End,\ T_1 = End + 1,\\&\qquad RentDays = End - T + 1,\\&\qquad car(CarPlate,\ Type),\ category(Type,\ Fee),\\&\qquad Amount = Fee * RentDays.\end{aligned} \tag{ex15}$$

Sentence (S3)-b instead establishes only one commitment, to substitute the car within two days from the user notification. Sentence (S4) however adds a further constraint: if the substitution happens later than the two days specified in (S3)-b, the agency will compensate with a further reimbursement. Axioms (ex16) and (ex17) capture the obligation of substituting the car by means of a commitment, while Axiom (ex18) focuses on the additional reimbursement.

$$\begin{aligned}&create(notify(C,\ A,\ broken(CarPlate)),\\&\quad c(A,\ C,\ property(e(T_1,\ T_2), sub(CarPlate))),\\&\quad T) \leftarrow\\&\qquad holds\_at(contract(C,\ A,\ CarPlate,\ Start,\ End),\ T),\\&\qquad T \geq Start,\ T \leq End - 2,\\&\qquad T_1 = T + 1,\ T_2 = T + 2.\end{aligned} \tag{ex16}$$

$$initiates(change(Broken,\ Newer),\ sub(Broken), \_). \tag{ex17}$$

$$\begin{aligned}&create(change(Broken,\ Newer),\\&\quad c(A,\ C,\ property(e(T_1,\ T_2), reimbursement(A,\ C,\ Amount))),\\&\quad T) \leftarrow\\&\qquad holds\_at(contract(C,\ A,\ Broken,\ Start,\ End),\ T),\\&\qquad violated(c(A,\ C,\ property(e(T_3, T_4),\ sub(Broken))),\ T),\\&\qquad Delay = T - T_4,\ Amount = 20 * Delay,\\&\qquad T_1 = End + 1,\ T_2 = End + 30.\end{aligned} \tag{ex18}$$

Table 2 A possible observed interaction

Event	Time
sign(ag, contract(john, ag, ab123cd, 10.0, 20.0))	3
sign(john, contract(john, ag, ab123cd, 10.0, 20.0))	5
payment(marge, 500.0)	9
withdraw(john, ab123cd)	10
notify(john, ag, broken(ab123cd))	13
payment(ag, john, 200.0)	14
change(ab123cd,ga999tt)	18
return(ga999tt)	20
payment(ag, john, 60.0)	30

Finally, although the example does not hint anything about, it appears reasonable to consider the following: if a car breaks down and it is changed with another one, then the commitment to return back the former broken car should be released. Moreover, a new commitment to bring back the latter car should be established. Axioms (ex19) and (ex20) capture such intuition.

$$\begin{aligned}&release(change(Broken,\ Newer),\\&\quad c(C,\ A,\ property(e(T_1,\ T_2),\ return(Broken))),\\&\quad \_).\end{aligned} \tag{ex19}$$

$$\begin{aligned}&create(change(Broken,\ Newer),\\&\quad c(C,\ A,\ property(e(T_1,\ T_2), return(Newer))),\\&\quad T) \leftarrow\\&\quad\ \ holds\_at(contract(C,\ A,\ Broken,\ Start,\ End),\ T),\\&\quad\ \ active(c(C,\ A,\ property(e(Tb_1,\ End),\ return(Broken))),\ T),\\&\quad T < End,\ T_1 = T+1,\ T_2 = End.\end{aligned} \tag{ex20}$$

4.2 Monitoring the car rental example

We show in Fig. 4 a screenshot of the output of an observed interaction between John, a customer, and a car rental agency. The output has been computed using the Axioms defined in 4, while the observed interaction is summarized in Table 2. In particular, the agency and John sign (at time 3 and 5, respectively) a contract for renting a medium size car from time 10 to time 20, hence generating the commitment to pay the fee of 500$ (Axiom (ex6)): his secretary Marge pays for him the rental fee at time 9, hence satisfying the commitment. Contextually with the sign of the contract, the agency becomes committed to guarantee that the rented car will work properly for the duration of the contract (Axiom (ex8)), i.e. from time 6 to time 20 (end of the rent). At time 10 John withdraws the car, becoming committed to bringing it back within time 20 (Axiom (ex3)).

Unfortunately, at time 13 the car breaks, and John immediately informs the agency. The agency becomes committed to paying a fixed reimbursement (Axiom (ex11)) of 200$, and becomes also committed to substituting the car within two days (Axiom (ex16)). The agency pays the fixed reimbursement at time 14. However, it does not manage to substitute the car in

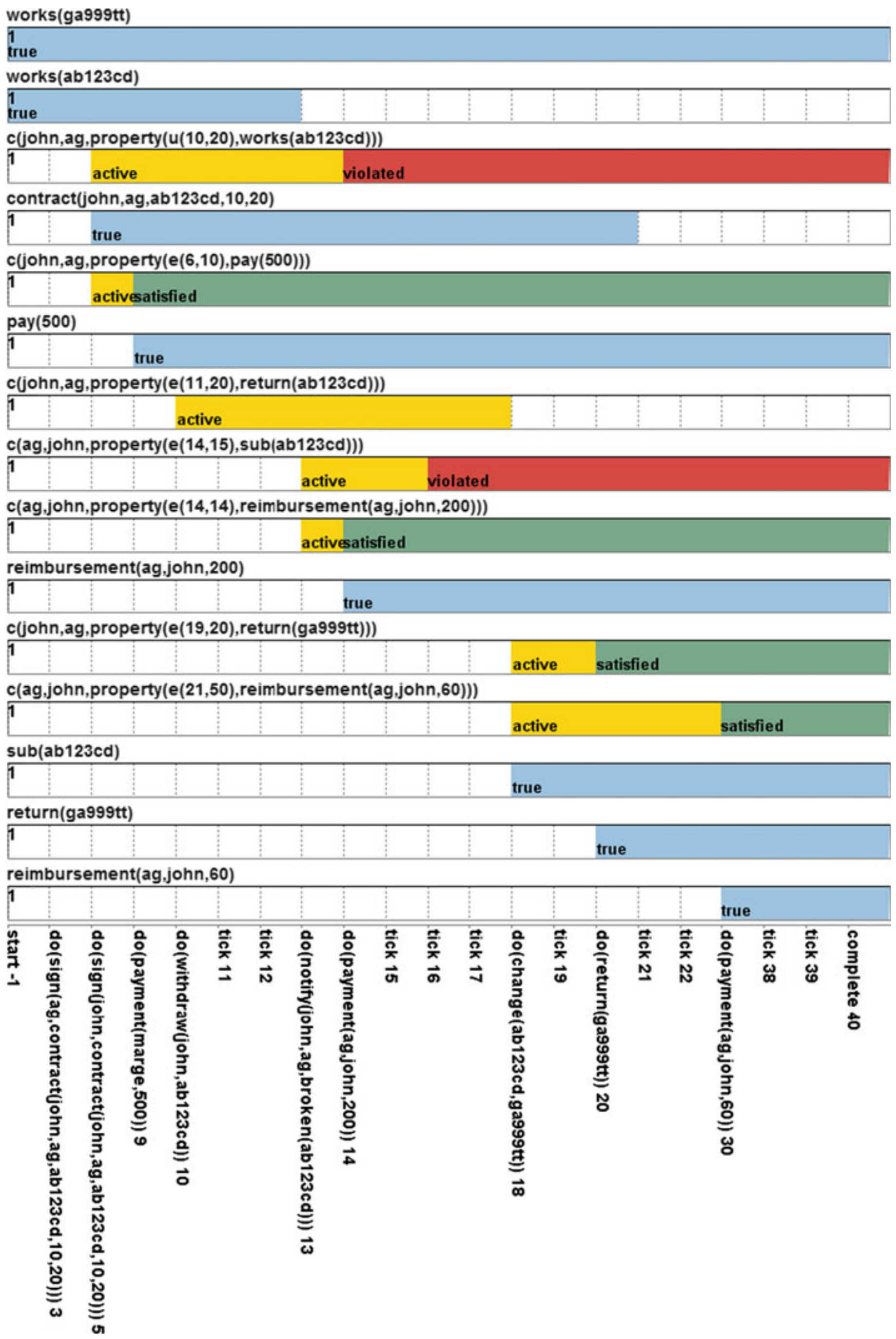

Fig. 4 Outcome shown by j$\mathcal{REC}$ (Color figure online)

two days, and John has to wait till time 18 for having a working car. Hence the commitment about the substitution gets violated (as well as the commitment to maintain the car working). In turn, this generates a further commitment for the agency to pay a variable reimbursement for the breaking of the car, that depends on how many days later the car has been substituted (Axiom (ex18)). In our case, the variable reimbursement amount to 60$. Moreover, since the car has been changed, the previous commitment to bring back the first car is released, and a new one to return the newer car is created.

Finally, John returns the car back to the agency at time 20, thus fulfilling the commitment to bring back the (new) car in time, and few days later the agency pays him also the variable reimbursement due to the car rupture, at time 30. So also the commitment about the variable reimbursement is satisfied.

Summing up, only two commitments get violated (the reader can see them in red colour in Fig. 4): the commitment to maintain the car working, and the commitment to substitute it in two days.

4.3 Experiments

We run experiments, based on the car rental illustration, to determine whether $\mathcal{REC}$ could actually be used for monitoring multi-agent systems at run-time. We did not resort to existing benchmarks for commitment monitoring tools because to the best of our knowledge there exist no such benchmarks. At least we could not find any benchmark or experimental evaluation among the works we surveyed (see Sect. 5). The only work we found with an empirical evaluation of a commitment modelling framework was done by El-Menshawy et al. [29,31], but such a framework is not concerned with online monitoring.

In particular, El-Menshawy et al. show that off-the-shelf symbolic model checkers can be used to verify properties of the NetBill protocol (e.g., reachability, safety, liveness, and fulfillment/violation of commitment) in about 18 minutes when the protocol involves up to 20 agents. It is well known that state-of-the-art model checkers can manage huge state spaces. However, it is also true that the number of states in the model generally grows very quickly. Thus 1 second is enough to verify said properties, against a commitment-based specification of the NetBill protocol involving 10 agents, producing a model with 11545 states, while to verify the same properties against a specification with twice as many agents requires about 18 minutes, and a model with over 400 million states. No results are shown for more than 20 agents, but execution time and number of reachable states increase exponentially when the number of agents increases. Moreover, the verified properties do not include metric time.

While the model checking approach is acceptable for static verification tasks, where protocols are defined once and for all and the roles of participants are but a few, and known a priori, a different level of scalability is needed for monitoring open multi-agent systems at run-time. Here we need to cope with a potentially unbound number of participants and actions, performed at specific points in time, and the time granularity may be such that the number of possible time points we need to represent is very large. In the same context, several types of contracts may coexist, some relevant to some actions performed by some agents, others irrelevant. The description of the world, accounting for all relevant information, may also be very large. For example, if we model a rental car scenario, we may have to deal with a very high number of available rental cars, potential customers, and types of rental contracts.

Thus we decided to test $\mathcal{REC}$'s performance along three different dimensions:

1. *number of events* occurring at runtime,
2. *number of contract clauses*, as a measure of the number of commitments/commitment state transitions per transaction, and

Table 3 $\mathcal{REC}$ performance in the car rental scenario for 6,000 events

N. cars	Total processing time (s)	**lept**(ms)
200	277	294
1,000	367	310
5,000	920	653
20,000	3,467	2,488

3. *number of cars* available to the rental agency, as a measure of the size of the description of the world at time zero.

Our performance metric of choice is the *longest* time needed to process an event (**lept**, for *l*ongest *e*vent *p*rocessing *t*ime), i.e., the worst case. This measure is arguably better than, e.g., the mean event processing time, because it determines the *flow* of events that can be successfully handled by the monitoring tool. It turns out that, under the assumption of total recall, the **lept**grows as the number of events grows, i.e., as time goes by. That is so, because each new event is added to the knowledge base, and may generate new fluents, which add to the complexity of reasoning. Hopefully, suitable garbage collection techniques, implementing a concept of limited recall or "statute of limitations," may be able to restrict the scope of relevant events for run-time monitoring to a given time window of limited size. We did not investigate this direction yet, which is subject for future work. Thus in a sense the results we show represent a worst-case scenario's upper bound.

We performed experiments on a MacBook Pro 2 GHz Intel Core i7 with 8 GB RAM, running Mac OS X 10.6 and YAP Prolog 6.2.2 (i386-darwin10.8.0).[9] The tables below show the **lept**, with respect to the number of events, number of contract types, and number of cars.[10]

Number of events. Our first experiment considers a single contract type, corresponding to the car rental scenario. The rental agency serves up to 200 customers. For each customer, a trace of events is generated. In total, in addition to the commitment model and the world model, the scenario consists of 15 contract clauses and 6000 events. The total processing time is below 5 minutes. The **lept** is 294 ms.

Number of contract clauses. We repeated the same scenario, by considering more contract types (i.e., slightly modifications of the contract type used for the first experiment). In particular we run the experiments by extending the number of contract clauses by a factor of 5, 10, and 50, and by keeping the same number of customers (200) and events (6,000). We registered no statistically significant variation of the total processing time, which stays in all cases below 5 minutes, and of the **lept**, which ranges in all cases between 281 and 304 ms. Thus in this setting the runtime depends on the number of events, but not on the size of the contract specification.

Number of cars. Finally, we repeated the first scenario, by keeping the same contract specifications, while increasing the number of cars from 200 to 20,000. The results are summarized in Table 3.

9 http://www.dcc.fc.up.pt/~vsc/Yap/.

10 The dataset we used is available from http://ai.unibo.it/projects/comMon.

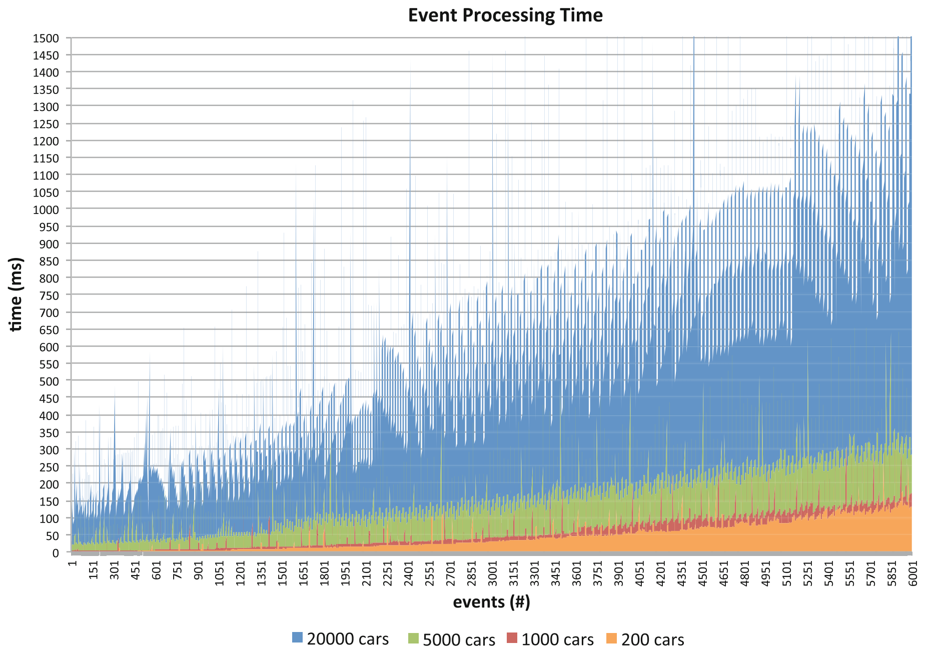

Fig. 5 Processing time of individual events with respect of the total number of events (Color figure online)

In this setting, we observe a sublinear increase in the total processing time and **lept** with respect to the increase of the size of the world model at time zero.

Figure 5 shows how the event processing time grows as the total number of events increases (x axis) for different sizes of the description of the work at time zero.

These experiments are by no means intended to be exhaustive. However they demonstrate that $\mathcal{REC}$ can be used for monitoring multi-agent systems at run-time, not only in theory, but also in practice.

5 Related work

The work presented here is mainly related to multiagent systems research on commitments.

The commitment operations presented in this work are largely inspired by the ones introduced by Singh [58]. We extend the commitment language by allowing properties that are related to temporal aspects and, more in general, to data. Moreover, with respect to [58], we relax some constraints regarding which agent can perform which operation.

The need for extending the commitment language with temporal aspects has been first recognized by Fornara and Colombetti [33], whose formalization includes time intervals, temporal quantifications and timeouts in the commitment language, but it does not formally propose a semantics, nor a reasoning procedure for reasoning about time. A more focussed discussion of temporal aspects attached to commitments can be found in [49,50], where Mallya et al. present the concepts of existentially and universally temporal-related properties, together with a formal declarative semantics in CTL^*. Besides some issues on the semantics, thoroughly discussed in [62], we greatly differ form those works in two fundamental aspects.

First, we give the commitment operations and the temporal constraints a unified semantics, based on the $\mathcal{EC}$, while [49,50] do not specify a particular semantics for the commitment, and propose CTL^* for the temporal propositions. Second, we provide and implement a proof-procedure, which we exploit to perform on-line monitoring based on the commitment models, whereas the cited approach does not address on-line reasoning, nor does it propose concrete automated procedures for temporal reasoning. Again with a CTL-based commitment language, Venkatraman and Singh [65] present an approach to run-time compliance verification. However, the representation of time is a simple ordering of events, i.e., a transition time, not a metric time, and as such it cannot be used to formalize deadlines because the timestamps of events are not absolute, but instead depend on the timestamps assigned to previous events.

Baldoni et al. [6,7] distinguish between the constitutive aspects of a protocol, and the regulative ones. The authors argue that the constitutive aspects are perfectly covered by the commitment paradigm, while the regulative aspects are not. Hence, they propose the new language 2CL that can be used to augment the commitment model with further information, such as the fact that a commitment can start to hold only after another one holds. Regulative aspects then are resolved to be expressed in terms of constraints about the existence of commitments, and in terms of constraints on the temporal ordering of the commitments. In a sense, also this work extends the commitment paradigm towards the temporal dimension, but in a orthogonal direction with respect to our framework. While Baldoni et al. focus on temporal properties about the commitments, we focus on temporal properties inside the commitments. The semantics of the temporal constraints is given in term of LTL, while we use $\mathcal{EC}$. In [5] a tool for supporting 2CL-based protocol specifications is presented: it aims to support static, a-priori analysis of the possible/allowed protocols interactions and outcomes envisaged by the 2CL specification.

Additional semantical considerations are proposed by Singh [59], who distinguishes between dialectical and practical commitments, and defines a number of reasoning postulates that reflect common reasoning patterns that apply to commitments. Some of these postulates are deemed natural for typical modelling situations, others less so. Our framework could be confronted with such postulates, although such an analysis is outside of the scope of this work. We conjecture that most of the "natural" postulates (e.g., B_1 to B_6, and B_8) apply–at least as far as simple properties– whereas some do not. Among the latter, B_7 (strong consistency) and B_9 (chain) do not seem to apply. It would be interesting to investigate how Singh's postulates could be extended to accommodate metric time, which does not appear to be a trivial task. Lorini [46] instead addresses the dynamic dimension of commitments, but only for a restricted commitment model.

The idea of using the $\mathcal{EC}$ as a formal semantics for the commitment paradigm is not new: Yolum and Singh proposed it in [66,67]. However, our formalization of commitments operations in terms of $\mathcal{EC}$ axioms is different for several reasons. We do not represent the commitments by fluents, but rather we represent the *state* of commitments by fluents, thus reifying such information. We provide extensions to the commitment model with respect to the treatment of temporal properties, which Yolum and Singh do not deal with. Moreover, our framework includes a reactive proof-procedure that enables run-time monitoring.

To the best of our knowledge, at the time of writing ours is the only framework comprising both a declarative semantics and a sound operational counterpart that can effectively be used at runtime. Fornara and Colombetti [35] use a well-known automated reasoning tool (the Discrete Event Calculus, [54]) to formalize a repertoire of abstract concepts like commitment, institutional power, role and norm, useful for the specification of artificial institutions. Monitoring is mentioned as a possible application but it is not explored. In fact, reasoning is deductive, and the irrevocability issue is not even raised. In more recent work [36], Fornara and Colombetti propose a method for commitment monitoring based on a loop, where at every step, all assertions that can be inferred are *deduced*, ontological information is updated (by closure), and the time of simulation is incremented by 1 unit to prepare the next iteration. This method should produce significant performance improvements with respect to the previous $\mathcal{EC}$-based solution, however its reasoning efficiency has not been evaluated yet.

Other relevant work addressing artificial institutions was carried out inside the ALFEBiiTE project[11] by Artikis et al. [4]. Though not directly dealing with social commitments, but rather with executable specifications of agent societies and on related normative aspects, that body of work is relevant because it also uses the $\mathcal{EC}$, alongside the C+ language. However, it is worthwhile noticing that in spite of the plethora of very expressive $\mathcal{EC}$ dialects already been proposed in the literature [51], there has been so far a lack of logic-based *and* reactive $\mathcal{EC}$ frameworks. The survey of $\mathcal{EC}$ applications presented by Miller and Shanahan [51] only mentions deductive and abductive reasoning (for planning). The $\mathcal{EC}$ variations they illustrate are surely expressive, but do not lend themselves well to runtime monitoring.

In more recent years, ad-hoc implementations, lacking an underlying formal basis, have been developed in order to target the run-time verification and monitoring of $\mathcal{EC}$-based specifications. For example, Chittaro and Montanari's Cached Event Calculus ($\mathcal{CEC}$) [18,19] relies on extra-logical features (assert and retract predicates in particular) to efficiently implement a reactive form of the $\mathcal{EC}$. We use some of the $\mathcal{CEC}$ concepts in the $\mathcal{REC}$ implementation discussed in Sect. 4.3. In [47], Mahbub and Spanoudakis present a framework for monitoring the compliance of a BPEL-based service composition with assumptions expressed by the user, or with behavioral properties automatically extracted from the composition process. The $\mathcal{EC}$ is exploited to monitor the actual behavior of interacting services and to report different kinds of violations. The approach is further investigated in [48], where the authors use an extension of WS-Agreement [3] to specify requirements. The monitoring framework relies on an ad-hoc event processing algorithm implemented in JAVA, which fetches the occurring events and updates the state of the affected fluents. Another JAVA-based ad-hoc implementation of the $\mathcal{EC}$, used to dynamically track the normative state of contracts, has been proposed by Farrel et al. in [32]. The common drawback of all these approaches is that they are not tailored to an underlying declarative semantics. That makes it impossible, or at least very hard, to assess important formal properties such as soundness, completeness, termination and irrevocability.

In [13] we discuss at length the need for reactive $\mathcal{EC}$ axiomatizations, as opposed to classical deductive ones, in the context of commitment monitoring. More about $\mathcal{REC}$ can be found in [11,15].

The run-time verification of event-based systems using logic-based approaches has been tackled not only by means of the $\mathcal{EC}$, but also using temporal logics, propositional linear temporal logic (LTL) in particular. Beside expressiveness issues (propositional LTL does not suffice to formalize the extended commitment machine presented in this paper) the construction of suitable monitors starting from an LTL formula poses a twofold challenge. First, the standard semantics of LTL must be modified to reflect that, during the execution, the reasoning process cannot always provide a definitive answer, but must be open to the acquisition of new knowledge, i.e., of new event occurrences. Let us for example consider a partial trace, which does not contain the execution of activity pay. A liveness property such as $\Diamond pay$ should not lead to evaluate the instance as noncompliant: there is still the possibility to meet

[11] ALFEBiiTE was a European project, whose goals included explaining the creation of trust, understanding the links between actions, contexts, social commitments and legal consequences, and how interactions between individuals and/or organizations are empowered by third party institutions (source: http://www.iis.ee.ic.ac.uk/~alfebiite/).

the property, by eventually executing the payment. To overcome this issue, Bauer et al. propose to switch to a three-valued semantics for the logic [8]; under a three-valued semantics, a run-time verifier would evaluate $\Diamond pay$ as unknown. Another possibility is to revise the LTL semantics so as to account for finite (partial) execution traces. Because partial ongoing executions are finite, Giannakopoulou and Havelund introduce a finite-trace semantics for LTL [41,37]. Their semantics would state that $\Diamond pay$ is violated, but in their setting the concept of violation is limited to the portion of execution observed so far. For a survey of the use of LTL for run-time verification, the interested reader can refer to [9]. The second critical aspect is related to the generation of a suitable run-time verifier. Automata-based techniques are usually employed, but they cannot take advantage from the standard transformations onto Büchi automata, because of the non standard three-valued or finite trace semantics. The automata generation algorithms require ad-hoc modifications, making it again difficult to prove the formal properties of the resulting approach. Among others, Spoletini and Verdicchio use automata to build a monitoring module aimed to support an agent during its life in a multi-agent system [61]. The method is not reactive (a new automaton is built every time an event deemed relevant is "sniffed" from the environment). We do not know how efficient such a method would be, if implemented, however we suspect scalability may be an issue.

The need for on-line monitoring facilities in multiagent systems was a major motivation behind the CONTRACT effort.[12] However, the on-line monitoring tool proposed in [52] implements a procedural monitoring control loop whose link with the semantics of the contract/commitment specification language is not easily proven, and therefore, to the best of our knowledge, no formal property has been proved so far in that respect. Our approach is based on the standpoint that soundness, completeness, and irrevocability, are fundamental conditions for the successful dissemination and acceptance of commitment technologies.

In the more general context of commitment frameworks, not addressing monitoring, many other authors considered the use of commitments to represent, model and verify protocols. Numerous commitment life cycles have been proposed, such as those cited in Sect. 2, including violation, cancelation, and other operations. Among them, Bentahar et al. [10] and El-Menshawy et al. propose several methods for commitment-based protocol verification using model checking [29–31]. In Sect. 4.3 we argued against the suitability of model checking approaches for run-time monitoring. In [53] we show evidence that in some domains, characterized by openness, top-down approaches can compete with model checking also for static verification. One thing we have in common with El-Menshawy et al.'s approach is instead the stance we take that clear, practical and verifiable semantics should be defined both for commitments, and for the operations used to manipulated commitments [29]. In particular, in this article we define the semantics for commitment operations in terms of $\mathcal{EC}$ axioms, and the semantics of commitments in terms of fluents with an underlying logic programming framework extended with the $\mathcal{EC}$ theory and CLP[13] constraints to represent metric time. El-Menshawy et al. instead use CTL^{*sc}, i.e., CTL^* extended with modalities for social commitments.

Another approach related to our framework is presented by Desai et al. in a number of works that apply commitment models to business processes [23,24]. There, the authors use the CCALC language [38] for modelling dynamics of commitments, in relation with the events occurring in the world (actions executed, in CCALC). CCALC is a very expressive

[12] CONTRACT was an European project, with a mixed academic/industrial consortium. Among its primary objectives was the synthesis of on-line monitoring systems that observe a trace of Web service behaviour generated at runtime and infer whether the service satisfies a contractual specification (source: Web, http://www.ist-contract.org/).

[13] CLP stands for Constraint Logic Programming [42].

system for representing commonsense knowledge about action and change, which also allows data variables.

However, Desai et al.'s aim is quite different from ours. Like other authors (see above), they are concerned with static verification of protocol properties, which we do not address. Conversely, there is no evidence that CCALC could be effectively used for run-time commitment monitoring.

Our commitment specification language is not exempt from shortcomings. Let us discuss them here as well. Because of the requirement that a commitment's property should be brought about *strictly after* the commitment is created (see Sect. 3, e.g., Axioms (2.1)–(2.3)), and not at the same time, we cannot express commitments where antecedent and consequent overlap ("*I will stay quiet if you also stay quiet!*"). The semantics of such commitments is also not straightforward and should be investigated. Similarly, we cannot model a commitment about a property that has to be brought about "*as of now*." For example, we cannot say "*you have time until tomorrow*," but we should instead choose a suitable time granularity and say "*you have time as of the next second/minute/hour, until tomorrow*." This is so, because of the $\mathcal{EC}$'s representation of time intervals: open on the left and closed on the right.

Another limitation of our approach is that we assume events to occur one at a time, and the time model underlying the framework is discrete. Dealing with multiple events at a time causes non-trivial problems (see Chopra and Singh, [21]). The difficulties here are not of a technical nature, but rather stem out directly from the semantics of commitments. Let us suppose, for example, that two events happens at the same time: one of them one cancels an active commitment, while the other satisfies the same active commitment. What would the "right" commitment state be, cancelled or satisfied? Once this question is answered, to extend our framework towards one or the other behavior can be easily accomplished. Desai et al. solve this problem by defining a overriding relationship between concurrent commitment operations [24]. Should two events happen at the same time (and in distributed environments, this is likely to happen), the overriding relationship specifies which commitment operation dominates. We are not fully convinced that such solution is the best in every application domain. Thus, in the current implementation we preferred to leave this problem open, and to add a constraint that prohibits concurrent events occurrences. It is worthwhile noticing, however, that the overriding relationship may be accommodated, by adding a further condition in all the axioms. Such a condition would check for concurrent events and, in case, would check the overriding relationship to establish which axioms to enable and which ones to inhibit. We consider this further extension out of the scope of this work, and leave it to future works.

Finally, since we rely on CLP libraries to express temporal expressions, we cannot easily express periodic commitments, such as "*gates close every day at 8pm and open every morning at 7am*," and other expressions that go beyond interval-based temporal reasoning. These are all open issues, to which we could not find an answer in the relevant literature. We believe that they deserve further investigation as they would improve the usability of a commitment language.

To conclude this section, fundamental considerations may be made about our choice of a first order logic framework with variable and fluents, as opposed to more limited, propositional languages and logics, e.g., the variations of CTL and LTL that we have mentioned in relation with other influential work on social commitments. The trade off is as usual between computational complexity and expressiveness. Thus with our logic of choice, static verification of protocol properties is possible but of limited applicability, because first order logic is undecidable. In particular, while some properties, under certain conditions, could effectively be verified using a first order logic programming framework such as SCIFF [1,53], it would be very hard if not utterly impossible to prove reachability, safety, and liveness properties typically studied using model checking approaches [29]. For the same reason, model checking seems to loose its effectiveness when a metric time representation is a domain requirement.

On the other hand, when the problem is run-time monitoring, our solution seems to have a potential to outperform other proposed solutions that use model checking using a propositional language. Thus in the commitment monitoring domain the trade off between expressiveness and complexity would be less of a problem, since a very expressive language, such as the one we propose, is also quite efficient in practice. This conjecture of ours is based on practical experience and empirical evidence, but clearly we can not make a definite statement, given the lack of practical experience and empirical evidence on the other side.

6 Conclusions

Given the wide popularity of the social commitments paradigm in multiagent research, the adoption of such a paradigm in practical agent applications is surprisingly limited. Our working hypothesis was that existing commitment frameworks lack appeal because of language limitations and/or because of technological shortcomings. In particular, commitments should be very interesting for concrete applications because they lend themselves well to external inspection and monitoring [57], but in concrete terms, effective monitoring tools are still missing, and simple though ubiquitous specifications such as those involving a deadline may represent a point where a gap is created between models and tools exploiting them.

In this article we proposed a solution to these issues. On the modelling side, we proposed a commitment modelling language that enables metric temporal reasoning, and allow the domain modeller to freely specify which roles the debtor and creditor agents shall play in the manipulation of commitments. We enabled reasoning with data variables and domains, thus opening up modelling possibilities that were precluded by previous formalizations grounded on propositional logics. We defined new commitment operations, and a new commitment life-cycle. Building on state of the art work, mainly Yolum and Singh's proposal, we adopted the $\mathcal{EC}$ to define a semantics for commitment operations, and a logic programming framework extended with the $\mathcal{EC}$ theory and CLP constraints to define the semantics of commitments.

On the technological side, we proposed and implemented the $\mathcal{REC}$: a reactive version of the $\mathcal{EC}$, designed to be used on-line, with a dynamically growing narrative of events. Empirical evidence suggests that $\mathcal{REC}$ can effectively be used to implement run-time monitoring services, exhibiting an unprecedented level of scalability.

We are currently extending the framework, to address the aspects as we discussed in Sect. 5 and throughout the paper. We hope that $\mathcal{REC}$ will be considered by researchers and practitioners interested in commitment monitoring as a simple and expressive enough modelling language, with and an effective operational tool to build upon. We and other authors are already using $\mathcal{REC}$ for detecting conflicts in commitments [40], for commitment exception diagnosis [43,44], and for monitoring commitment evolutions during simulated multiagent system executions [16].

Acknowledgments This work has been partially funded by the EU Project "Artifact-Centric Service Interoperation" (ACSI, FP7-ICT-257593). We wish to thank the anonymous referees for their insightful reviews.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic, 9*(4), 1–43.

2. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2005). The SCIFF abductive proof-procedure. In AI*IA 2005: Advances in artificial intelligence 9th congress of the Italian Association for Artificial Intelligence, Milan, Italy, September 21–32, 2005. Proceedings, Lecture notes in computer science (Vol. 3673, pp. 135–147). Berlin: Springer.
3. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., & Xu, M. (2007). Web services agreement specification (ws-agreement). Tech. rep., Grid Resource Allocation Agreement Protocol (GRAAP) WG.
4. Artikis, A., Sergot, M. J., & Pitt, J. V. (2009) Specifying norm-governed computational societies. *ACM Transactions on Computational Logic, 10*(1), 1–42.
5. Baldoni, M., Baroglio, C., Capuzzimati, F., Marengo, E., & Patti, V. (2012). A generalized commitment machine for 2cl protocols and its implementation. In M. Baldoni, L. Dennis, V. Mascardi, & W. Vasconcelos (Eds.), *Proceedings of the 10th international workshop on declarative agent languages and technologies (DALT 2012)*. Lecture notes in computer science. Berlin: Springer.
6. Baldoni, M., Baroglio, C., & Marengo, E. (2010). Behavior-oriented commitment-based protocols. In Proceedings of 19th European conference on artificial intelligence, ECAI 2010.
7. Baldoni, M., Baroglio, C., Marengo, E., & Patti, V. (2012). Constitutive and regulative specifications of commitment protocols: A decoupled approach. *ACM Transactions on Intelligent Systems and Technology*, to appear.
8. Bauer, A., Leucker, M., & Schallhart, C. (2006). Monitoring of real-time properties. In Proceedings of the 26th international conference on foundations of software technology and theoretical computer science (FSTTCS). Lecture notes in computer science (Vol. 4337, pp. 260–272). Berlin: Springer.
9. Bauer, A., Leucker, M., & Schallhart, C. (2010). Comparing LTL semantics for runtime verification. *Logic and Computation, 20*(3), 651–674.
10. Bentahar, J., Meyer, J. J. C., & Wan, W. (2009). Model checking communicative agent-based systems. *Knowledge-Based Systems, 22*(3), 142–159.
11. Bragaglia, S., Chesani, F., Mello, P., Montali, M., & Torroni, P. (2012). Reactive event calculus for monitoring global computing applications. In Essays in honour of Marek Sergot: Computational logic for normative systems. Lecture notes in computer science. Berlin: Springer.
12. Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. In V. R. Lesser & L. Gasser (Eds.), *Proceedings of the first international conference on multi agent systems* (pp. 41–48). Cambridge: The MIT Press.
13. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). Commitment tracking via the reactive event calculus. In *Proceedings of the 21st international joint conference on artificial intelligence* (pp. 91–96). San Francisco, CA.
14. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). A REC-based commitment tracking tool. System demonstration. In Proceedings of the 10th AI*IA/TABOO Italian Joint Workshop "From Objects to Agents" (WOA). http://cmt.math.unipr.it/woa09/papers/Chesani_Demo.pdf.
15. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2010). A logic-based, reactive calculus of events. *Fundamenta Informaticae, 105*(1–2), 135–161.
16. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2011). Monitoring time-aware commitments within agent-based simulation environments. *Cybernetics and Systems, 42*(7), 546–566.
17. Chesani, F., Montali, M., Mello, P., & Torroni, P. (2010). Monitoring time-aware social commitments with reactive event calculus. In *Proceedings of the 7th international symposium "From Agent Theory to Agent Implementation" (AT2AI-7)*.
18. Chittaro, L., & Montanari, A. (1994). Efficient handling of context-dependency in the cached event calculus. In *Proc. of TIME'94 - International Workshop on Temporal Representation and Reasoning* (pp. 103–112).
19. Chittaro, L., & Montanari, A. (1996). Efficient temporal reasoning in the cached event calculus. *Computational Intelligence, 12*, 359–382.
20. Chittaro, L., & Montanari, A. (2000). Temporal representation and reasoning in artificial intelligence: Issues and approaches. *Annals of Mathematics and Artificial Intelligence, 28*(1–4), 47–106.
21. Chopra, A. K., & Singh, M. P. (2009). Multiagent commitment alignment. In C. Sierra, C. Castelfranchi, K. S. Decker, & J. S. Sichman (Eds.), *Proceedings of the 8th international conference on autonomous agents and multiagent systems* (Vol. 2, pp. 937–944). Richland, SC: IFAAMAS.
22. Colombetti, M. (2000). A commitment-based approach to agent speech acts and conversations. In *Proceedings of workshop on agent languages and communication policies* (pp. 21–29). Barcelona, Spain.
23. Desai, N., Chopra, A. K., Arrott, M., Specht, B., & Singh, M. P. (2007). Engineering foreign exchange processes via commitment protocols. In *IEEE SCC* (pp. 514–521). Los Alamitos, CA: IEEE Computer Society.

24. Desai, N., Chopra, A. K., & Singh, M. P. (2007). Representing and reasoning about commitments in business processes. In *AAAI* (pp. 1328–1333). Menlo Park, CA: AAAI Press.
25. Desai, N., Chopra, A. K., & Singh, M. P. (2009). Amoeba: A methodology for modeling and evolving cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology, 19*(2), 1–45.
26. Desai, N., Narendra, N. C., & Singh, M. P. (2008). Checking correctness of business contracts via commitments. In L. Padgham, D. C. Parkes, J. Müller, & S. Parsons (Eds.), *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems* (Vol. 2, pp. 787–794). Richland, SC: IFAAMAS.
27. Desai, N., & Singh, M. P. (2008). On the enactability of business protocols. In D. Fox & C. P. Gomes (Eds.), *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)* (pp. 1126–1131). Menlo Park, CA: AAAI Press.
28. Dignum, V., & Dignum, F. (2007). Coordinating tasks in agent organizations. Or: Can we ask you to read this paper? In *COIN 2006 workshops, LNAI* (Vol. 4386, pp. 32–47). Berlin: Springer.
29. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2010). Verifiable semantic model for agent interactions using social commitments. In M. Dastani, A. E. Fallah-Seghrouchni, J. Leite, & P. Torroni (Eds.), *Languages, methodologies, and development tools for multi-agent systems, second international workshop, LADS 2009*, Torino, Italy, September 7–9, 2009, Revised Selected Papers. Lecture notes in computer science (Vol. 6039, pp. 128–152). Berlin: Springer.
30. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2011). Symbolic model checking commitment protocols using reduction. In A. Omicini, S. Sardiña, & W. W. Vasconcelos (Eds.), *Declarative agent languages and technologies VIII—8th international workshop, DALT 2010*, Toronto, Canada, May 10, 2010. Revised, Selected and Invited Papers. Lecture notes in computer science (Vol. 6619, pp. 185–203). Berlin: Springer.
31. El-Menshawy, M., Bentahar, J., Qu, H., & Dssouli, R. (2011). On the verification of social commitments and time. In L. Sonenberg, P. Stone, K. Tumer, & P. Yolum (Eds.), *10th international conference on autonomous agents and multiagent systems (AAMAS 2011)* (Vols. 1–3, pp. 483–490), Taipei, Taiwan, May 2–6, 2011.
32. Farrell, A. D. H., Sergot, M. J., Sallé, M., & Bartolini, C. (2005). Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems, 14*(2–3), 99–129.
33. Fornara, N., & Colombetti, M. (2004). A commitment-based approach to agent communication. *Applied Artificial Intelligence, 18*(9–10), 853–866.
34. Fornara, N., & Colombetti, M. (2007). Specifying and enforcing norms in artificial institutions. In G. Boella, L. van der Torre, & H. Verhagen (Eds.), *Normative multi-agent systems, no. 07122 in Dagstuhl Seminar Proceedings* (pp. 1–16). Germany: Schloss Dagstuhl. http://drops.dagstuhl.de/opus/volltexte/2007/909.
35. Fornara, N., & Colombetti, M. (2009). Specifying artificial institutions in the event calculus. In V. Dignum (Ed.), *Handbook of research on multi-agent systems: Semantics and dynamics of organizational models* (pp. 335–366). Hershey, PA: IGI Global.
36. Fornara, N., & Colombetti, M. (2010). Representation and monitoring of commitments and norms using owl. *AI Communications, 23*(4), 341–356.
37. Giannakopoulou, D., & Havelund, K. (2001). Automata-based verification of temporal properties on running programs. In *Proceedings of the 16th IEEE international conference on automated software engineering (ASE 2001)* (pp. 412–416). Providence, RI: IEEE Computer Society.
38. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence, 153*(1–2), 49–104.
39. Governatori, G., & Rotolo, A. (2010). Norm compliance in business process modeling. In M. Dean, J. Hall, A. Rotolo, & S. Tabet (Eds.), *RuleML*. Lecture notes in computer science (Vol. 6403, pp. 194–209). Berlin: Springer.
40. Gunay, A., & Yolum, P. (2012). Detecting conflicts in commitments. In C. Sakama, S. Sardina, W. Vasconcelos, & M. Winikoff (Eds.), *Proceedings of the 9th international workshop on declarative agent languages and technologies (DALT 2011)*. Lecture notes in computer science (Vol. 7169). Berlin: Springer.
41. Havelund, K., & Rosu, G. (2001). Testing linear temporal logic formulae on finite execution traces. Tech. Rep. TR 01-08, RIACS Technical Report.
42. Jaffar, J., & Lassez, J. L. (1987). Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on principles of programming languages, POPL '87* (pp. 111–119). New York: ACM.

43. Kafalı, Ö., & Torroni, P. (2011). Social commitment delegation and monitoring. In J. Leite, P. Torroni, T. Ågotnes, G. Boella, & L. van der Torre (Eds.), *Computational logic in multi-agent systems—12th international workshop, CLIMA XII*, Barcelona, Spain, 17–18 July 2011. Proceedings. Lecture notes in computer science (Vol. 6814, pp. 171–189). Berlin: Springer.
44. Kafali, Ö., & Torroni, P. (2012). Exception diagnosis in multiagent contract executions. *Annals of Mathematics and Artificial Intelligence, 64*(1), 73–107.
45. Kowalski, R. A., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing, 4*(1), 67–95.
46. Lorini, E. (2010). A logical analysis of commitment dynamics. In G. Governatori, & G. Sartor (Eds.), *Deontic logic in computer science, 10th international conference, DEON 2010*, Fiesole, Italy, 7–9 July 2010. Proceedings. Lecture notes in computer science (Vol. 6181, pp. 288–305). Berlin: Springer.
47. Mahbub, K., & Spanoudakis, G. (2005). Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience. In *Proceedings of the 3rd IEEE international conference on web services (ICWS)* (pp. 257–265). Los Alamitos, CA: IEEE Computer Society.
48. Mahbub, K., & Spanoudakis, G. (2007). Monitoring WS-agreements: An event calculus-based approach. In L. Baresi & E.D. Nitto (Eds.), *Test and analysis of web services* (pp. 265–306). Berlin: Springer.
49. Mallya, A. U., & Huhns, M. N. (2003). Commitments among agents. *IEEE Internet Computing, 7*(4), 90–93.
50. Mallya, A. U., Yolum, P., & Singh, M. P. (2004). Resolving commitments among autonomous agents. In *International workshop on agent communication languages and conversation policies*. Lecture notes in computer science (Vol. 2922, pp. 166–182). Berlin: Springer.
51. Miller, R., & Shanahan, M. (1999). The event calculus in classical logic—alternative axiomatisations. *Electronic Transactions on Artificial Intelligence, 3*(A), 77–105.
52. Modgil, S., Faci, N., Meneguzzi, F. R., Oren, N., Miles, S., & Luck, M. (2009). A framework for monitoring agent-based normative systems. In C. Sierra, C. Castelfranchi, K. S. Decker, & J. S. Sichman (Eds.), *Proceedings of the 8th international joint conference on autonomous agents and multiagent systems* (Vol. 1, pp. 153–160). Richland, SC: IFAAMAS.
53. Montali, M., Torroni, P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., & Mello, P. (2008). Verification from declarative specifications using logic programming. In M. G. de la Banda & E. Pontelli (Eds.), *Logic programming, 24th international conference, ICLP 2008*, Udine, Italy, 9–13 December 2008, Proceedings. Lecture notes in computer science (Vol. 5366, pp. 440–454). Berlin: Springer.
54. Mueller, E. T. (2006). *Commonsense reasoning*. San Francisco: Morgan Kaufmann.
55. Shanahan, M. (1999). The event calculus explained. In *Artificial intelligence today*. Lecture notes in artificial intelligence (Vol. 1600, pp. 409–430). Berlin: Springer.
56. Singh, M. P. (1991). Social and psychological commitments in multiagent systems. In *AAAI fall symposium on knowledge and action at social and organizational levels* (pp. 104–106). Menlo Park, CA: Menlo Park, CA
57. Singh, M. P. (1998). Agent communication language: Rethinking the principles. *IEEE Computer, 31*, 40–47.
58. Singh, M.P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law, 7*, 97–113.
59. Singh, M. P. (2008). Semantical considerations on dialectical and practical commitments. In D. Fox & C. P. Gomes (Eds.), *Proceedings of the twenty-third AAAI conference on artificial intelligence, AAAI 2008*, Chicago, IL, USA, 13–17 July 2008 (pp. 176–181). Menlo Park, CA: AAAI Press.
60. Singh, M. P., Chopra, A. K., & Desai, N. (2009). Commitment-based service-oriented architecture. *IEEE Computer, 42*(11), 72–79.
61. Spoletini, P., & Verdicchio, M. (2009). An automata-based monitoring technique for commitment-based multi-agent systems. In J. F. Hübner, E. T. Matson, O. Boissier, & V. Dignum (Eds.), *Coordination, organizations, institutions and norms in agent systems IV, COIN 2008 International Workshops, COIN@AAMAS 2008*, Estoril, Portugal, May 12, 2008. COIN@AAAI 2008, Chicago, USA, 14 July 2008. Revised Selected Papers. Lecture notes in computer science (Vol. 5428, pp. 172–187). Berlin: Springer.
62. Torroni, P., Chesani, F., Mello, P., & Montali, M. (2010). Social commitments in time: Satisfied or compensated. In M. Baldoni, J. Bentahar, M. B. van Riemsdijk, & J. Lloyd (Eds.), *Declarative agent languages and technologies VII*. Revised Selected and Invited Papers. Lecture notes in computer science (Vol. 5948, pp. 228–243). Berlin: Springer.
63. Torroni, P., Chesani, F., Mello, P., & Montali, M. (2012). A retrospective on the Reactive Event Calculus and Commitment Modeling Language. In C. Sakama, S. Sardina, W. Vasconcelos,

& M. Winikoff (Eds.), *Proceedings of the 9th international workshop on declarative agent languages and technologies (DALT 2011)*. Lecture notes in computer science (Vol. 7169, pp. 120–127). Berlin: Springer.

64. Torroni, P., Yolum, P., Singh, M.P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., & Mello, P. (2009). Modelling interactions via commitments and expectations. In V. Dignum (Ed.), *Handbook of research on multi-agent systems: Semantics and dynamics of organizational models* (pp. 263–284). Hershey, PA: IGI Global.
65. Venkatraman, M., & Singh M., P. (1999). Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems, 2*(3), 217–236.
66. Yolum, P., & Singh, M. P. (2002). Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the first international joint conference on autonomous agents & multiagent systems* (pp. 527–534). New York, NY: ACM Press.
67. Yolum, P., & Singh, M. P. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence, 42*(1–3), 227–253.