Ann Math Artif Intell (2012) 64:73–107
DOI 10.1007/s10472-012-9282-1

Exception diagnosis in multiagent contract executions

Özgür Kafalı · Paolo Torroni

Published online: 8 March 2012

Abstract We propose a diagnosis procedure that agents can use to explain exceptions to contract executions. Contracts are expressed by social commitments associated with temporal constraints. The procedure reasons from the relations among such commitments, and returns one amongst different possible mismatches that may have caused an exception. In particular, we consider two possibilities: *misalignment*, when two agents have two different views of the same commitment, and *misbehavior*, when there is no misalignment, but a debtor agent fails to oblige. We also provide a realignment policy that can be applied in case of a misalignment. Our formalization uses a reactive form of Event Calculus. We illustrate the workings of our approach by discussing a delivery process from e-commerce as a case study.

Keywords Commitments · Exception diagnosis · Alignment · Event calculus

Mathematics Subject Classifications (2000) 68T42 · 68T27 · 68T30

1 Introduction

Contracts are a traditional means to regulate and secure business transactions in open electronic markets. In such markets, agents may take on different roles such as buyers, sellers, auditors, information vendors, financial institutions and other intermediaries. Contracts make the dependencies between the contract participants explicit, and contain the norms that govern relevant interaction [35].

Ö. Kafalı
Department of Computer Engineering, Boğaziçi University, 34342, Bebek, İstanbul, Turkey
e-mail: ozgurkafali@gmail.com

P. Torroni (✉)
DEIS, University of Bologna, V.le Risorgimento, 2, 40136, Bologna, Italy
e-mail: paolo.torroni@unibo.it

Social commitments [47] are an increasingly common way of representing contracts among software agents. A social commitment (or simply, a commitment) is formed by a debtor agent towards a creditor agent about a specific property of interest. In realistic environments, a commitment is associated with a deadline, telling that its property has to be brought about within that time bound [48]. Otherwise, a violation occurs regarding that commitment. A violation of a commitment means an *exception* for its creditor.

In order to prevent a commitment violation, it is essential that agents have some kind of control over their trading partners (e.g., the creditor agent works with a trustworthy debtor as the contractor of its commitment). However, it is not possible to predict the behavior of the agents in open multiagent systems. Moreover, the environment can also be unpredictable, and cause a mismatch between the agents' contracts even though the agents are trustworthy. Therefore, we need to monitor and handle exceptions in order to ensure safe contract execution.

Contracts are manipulated in two phases; contract negotiation and contract execution [26, 35]. Contract negotiation is out of the scope of this paper. Rather, we assume that agents start executing the protocol with predetermined (and possibly negotiated) contracts. For contract execution, we do not consider an authority that supervises the interactions of the agents regarding their contracts. Instead, agents may run distributed/local monitoring on the environment, and verify their contracts accordingly. Another thing to consider about contract violation is sanctions. That is, once a commitment is violated, a suitable penalty is applied to the debtor. In order for sanctions to function properly, timely monitoring/diagnosis is required, since there may be deadlines associated with issuing sanctioning procedures.

In a distributed contract-based setting, each agent keeps track of its own commitments. Thus, the cause of an exception is often a misalignment between the debtor and the creditor's individual copies of the same commitment. Example 1 presents such a scenario from a real-life delivery process.

Example 1 The customer, the bank, the store and its courier: a book's online purchase. Let us consider a scenario with a *customer*, Federico, who wishes to buy a copy of a book, "Harry Potter and the Deathly Hallows" (HP7), from an online *store* (Amazon). The transaction needs two additional parties: a *bank* (HSBC) to carry out the payment and a *courier* (UPS) to deliver HP7 to Federico.

The process begins with Federico paying for HP7 using HSBC on Monday. The contract between Federico and Amazon states that HP7 will be delivered in five business days as of the time HSBC verifies Federico's payment. Assume that HSBC verifies Federico's payment on Wednesday. Now, Federico expects a delivery by the following Wednesday. But what if HSBC does not notify Amazon about the verification of Federico's payment until Friday? When Amazon receives the notification, it infers the deadline for delivery as the following Friday (two days later than Federico has previously inferred!). When Federico does not receive HP7 on Wednesday, he contacts Amazon to ask the reason of the delay. Amazon tells Federico that there are two more days until the deadline. At this point, Federico understands that there is a mismatch between their copies of the contracts. He has to decide what to do next. One such possibility is to align his contract with Amazon's, by changing his deadline, and wait a bit longer.

One key point in Example 1 is the ability of parties to reason about contractual obligations, based on contractual clauses, i.e., implications and facts that specify the contract, and on relevant events that occur at specific points in time. The presence of domain-related as well as general-purpose knowledge that agents can use to make inferences about the state of their commitments is a common setting in realistic e-commerce scenarios. It is also important to stress that parts of such knowledge, such as contract specifications, are agreed upon, *shared*, by the interested agents, whereas other parts of it, such as knowledge about the occurrence of relevant events, depend on observations made by agents autonomously and independently, and are thus a potential source of mismatch.

For instance, at some point Federico becomes aware of a mismatch between Amazon's understanding of the commitment about HP7 and his own. In particular, he realizes that they are inferring different deadlines. That is a typical *misalignment* of commitments, due to differences between the debtor's and the creditor's observations [17].

Another possible mismatch can be due to the debtor's *misbehavior*, e.g., Amazon delegates its commitment to UPS but then gives a wrong deadline. Finally, a mismatch may be caused by a simple *misunderstanding* among agents, e.g., Federico receives another book instead of HP7. Schroeder and Schweimeier [45] study such misunderstandings using negotiation and argumentation theory. In this work, we only consider misalignment caused by different observations of the agents, and misbehavior caused by the debtor failing to oblige.[1]

Among the few related works, Chopra and Singh [17] formalize commitment alignment in multiagent systems. They consider misalignment of commitments that arise from different observations of the debtor and the creditor. They show how the creditor can *prevent* misalignments, by informing the debtor when the condition of each conditional commitment is satisfied, so that debtor and creditor can infer the same base-level commitments. This approach requires extra communication. If we assume exceptions to be rare events in process executions, such a communication overhead may be unjustified. Accordingly, we choose to verify alignment *on demand*, i.e., when an exception occurs. Besides, Chopra and Singh do not formalize commitment deadlines. We instead accommodate time-aware commitments.

Accordingly, we propose a distributed collaborative process to diagnose exceptions due to misalignment or misbehavior. When the creditor agent detects that one of its commitments is violated, it initiates the diagnosis process by making a diagnosis *request* to the commitment's debtor. As diagnosis proceeds, agents exchange information about relevant commitments. In Example 1, Federico makes a diagnosis request to Amazon about his violated commitment. The diagnosis process may involve a larger set of agents, e.g., Amazon may have delegated its commitment to UPS. In the end, such a process results in one of the following outcomes:

1. a *misalignment* is found, with a possible commitment to be aligned with (if any exists), or
2. a *misbehavior* is found, with the identification of a "culprit" agent.

[1] Here, "misbehavior" must be literally intended as a *failure to function correctly*. We do not mean to imply intentionality in such failures. The question *why* an agent misbehaves is entirely outside of the scope of this paper.

There are two cases of misalignment: (1) the one described by Chopra and Singh [17] where the creditor infers the commitment, but the debtor does not, and (2) a temporal misalignment which we describe here, where the debtor infers a later deadline for the commitment than the creditor. In the case of a temporal misalignment, the agents can maintain alignment via an alignment policy described by a set of commitment *update* rules. This is often the case for real-life delivery scenarios; the customer may accept to wait a bit longer, if it is a matter of adjusting a deadline. Alternatively, agents can start negotiating about what to do next. That can solve both situations of misalignment and misbehavior. More elaborate solutions are indeed possible. We do not address negotiation in this paper, and this trivial form of automatic realignment is simply a by-product of our diagnosis procedure.

Our diagnosis architecture includes "coupled" knowledge-bases, in which agents store the protocol rules and contracts they agree upon. That is, a part of a protocol formalization concerning two agents is stored in a knowledge-base containing agreed-upon protocol rules, that are accessible by and thus shared between these two agents. Similarly, a contract is contained in the knowledge-base shared between its participants. In particular, these coupled knowledge-bases contain commitment and protocol rules and facts agreed upon by the interested parties. They do not contain an extensional description of the commitments, e.g., the current states of the commitments. These are elaborated individually by the agents.

Each agent has a separate trace of happened events according to what it observes. Thus, an agent can only track down the status of its own commitments. We assume that agents are always honest and collaborative during diagnosis. That is, when an agent is requested to take part in the diagnosis process, it does so. It is worthwhile stressing that identifying an agent as being a culprit for misbehavior does not necessarily make that agent dishonest or malicious. The agent may have unintentionally caused an exception, but it can still be motivated to help identifying its cause, e.g., in order to keep its good reputation.

The procedure we propose always terminates, is sound and, if associated with suitable agent policies, is capable of removing misalignment whenever possible. We discuss these and other results. In order to evaluate our approach, we extend the scenario described in Example 1 by presenting, as a case study, two different traces of events that lead to separate diagnosis results.

We formalize the agents' interactions in $\mathcal{REC}$ [5, 8], a reactive form of *Event Calculus* [36]. We shall emphasize that our diagnosis framework is orthogonal to the choice of commitment specification language. In particular, we could have used any other specification language that enables modeling time-aware commitments with metric time. However, $\mathcal{REC}$ turned out to be the simplest such language. Besides, $\mathcal{REC}$'s embedding in an existing tool for run-time commitment monitoring [9, 49] allowed us to test our method and evaluate it empirically.

The rest of the paper is structured as follows. Section 2 reviews related work in multiagent diagnosis literature. Section 3 describes commitments. Section 4 describes the delivery process running example. Section 5 describes similarity relations used by the diagnosis procedure, which is explained in Section 6. Section 7 illustrates the case study, and Section 8 concludes the paper.

2 Related work

Our work relates to three important research directions. The first one is towards exception handling in contract-based multiagent systems (e.g., e-commerce and e-business applications). A business process can be designed as a single protocol, or a composition of protocols, each describing a certain transaction amongst some of its participants. One of the key properties of business protocols is privacy, which is preserved by agents' policies [21]. In multi-party business processes, the compliance level of agents to their protocols determines how often exceptions arise. Roughly, compliance is related to conformance and interoperability [15, 24]. First, the agent's design must conform to the protocol's standards. However, this is a necessary, but not sufficient, condition. The agent should also be interoperable with other agents whom it interacts with. An agent's conformance to its protocol can be determined regarding the path it follows in its protocol. Nevertheless, an agent can conform to its role without following the exact protocol path it is supposed to follow, as long as it satisfies all of its commitments. This flexibility is due to commitment protocols.

Exception handling in business processes is a multi-disciplinary task which attracts the attention of computer science, management, economics, and several other disciplines. The MIT Process Handbook provides a knowledge source on business processes that aims at combining the efforts developed so far by separate disciplines [34]. It proposes a taxonomy of business processes, and proposes dependence relations between them; (1) *flow*, if a process consumes the resource produced by another process; (2) *fit*, if two processes together produce a resource; and (3) *sharing*, if two processes consume the same resource. In this paper, we have chosen a delivery process since it mimics the exception-prone structure we look for. The distributed nature of the delivery protocol enables us to define role-based protocols for each business party, and reason on exceptions with partial knowledge. The delivery process consists of flow type dependencies. That is, the process evolves sequentially among the participants. The details of the process are given in Section 4.

The MIT Exception Repository is an extension to the MIT Process Handbook whose aim is to relate exception handling with business processes, and make exception expertise available to each process [33]. The repository focuses on coordination based exceptions which are initiated from dependency relations among processes. Similar to the process taxonomy, an exception taxonomy describes the types of exceptions from the most general to the most specialized. Related with each coordination mechanism in the process handbook is a set of exception types described in the exception repository. Associated with each exception type, there is an exception handling process used to recover from the exception. In contrast to such static recovery schemes, where all the faults are predetermined at design-time, here we deal with run-time exceptions that need to be detected and diagnosed during execution.

The second direction is towards diagnosis. Diagnosis is a process that starts from observing a deviation from the expected behavior of a given system, and whose purpose is to identify the reason of the exception, i.e., to locate those subsystems whose abnormal behavior accounts for the observed behavior [38]. In multiagent systems, diagnosis is typically initiated by one specific agent, who detects an exception in

the system, and interacts with other agents in order to isolate a problem in its own behavior or in the behavior of other agents.

The diagnosis problem is in general a hard one. For example, component-based diagnosis is in the worst case exponential in the number of annotated components [7]. Multiagent diagnosis presents additional problems, depending on the setting. In closed systems, such as teams, we may need to avoid information flooding. In open domain, such as e-commerce settings, we may care for privacy of information and for the trust we put on autonomous, unknown agents.

There are two fundamentally different approaches to diagnostic reasoning: heuristic approaches, such as *fault-based diagnosis* (FBD) and diagnosis from the first principles or *model-based diagnosis* (MBD) [38]. In FBD, the idea is to encode the diagnostic reasoning of human experts in a given domain. The real-world system is not modeled. All known faults are instead modeled. Conversely, MBD starts from a model of the structure (components and their connections) and function (or behavior) of the system, and a set of observations indicating an abnormal behavior. A system is faulty if the observed behavior of the system contradicts the behavior predicted by assuming that all of its components are correct [18].

As pointed out by Kalech and Kaminka following Micalizio et al. [40], fault-based techniques [20, 25, 37, 41], in which faults are modeled in advance, cannot be used for multiagent diagnosis, because the interactions in multiagent systems are unpredictable. For this reason, multiagent diagnosis is typically model-based. This is especially true in open systems, where the idea of social commitments is precisely to avoid enumerating all possible ways agents can interact in order to fulfill a contract, thus providing agents with more flexibility and opportunities [46].

Thus in recent years, the MBD approach has been applied to MAS diagnosis by several research groups, including Console et al. [19] with applications in the automotive industry [42], Roos et al. [44, 51] for plan diagnosis, and Kalech and Kaminka [29, 30] for coordination failures in agent teams. These are in general closed systems. For example, in [29], a coordination model is described by way of concurrency and mutual exclusion constraints. The approach assumes that each agent has knowledge of all the possible behaviors available to each team-member, i.e., their *behavior library*. In this way, each observing agent creates a model of other agents in the team. Transitions between one behavior to another are described in terms of preconditions and termination conditions. Then, a "social" diagnosis is initiated as a collaborative process aimed at finding causes of failures to maintain designer-specific social relationships [31]. More specifically to the case of team-work, a diagnosis is a set of contradictions in beliefs that accounts for the selection of different team behaviors by different agents [29]. MBD has also been used by Ardissono et al. to enable Web services with diagnostic capabilities [3]. Thus when defining complex Web services, each component (a simple Web service) is associated with a *local diagnoser*, whereas the complex Web service is associated with a *global diagnoser* service. Differently from Web service compositions, multi-agent interaction is not orchestrated (as in BPEL programs), but autonomous. Diagnosis exception in such an open and flexible setting calls for new approaches, such as the one we present here.

The final direction is towards commitment protocols and misalignment. Commitments are widely used to model agent interactions. They are live objects that change state based on occurring events or on actions being executed [47]. In its

traditional definition, a commitment does not include a notion of time, e.g., a deadline. That is, the content (i.e., proposition) of the commitment is a logical formula which is not necessarily associated with time. However, time is essential when considering whether commitments are violated or fulfilled [48]. A commitment has three fundamental states; *active*, *fulfilled*, and *violated* [22, 23]. In a centralized monitoring system where all the interactions of agents are observable, commitment tracking is an effective way to detect protocol exceptions [8]. While the commitment description does not need any modifications for this process, temporal considerations are required to enable deadlines for commitments.

The Event Calculus is used for representing events and their outcomes [36]. Commitments can be created in the Event Calculus, and protocol evolution can be observed based on actions that manipulate commitments. Event Calculus, in its nature, is used for backward reasoning, thus it is goal-driven. But Reactive Event Calculus [8, 48], on the other hand, is event-driven. That is, it enables forward reasoning in time which allows commitment tracking during protocol execution. In particular, it can tell which properties change state as a new event occurs. The SCIFF framework combines backward and forward reasoning exactly for the purpose of runtime verification [1]. It models the agents' interactions through expectations rather than commitments. However, commitments can also be modeled in SCIFF using expectations [50]. Once commitments and deadlines are created, the tracking procedure just runs a monitoring process based on the happened events [48]. This monitoring process determines which commitments are violated. In addition, compensation rules describe how the system will behave when commitments are violated (i.e., their deadlines have passed). Thus, the procedure also offers a solution for the exceptions that occur due to commitment violations. Misalignment of commitments often causes exceptions for the involved agents [16, 17, 45] as we have described in the previous section.

To the best of our knowledge, no existing approach to distributed diagnosis is able to distinguish between misalignment and misbehavior, as we do in this paper. Here, we propose a logic-based framework to track down agents' commitments and diagnose faults via similarity relations described over those commitments. Moreover, we provide a means of recovery from misalignment situations, whenever possible (e.g., extend a commitment's deadline to a newly discovered time point).

3 Commitments in $\mathcal{REC}$

We use social commitments [47] to represent agent contracts. A social commitment is formed between a debtor and a creditor, in which the debtor is committed to the creditor for bringing about a property. Singh [47] considers two types of social commitments:

– *c(x, y, p)* is a **base-level commitment** between the debtor agent x and the creditor agent y to bring about the property described by the proposition p. When this commitment is created, it is said to be *active*, and x is committed to y for satisfying p. *c(x, y, p)* remains *active* until p gets either satisfied, in which case the commitment becomes *fulfilled*, otherwise, it becomes *violated* [52].

– *cc(x, y, q, p)* is a **conditional commitment** between the debtor agent x and the creditor agent y to bring about the property described by the proposition p when the condition q holds. When this commitment is active, namely when q is satisfied, x will become committed to y for satisfying p. Thus, a new base-level commitment *c(x, y, p)* is created, and we say that the conditional commitment is detached. Note that a conditional commitment may never be violated.

It is more realistic to consider commitments with time [48]. That is, the debtor is committed to satisfy the property for the creditor within a predefined deadline. In this paper, we use the Reactive Event Calculus ($\mathcal{REC}$) [8] to model *time-aware* commitments (i.e., commitments with temporal constraints). $\mathcal{REC}$ extends the Event Calculus, which is based on Prolog,[2] to monitor commitments at run-time. $\mathcal{REC}$ models two types of temporal constraints on commitments: (1) an existential temporal constraint where the property of the commitment has to be brought about inside a time interval, and (2) a universal temporal constraint where the property of the commitment has to be maintained valid along a time interval. In this paper, we focus on base-level commitments with existential temporal constraints.

We use the following syntax to represent an existential base-level commitment throughout the paper:

$$s(c(x, y, property(e(t_1, t_2), p))),$$

where:

– s is a label identifying the state of the commitment at a specific time point. It can be *active*, *fulfilled*, or *violated*;[3]
– x and y are the debtor and the creditor of the commitment, respectively;
– the existential temporal constraint $e(t_1, t_2)$ on the property p, which is represented by a logic formula, means that p must be satisfied at some time t, $t_1 \leq t \leq t_2$.

When the commitment is first created by the *create* operation [52], the commitment's state s is *active*. It remains active until t_1. After t_1, if p is satisfied between t_1 and t_2, the commitment's state s becomes *fulfilled* as soon as p is satisfied. Otherwise, it becomes *violated* as soon as t_2 is past. A detailed explanation of how $\mathcal{REC}$ manipulates commitment states can be found in [10].

In $\mathcal{REC}$, we can express that an event *initiates* (or *terminates*) a *fluent* or *property* of the system, by way of *initiates(Event, Fluent, Time)* relations (see an example in Listing 5 below). We use the following syntax to represent a happened event:

$$hap(event(exec(e(x, y, \chi_1, \ldots, \chi_n))), t).$$

Each event is thus represented as an *exec* message between an agents x, the sender, and y, the receiver. The event description is represented by e, and χ_1 through χ_n are the parameters associated with e. The time in which the event has occurred is represented by t.

[2] In the sequel, we assume that the reader has some basic knowledge of Prolog.

[3] We use this notation for presentation purposes only. In $\mathcal{REC}$, the state is also a parameter of the commitment description, and any number of states can be accommodated.

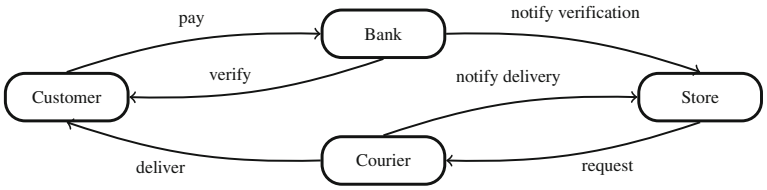


Fig. 1 Delivery process

4 Running example

Figure 1 shows the delivery process introduced in Example 1. We assume that the customer has already placed the order, by direct or indirect interaction with the store, e.g., via an e-commerce Web site. We thus focus on the subsequent phases (payment & delivery).

In a desired execution, first the customer sends the payment to the bank regarding its purchase from the store (*pay*). Then, the bank verifies the payment of the customer (*verify*), and informs the store about the verification (*notify verification*). Upon receiving the verification, the store requests the delivery of the book from the courier (*request*). Finally, the courier delivers the book to the customer (*deliver*), and informs the store about the delivery (*notify delivery*). Listings 1 through 5 show how this process is formalized in $\mathcal{REC}$, in coupled knowledge-bases. Recall that the coupled knowledge-base of two agents contain the protocol rules and contract descriptions involving those agents.

Listing 1 shows the $\mathcal{REC}$ rules shared among the customer and the bank agents. We model the agents' interactions as *exec* events from a sender towards a receiver. The first two rules (CB_1 and CB_2) describe the effects of such events in terms of fluents *paid* and *verified*. When the payment is sent from the customer to the bank, the fluent *paid* starts to hold (CB_1). Likewise, when bank verifies the customer's payment, the fluent *verified* starts to hold (CB_2). The last parameter for both *initiates* rules is a blank variable, telling that the time of event is not significant for its effect to happen. The last rule (CB_3) corresponds to the *create* operation described for commitments [48, 52]. Here, the syntax of *create* in CB_3 follows the literature; the first parameter is the event that initiates the base-level commitment (this is the requirement for Event Calculus), the second parameter is the debtor of the commitment (a commitment can only be created by its debtor), and the last parameter is the commitment itself. Since we deal with time-aware commitments here, the body

Listing 1 Coupled knowledge-base of the customer and the bank

```
% CB1: pay
initiates(exec(pay(Customer,Bank,Item)),paid(Item),_).

% CB2: verify payment
initiates(exec(verify(Bank,Customer,Item)),verified(Item),_).

% CB3: verify commitment
create(exec(pay(Customer,Bank,Item)),Bank,
        c(Bank,Customer,property(e(Ts,Te),verified(Item))),Ts):-
        Te is Ts + 3.
```

of the rule handles the time constraints associated with the commitment. That is, when the customer sends the payment for an item to the bank, then the bank will be committed to verifying that payment in three time units (without loss of generality, we use days as the time unit from now on). Lines starting with % are comments.

Listing 2 shows the $\mathcal{REC}$ rules shared among the bank and the store agents. The only rule, BS_1, is similar to CB_2, the only difference being the receiver. That is, the bank notifies the store about the verification of the customer's payment.[4]

Listing 2 Coupled knowledge-base of the bank and the store

```
% BS1: verify payment
initiates(exec(verify(Bank,Store,Item)),verified(Item),_).
```

Listing 3 shows the $\mathcal{REC}$ rules shared among the customer and the store agents. The only rule CS_1 describes the commitment between them. The semantics is that when the bank sends the payment verification, then the store will be committed to deliver the item in five days.[5]

Listing 3 Coupled knowledge-base of the customer and the store

```
% CS1: verify–deliver commitment
create(exec(verify(Bank,_,Item)),Store,
    c(Store,Customer,property(e(Ts,Te),delivered(Item))),Ts):-
    Te is Ts + 5, holds_at(in_stock(Item,Store),Ts).
```

Listing 4 shows the $\mathcal{REC}$ rules shared among the store and the courier agents.

Listing 4 Coupled knowledge-base of the store and the courier

```
% SD1: send for delivery
initiates(exec(request(Store,Courier,Item)),requested(Item),_).
% SD2: deliver
initiates(exec(deliver(Courier,Store,Item)),delivered(Item),_).
% SD3: send–deliver commitment
create(exec(request(Store,Courier,Item)),Courier,
    c(Courier,Store,property(e(Ts,Te),delivered(Item))),Ts):-
    Te is Ts + 3.
```

The first two rules (SD_1 and SD_2) describe the events for the request of a delivery, and the delivery itself. The last rule SD_3 describes the commitment between the two agents. The semantics is that when the store requests the delivery of an item, then the courier will be committed to deliver that item in three days.

[4]For the sake of simplicity, we do not indicate further conditions on *Bank*, *Store*, and *Item*, which are free variables. A detailed implementation would require to express restrictions on such variables, i.e., to define the "context" [12].

[5]The blank variable in the receiver location of the *exec* message accounts for either the customer or the store. When the customer receives the message, it will infer the commitment. Similarly, the store will infer the commitment when it is the receiver of that message.

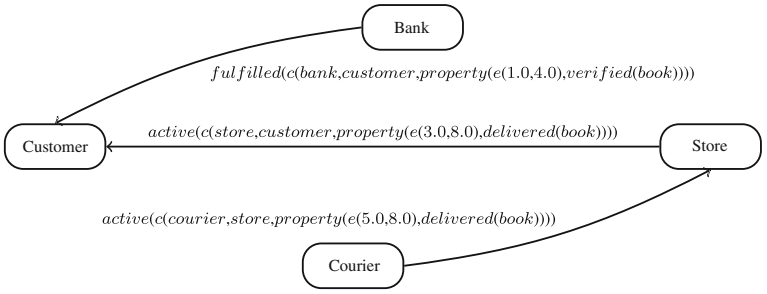


Fig. 2 Commitments in the delivery process at time 6.0

Listing 5 shows the $\mathcal{REC}$ rules shared among the courier and the customer agents. The only rule DC_1 is similar to the rule SD_2, in which the only difference is the receiver.

Listing 5 Coupled knowledge-base of the courier and the customer

```
% DC1: deliver
initiates(exec(deliver(Courier,Customer,Item)),delivered(Item),_).
```

For illustration purposes, the commitment rules contain the parties *Customer*, *Bank*, *Store*, and *Courier*. These parties represent the roles that agents will enact during the protocol's execution. The real contracts, however, should define explicitly which agents are involved in the contract, e.g., they should include a specific customer name. Such definitions could be written modularly in $\mathcal{REC}$, by resorting to the concept of role [11].

We have implemented the diagnosis framework in the j-$\mathcal{REC}$ tool for run-time monitoring[6] which embeds a tuProlog reasoner.[7] We could thus run experiments in a simulated environment. In this paper, we only show some excerpts of the whole code. A running prototype with a full implementation can be downloaded from a dedicated Web page.[8]

Figure 2 demonstrates a snapshot of the agents' commitments in the delivery process. The snapshot is taken at time 6.0. You can see that the bank has already fulfilled the commitment towards verifying the customer's payment. Upon receiving the verification of payment, the store has created the commitment towards delivering the book to the customer. In addition, it has delegated the commitment to the courier.

[6] http://www.inf.unibz.it/~montali/tools.html#jREC

[7] http://sourceforge.net/projects/tuprolog

[8] http://mas.cmpe.boun.edu.tr/ozgur/code.html. See Appendix B for further information.

5 Commitment similarity

Chopra and Singh [17] propose a stronger-weaker relation for commitments using the commitments' conditions and propositions (i.e., properties). However, we do not focus on the properties of the commitments. Instead, we make comparisons based on the temporal constraints associated with their properties (i.e., deadlines), and the agents involved (i.e., debtor and creditor). Accordingly, we propose the following similarity levels for commitments. We consider two commitments about the same transaction (e.g., payment and delivery of a certain book) to be *relevant* to each other. A commitment's property suffices to identify a specific transaction. Thus the following definition:

Definition 1 (Relevance) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is relevant to commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if property p_1 is identical[9] to property p_2.

Example 2 Table 1(a) shows an example of relevance; c_1 and c_2 are relevant since their properties are identical (even though their state, debtor, creditor and temporal properties may differ).

Remark 1 Relevance is an equivalence relation, i.e., it is reflexive, symmetric and transitive.

Definition 2 (Cover) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ covers commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 \geq t_3$, and $t_2 \leq t_4$.

Definition 3 (Extension) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is an extension of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 < t_3$, and $t_2 > t_4$.

Example 3 Table 1(b) shows an example of cover; c_3 covers c_4 since they are relevant to each other, and the time span of c_3 is within that of c_4. That is, if the customer wants delivery to be performed between times 3–10, then he will also accept delivery between 5–8. Conversely, c_4 is an extension of c_3, in which the customer may not accept an extended deadline for delivery.

Definition 4 (Forward-shift) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a forward-shift of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 \geq t_3$, and $t_2 > t_4$.

[9] In this work, by "identical" we mean *literally* (syntactically) identical. This choice is dictated by our intention to keep our focus on temporal and delegation aspects of misalignment. In future versions, the framework could be extended to encompass more elaborate similarity relations, such as subsumption and part-of relations, and others based on edit distance [45].

Table 1 Similarity relations

(a) $c_1 = active(c(courier, store, property(e(7.0,10.0), delivered(book))))$
$c_2 = violated(c(store, customer, property(e(3.0,8.0), delivered(book))))$
(b) $c_3 = active(c(store, customer, property(e(5.0,8.0), delivered(book))))$
$c_4 = active(c(store, customer, property(e(3.0,10.0), delivered(book))))$
(c) $c_5 = active(c(store, customer, property(e(5.0,10.0), delivered(book))))$
$c_6 = violated(c(store, customer, property(e(3.0,8.0), delivered(book))))$
(d) $c_7 = active(c(courier, store, property(e(5.0,8.0), delivered(book))))$
$c_8 = active(c(store, customer, property(e(3.0,8.0), delivered(book))))$
(e) $c_9 = active(c(courier, store, property(e(7.0,10.0), delivered(book))))$
$c_{10} = violated(c(store, customer, property(e(3.0, 8.0), delivered(book))))$
(f) $c_{11} = active(c(courier, store, property(e(4.0,7.0), delivered(book))))$
$c_{12} = active(c(store, customer, property(e(5.0, 10.0), delivered(book))))$

Definition 5 (Backward-shift) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a backward-shift of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 < t_3$, and $t_2 \leq t_4$.

Example 4 Table 1(c) shows an example of forward-shift; c_5 is a forward-shift of c_6 since they are relevant to each other, the starting point of c_5's time span is no less than that of c_6, and its end point (deadline) is strictly greater than c_6's deadline. Conversely, c_6 is a backward-shift of c_5.

Note that forward-shift is *not* the inverse of backward-shift. For instance, c_4 from Example 3 is a backward-shift of c_5 from Example 4, but c_5 is not a forward-shift of c_4.

Remark 2 The following properties hold for the relations introduced so far. *Cover* is reflexive, asymmetric, and transitive. *Extension*, *forward-shift*, and *backward-shift* are all anti-symmetric and thus non reflexive, but they are transitive.

Lemma 1 *Any two commitments c_i and c_j that are relevant to each other, are in one* (*and only one*) *of the four temporal relations specified in Definitions* 2–5. *In other words, if commitment c_i is relevant to commitment c_j, then one and only one of the following relations holds:*

- *c_i covers c_j;*
- *c_i is an extension of c_j;*
- *c_i is a forward-shift of c_j;*
- *c_i is a backward-shift of c_j.*

(*one option excludes the other ones*).

Proof There are 13 possible (exclusive) relations between two time intervals according to Allen [2]. We show that our temporal relations cover all of those. Let us now

enumerate each possible case; below $[t_1, t_2]$ and $[t_3, t_4]$ are the temporal constraints of c_i and c_j, respectively.

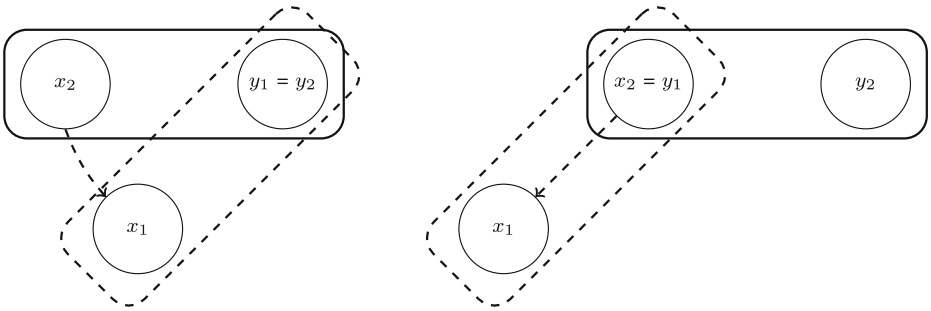
1. $t_2 < t_3$: c_i is a backward-shift of c_j,
2. $t_1 > t_4$: c_i is a forward-shift of c_j,
3. $t_1 = t_3$ and $t_2 = t_4$: c_i covers c_j,
4. $t_2 = t_3$: c_i is a backward-shift of c_j,
5. $t_1 = t_4$: c_i is a forward-shift of c_j,
6. $t_1 > t_3$ and $t_2 < t_4$: c_i covers c_j,
7. $t_1 < t_3$ and $t_2 > t_4$: c_i is an extension of c_j,
8. $t_1 < t_3 < t_2 < t_4$: c_i is a backward-shift of c_j,
9. $t_3 < t_1 < t_4 < t_2$: c_i is a forward-shift of c_j,
10. $t_1 = t_3$ and $t_2 < t_4$: c_i covers c_j,
11. $t_1 = t_3$ and $t_2 > t_4$: c_i is a forward-shift of c_j,
12. $t_1 > t_3$ and $t_2 = t_4$: c_i covers c_j,
13. $t_1 < t_3$ and $t_2 = t_4$: c_i is a backward-shift of c_j.

The proof that the relations above are mutually exclusive trivially follows their definitions. □

Lemma 1 states that we can compare any two relevant commitments in terms of the (temporal) similarity relations we propose, and that we can partition the set of relevant commitments based on such relations. Let us now introduce the notion of delegatee, which we need in the following definitions regarding commitment delegation.

Definition 6 (Delegatee) The debtor-creditor couple (x_1, y_1) is a delegatee of the debtor-creditor couple (x_2, y_2) if (a) $x_1 \neq x_2$ and $y_1 = y_2$, or (b) $x_1 \neq y_2$ and $x_2 = y_1$.

Figure 3 demonstrates the delegatee relation. There are two cases; (a) the debtor of the commitment delegates it to a new debtor (the debtor of the former commitment has no longer the responsibility), (b) the debtor of the commitment gets involved in a new commitment with another agent towards the same property, this time it is the creditor of the new commitment.



(a) new debtor for the same creditor (b) debtor is the new creditor

Fig. 3 Delegatee

Example 5 As an example of case (a), consider the following scenario: the store is committed to the customer for delivering the book. Now assume that the store does not currently have the book in stock. Thus, the store delegates its commitment to another store, and informs the customer about the new commitment. For case (b), consider again that the store has the commitment to the customer, but this time it needs a courier in order to deliver the book to the customer. Thus, it gets involved in another commitment with the courier towards delivery. The latter commitment is again a delegation of the former commitment.

Some authors [17, 52] propose a more restricted definition of delegation which is limited to case (a). There, when a delegation occurs, only the debtor of the commitment changes. Definition 6 extends the notion of delegation by case (b). This provides a way to trace a set of delegated commitments when diagnosing an exception (e.g., identify the sequence of delegations). The first case does not support this by just looking at the commitments themselves. That is, if the commitment is delegated several times, it is not possible to keep track of the delegation sequence.

Remark 3 *Delegatee* is anti-symmetric and thus non reflexive. It is also not transitive.

The delegatee relation only makes sense when embedded in a commitment, as described next.

Definition 7 (Delegation) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a (proper) delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 covers c_2, and (x_1, y_1) is a delegatee of (x_2, y_2).

Definition 7 describes commitment delegation; commitment c_2 is delegated to agent x_1. The delegation of a commitment usually has the same state as the commitment itself.

Example 6 Table 1(d) shows an example of delegation; c_7 is a delegation of c_8 since c_7 covers c_8, and the debtor-creditor couple of c_7 is a delegatee of that of c_8.

Alongside with proper delegations of commitments there may be "improper" delegations, which may cause an exception. The exception is usually due to an agent failing to bring about a property within a given time interval. Such improper delegations may give rise to exceptions because they extend in one way or another the time interval specified in the initial commitment. We distinguish between three different types of improper delegation: forward-shift delegation, extension delegation, and backward-shift delegation.

Definition 8 (Forward-shift delegation) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a forward-shift delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is a forward-shift of c_2, and (x_1, y_1) is a delegatee of (x_2, y_2).

Definition 9 (Extension delegation) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is an extension delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is an extension of c_2, and (x_1, y_1) is a delegatee of (x_2, y_2).

Example 7 Table 1(e) shows an example of forward-shift delegation; c_9 is a forward-shift delegation of c_{10} since c_9 is a forward-shift of c_{10}, and the debtor-creditor couple of c_9 is a delegatee of that of c_{10}.

Definition 10 (Backward-shift delegation) Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a backward-shift delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is a backward-shift of c_2, and (x_1, y_1) is a delegatee of (x_2, y_2).

Example 8 Table 1(f) shows an example of backward-shift delegation; c_{11} is a backward-shift delegation of c_{12} since c_{11} is a backward-shift of c_{12}, and the debtor-creditor couple of c_{11} is a delegatee of that of c_{12}.

Remark 4 Delegation, forward-shift delegation, and backward-shift delegation are all anti-symmetric and thus non reflexive, they are also not transitive.

Lemma 2 *Any two commitments, c_i and c_j, such that c_i's debtor and creditor are delegatees of c_j's debtor and creditor, are in one* (*and only one*) *of the four delegation relations specified in Definitions* 7–10. *In other words, if $c_i = s_i(c(x_i, y_i, p))$, $c_j = s_j(c(x_j, y_j, q))$, and (x_i, y_i) is a delegatee of (x_j, y_j), then one and only one of the following relations holds:*

- *c_i is a delegation of c_j;*
- *c_i is an extension delegation of c_j;*
- *c_i is a forward-shift delegation of c_j;*
- *c_i is a backward-shift delegation of c_j.*

(*one option excludes the other ones*).

Proof The proof trivially follows from Lemma 1 and from Definitions 7–10. □

Thanks to Lemma 2, we can partition the set of possible commitments linked by delegation to a given commitment based on their mutual temporal relations.

Listing 6 shows some of the $\mathcal{REC}$ rules for the similarity relations.

6 Diagnosis process: architecture, procedure, and properties

The purpose of diagnosis is to investigate the state of commitments in the system, and return a possible cause of violation; either a misalignment or a misbehavior. Throughout this section, we provide the details regarding our diagnosis process.

Let $\mathcal{C}$ be the set of all commitments in the system and $\mathcal{A}$ be the set of all agents in the system. When a diagnosis process is initiated by an agent $A \in \mathcal{A}$, we are interested in identifying the cause of violation of a specific commitment $C \in \mathcal{C}$. We denote by $\mathcal{C}_A \subseteq \mathcal{C}$ the set of commitments A is aware of. By definition, $C \in \mathcal{C}_A$.

With respect to Definition 1, $\mathcal{C}_A^C \subseteq \mathcal{C}_A$ is the set of all and only the commitments that are relevant to C. Moreover, members of $\mathcal{C}_A^C$ that are a delegation of C by Definition 7 belong to $\mathcal{C}_A^{CX}$; those that are a forward-shift of C by Definition 4 belong to $\mathcal{C}_A^{Cf}$, and those that are a forward-shift delegation of C by Definition 8 belong to

Listing 6 Commitment similarity in $\mathcal{REC}$ (excerpt)

```
% S1: relevant
relevant(c(X1,Y1,property(e(Ts1,Te1),P)),
    c(X2,Y2,property(e(Ts2,Te2),P))).

% S2: forward–shift
fshift(c(X1,Y2,property(e(Ts1,Te1),P1)),
    c(X2,Y2,property(e(Ts2,Te2),P2))):-
    relevant(c(X1,Y1,property(e(Ts1,Te1),P1)),
        c(X2,Y2,property(e(Ts2,Te2),P2))),
    Ts1 > Ts2, Te1 > Te2.

% S3: delegatee
delegatee((X1,Y),(X2,Y)):- X1 X̄2. delegatee((X1,Y),(Y,Y2)):- X1 Ȳ2.

% S4: forward–shift delegation
fshift_delegation(c(X1,Y1,property(e(Ts1,Te1),P1)),
        c(X2,Y2,property(e(Ts2,Te2),P2))):-
        fshift(c(X1,Y1,property(e(Ts1,Te1),P1)),
            c(X2,Y2,property(e(Ts2,Te2),P2))),
        delegatee((X1,Y1),(X2,Y2)).
```

$\mathcal{C}_A^{CfX}$. By definition, $\mathcal{C}_A^{CX} \subset \mathcal{C}_A^{C}$ and $\mathcal{C}_A^{CfX} \subset \mathcal{C}_A^{Cf} \subset \mathcal{C}_A^{C}$, while $C \in \mathcal{C}_A^{C}$ by reflexivity (see Remark 1) and $C \notin \mathcal{C}_A^{Cf}$, $C \notin \mathcal{C}_A^{CX}$ because forward-shift and delegation relations are anti-symmetric (see Remarks 2 and 4).

For ease of reference, we summarize our notation in Table 3 (see Appendix A), and graphically illustrate the relations among commitments in Fig. 4.

Definition 11 below formally defines the concept of diagnosis. A diagnosis *process* starts from a violated commitment C, and aims to identify the reason behind that violation, either as a misalignment among C and a relevant commitment c, or as a misbehavior of the debtor x of a relevant commitment. The outcome of such a process is what we call diagnosis.

Definition 11 Given a set of agents $\mathcal{A}$, an agent $A \in \mathcal{A}$, and a violated commitment $C \in \mathcal{C}_A$, we call *diagnosis* of C by A an atom $\delta \in \{\text{misalignment}(c), \text{misbehavior}(x)\}$, for some $c \in \mathcal{C} \cup \{\emptyset\}$, or $x \in \mathcal{A}$.

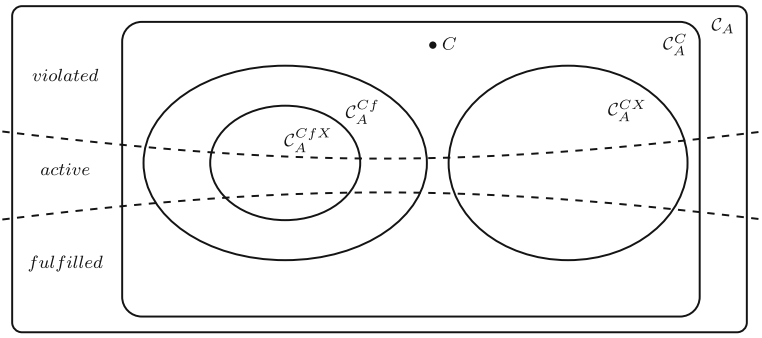


Fig. 4 Commitment relations

To denote that a given atom δ represents such a diagnosis, we write $\langle A, \mathcal{A}\rangle \overset{d(C)}{\models} \delta$, or simply $A \overset{d(C)}{\models} \delta$ when the set of agents in the system is clear from the context. Sometimes it is useful to focus the diagnosis on a specific set of commitments. This happens when some of the commitments relevant to C have been found not to be relevant to the diagnosis process. Thus to prevent unnecessary iterations and ensure that the diagnosis process always terminates, we keep track of already diagnosed commitments for exclusionary purposes.

We will write $A \overset{d(C)}{\models_{\Delta}} \delta$ to indicate that δ is a diagnosis of C which excludes a given set Δ of commitments from the set of commitments relevant to C.

While Definition 11 describes the generic outcome of a diagnosis process, we are interested only a diagnosis that actually identifies the reason of C's violation. We call it a *correct* diagnosis. The two possible outcomes of a correct diagnosis are:

1. *Misalignment*: When we are in the presence of a (correct) misalignment diagnosis, one of the following applies:
   (a) there is a commitment that is relevant to C, such that **its creditor infers the commitment but its debtor does not**,
   (b) there is a **violated** commitment that is relevant to C, such that its debtor infers that the commitment is **active** (i.e., forward-shift or extension).

   In case (a), the debtor's and creditor's event traces are misaligned. In particular, debtor does not infer the creditor's commitment because it does not have the event in its trace that would create such a commitment (e.g., the bank never informs the store about the customer's payment, thus the store is not committed to deliver).
   In case (b), the debtor infers the creditor's commitment with a shifted deadline (e.g., the bank indeed informs the store, but at a later time than the customer).
   Both cases are considered as misalignments in our diagnosis process, since the debtor is not directly responsible for the violation of the commitment.
2. *Misbehavior*: When we are in the presence of a (correct) misbehavior diagnosis, one of the following applies:
   (a) there is a violated commitment that is relevant to C, such that its debtor infers the commitment, and the debtor **has not delegated** the commitment, or
   (b) there is a violated commitment that is relevant to C, such that its debtor infers the commitment, and the debtor **has delegated** the commitment **without respecting its deadline** (i.e., forward-shift delegation or extension delegation).

   In case (a), the debtor also infers the violated commitment. Since the debtor has not delegated the commitment to another agent, it has the full responsibility for the violation (e.g., the store does not deliver in time).
   In case (b), the debtor has not violated the commitment itself since it has delegated the commitment to another agent. Nevertheless, it is still responsible for the violation since the deadline of the commitment is shifted during delegation (e.g., the store gives the delivery package to the courier too late).
   Thus, both cases are considered as misbehaviors in our diagnosis process.

Definition 12 describes correct diagnosis formally taking into account the above cases.

Definition 12 A *correct diagnosis* of $C \in \mathcal{C}_A$ by $A \in \mathcal{A}$ is a δ such that $A \overset{d(C)}{\models} \delta$ and:

1. (a) if $\delta = \text{misalignment}(\emptyset)$, then
   $\exists x, y \in \mathcal{A}, c = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p))) \in \mathcal{C}^C_{\mathcal{A}}$, such that
   - $c \in \mathcal{C}_y$ (c is known to its creditor, y), and
   - $\mathcal{C}^C_x = \emptyset$ (c is not known to its debitor, x);

   (b) if $\delta = \text{misalignment}(c)$, $c \in \mathcal{C}$, then
   $\exists x, y \in \mathcal{A}, c = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p)))$,
   $c' = s(c(x, y, \mathit{property}(\ e(t_3, t_4), p))) \in \mathcal{C}^C_{\mathcal{A}}$, such that
   - $s \in \{\mathit{active}, \mathit{fulfilled}\}$,
   - $c \in \mathcal{C}_y$ (c is known to be active or fulfilled to its creditor, y),
   - $c' \in \mathcal{C}_x$ (its debitor, x, considers it violated), and
   - c' is a forward shift or an extension of c (thus the misalignment);
2. if $\delta = \text{misbehavior}(x)$, then
   $\exists x, y \in \mathcal{A}, c = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p))) \in \mathcal{C}^C_{\mathcal{A}}$, such that
   - $c \in \mathcal{C}_x \cap \mathcal{C}_y$ (both debtor and creditor know c is violated), and either
     (a) $\mathcal{C}^{cY}_x = \emptyset$ for all $Y \in \mathcal{A}$ (the debtor, x, has not delegated c to anyone), or
     (b) $\mathcal{C}^{cfY}_x \cup \mathcal{C}^{ceY}_x \neq \emptyset$ for some $Y \in \mathcal{A}$ (x has delegated c using wrong deadlines).

Note that Definition 12 describes diagnosis by looking at the multiagent system as a whole, i.e., it gives an account of what has gone wrong in the system, by considering all the commitments and the agents, and tell what a diagnosis should be in each case.

We will now propose a procedure that can achieve this result in a distributed way, in which no agent has a global view.

Figure 5 describes the architecture used to perform such distributed diagnosis. Agents are equipped with local/private knowledge-bases and may possess other reasoning capabilities, not shown in the figure. Agents are connected to others through the coupled knowledge-bases. Recall that we use such knowledge-bases to represent knowledge shared by agents pairwise or group-wise, such as protocol rules, commitment rules and 2-party contracts. However, the actual instances of commitments (e.g., an active commitment of the store towards delivery) are not stored within those knowledge-bases, but they are instead inferred based on the existing knowledge at run time and on suitable monitoring capabilities. Each agent individually monitors the state of commitments within its own scope. To this end, it may use a $\mathcal{REC}$ engine such as j-$\mathcal{REC}$, or any other suitable run-time monitoring application equipped with a commitment theory and a domain representation.

For example, the customer can run its $\mathcal{REC}$ engine to track down the state of its commitment for delivery, to make sure that nothing is wrong. If it detects that the commitment is violated, it can verify whether its violated commitment is aligned with the other agents' commitments. In such a case, the customer initiates a diagnosis process to find out what has happened. The diagnosis process may involve a number

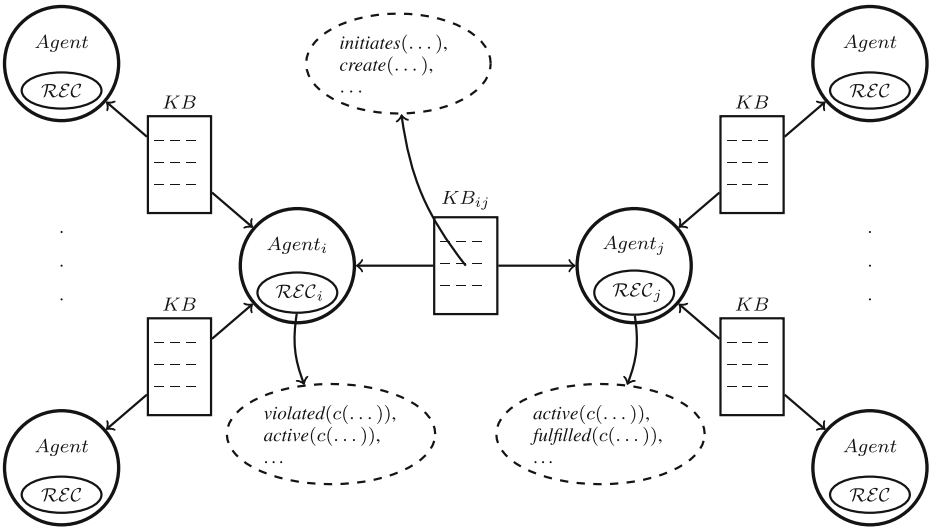


Fig. 5 Diagnosis architecture

of other agents depending on where the exception originated from. We cannot determine a priori who will be involved; if all agents in $\mathcal{A}$ or only a subset.

The diagnosis process will step through a series of diagnosis requests among agents in $\mathcal{A}$ until one of the outcomes in Definition 12 is reached and returned back recursively to the initiator. In each iteration, the diagnosing agent processes through its commitments, identifies the ones that are relevant to the subject commitment, and checks to see if it can find a relation among them that fits one of the cases as described in Definition 12. If so, the diagnosis ends with a result. Otherwise, it means that the agent has delegated the commitment faultlessly (i.e., there is nothing wrong the current diagnosing agent's actions). Now, the agent delegates the diagnosis process to the delegatee of the commitment. Recall that we do not allow multiple delegations of the same commitment to several agents. Thus, an agent can delegate its commitment to a single agent only. Thus, the next iteration of diagnosis always continues with only one branch.

Definitions 13–18 specify the diagnosis process declaratively.

Definition 13 (*Misalignment* diagnosis without further evidence)

$$\frac{\mathcal{C}_A^C \setminus \Delta = \emptyset}{A \overset{d(C)}{\vDash_\Delta} \mathit{misalignment}(\emptyset)}$$

If no commitment is found that is relevant to C, diagnosis identifies a misalignment as described in case (1a) of Definition 12. This is the case when the debtor does not infer the creditor's commitment at all. Note that there is no candidate commitment for realignment in this case.

Definition 14 (*Misalignment* diagnosis following forward-shift)

$$\frac{c \in \mathcal{C}_A^{Cf} \cup \mathcal{C}_A^{Ce} \setminus \Delta \wedge \mathcal{C}_A^{C*X} \setminus \Delta = \emptyset \wedge \{c = active(C') \vee c = fulfilled(C')\}}{A \overset{d(C)}{\vDash_\Delta} misalignment(C')}$$

If a commitment C' is found that is a forward-shift or an extension of C, and such a commitment is not violated, but instead still active, or even fulfilled, this means that there is a temporal misalignment between the creditor's and the debtor's copies of the same commitment.[10] The debtor may be intending to bring about the property of the commitment. Alas, too late! The diagnosis returns the debtor's copy as a candidate for alignment as described in case (1b) of Definition 12.

Definition 15 (*Misalignment* diagnosis following wrong delegation)

$$\frac{c \in \mathcal{C}_A^{CfX} \cup \mathcal{C}_A^{CeX} \setminus \Delta \wedge \{c = active(C') \vee c = fulfilled(C')\}}{A \overset{d(C)}{\vDash_\Delta} misalignment(C')}$$

Similar to Definition 14, if agent A finds a commitment C' which is still active, or already fulfilled, it means that C' has been delegated away. Again, there is a possibility of recovering from the violation (e.g., the creditor may not mind the delay). The diagnosis returns the debtor's copy as a candidate for alignment.

Definition 16 (*Misbehavior* diagnosis following failure to meet deadline)

$$\frac{violated(C') \in \mathcal{C}_A^{C} \setminus \Delta \wedge \mathcal{C}_A^{C*X} \setminus \Delta = \emptyset}{A \overset{d(C)}{\vDash_\Delta} misbehavior(A)}$$

If an agent finds a violated commitment relevant to C, but no delegation of it, it means that the debtor failed to fulfill the commitment in time, and that it is no one else's responsibility. Thus, diagnosis returns the debtor as a culprit as described in case (2a) of Definition 12.

Definition 17 (*Misbehavior* diagnosis following wrong delegation)

$$\frac{violated(C') \in \mathcal{C}_A^{CfX} \cup \mathcal{C}_A^{CeX} \setminus \Delta}{A \overset{d(C)}{\vDash_\Delta} misbehavior(A)}$$

If a violated commitment is found that is a forward-shift delegation, or an extension delegation of C, this means that the debtor has delegated its commitment without respecting its deadline. Accordingly, diagnosis reports the debtor as a culprit as described in case (2b) of Definition 12.

[10] A diagnosis process is always initiated following the violation of a commitment. Thus, the backward-shift of a violated commitment will also be violated, and cannot cause a misalignment. For this reason, we do not inspect backward-shift-related commitments in the diagnosis process. More about this topic in Section 8.

Definition 18 (Propagation of diagnosis following successful local check)

$$\frac{c \in \mathcal{C}_A^{CX} \cup \mathcal{C}_A^{Cb\,X} \setminus \Delta \wedge X \overset{d(C)}{\vDash}_{\Delta \cup c} \delta}{A \overset{d(C)}{\vDash}_{\Delta} \delta}$$

Definition 18 covers a last case, in which a violated commitment is found, that is either a delegation or a backward-shift delegation of C. This means that the debtor has correctly delegated its commitment, but the debtor of the delegated commitment (delegatee) has not fulfilled the commitment as it should have. Diagnosis continues with the delegatee. Recall that we allow only a single delegation of the same commitment. Thus, diagnosis continues with one branch only. To prevent infinite loops, the next agent to continue diagnosis will exclude those commitments in Δ, since they have already been inspected.

To execute such a diagnosis process, agents only need to be able to infer the state of commitments at run-time, and have basic inference capabilities such as simple operations with sets. Importantly, since each inference rule involves only one agent making an inference about the state of its commitments, and possibly a request to another specific agent (the delegatee of one of its commitments), it is possible to implement the diagnosis process in a distributed way. In particular, a j-$\mathcal{REC}$ implementation of such a process is available (see Appendix B).

Let us now analyze and discuss the behaviour and outcomes of the diagnosis process. Let $\mathcal{M}_1$ be any method that implements a diagnosis process following Definitions 13–18. The following properties hold for $\mathcal{M}_1$:

Property 1 ($\mathcal{M}_1$ terminates) We consider two cases for termination: (1) there does not exist any circular chain of delegations, and (2) there exists a circular chain of delegations. Termination for the former case is trivial since the number of iterations is bounded with the number of agents in the system. For the latter case, consider the following circular chain of delegations among the commitments; $c_1 = \mathrm{c}(x, y, ...)$, $c_2 = \mathrm{c}(z, x, ...)$, ..., $c_{n-1} = \mathrm{c}(w, u, ...)$, $c_n = \mathrm{c}(y, w, ...)$. After each agent takes one diagnosis turn, agent y is requested to diagnose on commitment c_n. Now, y cannot request a further diagnosis from agent x on c_1 since c_1 is already contained in the set of commitments that are previously diagnosed on (by Definition 18).

Property 2 ($\mathcal{M}_1$ makes a correct diagnosis) If $\mathcal{M}_1$ returns a misalignment, then a misalignment has occurred in the system. Similarly, if $\mathcal{M}_1$ returns a misbehavior, then a misbehavior has occurred in the system. In other words, $\mathcal{M}_1$'s outcome satisfies the conditions stated in Definition 12. However, the other direction is not always true. That is, if a misalignment has occurred in the system, $\mathcal{M}_1$ may return a misbehavior if it is also the case that a misbehavior has occurred prior to the misalignment. Similarly, if a misbehavior has occurred in the system, $\mathcal{M}_1$ may return a misalignment if it is also the case that a misalignment has occurred prior to the misbehavior. This is intuitive as we try to deal with the first possible reason for the exception.

We provide a notion of restricted completeness for $\mathcal{M}_1$, in the sense that our diagnosis process only identifies the first exception (either a misalignment or a misbehavior) that has possibly occurred in a chain of delegations. An alternative would be that, Definitions 13–18 would exhaustively account for every possible exception that may have occurred during the process. However, the motivation

behind our reasoning is that once an exception is identified, it does not seem interesting to look for further misalignments or misbehaviors. As a matter of fact, every other exception that occurs afterwards can be considered as a consequence of the first one. Since $\mathcal{M}_1$ identifies that first exception (i.e., the most significant one), we say that it is complete in that sense.

The conditions of Definitions 13–18 are also mutually exclusive. Therefore, the following property holds:

Property 3 ($\mathcal{M}_1$ is deterministic) Let us now discuss the outcomes of this diagnosis process in case of misbehavior or misalignment. In the case of a temporal misalignment, $\mathcal{M}_1$ returns a commitment C' which is the reason of the misalignment. If that is the only reason of violation in the system (i.e., if there are no other misbehaviors nor misalignments), a simple way to achieve realignment is the following Policy $\mathcal{P}_1$. Agents following $\mathcal{P}_1$ will align their violated commitments with the one that is presented as the outcome of the diagnosis procedure, by following these *commitment update rules*:

- *Alignment with forward-shift or extension:* If the agent has commitment c_1, and the diagnosis process has proposed a commitment c_2 which is a forward-shift or an extension of c_1, then the agent will replace its commitment c_1 with c_2.
- *Alignment with forward-shift delegation or extension delegation:* If the agent has commitment $c_1 = s_1(c(x_1, y_1, \mathit{property}(e(t_1, t_2), p_1)))$, and the diagnosis process has proposed a commitment $c_2 = s_2(c(x_2, y_2, \mathit{property}(e(t_3, t_4), p_2)))$ which is a forward-shift delegation or an extension delegation of c_1, then the agent will replace c_1 with $c_3 = s_2(c(x_1, y_1, \mathit{property}(e(t_3, t_4), p_2)))$.

The adoption of Policy $\mathcal{P}_1$ amounts to an implicit acceptance of a delayed commitment fulfillment.

The two final results below are a consequence of $\mathcal{P}_1$'s soundness, determinism and (restricted) completeness results.

Property 4 ($\mathcal{M}_1$ and $\mathcal{P}_1$ provide a means of alignment) This is true when a single misalignment has occurred in the system, and no misbehaviors have occurred prior to that misalignment. In that case, if all agents involved in the diagnosis process adopt $\mathcal{P}_1$, once $\mathcal{M}_1$ terminates and all applicable $\mathcal{P}_1$ rules have been applied, there will be no more violated commitment in the system.

Example 9 Assume that the agent has *violated*(*c*(*store*, *customer*, *property* (*e*(3.0, 8.0), *delivered*(*book*)))), and it is presented with a forward-shift delegation of this commitment, *active*(*c*(*courier*, *store*, *property*(*e*(7.0,10.0), *delivered*(*book*)))). Then, the agent will update its commitment to *active*(*c*(*store*, *customer*, *property* (*e*(7.0, 10.0), *delivered*(*book*)))) following the rule for alignment with forward-shift delegation.

Example 9 describes a case where the store makes a delegation to the courier without respecting the deadline of the customer's commitment. When such cases

occur in real life, the customer often chooses to adapt to the new deadline discovered. This is a means of *alignment* for the exception faced.

The update rules we propose provide a sample policy that agents can adapt in order to realign their commitments with other agents. This ensures that no commitment violations occur due to misalignment. Policy $\mathcal{P}_1$ can also be combined with a form of compensation, e.g., the agent will adopt the commitment update rules, in exchange for a monetary compensation. The compensation may also apply in the case of a propagated diagnosis process. For example, the customer has asked the store to diagnose its violated commitment. The store, in turn, has asked the courier for a diagnosis. As a result, assume that the courier has identified a misalignment. Now, both the store and the customer may realign their commitments regarding the courier's commitment. In such a case, who will get the compensation? This is an interesting issue that we plan to investigate when considering recovery scenarios.

Property 5 ($\mathcal{M}_1$ "finds the culprit") This is true when a single misbehavior has occurred in the system, and no misalignments have occurred. In that case, $\mathcal{M}_1$ returns an answer $\delta(X)$, $X \in \mathcal{A}$, such that there exists an alternative possible trace of events in X's execution which will lead to no violation.

7 Case study

Next, we present two separate traces of happened events from the delivery process, each leading to a different outcome of diagnosis. We assume that all agents adopt $\mathcal{P}_1$, whenever there is a misalignment issue.

7.1 Case I: misalignment

Figure 6 shows the trace of events for the first case. This case represents a misalignment of commitments between the customer (Federico) and store (Amazon) agents.

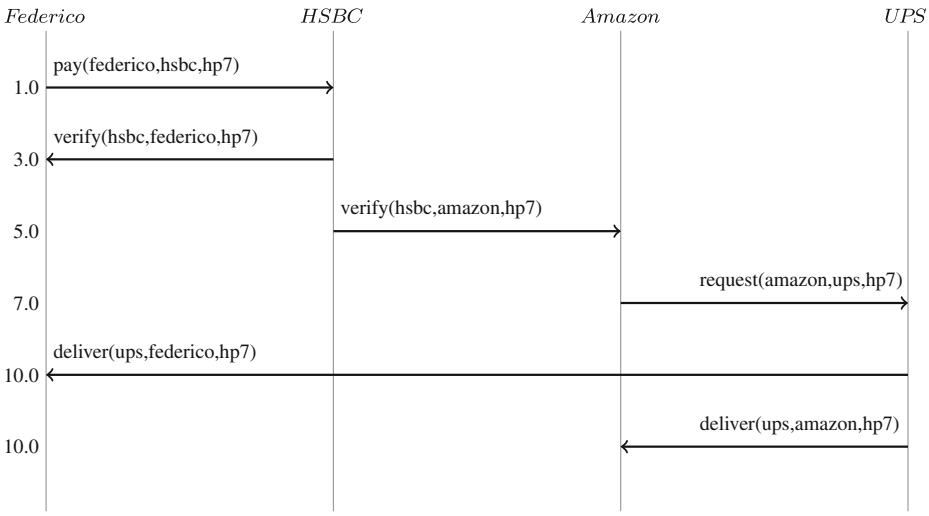


Fig. 6 Trace of events for Case I

The misalignment is caused by the late notification of the bank (HSBC) agent. Let us review the trace of events. Federico sends the payment to HSBC at day 1, regarding its purchase of *HP7* from Amazon. At day 3, HSBC verifies Federico's payment. However, HSBC sends the notification of Federico's payment to Amazon at day 5. This is where the misalignment occurs between Federico and Amazon. Recall that their commitment is triggered by HSBC's confirmation of payment. Then, Amazon requests the delivery of HP7 from the courier (UPS) agent at day 7. Finally, UPS delivers HP7 to Federico at day 10. At the same time, UPS notifies Amazon about the delivery.

Let us now track the commitments of agents in time. At day 3, Federico infers $c_1 = active\ (c(amazon,\ federico,\ property(e(3.0,\ 8.0),\ delivered\ (hp7))))$. At day 5, Amazon infers $c_2 = active\ (c(amazon,\ federico,\ property(e(5.0,\ 10.0),\ delivered\ (hp7))))$. Notice the deadline shift between those two commitments. At day 7, both Amazon and UPS infer $c_3 = active\ (c(ups,\ amazon,\ property(e(7.0,\ 10.0),\ delivered\ (hp7))))$. At day 9, when Federico runs its $\mathcal{REC}$ engine, it detects that c_1 has been violated. Figures 8 through 11 in Appendix B show the j-$\mathcal{REC}$ output for this instance.

Since there is a commitment violation, Federico initiates the diagnosis process with a diagnosis request for Amazon, i.e., by asking Amazon to find an answer δ such that

$$amazon \overset{d(c_1)}{\vDash_{\emptyset}} \delta.$$

According to the output of its monitoring facility, Amazon finds c_2, which is a (still active) forward-shift of c_1. Things could still be OK for Amazon if Amazon had delegated the task with a correct deadline. Unfortunately, this is not the case. Based on its records, Amazon verifies that an active commitment, c_3, is a forward-shift delegation of c_1. Thus, Amazon will immediately inform Federico of a misalignment following wrong delegation (see Definition 15): $\delta = misalignment(c_3)$. Federico then updates its commitment c_1 with c_3 via the alignment policy $\mathcal{P}_1$, and waits for the new deadline. At day 10, the updated commitment is fulfilled since UPS makes the delivery.

7.2 Case II: misbehavior

Figure 7 shows the trace of events for a case of misbehavior, which is diagnosed via a propagation of the diagnosis request. Let us review the trace of events. Federico sends the payment to HSBC at day 1, regarding its purchase of HP7 from Amazon. At day 3, HSBC verifies Federico's payment, and sends the notification to Amazon. Then, Amazon requests the delivery of HP7 from UPS at day 5. UPS delivers HP7 to Federico at day 10, and notifies Amazon about the delivery. This time, however, UPS should have delivered earlier. Thus, this is where UPS violates its commitment by disrespecting its deadline.

Let us track the commitments of the agents in time. At day 3, both Federico and Amazon infer $c_1 = active\ (c(amazon,\ federico,\ property(e(3.0,\ 8.0),\ delivered\ (hp7))))$. At day 5, both Amazon and UPS infer $c_2 = active\ (c(ups,\ amazon,\ property(e(5.0, 8.0),\ delivered\ (hp7))))$. Similar to the first case, Federico detects that c_1 is violated when it runs its $\mathcal{REC}$ engine at day 9.

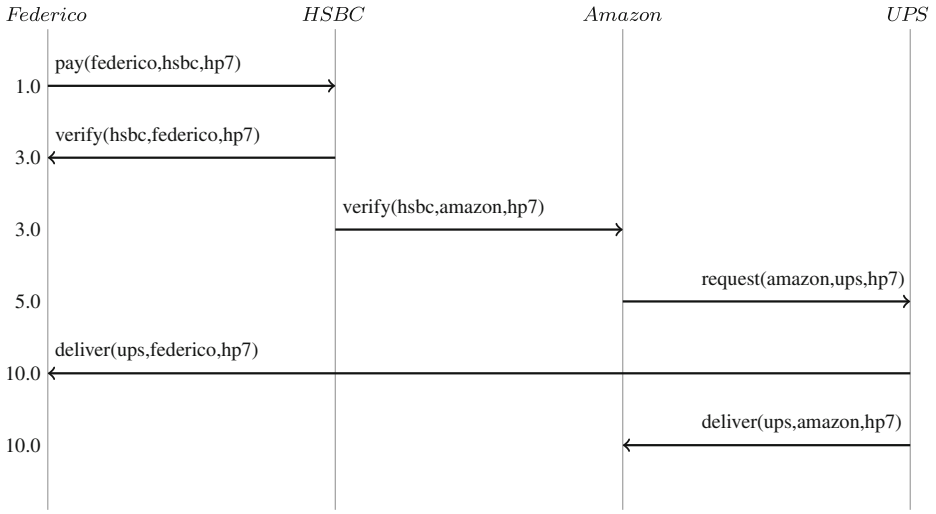


Fig. 7 Trace of events for Case II

Federico initiates the diagnosis process with the diagnosis request

$$amazon \overset{d(c_1)}{\models_{\emptyset}} \delta.$$

Accordingly, Amazon will find a violated commitment c_1, and a correct delegation c_2 of c_1, which is also violated. This eliminates all the diagnosis cases in $\mathcal{M}_1$, except the last one. Thus, Amazon redirects the diagnosis request to UPS (see Definition 18):

$$ups \overset{d(c_2)}{\models_{\{c_1\}}} \delta.$$

UPS finds that c_2 is violated. Since there is no delegation of c_2, UPS is the cause of the exception (by Definition 16). The answer $\delta = misbehavior(ups)$ is passed from UPS to Amazon, and from Amazon back to Federico. Figures 12 through 15 in Appendix B show the j-$\mathcal{REC}$ output for this instance.

8 Discussion and future work

In this paper, we have mainly studied diagnosis of exceptions when the commitments of agents are misaligned with each other. Among the set of possible causes for misalignment [16, 17, 45], we are interested in the temporal aspects. That is, we aimed at fixing misalignments that are caused by conflicts in the commitments' deadlines. We have argued that a conflict of deadlines among two relevant commitments may be caused either by individual observations of agents that are in conflict with each other (i.e., misalignment), or by a delegation that does not respect a previously established deadline (i.e., misbehavior). We have proposed commitment similarity relations that can be used to verify if two commitments are aligned in time. A more comprehensive account of delegation similarity is presented in a companion paper [27]. In the case of misalignment, the agents can update their commitments based on

the alignment policy we have proposed. Providing an update of contract deadlines is an effective way of compensation that mimics real-life situations very closely. While this constitutes one step of diagnosis, we also provide the culprit agent in the case of misbehavior.

Alignment in commitment protocols has been thoroughly investigated by Chopra et al. [13, 16, 17]. Their work accounts for constitutive protocol specifications that are regulated by commitments. In other words, once the commitments are fulfilled, then the protocol is assumed to be successful. Baldoni et al. [4], however, draw attention to the regulative side of commitment protocols. In their work, they propose (causal) rules among commitments that will further constrain an agent's behavior during the execution of a protocol.

Among others, the most similar work in commitment alignment to ours is that of Chopra and Singh's [17]. The key points regarding our formalization and theirs are:

- There are no temporal constraints on commitments in Chopra and Singh's formalization. However, without an explicit notion of time, it is hard to capture the scenarios that are presented in this paper.
- Chopra and Singh propose a *strength* relation for commitments based on their properties (conditions and propositions). They also handle asynchronous messaging and have mechanisms of cancel and release of commitment, which we do not. Currently, we consider only base-level commitments with single properties. However, we focus on the *similarity* relation for commitments since it provides a basis for verifying alignment. On one hand, the similarity relation takes into account the deadlines associated with commitments when verifying alignment in time. On the other hand, it takes into account the agents associated with commitments when tracing for delegations.
- Chopra and Singh propose a solution for misalignment by ensuring that the creditor of a commitment informs the debtor when the condition of the commitment is brought about. So, the debtor of the commitment will also infer the same base-level commitment the creditor infers. We believe that this solution may be an overkill in large-scale e-commerce applications. Under normal conditions, the execution will proceed as desired and the agents will infer the same commitments most of the times. Thus, it may be more efficient to verify alignment if something goes wrong (i.e., in the case of an exception). Moreover, when deadlines are involved, a delay in such a notification message will also cause a similar misalignment between the debtor and the creditor's individual commitments.

Agents' goals are also important when considering the commitments in a protocol [14, 43]. Agents try to manipulate (e.g., create or delegate) their commitments with each other in order to satisfy their goals. An agent's goal can be an achievement goal, or a maintenance goal [43]. In this paper, we have in a way dealt with achievement goals. That is, the existential temporal constraint on a base-level commitment corresponds to an achievement goal, in which the debtor of the commitment has to satisfy a property for the creditor within a deadline. What we have not dealt with here are maintenance goals, which can be represented by universal temporal constraints on commitments. That is, the debtor of the commitment has to ensure that the property of the commitment is valid throughout a certain period of time.

There are other settings, different from e-commerce as we experiment here, that monitoring and diagnosis is essential in identifying exceptions. Kalech and Kaminka

[29, 30] consider diagnosis of exceptions arising from disagreements due to different observations in multiagent teamwork, e.g., in robotics or in a combat-field setting. Agent death, or unreachable agents, is an important issue in such settings in the sense that an unreachable agent will obviously not participate in the diagnosis. Since we assume that our diagnosis process needs the collaboration of the agents that are involved in the mismatch, it is not possible to diagnose cases of agent death [32].

Mallya and Singh propose similarity relations for protocol states [39], not directly for commitments. However, some of the similarity relations involve the similarity of commitments in order to make reasoning possible on states. Specifically, *creditor-similarity* relation is very similar to our *delegation* relation; it aims at identifying similar states that differ only in some commitments that are delegated. On the contrary, their primary aim is completely different. They try to extend protocols via the similarity relations (e.g., merge two protocols). We, on the other hand, diagnose exceptions based on similar commitments.

In order to demonstrate how our approach works, we have extended a delivery process description [28] by involving temporal constraints, and formalized it in $\mathcal{REC}$ [8, 48]. We have designed and presented two different exception cases according to the two possible outcomes of our diagnosis procedure. The *forward-shift* relation we have proposed is the main cause of exceptions triggered by misalignment. In particular, forward-shift can further have different levels. Table 2 shows three commitments; both c_2 and c_3 are forward-shifts of c_1. For diagnosis purposes, each one is considered as misalignment. However, assume that the current time is 4.0 and the customer has the commitment c_1; it wishes to understand whether the delivery will take place at time 8.0. If the store has the commitment c_2, then the customer may think that an exception is probable. But, there is still a chance that delivery will take place at time 8.0, since 8.0 is within the temporal interval of c_2. On the other hand, if the store has the commitment c_3, then the customer can understand that there is no chance of delivery taking place at time 8.0. This type of reasoning can be used to predict exceptions, e.g., prognosis. As for future work, we plan to investigate how our similarity relations can be used for prognosis. In addition, we plan to extend our commitment similarity relation to cover the *strength* relation of Chopra and Singh. This will also allow us to investigate cases where multiple delegations are possible for the same commitment. We plan to look at the Tropos framework for complex delegation schemes [6].

The backward-shift and the backward-shift delegation relations we propose here are not operational for diagnosis purposes. Since a diagnosis process is always initiated after a commitment violation is detected, the backward-shift (delegation) of the violated commitment will also be violated, thus cannot cause a misalignment. However, when prognosis is also involved, backward-shift (delegation) may also be the cause of a misalignment. For example, the customer may not wish to receive the

Table 2 Levels of forward-shift

$c_1 = active(c(store, customer, property(e(3.0,8.0), delivered(book))))$
$c_2 = active(c(store, customer, property(e(5.0,10.0), delivered(book))))$
$c_3 = active(c(store, customer, property(e(9.0,14.0), delivered(book))))$

delivery earlier than a specific date. We will investigate such cases when we consider prognosis. Another possible direction for future work is to decide what to do next with the culprit agent identified (e.g., recovery). We are currently working on how to proceed with such diagnosis via the exchange of happened events. That is, the agents should reason both on the similarity among events and the relevance between commitments and events in order to find a suitable recovery.

Acknowledgements The first author is supported by Boğaziçi University Research Fund under grant BAP5694, and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610. We thank Marco Montali and Federico Chesani for providing us with a working implementation of j-$\mathcal{REC}$ (http://www.inf.unibz.it/~montali/tools.html#jREC), which enabled us to run experiments.

Appendix A Notation

Table 3 Notation used for diagnosis process

Symbol	Description
$\mathcal{C}$	The domain of commitments
$\mathcal{A}$	The domain of agents
$\mathcal{C}_A$	The set of commitments that agent A is aware of
$\mathcal{C}_A^C$	The set of commitments in $\mathcal{C}_A$ that are relevant to C
$\mathcal{C}_A^{Ce}$	The set of commitments in $\mathcal{C}_A$ that are an extension of C
$\mathcal{C}_A^{Cf}$	The set of commitments in $\mathcal{C}_A$ that are a forward-shift of C
$\mathcal{C}_A^{Cb}$	The set of commitments in $\mathcal{C}_A$ that are a backward-shift of C
$\mathcal{C}_A^{CX}$	The set of commitments in $\mathcal{C}_A$ that are a (proper) delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{CeX}$	The set of commitments in $\mathcal{C}_A$ that are an extension delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{CfX}$	The set of commitments in $\mathcal{C}_A$ that are a forward-shift delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{CbX}$	The set of commitments in $\mathcal{C}_A$ that are a backward-shift delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{C*X}$	$\mathcal{C}_A^{CX} \cup \mathcal{C}_A^{CeX} \cup \mathcal{C}_A^{CfX} \cup \mathcal{C}_A^{CbX}$
$\mathcal{C}_{\mathcal{A}}$	The set of all commitments ($\mathcal{C}_{\mathcal{A}} = \bigcup_{A \in \mathcal{A}} \mathcal{C}_A$)
$\mathcal{C}_{\mathcal{A}}^C$	The set of commitments in $\mathcal{C}_{\mathcal{A}}$ that are relevant to C

Appendix B Implementation and testing

A running prototype with a full implementation can be downloaded from http://mas.cmpe.boun.edu.tr/ozgur/code.html. Below we display some screenshots of the testing done on the case studied presented here, using the j-$\mathcal{REC}$ tool. The code also contains a commitment-based definition of the diagnosis method. We use commitments to model collaboration. In particular, each agent is committed to answer (faithfully) to a diagnosis request within a certain deadline. Different evolutions of commitments related to the business (as shown in the pictures below) and to the diagnosis can be tested by trying various sequences of event histories, such as those contained in each eventTrace.txt file of each agent's folder. The j-$\mathcal{REC}$ tool only needs Java. For simplicity, there is one customized version of j-$\mathcal{REC}$ Tool

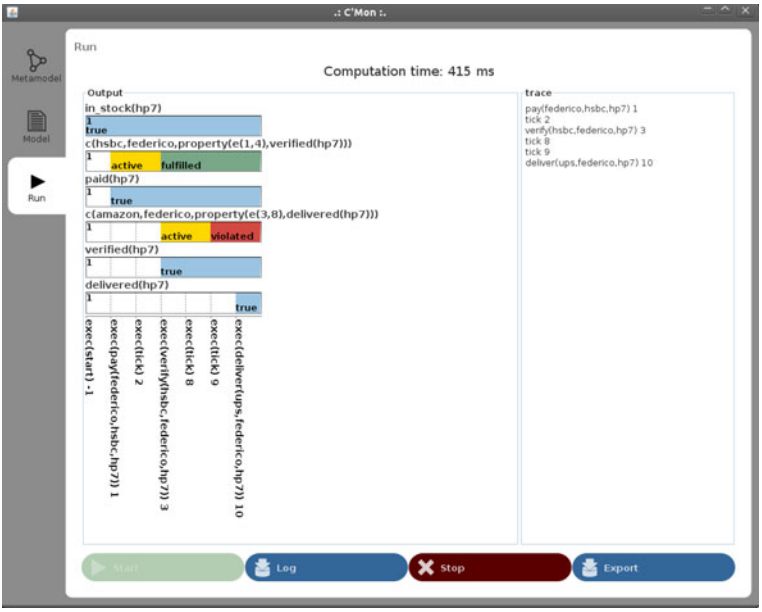


Fig. 8 Case I: j-$\mathcal{REC}$ output for customer

in each of 4 agents (federico, hsbc, amazon, and ups). The simplest way to run the example is to execute `java -jar CMon.jar` (or double-click on the `CMon.jar` file icon) on a selected agent folder.

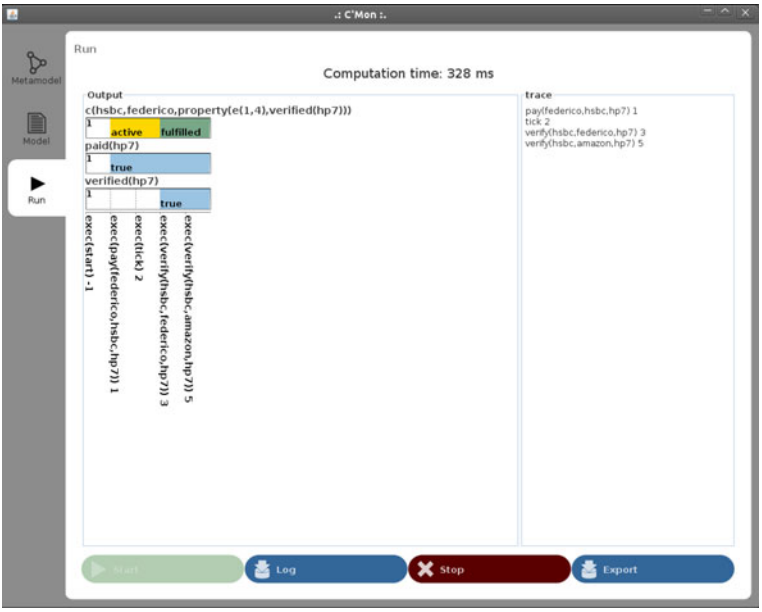


Fig. 9 Case I: j-$\mathcal{REC}$ output for bank

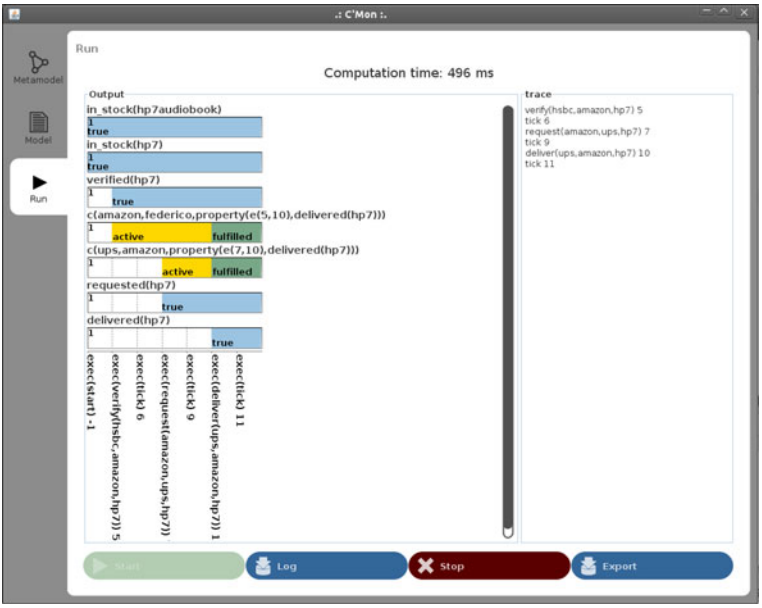


Fig. 10 Case I: j-$\mathcal{REC}$ output for Store

Below is a screenshot of j-$\mathcal{REC}$ showing a version of Case I with compensation by $\mathcal{P}_1$ following diagnosis. The compensated commitment c_1 is shown in orange color by the time Federico accepts to switch to a new commitment, with an extended deadline.

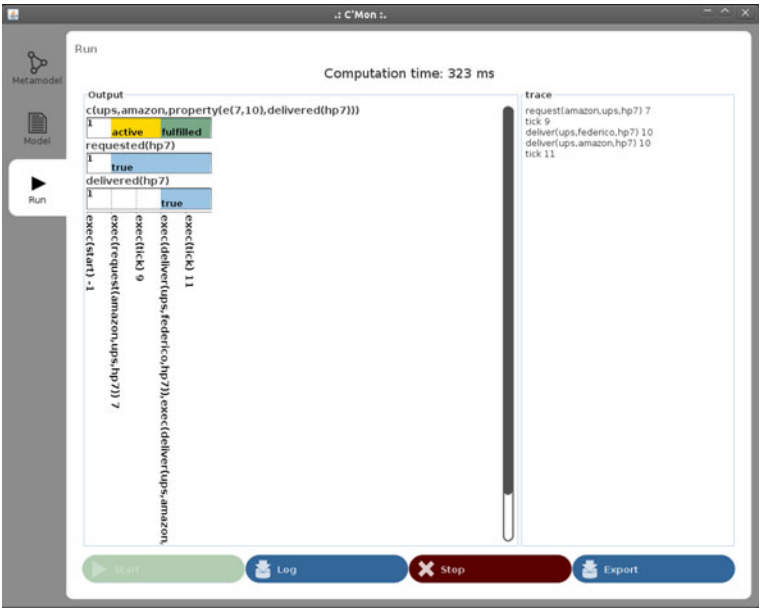


Fig. 11 Case I: j-$\mathcal{REC}$ output for courier

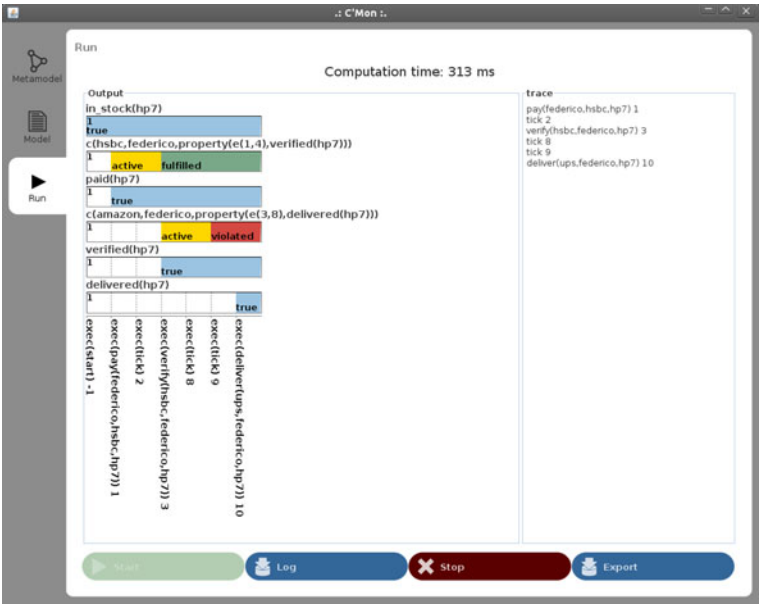


Fig. 12 Case II: j-$\mathcal{REC}$ output for customer

To run tests such as this one, select tab (Model) from the left-hand side menu and copy-paste the KB of your agent of choice (e.g., file `Customer (federico)/model.txt`). Then hit the Run, and copy-paste on the right-hand

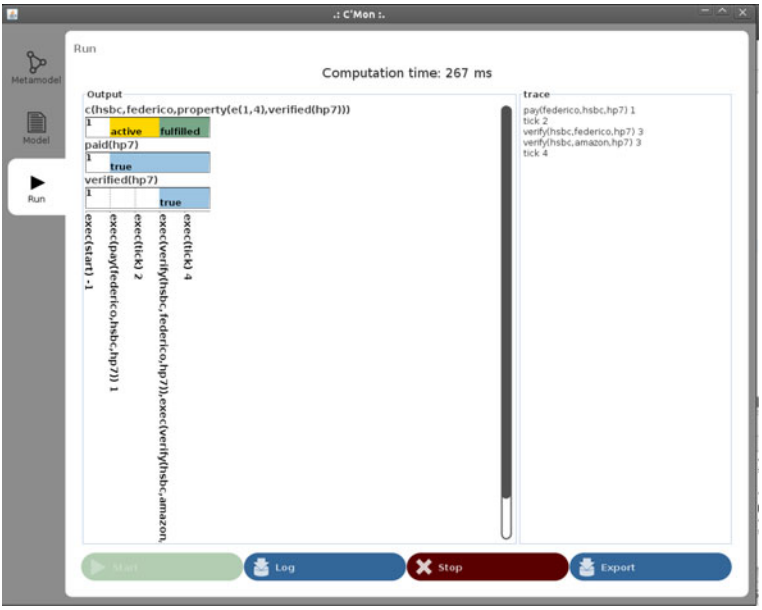


Fig. 13 Case II: j-$\mathcal{REC}$ output for bank

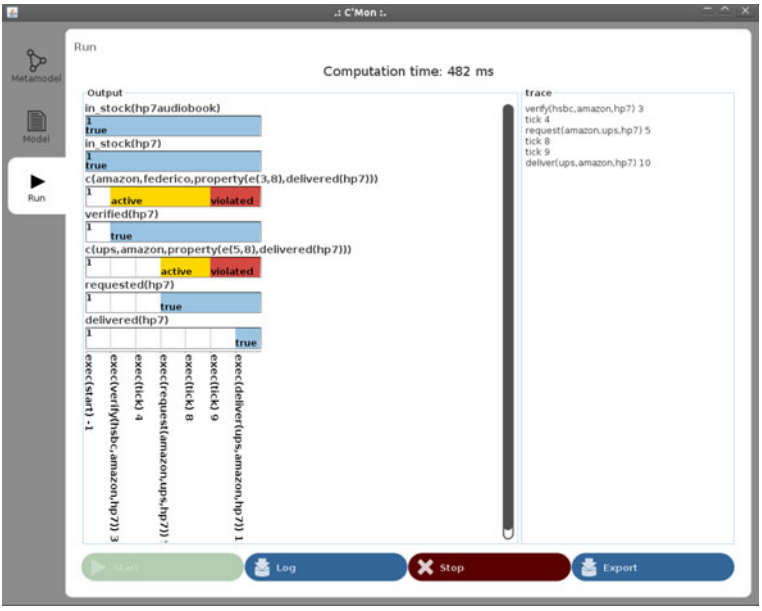


Fig. 14 Case II: j-$\mathcal{REC}$ output for store

box called trace the desired evolution of events. Once the events are in place, select Start and then Log from the bottom. Use Stop to restart and Export to save the output on a file.

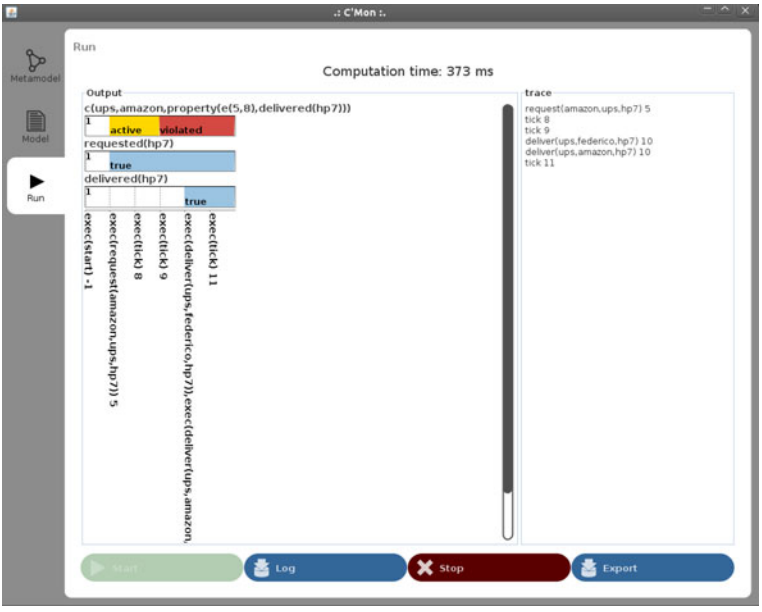


Fig. 15 Case II: j-$\mathcal{REC}$ output for courier

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Trans. Comput. Log. **9**(4), Article 29, 43 pp. (2008). doi:10.1145/1380572.1380578
2. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**, 832–843 (1983)
3. Ardissono, L., Console, L., Goy, A., Petrone, G., Picardi, C., Segnan, M., Dupré, D.T.: Enhancing Web services with diagnostic capabilities. In: Proc. 2005 IEEE International Conference on Web Services (ICWS 2005), pp. 182–191. IEEE Computer Society (2005)
4. Baldoni, M., Baroglio, C., Marengo, E.: Behavior-oriented Commitment-based Protocols. In: Proc. 19th ECAI. IOS Press (2010)
5. Bragaglia, S., Chesani, F., Mello, P., Montali, M., Torroni, P.: Reactive Event Calculus for monitoring global computing applications. In: Essays in Honour of Marek Sergot: Computational Logic for Normative Systems. LNCS, Springer-Verlag (2012)
6. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: an agent-oriented software development methodology. Auton. Agent. Multi-Ag. **8**, 203–236 (2004)
7. Bylander, T., Allemang, D., Tanner, M.C., Josephson, J.R.: The computational complexity of abduction. Artif. Intell. **49**(1–3), 25–60 (1991)
8. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus. In: Proc. 21st IJCAI, pp. 91–96 (2009)
9. Chesani, F., Mello, P., Montali, M., Torroni, P.: A REC-based commitment tracking tool. System demonstration. In: Proc. 10th WOA (2009). http://cmt.math.unipr.it/woa09/papers/Chesani_Demo.pdf
10. Chesani, F., Mello, P., Montali, M., Torroni, P.: Monitoring time-aware social commitments with reactive event calculus. In: Proc. 7th International Symposium "From Agent Theory to Agent Implementation" (AT2AI-7), pp. 447–452 (2010)
11. Chesani, F., Mello, P., Montali, M., Torroni, P.: Role monitoring in open agent societies. In: Proc. 4th KES-AMSTA, Part I. LNCS, vol. 6070, pp. 112–121. Springer-Verlag (2010)
12. Chittaro, L., Montanari, A.: Temporal representation and reasoning in artificial intelligence: Issues and approaches. Ann. Math. Artif. Intell. **28**(1–4), 47–106 (2000)
13. Chopra, A.K.: Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing. PhD Dissertation, NCSU (2009)
14. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Reasoning about agents and protocols via goals and commitments. In: Proc. 9th AAMAS, pp. 457–464 (2010)
15. Chopra, A.K., Singh, M.P.: Producing compliant interactions: Conformance, coverage, and interoperability. In: Proc. 4th DALT. LNCS, vol. 4327, pp. 1–15. Springer-Verlag (2006)
16. Chopra, A.K., Singh, M.P.: Constitutive interoperability. In: Proc. 7th AAMAS, pp. 797–804 (2008)
17. Chopra, A.K., Singh, M.P.: Multiagent commitment alignment. In: Proc. 8th AAMAS, pp. 937–944 (2009)
18. Console, L., Dressler, O.: Model-based diagnosis in the real world: lessons learned and challenges remaining. In: Proc. 16th IJCAI, pp. 1393–1400 (1999)
19. Console, L., Dupré, D.T., Torasso, P.: Towards the integration of different knowledge sources in model-based diagnosis. In: Proc. 2nd AI*IA. LNCS, vol. 549, pp. 177–186. Springer-Verlag (1991)
20. Dellarocas, C., Klein, M., Rodríguez-Aguilar, J.A.: An exception-handling architecture for open electronic marketplaces of contract net software agents. In: Proc. 2nd EC, pp. 225–232. ACM (2000)
21. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Processes = protocols + policies: a methodology for business process development. Tech. Rep., NC State University, TR2004-34 (2004)
22. Desai, N., Chopra, A.K., Singh, M.P.: Representing and reasoning about commitments in business processes. In: Proc. 22nd AAAI, pp. 1328–1333 (2007)
23. Fornara, N., Colombetti, M.: Defining interaction protocols using a commitment-based agent communication language. In: Proc. 2nd AAMAS, pp. 520–527 (2003)
24. Giordano, L., Martelli, A.: Verifying agents' conformance with multiparty protocols. In: Proc. CLIMA IX. LNCS, vol. 5405, pp. 17–36. Springer-Verlag (2009)
25. Horling, B., Benyo, B., Lesser, V.R.: Using self-diagnosis to adapt organizational structures. In: Proc. 5th Autonomous Agents, pp. 529–536. ACM (2001)

26. Jennings, N.R., Faratin, P., Norman, T.J., O'Brien, P., Odgers, B.: Autonomous agents for business process management. Appl. Artif. Intell. **14**(2), 145–189 (2000)
27. Kafali, Ö., Torroni, P.: Social commitment delegation and monitoring. In: Proc. CLIMA XII. LNCS, vol. 6814, pp. 171–189. Springer-Verlag (2011)
28. Kafalı, Ö., Yolum, P.: Detecting exceptions in commitment protocols: Discovering hidden states. In: Proc. 2nd LADS. LNCS, vol. 6039, pp. 112-127, Springer-Verlag (2009)
29. Kalech, M., Kaminka, G.A.: On the design of social diagnosis algorithms for multi-agent teams. In: Proc. 18th IJCAI, pp. 370–375 (2003)
30. Kalech, M., Kaminka, G.A.: Towards model-based diagnosis of coordination failures. In: Proc. 20th AAAI, pp. 102–107 (2005)
31. Kaminka, G.A., Tambe, M.: Robust agent teams via socially-attentive monitoring. J. Artif. Intell. Res. **12**, 105–147 (2000)
32. Klein, M., Rodriguez-Aguilar, J., Dellarocas, C.: Using domain-independent exception handling services to enable robust open multi-agent systems: the case of agent death. Auton. Agent. Multi-Ag. **7**(1–2), 179–189 (2003)
33. Klein, M., Dellarocas, C.: A knowledge-based approach to handling exceptions in workflow systems. Comput. Support. Coop. Work 9(3/4), 399–412 (2000)
34. Klein, M., Dellarocas, C.: A systematic repository of knowledge about handling exceptions in business processes. ASES Working Report. MIT (2000)
35. Kollingbaum, M., Norman, T.: A contract management framework for supervised interaction. In: UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK (2002)
36. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Gener. Comput. **4**(1), 67–95 (1986)
37. Lamperti, G., Zanella, M.: Eden: An intelligent software environment for diagnosis of discrete-event systems. Appl. Intell., **18**(1), 55–77 (2003). doi:10.1023/A:1020974704946
38. Lucas, P.J.F.: Analysis of notions of diagnosis. Artif. Intell. **105**(1–2), 295–343 (1998)
39. Mallya, A.U., Singh, M.P.: An algebra for commitment protocols. Auton. Agent. Multi-Ag. **14**(2), 143–163 (2007)
40. Micalizio, R., Torasso, P., Torta, G.: On-line monitoring and diagnosis of a team of service robots: a model-based approach. AI Commun. **19**, 313–340 (2006)
41. Pencole, Y., Cordier, M.O., Roze, L.: Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In: IEEE Conference on Decision and Control, pp. 435–440 (2002)
42. Picardi, C., Bray, R., Cascio, F., Console, L., Dague, P., Millet, D., Rehfus, B., Struss, P., Vallée, C.: Idd: Integrating diagnosis in the design of automotive systems. In: Proc. 15th ECAI, pp. 628–632. IOS Press (2002)
43. van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in agent systems: a unifying framework. In: Proc. 7th AAMAS, pp. 713–720 (2008)
44. Roos, N., Witteveen, C.: Models and methods for plan diagnosis. Auton. Agent. Multi-Ag. **19**(1), 30–52 (2009)
45. Schroeder, M., Schweimeier, R.: Arguments and misunderstandings: fuzzy unification for negotiating agents. Electr. Notes Theor. Comput. Sci. **70**(5), 1–19 (2002)
46. Singh, M.P.: Agent communication languages: rethinking the principles. IEEE Comput. **31**, 40–47 (1998)
47. Singh, M.P.: An ontology for commitments in multiagent systems: toward a unification of normative concepts. Artif. Intell. Law **7**, 97–113 (1999)
48. Torroni, P., Chesani, F., Mello, P., Montali, M.: Social commitments in time: satisfied or compensated. In: Proc. 7th DALT. LNCS, vol. 5948, pp. 228–243. Springer-Verlag (2009)
49. Torroni, P., Chesani, F., Mello, P., Montali, M.: A retrospective on the reactive event calculus and commitment modeling language. In: Proc. 9th DALT. LNCS, vol. 7169, pp. 120–127. Springer-Verlag (2012)
50. Torroni, P., Yolum, P., Singh, M.P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: Modelling interactions via commitments and expectations. In: Handbook of Research on Multi-Agent Systems: semantics and Dynamics of Organizational Models, pp. 263–284. IGI Global (2009)
51. Witteveen, C., Roos, N., van der Krogt, R., de Weerdt, M.: Diagnosis of single and multi-agent plans. In: Proc. 4th AAMAS, pp. 805–812. ACM (2005)
52. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: Proc. 1st AAMAS, pp. 527–534 (2002)