COMODO: Collaborative Monitoring of Commitment Delegations

Özgür Kafalı a, Paolo Torroni b

a School of Computing, University of Kent
Canterbury, CT2 7NF, UK
b DISI, University of Bologna
Vle Risorgimento, 2, 40136, Bologna, Italy

Abstract

Understanding accountability in contract violations, e.g., whom is accountable for what, is a tedious, time-consuming, and costly task for human decision-making, especially when contractual responsibilities are delegated among parties. Intelligent software agents equipped with expert capabilities such as monitoring and diagnosis help save time and improve accuracy of diagnosis by formal reasoning upon electronic contracts. Such contracts are represented as commitment norms, a well studied artifact in multi-agent systems, which provide semantics for agent interactions. Due to the open and heterogeneous nature of multi-agent systems, commitments are often violated. When a commitment is violated, e.g., an exception occurs, agents need to collaborate to understand what went wrong and which agent is responsible. We propose COMODO: a framework for monitoring commitment delegations and detecting violations. We define a complete set of possible rational delegation schemes for commitments, identifying for each combination of delegations what critical situations may lead to an improper delegation and potentially to a commitment violation. COMODO provides a sound and complete distributed reasoning procedure that is able to find all improper delegations of a given commitment. We provide the complete implementation of COMODO using the Reactive Event Calculus, and present an e-commerce case study to demonstrate its workings. Due to its generic nature, we discuss the application of our approach to other distributed diagnosis problems in emergency healthcare, Internet of Things and smart environments, and security, privacy, and accountability in the context of socio-technical systems.

Keywords: Agent-based commerce, Norms, Commitment delegation, Diagnosis, Computational logic

1. Introduction

A commitment describes a contract between two agents: the debtor commits to bringing about a property for the creditor (Singh, 1999). Commitments are used to give agent interaction a social semantics (Torroni et al., 2009). The idea of a social semantics is to abstract away from the agent internals and provide a social meaning to agent message exchanges. In a contract-based multi-agent system, several such commitments are in effect. Consider Amazon Prime’s
next-day delivery scheme: Amazon is committed to provide a one day delivery as long as the customer pays until noon. This can be represented by a conditional commitment:

\[ C(\text{amazon}, \text{customer}, \text{paid}(\text{customer}, \text{item}) \land \text{prime}(\text{customer}), a[1, 12], \text{delivered}(\text{item}), a[12, 36]) \]

where \( \text{paid}(\text{customer}, \text{item}) \land \text{prime}(\text{customer}) \) is the condition enabling the commitment (antecedent), and \( a[1, 12] \) specifies the absolute time window for the payment (in this example, each unit represents one hour). The conditional commitment states that, if the customer pays until noon and is an Amazon Prime customer, then \( \text{amazon} \) (the debtor) commits to the delivery of the item (\( \text{delivered}(\text{item}) \) is the consequent of this commitment) in the absolute time interval \( a[12, 36] \). Now, let’s consider another commitment where the bank is committed to verify the customer’s payment in three days after the customer initiates the payment. We can represent this commitment with a relative deadline for the consequent as follows:

\[ C(\text{bank}, \text{customer}, \text{paid}, a[\infty, \infty], \text{verified}, r[0, 3]). \]

The use of commitments to model agent interaction has been advocated especially in heterogeneous and open settings where autonomous agents must interact flexibly so as to handle exceptions and seize opportunities (Yolum & Singh, 2002). In these settings, traditional representations of protocols, such as finite state machines or AUML interaction protocol diagrams, are inadequate. One reason why commitment-based approaches are more flexible than traditional approaches is that they enable the stakeholders to delegate their commitments.

Delegation, the act of giving control or authority (e.g., a job, a duty, etc.) to another agent (Castelfranchi & Falcone, 1998; Norman & Reed, 2010), may be desirable for several reasons. For instance, agents may not be capable of satisfying the properties they are committed to bringing about. This is a very common case in e-commerce scenarios. Amazon delegates its deliveries to a courier (e.g., UPS). In our example, a delegation of delivery by Amazon (delegator) creates a new commitment:

\[ C(\text{ups}, \text{amazon}, \top, a[\infty, \infty], \text{delivered}, a[31, 45]) \]

whereby a new agent, in this case UPS (delegatee), commits to Amazon to carrying out the delivery (delegandum). Note that the absence of a time reference in the antecedent is represented by \( a[\infty, \infty] \). Here, the commitment for delivery between Amazon and the customer is extended with UPS. The customer may not be aware of this extension until the delivery is completed, or something goes wrong (e.g., the deadline passes). In that case, this connection should be revealed so that if the problem is related to UPS, it can be identified.

We call such problems exceptions in the sense that they do not account for the expected outcome of a commitment. Some causes of exceptions have been identified by related work as (i) a violation where a commitment is violated by its debtor (Singh, 1999), (ii) a misalignment where two commitments are not aligned with each other due to different observations of the participating agents (Kafalı & Torroni, 2012), or (iii) an improper delegation where a commitment is delegated without respecting the delegandum’s deadline (Kafalı & Torroni, 2011).

This paper significantly extends our previous work on commitment delegation (Kafalı & Torroni, 2011), and
provides a distributed framework, COMODO, for detecting exceptions caused by inconsistencies regarding such delegations. When there are many commitments in the system at hand, in order to identify an exception we need effective ways to explore the space of commitments. In particular, we need to identify links between commitments and exclude irrelevant instances from our search. The process of tracking individual commitment states is called commitment monitoring (Chesani et al., 2009). We extend monitoring to enable tracking down exceptions by following the links between commitments of different agents. To this end, we define a language for commitments and deadlines, which enables modeling a variety of interesting e-commerce situations. The language allows us to define properties as conjunctions of atomic propositions. Properties are associated with absolute or relative deadlines. We define a complete set of possible rational delegation schemes, identifying for each combination of delegations what critical situations may lead to improper delegations and possibly to commitment violations. A naive way of diagnosing exceptions would be to scan the set of commitments relative to a given transaction, and to look for the property that has been violated. However, this does not solve the problem if the property changes because of delegations, and if some knowledge is local to agents. Thus, we need to identify types of delegations and define a distributed algorithm that only makes use of the knowledge that is locally available to the agent. COMODO provides such an algorithm. We prove that our algorithm is sound and complete. Alongside with the theoretical results, we also provide an implementation for COMODO based on the Reactive Event Calculus. Finally, we present a case study to demonstrate its workings.

**Contributions and Implications:** Our core contribution is a fully distributed monitoring and diagnosis procedure for intelligent agents that mimics the reasoning of a human expert (Jackson, 1986) in the context of e-commerce exceptions. The knowledge base of an agent contains stateful commitments that keep track of its interactions with other agents. Using the facts contained in its knowledge base, the agent makes inference using temporal reasoning via the Reactive Event Calculus engine. Additional contributions include: (i) the extension of the commitment language presented in (Kafalı & Torroni, 2011) with deadlines for antecedents of commitments; (ii) an exhaustive list of multi-agent commitment delegation schemes; and (iii) a complete implementation of the diagnosis procedure in the Reactive Event Calculus. The proposed diagnosis procedure can be extended with an explanation capability since commitments provide semantics for agents’ interactions. Such explanations would enable human users to understand what has transcribed during a transaction and help introspection about the exception situation. Intelligent agents equipped with such expert capabilities have potential implications on several important domains regarding all phases of distributed exception handling (Kafalı et al., 2017b; Soeanu et al., 2016; Sun et al., 2012; Vasconcelos et al., 2009; Xu et al., 2011), e.g., planning, monitoring, conflict resolution, semantic reasoning, and diagnosis. While we have applied the diagnosis procedure on an e-commerce case study, its generic nature enables deployment in application areas including emergency healthcare, Internet of Things and smart environments, and security and privacy in the context of sociotechnical systems.

The rest of the paper is structured as follows. Section 2 describes our extensions to commitments. Section 3 discusses commitment delegation with a temporal analysis. Section 4 introduces COMODO’s distributed monitoring
procedure and its formal properties. Section 5 presents a case study. Section 6 reviews the relevant literature and places our contribution in the context of expert and intelligent systems. Section 7 discusses the limitations of COMODO and presents potential future directions. The Appendix provides proofs of formal properties as well as COMODO’s implementation.

2. Commitments

Commitments represent contracts or protocols (Yolum & Singh, 2002; Chopra et al., 2010). A commitment, as originally defined by Singh (Singh, 1999), is a directed obligation between two agents: the debtor commits to the creditor to bringing about a given property. Our definition of commitments extends Singh’s definition with the notion of a deadline, following a recent line of research on reasoning with commitments in time (Chesani et al., 2009; Kafalı & Torroni, 2012).

Definition 1. \( C(X, Y, Q, a[t_1, t_2], P, \gamma[t_3, t_4]) \) represents a commitment, where the debtor \( X \) commits to the creditor \( Y \) to satisfying the consequent \( P \) when the antecedent \( Q \) holds. If \( Q \) is \( \top \), then \( X \) is committed to \( Y \) unconditionally.\(^1\) When \( Q \) is not \( \top \), it is associated with the temporal constraint \( a[t_1, t_2] \). Similarly, \( P \) is associated with the temporal constraint \( \gamma[t_3, t_4] \), where \( \gamma[t_3, t_4] \) can be one of the following:

- \( a[t_3, t_4] \) defines an absolute deadline, where \( P \) has to be brought about at some point between \( t_3 \) and \( t_4 \).
- \( r[t_3, t_4] \) defines a relative deadline, where \( P \) has to be satisfied between \( t_3 \) and \( t_4 \) time units as of the time \( t \) \( Q \) gets satisfied, i.e., \( P \) has to be brought about at some point between \( t + t_3 \) and \( t + t_4 \).

In Definition 1, \( X, Y \) are agents, and \( Q, P \) are atomic (or conjunctions of) propositions. Note that the antecedent can only have an absolute deadline, and a relative deadline is only defined for the consequent when the antecedent is not \( \top \). In the remainder of the paper, we sometimes call commitments whose antecedent is \( \top \) base-level, and whose antecedent is not \( \top \) conditional (Yolum & Singh, 2002). When we discuss commitments independently of the temporal constraints, we use the simplified notation \( C(X, Y, Q, P) \). Our commitment language currently does not support negation or disjunction of propositions, nor nested commitments. The reason for this is purely pragmatic, most realistic e-commerce scenarios can be represented with conjunction, and adding negation or disjunction will reduce efficiency. When \( P \) is a conjunction of propositions, we assume that all the atomic propositions in \( P \) have the same deadline. A commitment is a live object and changes state through its life-cycle (Yolum & Singh, 2002). We make use of the following five commitment states:

- **conditional**, when \( Q \) is not yet satisfied,
- **expired**, when \( a[t_1, t_2] \) has expired and \( Q \) is not satisfied,

\(^1\)\( \top \) is a constant symbol indicating a fictitious property that does not need to be satisfied because it is already true.
• fulfilled, when $P$ is satisfied with respect to $\gamma [t_3, t_4]$.

• violated, when $\gamma [t_3, t_4]$ has expired and $P$ is not satisfied, and

• active, otherwise.

**Example 1.** Consider $C(\text{amazon, customer, paid(customer, item)} \land \text{clothing(item)}, a[\infty, \infty], \text{discount(customer, 20, clothing)}, r[0, 1])$. This conditional commitment states that if the customer purchases a clothing item, then Amazon will issue a discount for the next clothing purchase in an hour from the time the payment is made. For example, if the customer pays at time 3, this commitment will become the active base-level commitment $C(\text{amazon, customer, } \top, a[\infty, \infty], \text{discount(customer, 20, clothing)}, a[3, 4])$.

3. Delegation

When an agent $X$ is bound to a commitment $C$, it may decide to carry out the consequent (if $X$ is the debtor) or the antecedent (if $X$ is the creditor of a conditional commitment) only by itself, or by delegating $C$ in part, or in full, to other agents, which will act as subcontractors. Multiple commitments may then originate from $C$. These will be, directly or indirectly, related to $C$.

Previous work has looked at commitments and their relations from different angles. In particular, Chopra and Singh (2009) compare commitments via a **strength** relation using the commitments’ properties, whereas Kafalı et al. (2012) focus on the temporal aspects of commitments and provide similarity relations based on the commitments’ deadlines. We combine both approaches, propose direct and indirect delegation relations, and show which cases are relevant to monitoring.

**Definition 2.** A **delegation** of a commitment $C(X, Y, Q, P)$, called **primary**, is a new commitment where either $X$ or $Y$ plays the role of the creditor or debtor (**delegator**), and a new agent $Z$ (**delegatee**) is responsible for bringing about the antecedent $Q$ or part of $Q$ (for conditional commitments only), or the consequent $P$, or part of $P$. The common property between primary and delegation is called **delegandum**.

In the sequel, we use the notation debtor($C, X$) to indicate that $C$’s debtor is $X$, and delegatee($C, C_j, Y$) to indicate that the delegatee of $C$’s delegation $C_j$ is $Y$. Next, we show how different kinds of delegations are defined and combined, considering variations of $C(\text{amazon, customer, paid, delivered})$ as our **primary** ($C_{\text{prim}}$).

3.1. Basic Delegations

Basic delegations are instances of Definition 2 involving two commitments. They can be of six types: explicit, implicit, antecedent, and their duals (inverse delegations).
Definition 3. Commitment $C(Z, Y, Q', P')$ is an explicit delegation of commitment $C(X, Y, Q, P)$ iff $P \models P'$.

This type of delegation was proposed by Yolum and Singh (2002) as the result of a “Delegate” operation. A new commitment is created, whereby the new debtor is committed to the same creditor, and if $P = P'$, the primary is canceled following a “Cancel” operation (Yolum & Singh, 2002). An explicit delegation is shown in Figure 1. The new debtor UPS replaces the former debtor Amazon.

Definition 4. Commitment $C(Y, Z, Q', P')$ is an inverse explicit delegation of commitment $C(X, Y, Q, P)$ iff $P \models Q'$.

The creditor $Y$ of the primary is now the debtor of the new commitment, and $Y$ wishes to achieve $P$ (or part of it) via a new creditor $Z$. This is an inverse delegation to achieve $P$ since there is no obligation for $Z$ to satisfy $P$, still, $Z$ is motivated to satisfy $P$ if he wishes $Q$ to be satisfied. The concept of inverse delegation was introduced by Kafalı and Torroni (2011), inspired by the work of Chopra et al. (2010). An inverse explicit delegation is shown in Figure 1. Note that the roles of creditor and debtor are reversed, and accordingly the antecedent and the consequent are reversed as well.

Definition 5. Commitment $C(Z, X, Q', P')$ is an implicit delegation of commitment $C(X, Y, Q, P)$ iff $P \models P'$.

The debtor of the primary is now the creditor of a new commitment for (part of) the consequent $P$. The primary becomes dependent on the delegation, as proposed by Kafalı et al. (2012). An implicit delegation is shown in Figure 2. Note that the creditor is Amazon, which is the primary’s debtor. The primary is not cancelled, because a commitment must be kept to the former creditor (the customer).

Definition 6. Commitment $C(X, Z, Q', P')$ is an inverse implicit delegation of commitment $C(X, Y, Q, P)$ iff $P \models Q'$.

---

The semantics of $\models$ will depend on the language of the antecedent/consequent properties. In this paper for simplicity we consider properties to be conjunctions of propositions, therefore $P \models P' \iff (P = P') \lor (P = P' \land P'')$, where $P, P', P''$ are all (conjunctions of) propositions.
Figure 2: Implicit delegation. The inverse case is shown with a dashed arrow.

The debtor of the primary also becomes the debtor of a new commitment where the antecedent is (part of) the primary’s consequent. This type of delegation, as well as the next two ones (antecedent and inverse antecedent delegation), were introduced by Kafalı and Torroni (2011). An inverse implicit delegation is shown in Figure 2.

**Definition 7.** Commitment $C(Z, Y, Q', P')$ is an antecedent delegation of commitment $C(X, Y, Q, P)$ iff $Q$ is not $\top$ and $Q = Q'$.

The creditor of the primary also becomes the creditor of a new commitment for (part of) the antecedent of the primary. An antecedent delegation is shown in Figure 3. Since the former consequent (delivered) does not appear in the antecedent delegation, in order to maintain a commitment about the former consequent, the primary is not cancelled. Antecedent delegations and implicit delegations can be combined together and bind multiple commitments into causal relations (see Section 3.2). The last type of delegation we consider is the inverse variant of antecedent delegation.

**Definition 8.** Commitment $C(Y, Z, Q', P')$ is an inverse antecedent delegation of commitment $C(X, Y, Q, P)$ iff $Q$ is not $\top$ and $Q = Q'$.

The creditor of the primary is now the debtor of a new commitment whose antecedent is (part of) the antecedent of the primary. As in the previous case, the primary is not canceled. An inverse antecedent delegation of the primary is shown in Figure 3.

Definitions 3 - 8 give an exhaustive account of how a commitment can be rationally delegated, i.e., so as to preserve the responsibilities of roles in relation with the primary’s properties (Kafalı & Torroni, 2011). We denote via $\text{dlg}(C_i, C)$, that $C_i$ is a delegation of $C$.

3.2. Causal Delegations

We will now shift the focus to commitments that are linked to each other via other commitments. To this end, in (Kafalı & Torroni, 2011) we introduced the concept of commitment similarity. Here we extend similarity, in order to
capture the notion of chains of delegations. We are interested in cases where two seemingly unrelated commitments are connected to each other via a third commitment. Let us review all possible combinations of delegations described previously:

I. **Explicit delegations:** There are no possible chains with explicit delegations since the primary is cancelled in the process.

II. **Implicit delegations:** Consider the case where an implicit delegation is followed by another implicit delegation as depicted below.

\[
C_1 = C(X, Y, Q, P) \\
C_{2i} = C(Z, X, R, P) \\
C_{3i} = C(W, Z, O, P)
\]

Now, assume that \( X \) respects the deadline for \( P \) when delegating to \( Z \), but \( Z \) does not when delegating to \( W \). Eventually, this would lead to the violation \( C_1 \). But, it would also violate \( C_{2i} \). Thus, this can be identified by only looking at the individual delegation of \( C_{2i} \) to \( C_{3i} \). That is, we do not need to consider more than one implicit delegation at a time.

III. **Antecedent delegations:** Consider the case where an antecedent delegation is followed by another antecedent delegation as depicted below.

\[
C_1 = C(X, Y, Q, P) \\
C_{2a} = C(Z, Y, R, Q) \\
C_{3a} = C(W, Y, O, R)
\]

Assume that \( Y \) delegates \( Q \) to \( Z \) with a deadline relative to \( R \), and \( Y \) further delegates \( R \) to \( W \) with an absolute deadline. Now, there is no way of knowing \( C_1 \) would be violated due to \( C_{2a} \) (since it has a relative deadline) unless we take into account \( C_{3a} \). Thus, all three commitments are connected. Accordingly, we may need to consider more than one antecedent delegation at a time, in order to identify the source of an exception.

---

**Figure 3:** Antecedent delegation. The inverse case is shown with a dashed arrow.
IV. Implicit delegation followed by antecedent delegation:

\[ C_1 = C(X, Y, Q, P) \]
\[ C_{2i} = C(Z, X, R, P) \]
\[ C_{3a} = C(W, X, O, R) \]

Now, assume that \( X \) delegates \( P \) to \( Z \) with a deadline relative to \( R \), and \( X \) further delegates \( R \) to \( W \) with an absolute deadline. Similar to the antecedent delegations case, all three commitments should be considered in order to identify a problem.

V. Antecedent delegation followed by implicit delegation:

\[ C_1 = C(X, Y, Q, P) \]
\[ C_{2o} = C(Z, Y, R, Q) \]
\[ C_{3i} = C(W, Z, O, Q) \]

Again, a problem with this case can be identified by looking at two commitments at a time as in the implicit delegations case.

**Definition 9.** Commitment \( C_1 = C(X_1, Y_1, Q_1, P_1) \) is a causal delegation of commitment \( C_2 = C(X_2, Y_2, Q_2, P_2) \) via commitment \( C_3 = C(X_3, Y_3, Q_3, P_3) \) if

(a) \( P_2 \models P_3 \) and \( X_2 = Y_3 \) (implicit delegation), and \( Q_3 \models P_1 \) and \( Y_3 = Y_1 \) (antecedent delegation), or

(b) \( Q_2 \models P_3 \) and \( Y_2 = Y_3 \) (antecedent delegation), and \( Q_3 \models P_1 \) and \( Y_3 = Y_1 \) (antecedent delegation).

We call \( C_1 \) cause, \( C_2 \) outcome, and \( C_3 \) connective. The sequence of delegations from the outcome to the cause forms a causal delegation chain. Note that the number of connectives might increase making the delegation chain longer, e.g., a series of implicit delegations followed by a series of antecedent delegations. Note that the first part (series of implicit delegations) is Case II, and can be tackled by looking at commitments pairwise.

Definition 9 connects three commitments through two delegations; either (i) an implicit delegation followed by an antecedent delegation, or (ii) an antecedent delegation followed by another antecedent delegation. This allows us to trace chained commitments where the delegandum changes along the delegation chain. We account for all relations between a given commitment (primary) and its direct and indirect delegations, within the scope of a single agent.

Now, let us demonstrate a typical case for Definition 9.

**Example 2.** Consider the following set of commitments as depicted in Figure 4:

\[ C_{2,1} = C(amazon, customer, paid, delivered) \]
\[ C_{2,2} = C(ups, amazon, deliveryFee, delivered) \]
\[ C_{2,3} = C(bank, amazon, account, deliveryFee) \]
According to $C_{2.1}$ ($C_{\text{prim}}$), once the customer pays, Amazon will have the goods delivered. Now, Amazon delegates the delivery to UPS via $C_{2.2}$. However, in order to deliver, UPS needs payment for delivery. Amazon makes another delegation to the bank for payment via $C_{2.3}$. Thus, Amazon’s delivery ($C_{2.1}$, outcome) is now connected via $C_{2.2}$ (connective) to bank’s payment ($C_{2.3}$, cause), which is an example of case (a) in Definition 9.

For each commitment whose antecedent or consequent is a conjunction of properties, there may be more than one delegation. We thus obtain a delegation tree. We can trace all delegations of a given commitment by exhaustive search of the delegation tree.

3.3. Delegation Tree

A delegation tree is a set of connected delegations. This is formally described in the next two definitions.

Definition 10. A delegation chain $\sigma = C_1, \ldots, C_j \subseteq C$ is a set of commitments such that $\forall i, 1 < i \leq j, C_i$ is a direct\(^3\) delegation of $C_{i-1}$. The first element is called the root of the chain.

Definition 11. A delegation tree $\tau = (V, A)$ is a tree, whose nodes are commitments, $V \subseteq C$, such that for every edge $(C_i, C_j) \in A, C_j$ is a direct delegation of $C_i$.

3.4. Temporal Analysis

We will now enrich the relations we have defined so far, by taking into account temporal constraints. We seek to identify and understand the reasons behind exceptional situations that can lead to faulty behaviour. To this end we will define cases of delegations where the deadline of the primary is not properly propagated onto the delegation. We will use the term improper to label a delegation whose deadline exceeds the primary’s deadline, and that can possibly

\(^3\)A direct delegation is one of the cases described in Definitions 3 - 8.
cause a mismatch between the satisfaction conditions believed by the delegator and delegatee. We will consider the overall system as it is observed at a specific time, the time of observation. First, we describe how two time intervals are compared.

**Definition 12.** Let $t$ be the time of observation. An interval $I_1 = \gamma_1 [t_1, t_2]$ exceeds an interval $I_2 = \gamma_2 [t_3, t_4]$, iff either of the following holds:

- $I_1$ is absolute, $I_2$ is absolute, and $t_1 < t_3$ or $t_2 > t_4$,
- $I_1$ is absolute, $I_2$ is relative, and $t_1 < t_3 + t$ or $t_2 > t_4 + t$,
- $I_1$ is relative, $I_2$ is absolute, and $t + t_1 < t_3$ or $t + t_2 > t_4$,
- $I_1$ is relative, $I_2$ is relative, and $t_1 < t_3$ or $t_2 > t_4$.

Next, we use the notion of exceeding intervals to define improper delegations.

**Definition 13.** Let $C_{id} = C(Z, X, Q', \ldots, P', T')$ be an implicit delegation of $C_{prim} = C(X, Y, Q, \ldots, P, T)$. $C_{id}$ is an improper consequent delegation of $C_{prim}$ iff $T'$ exceeds $T$. The inverse case is defined similarly.

**Example 3.** Consider the following set of commitments:

- $C_{3.1} = C(amazon, customer, paid, a[1, 12], discount \land delivered, a[31, 45])$
- $C_{3.2} = C(office, amazon, \top, a[\infty, \infty], discount, [31, 31])$
- $C_{3.3} = C(ups, amazon, \top, a[\infty, \infty], delivered, a[35, 50])$

Now, discount has been delegated correctly, since $C_{3.2}$’s deadline does not exceed that of $C_{3.1}$. $C_{3.3}$ instead is an improper delegation, whose deadline exceeds that of $C_{3.1}$. Note that the occurrence of an exception is not inevitable, since UPS may still complete delivery before time 45. However, $C_{3.3}$ creates a vulnerability, and may be the root of future exceptions.

**Definition 14.** Let $C_{ad} = C(Z, X, Q', \ldots, P', T')$ be an antecedent delegation of $C_{prim} = C(X, Y, Q, T, P, \ldots)$. $C_{ad}$ is an improper antecedent delegation of $C_{prim}$ iff $T'$ exceeds $T$. The inverse case is defined similarly.

**Definition 15.** Let $\sigma$ be a delegation chain rooted in $C$, and let $C_j \in \sigma$ be a (direct or causal) delegation of $C_i \in \sigma$. Let $I = \gamma[t_s, t_e]$ be $C$’s interval and $I_i = \gamma_i [t_{i,s}, t_{i,e}]$ be $C_i$’s interval, for each $C_i$ in $\sigma$. Let $t$ be the time of observation. $C_j$ is an improper causal delegation of $C$ iff either of the following holds:

- $I$ and all $I_i$ are relative, and $\sum_{i=0}^{j} t_{i,s} < t_s$ or $\sum_{i=0}^{j} t_{i,e} > t_e$,
- assuming $I_k$ is the last absolute deadline in $\sigma$, and $t_k$ is the time $C_k$’s property is satisfied

---

4We omit the deadline interval in the commitment with ’’ when it is not relevant to the discussion.
Symbol | Description
--- | ---
δ_{pro} | set of proper delegations of a given commitment
δ_{imp} | set of improper delegations of a given commitment
C = \{⟨C, δ_{pro}, δ_{imp}⟩, \ldots\} | for each known commitment C, a triplet consisting of: C, its proper delegations δ_{pro}, and its improper delegations, δ_{imp}
X ∨REC C | C contains all (locally) known information about X’s commitments, extracted by a REC reasoner such as ComMon
δ_{exc} | commitments to be excluded from a monitoring process
δ_{out}, δ_j, δ_k | output of monitoring processes (sets of improper delegations)
X ▷_{δ_{exc}} C | the result of a monitoring process issued by X about C and excluding δ_{exc}, is the set of improper delegations δ_{out}
X ▷_{δ_{out}} Y ≫ C | the result of a monitoring process requested by X to agent Y about C and excluding δ_{exc}, is the set of improper delegations δ_{out}

| Table 1: Notation for COMODO’s monitoring process. |

**Definition 16.** Let C, C_j ∈ C. C_j is an improper delegation of C, denoted dlg_{imp}(C_j, C), iff

- \( I \) is absolute and \( t_k + \sum_{i=k+1}^{l} t_{i,s} < t_s \) or \( t_k + \sum_{i=k+1}^{l} t_{i,e} > t_e \), or
- \( I \) is relative and \( t_k + \sum_{i=k+1}^{l} t_{i,s} < t + t_s \) or \( t_k + \sum_{i=k+1}^{l} t_{i,e} > t + t_e \).

**Definition 17.** A monitoring process \( M \) is a process whose inputs are

- **C** is an improper consequent delegation of C, or
- **C** is an improper antecedent delegation of C, or
- **C** is an improper causal delegation of C, or
- \( \exists C_k \in C \) such that \( dlg(C_k, C) \) and \(dlg_{imp}(C_j, C_k)\).

A delegation which is not improper is called a proper delegation and denoted by \(dlg_{prop}(C_j, C)\) (meaning that \( C_j \) is a proper delegation of C).

4. Monitoring

In this section, we describe COMODO’s distributed monitoring procedure. At a given time of observation \( t \), a monitoring process \( M \) records all the improper delegations \( M_t \), that occurred up to \( t \).

**Definition 17.** A monitoring process \( M \) is a process whose inputs are
• a set \( \mathcal{A} \) of agents,

• a set \( \mathcal{C} \) of commitments among agents in \( \mathcal{A} \),

• a narrative \( \mathcal{T}_t \) of events up to a given time of observation \( t \), and

• a commitment model and domain knowledge defining the states of commitments in \( \mathcal{C} \) based on \( \mathcal{T}_t \),

and whose output is \( \mathcal{M}_t = \{(C_i, C_j) | \text{dlg}\_\text{imp}(C_i, C_j)\} \), where \( C_i, C_j \in \mathcal{C} \).

The purpose of monitoring is to identify all the exceptions (e.g., improper delegations) among agents’ commitments at a given time of observation, considering the available knowledge. Note that \( \mathcal{M} \) is an abstract concept, as we cannot assume that there is always an agent who has complete global knowledge. We use this to demonstrate that our distributed monitoring procedure produces the same output as the global monitoring process.

Typically, a monitoring process starts from a specific commitment \( C_m \) whose delegations need to be analysed. For example, \( C_m \)’s creditor \( X \) might want to check the situation with \( C_m \) some time before \( P \)’s (the property of \( C_m \)) deadline expires, in order to prevent potential problems. So, \( X \) will ask \( Y \)’s collaboration (as the debtor of \( C_m \)). Accordingly, \( Y \) will run a local monitoring process about \( P \), and report back to \( X \). The initial commitment about \( P \) may in turn be linked to a number of other commitments, thus originating a chain of commitments, possibly involving additional agents, other than \( X \) and \( Y \).

We refer to a narrative \( \mathcal{T}_t \) of events to trace a protocol execution. In particular, the successful completion of a given action by a given agent will be represented by a particular event in \( \mathcal{T}_t \). We do not model action duration, but only completion. \( \mathcal{T}_t \) contains all the elements that describe a specific protocol execution up to time point \( t \).

### 4.1. Distributed Monitoring

We will now describe the distributed monitoring procedure that agents follow to detect improper delegations. The monitoring procedure is a derivation process, described by the local rules \( L_1 \) and \( L_2 \) (intra-agent reasoning) and the social rules \( S_1 - S_3 \) (inter-agent reasoning). Table 1 summarizes the notation.

Given an agent \( X \in \mathcal{A} \) and a commitment \( C_m \in \mathcal{C} \), a derivation \( X \triangleright_{\delta_{\text{out}}}^\theta C_m \) starts when \( X \) decides to monitor one of its commitments \( C_m \). The \( \theta \) symbol (which is an input to the derivation) signifies that no commitment is initially excluded from the monitoring process, because no commitment has been analysed yet. The output \( \delta_{\text{out}} \) is a set of improper delegations, which might be empty in some cases. The monitoring procedure may propagate from agent to agent, as described by the social rules. As commitments get analysed by the agents involved in the monitoring process, they are included in the set \( \delta_{\text{exc}} \) when performing further derivation. In this way, we prevent agents from analysing the same commitment more than once. In a concrete implementation, answering to a monitoring request could be implemented as a background agent behaviour, whereas issuing a monitoring request could be implemented by a communicative act from an agent \( X \) to an agent \( Y \), that implements the “answering to a monitoring request” behaviour. A possible architecture for distributed monitoring is described in (Kafali & Torroni, 2012), where observations are
local to the agent, and commitment-based contract specifications are instead shared, i.e., accessible to both the debtor and the creditor of each commitment.

Each agent involved will only use its local knowledge of commitments to contribute to the derivation by applying local and social monitoring rules. Part of the local reasoning amounts to checking which commitments are linked to the subject commitment, via proper or improper delegations. This is defined in the \( \mathcal{REC} \) language, assuming that for commitment tracking purposes each agent relies upon tools such as ComMon. However, in the general case, the delegation check could be done by using any procedure that queries a local database of commitments.

**Local monitoring:** These are the rules used for monitoring the agent’s commitments locally. They describe intra-agent reasoning, which is based on the agent’s local knowledge base (i.e., own commitments and fluents). This is performed via the agent’s internal \( \mathcal{REC} \) engine, for which the details will be given in the implementation part (see Section 4.2).

\[
L_1) \quad X \triangleright \mathcal{REC} \ C \land (C_m, \delta_{pro}, \delta_{imp}) \in C \land \delta_{out} = \delta_{imp} \setminus \delta_{exc} \land \delta_{out} \neq \emptyset \quad \Rightarrow \quad X \triangleright \delta_{out} \ C_m
\]

By rule \( L_1 \), if the agent identifies any improper delegations of the currently monitored commitment \( C_m \) via querying its \( \mathcal{REC} \) engine locally, and these commitments are not already contained in \( \delta_{exc} \) (the set containing the commitments that are already processed during monitoring), then they are added to the output (\( \delta_{out} \)) of the monitoring process. Consider the commitments in Example 3: let \( X \) be Amazon, \( C_m \) be \( C_3 \), and \( \delta_{exc} = \emptyset \). Now, when Amazon queries their \( \mathcal{REC} \) engine, he will find out that \( \delta_{pro} = \{C_3, 2\} \) and \( \delta_{imp} = \{C_3, 3\} \). Thus, \( \delta_{out} = \{C_3, 3\} \) which contains the only improper delegation of \( C_3, 1 \).

\[
L_2) \quad X \triangleright \mathcal{REC} \ C \land (C_m, \delta_{pro}, \delta_{imp}) \in C \land (\delta_{pro} \cup \delta_{imp}) \setminus \delta_{exc} = \emptyset \land \text{debtor}(C_m, X) \quad \Rightarrow \quad X \triangleright \delta_{exc} \ C_m
\]

By rule \( L_2 \), if there are no locally known delegations of the monitored commitment \( C_m \), and \( X \) is \( C_m \)'s debtor, the result is an empty set. This rule complements \( L_1 \), and is a termination condition for some branches of the distributed monitoring process, when there are no more delegations left in the corresponding delegation chain for the subject commitment.

**Social monitoring:** These rules describe how the derivation process propagates from one agent to another agent, and how the results are combined. They describe the inter-agent reasoning, which is based on the monitoring interactions (e.g., requests and responses) among the agents. In the following rules, we use the notation \( \triangleright \), whose semantics is given in Table 1, to indicate a request for monitoring.

---

\(^5\)\( \mathcal{REC} \) (Reactive Event Calculus) is an event calculus-based language and reasoning framework (Chesani et al., 2009). ComMon is a \( \mathcal{REC} \)-based monitoring engine that can be downloaded from [http://ai.unibo.it/projects/comMon](http://ai.unibo.it/projects/comMon).
By rule $S_1$, if there is a locally known proper delegation $C_j$ which is not to be excluded ($C_j \in \delta_{pro} \setminus \delta_{exc}$), $X$ delegates monitoring to $C_j$’s delegatee $Y$, thereby obtaining a result $\delta_j$. $X$ will then continue monitoring its other delegations, excluding $C_j$ from the process, thereby obtaining a result $\delta_k$. The final result $\delta_{out}$ is the union of the two partial results, $\delta_j \cup \delta_k$.

By rule $S_2$, if there is no locally known delegation of the monitored commitment $C_m$, $C_m$’s creditor $X$ makes a monitoring request to $C_m$’s debtor $Y$, and the result $\delta_{out}$ is provided by $Y$ as the response.

By rule $S_3$, an agent $Y$ answers to $X$’s request for monitoring concerning a given commitment $C_m$ by executing a monitoring process about $C_m$ and propagating the result back to $X$.

This procedure relies on local reasoning and collaboration among agents to produce monitoring results that, ideally, should be equivalent to the global results produced by the abstract monitoring process $M$. Under the assumption that the $REC$ reasoner provides sound and complete results, we can prove the following theorems (see Appendix A):

**Theorem 1 (Soundness).** Given a commitment $C_m \in C_T$, and an agent $X \in A$, if $X \vdash_{\delta_{out}} C_m$ and $C_i \in \delta_{out}$, then $(C_i, C_m) \in M_T$.

By Theorem 1, if the distributed monitoring process identifies an exception in the form of an improper delegation $C_i$ of a given commitment $C_m$, then $(C_i, C_m)$ is an outcome of the global monitoring (see Definition 17).

**Theorem 2 (Completeness).** $\forall (C_i, C_m) \in M_T, \exists$ an agent $X \in A$ and a derivation $X \vdash_{\delta_{out}} C_m$ such that $C_i \in \delta_{out}$.

By Theorem 2, for any two given commitments $C_i, C_m \in C_T$, if $C_i$ is an improper delegation of $C_m$, then there is a possible run of the distributed monitoring process starting from some agent $X$ that identifies it as such.

### 4.2. Implementation

We have provided an implementation for COMODO. We have written specifications in the $REC$ language, and utilised ComMon for monitoring of commitments. More specifically, the input to ComMon is the following:
• a commitment theory that contains the rules for manipulation of commitments,

• a domain model that contains the protocol rules that describe the agents’ domain,

• an event trace that contains the actions of the agents throughout time.

Given these inputs, ComMon produces an outcome that displays the agents’ fluents through time. This is used to monitor the individual states of the commitments at run-time. Moreover, we have defined a subset of the commitment relations introduced in this paper in the REC language, thus extending the commitment model with a delegation model and an exception model, in order to accommodate local reasoning. Additional details on the implementation can be found in Appendix B.

5. Case Study

Let us now use Amazon’s Prime next-day delivery scheme\(^6\) to demonstrate how COMODO works. We have the following three commitments to represent the process for the customer to order an item from Amazon:

• \(C_1 = C(amazon, customer, paid \land prime, a[1, 12], delivered, r[12, 36])\): Amazon must deliver the client’s order within the following day,

• \(C_2 = C(ups, amazon, packaged, a[1, \infty], delivered, r[6, 24])\): When the item is packaged, UPS can deliver it in the next 24 hours,

• \(C_3 = C-office, amazon, confirmed, a[1, \infty], packaged, r[6, 24])\): Confirmed orders are packaged in the next 24 hours.

Note that the customer only knows about the first commitment \(C_1\). In addition, the following two actions are known to the customer:

• \(pay(customer, amazon) \rightarrow paid\).

• \(deliver(ups, customer) \rightarrow delivered\).

The semantics of the actions given by the above rules is that when the action on the left-hand side is executed, then the fluent on the right-hand side holds. For the sake of simplicity, we assume that action executions are successful and their effects are independent of the context. Now, consider the following trace of events:

12 pay(customer, amazon)
17 confirm(amazon, office)
30 package(office, amazon)

That is, customer pays for the item at noon. Amazon confirms the order at 5 pm, and the item is packaged next morning at 6 am. The following commitments are in place at time 30:

\(^{6}\)www.amazon.com/prime
$C_1 = C(\text{amazon, customer, } T, a[\infty, \infty], \text{delivered, } a[24, 48])$

$C_2 = C(\text{ups, amazon, } T, a[\infty, \infty], \text{delivered, } a[36, 54])$

$C_3 = C(\text{office, amazon, } T, a[\infty, \infty], \text{packaged, } a[23, 41])$

Notice the pattern among these three commitments; $C_2$ is an implicit delegation of $C_1$ (Definition 5), and $C_3$ is an antecedent delegation of $C_2$ (Definition 7). Then, $C_3$ is a causal delegation of $C_1$ via $C_2$ (Definition 9).

First, we look at the global monitoring result considering all the commitments in the system. Assume that no delivery has occurred until time 48. $C_1$ is indeed violated since its deadline has passed. Because of the causal delegation, $C_2$ and $C_3$’s deadlines together affect that of $C_1$. Even though the packaging of the item is completed at time 30, UPS has 24 hours for delivery, which will eventually exceed $C_1$’s deadline. If the delivery is completed at time 54, $C_2$ is fulfilled. However, $C_1$ is still violated. Here, Amazon should have confirmed customer’s order earlier, or set a tighter deadline for $C_2$. Next, we look at the agents’ local reasoning:

- **Customer:** $C = \{\{C_1, \{\}, \{\}\}\}$
  - Rules $L_1$, $L_2$, and $S_1$ do not apply,
  - Rule $S_2$ delegates to Amazon.

- **Amazon:** $C = \{\{C_1, \{\}, \{C_2, C_3\}\}, \ldots\}$
  - Rule $L_1$ applies, and finds an improper delegation,
  - Rule $S_3$ propagates the result to the customer.

Now, let us change the trace of events so that the protocol will not lead to any improper delegations. Consider the following trace:

12 pay(customer, amazon)
13 confirm(amazon, office)
16 package(office, amazon)

Notice that the confirmation of the order is performed earlier in this case, which leads to an earlier deadline for packaging ($C_3$). This also affects the delivery of the item which depends on packaging $C_2$. Figure 5 shows the output of the ComMon tool for Amazon and the customer’s reasoning. The horizontal axis shows the timeline of events that have occurred during execution. Alongside such events, we inserted additional tick events, whose only purpose is to force ComMon to update and display the state of commitments and fluents at every time point. The commitments and fluents are shown alongside the vertical axis together with how their states change over time. Note that we omit antecedent deadlines for brevity.
Figure 5: ComMon output for Amazon’s Prime next-day delivery.

6. Related Work

We have presented a commitment-based approach to support automated reasoning in expert systems, where intelligent agents employ distributed monitoring and diagnosis to resolve exceptions caused by commitment delegations. Commitments are a specific type of norm to regulate interactions among various stakeholders in sociotechnical systems (Barth et al., 2006; Singh, 2013; Hao et al., 2016; Kafalı et al., 2016, 2017a,b; V asconcelos et al., 2009). In the rest of this section, we first summarize the strengths and limitations of relevant approaches from the literature (including COMODO) in Table 2, and then we review each approach in more depth.

A good deal of related work on commitments investigates formal properties of commitments (Lorini, 2010; Singh, 2008; Khan & Lespérance, 2006; Verdicchio & Colombetti, 2003), temporal extensions of commitment languages
<table>
<thead>
<tr>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chesani et al. (2013); Kafali &amp; Torroni (2012)</td>
<td>Temporal commitments</td>
<td>Centralised approach</td>
</tr>
<tr>
<td>Sun et al. (2012); Xu et al. (2011)</td>
<td>Domain ontologies; Automated negotiation</td>
<td>Focused on e-commerce domain</td>
</tr>
<tr>
<td>Ajmeri et al. (2016); Gunay &amp; Yolum (2013); Vasconcelos et al. (2009)</td>
<td>Conflict resolution; Generalizable to all norm types</td>
<td>Centralised approach</td>
</tr>
<tr>
<td>Al-Saqqa et al. (2016); El-Menshawy et al. (2013); Herd et al. (2015)</td>
<td>Formal verification of protocol properties</td>
<td>Design-time approach</td>
</tr>
<tr>
<td>Castelfranchi &amp; Falcone (1998); Norman &amp; Reed (2010); Falcone &amp; Castelfranchi (2001); Santos et al. (1997)</td>
<td>Attribution of responsibility and accountability; Foundational concepts; Theoretical analysis</td>
<td>No practical application explored</td>
</tr>
<tr>
<td>Gao &amp; Singh (2014)</td>
<td>Automated extraction of norms from business contracts Both practical and dialectical commitments</td>
<td>No exception handling or diagnosis; Design-time approach</td>
</tr>
<tr>
<td>COMODO</td>
<td>Distributed monitoring and diagnosis; Generic and generalizable process; Run-time approach; Flexible agent execution</td>
<td>Single application domain; Single norm type</td>
</tr>
</tbody>
</table>

Table 2: Comparison of COMODO with relevant literature.

(Fornara & Colombetti, 2004; Mallya et al., 2004; Venkatraman & Singh, 1999), normative models (Antoniou et al., 2009; Kafali et al., 2017a; Vasconcelos et al., 2012), and norm monitoring and planning (Alechina et al., 2016; Fornara & Colombetti, 2010; Gasparini et al., 2016; Meneguzzi et al., 2015; Spoletini & Verdicchio, 2009). The temporal extensions to commitment languages and monitoring procedures proposed in (Chesani et al., 2009, 2013) are used in this paper for local reasoning. Temporal constraints are also used in (Kafali & Torroni, 2012) to compare commitments. This approach, as well as others, mainly focuses on the relations between pairs of commitments in two-agent interactions. In (Chopra & Singh, 2015), delegation is also taken into account from the perspective of
commitment alignment, but no temporal aspects are considered.

Agent-based approaches have been utilized for monitoring, detection, and handling of exceptions. Kafali and Yolum (2016) propose an approach for monitoring an agent’s interactions to determine whether the agent is progressing as expected. In particular, they verify whether the agent’s expectations (represented by a set of propositions and commitments) are satisfiable by its current state. Sun et al. (2012) propose a distributed environment for e-procurement processes, where each agent is assigned to different a task such as search, negotiation, monitoring, or exception handling. Xu et al. (2011) propose a taxonomy of logistics exceptions based on previous work in the literature. They differentiate among potential problems related to deliveries such as late or partial delivery, and explore dependencies between all logistics components. It would be interesting to apply COMODO on these domains as well as extend our delivery scheme with a domain ontology.

Günay and Yolum (2013) discuss the feasibility of a set of commitments, i.e., whether it is possible for an agent to honor all its (existing and prospective) commitments. They formulate feasibility as a constraint satisfaction problem. Vasconcelos et al. (2009) propose methods for resolving conflicts among norms. Their resolution method, norm curtailment, manipulates the constraints associated with norms, e.g., reduce the scope of a prohibition to avoid conflict with an obligation. Ajmeri et al. (2016) propose Coco, a formalism to express and reason about conflicting commitment instances at runtime, and dominance among them. Coco employs Answer Set Programming to compute nondominated commitment instances and uses Alechina et al.’s (2013) framework to determine compliance of actions with nondominated commitment instances. Compared to Coco, while we do not explicitly deal with conflicts among commitment delegations, a detected violation during monitoring indicates a potential conflict.

In the general context of commitment frameworks, not addressing monitoring, many authors considered the use of commitments to represent, model and verify protocols. Among them, El-Menshawy et al. (2013) propose several methods for commitment-based protocol verification using model checking. Similar verification based approaches (Al-Saqqar et al., 2016; Herd et al., 2015) are performed at design time. We focus instead on run-time verification. We do not elaborate here on how to use our analysis at design time, although that may be a possible application.

The concept of commitment delegation has been proposed by Singh and Yolum (2002). The delegation mechanism gives great flexibility to commitment-based protocols. However, it also lays itself open to misuse and may induce possible mismatches among agent beliefs about deadlines associated with properties. Improper delegations eventually drive the system into a state of violation, where some agents believe that there has been no violation at all. In this work, we presented an in-depth analysis of improper delegations, and proposed an effective distributed reasoning procedure for finding all improper delegations of a given commitment.

Santos et al. (1997) investigate aspects of organised interaction and propose a characterisation of “transmission of agency” (a concept related to delegation and attribution of responsibility) and the analysis of the conditions under which a given organisation recognises that an agent has fulfilled his responsibilities. Castelfranchi and Falcone (1998; 2001) develop a theory of delegation and adoption, using a plan-based approach. The authors’ perspective is more general than that of Santos et al. It does not necessarily target an institutionalised environment, but it considers task
and delegation to be foundational concepts of agency and autonomy in the broader perspective. In any case, when an agent delegates a task to another agent, the latter takes care of the interests of goals of the former remotely, i.e., far from it, and without its monitoring and intervention (control). According to Castelfranchi and Falcone, in delegation, an agent A tries to achieve some of its goals through another agent B’s actions, thus A has the goal that B performs a given action. Delegation and adoption are thus characterised in terms of mental states of the agents involved in the interaction. The authors distinguish between “weak” adoption, i.e., based on spontaneous initiative, and “strict” adoption, i.e., accompanied by a formal agreement (contract). Both Santos et al. and Castelfranchi and Falcone stress the fundamental importance of expressing agent behaviour without referring to concrete actions when delegating, as a basis for flexibility (“open” delegation in (Castelfranchi & Falcone, 1998)). The relationship between openness and control is further explored in (Falcone & Castelfranchi, 2001), where Falcone and Castelfranchi propose a theory of adjustable autonomy, where trust is the cognitive basis for adjusting autonomy.

Gelati et al. (2004) provide a formal analysis of the idea of normative coordination, in the belief that the adoption of a normative perspective would allow a substantial progress in the creation of agent societies. Agents can achieve flexible co-ordination by conferring normative positions to other agents. The building blocks of their analysis are declarative power (the capacity of the power-holder of creating normative positions by proclaiming such positions), representation (the representative’s capacity of acting in the name of its principal) and mandate (the mandator’s duty to act as the mandator has requested). Norms are also investigated in agent-based supply-chain environments (Vasconcellos et al., 2012), and conflicts are discussed in the form of exceptions. Moreover, agents are used to resolve exceptions in e-procurement systems, where several agents enact different roles (Sun et al., 2012). Another use for agents is discussed in (Chen & Nof, 2012), where agents detect and prevent errors in sequential production lines. Governatori (2013) proposes a conceptual abstract framework to model normative requirements, formalizes different types of obligations, and verifies whether a business process is compliant with requirements (set of obligations). Integrating delegations into the above approaches would be an interesting directions to pursue.

Several authors, including Lorini et al. (2007; 2009) and Norman and Reed (2010), proposed a semantic characterisation of delegation in relation with agent mental states such as beliefs and intentions (Lorini et al., 2007), with the semantics of speech acts (Longin et al., 2009), and with the concept of responsibility (Norman & Reed, 2010).

In contrast to these approaches, we do not follow a normative perspective, and we do not make any formal reference to the foundational concepts above. We refer instead to the intuitive notion of responsibility and accountability, to justify the links that may bind two commitments together. In particular, we will say that when a commitment between two agents X and Y is delegated, a third agent (the delegatee), will be responsible for bringing about a property derived from the initial commitment (the delegandum). To the best of our knowledge, there are no other works in the literature that propose ways to reason about chains of commitment delegations and identify (potentially) problematic situations. Our work also has a practical interest. We show how the monitoring process can be implemented using efficient, off-the-shelf tools. We are not aware of other works that cover both theoretical and practical aspects of the problem we address.
Verifying agent executions against commitment specifications (or interaction protocols in general) has been the focus of recent research, both at design-time as well as run-time. Gao and Singh (2014) propose a method for extracting contracts from actual business relationships. Contracts are represented as both practical and dialectical commitments. In this paper, we only focus on practical commitments since our focus is on e-commerce protocols. Kafalı et al. (2014) propose a distributed algorithm to verify at run-time whether the goals of the agent will be satisfied via its commitments. They make use of temporal constraints and implement their work with REC like we do here. However, their commitment relations are basic compared to our extensive study of commitment delegations. Abushark et al. (2014) also focus on detecting exceptions in the form of defects. However, this is not designed for run-time detection as we do here. They compare agent designs with protocol specifications, and aim to help agent developers to reduce defects in design.

7. Discussion

In this paper, we have built upon previous work (Kafalı & Torroni, 2011), where we discuss a systematic classification of commitment delegation types using a simple commitment language. There, we have used motivating examples inspired from an e-commerce scenario, to show that delegation can follow meaningful patterns, other than the traditional way of delegating commitments proposed in the literature (Yolum & Singh, 2002). Moreover, we have introduced the concept of similarity, improper delegation, and monitoring process. In COMODO, we have extended the language for commitments by introducing relative and absolute deadlines represented as time intervals. We have further explored the concepts of similarity and improper delegation by giving an exhaustive account of all possible improper delegations, and we have provided a sound and complete distributed reasoning procedure that is able to find all improper delegations of a given commitment.

Limitations

• In this work, we have focused on a specific type of norm, commitment, which is a dominant artifact in e-commerce contracts. However, in other domains such as healthcare, authorization and prohibition norms are commonly used to represent regulations.

• We have not evaluated the expressiveness of our commitment language in multiple contract domains. While the deadline conditions we introduced are helpful in representing e-commerce contracts, extending the language with disjunction and maintenance properties would further increase expressiveness. Maintenance properties can be related with a maintenance goal (Chesani et al., 2009), where a certain property should hold at all times during a specified interval. Moreover, temporal constraints such as those proposed by several languages for temporal representation and reasoning (e.g., before, after, until) may indeed be useful in some applications (Marengo et al., 2011). However, this would significantly increase the complexity of our temporal reasoning agents.
• We have demonstrated the working of COMODO on one case study from e-commerce, which constitutes a threat to external validity. Other works on normative models explore emergency healthcare (Kafalı et al., 2016, 2017a), and security and privacy (Barth et al., 2006; Kafalı et al., 2017b). Exploring norm delegations in such settings would provide valuable insight to our distributed diagnosis procedure.

Implications

• All phases of distributed exception handling (Soeanu et al., 2016; Sun et al., 2012; Vasconcelos et al., 2009; Xu et al., 2011), e.g., planning, monitoring, conflict resolution, and diagnosis, are crucial capabilities for any expert system that deals with private and confidential information. Intelligent agents should detect and resolve inconsistencies that arise from their interactions using partial information and without violating their users’ privacy. In this work, we focused on an e-commerce application by representing electronic contracts with commitments, which provide flexible execution for software agents. Apart from the e-commerce domain, our distributed monitoring and diagnosis process can be adopted in safety-critical domains such as intrusion detection (Geib & Goldman, 2001) and other crime detection (Jarvis et al., 2005) by integrating it with additional AI-based methods. Commitments and additional normative representations can be combined with ontologies and semantic reasoning to provide additional expert capabilities to agents (Kafalı et al., 2017b; Xu et al., 2011).

• Explainable AI is a great concern in existing machine learning applications. The diagnosis process we have proposed can be extended with additional explanation capabilities since commitments add semantics to agents’ interactions, and help human users understand exception situations. Intelligent agents equipped with such expert capabilities have potential implications on several other important domains including emergency healthcare, Internet of Things and smart environments, and security and privacy in the context of sociotechnical systems.

Future Work

• In principle, some of the notions that we introduced for the purpose of run-time monitoring could also be used for auditing or at design-time. For example, it may be useful to introduce design constraints, or guarantee mechanisms to prevent agents from causing improper delegations. We do not deal with design issues here, but as a future work it would be interesting to study the application of the improper delegation notion in contexts other than monitoring, or monitoring in the planning domain (Soeanu et al., 2016). Moreover, extending commitments with sanctions (Nardin et al., 2016) would add another dimension to COMODO’s delegation monitoring procedure. Sanctions provide compensation for commitment violations, therefore act as deterrence against violating commitments.

• It would be interesting to elaborate on how agents use the outcome of monitoring, e.g., in order to detect inconsistencies, contradictory facts or inappropriate situations, as well as to investigate the integration of the agent’s monitoring capability with other reasoning capabilities in concrete agent architectures.
• Our diagnosis process can be extended with other AI-based approaches such as plan recognition (Kautz, 1987), goal recognition (Lesh & Etzioni, 1995), and intention recognition (Sadri, 2012) to extend the application domain beyond e-commerce. Such recognition approaches often have implementations in the Event Calculus (EC), therefore we can seamlessly integrate those into the COMODO framework.

References


Appendix A. Proofs

Proof of Theorem 1

Given a commitment $C_m \in C_T$, and an agent $X \in A$, if $X \triangleright_{\delta_{out}}^0 C_m$ and $C_i \in \delta_{out}$, then $(C_i, C_m) \in M_T$.

Proof 1. We prove soundness by contradiction. Given an agent $X$ and two commitments $C_i, C_m$, assume that $X \triangleright_{\delta_{out}}^0 C_m$ and $C_i \in \delta_{out}$, and $(C_i, C_m) \notin M_T$. We have the following possibilities:

• Rule $L_1$ applies, and $C_i$ is an improper delegation of $C_m$. $M_T$ finds it since both $C_m$ and $C_i$ are members of $C_T$. Thus, $(C_i, C_m) \in M_T$. We reach a contradiction.

• Rule $S_1$ applies. There is no improper delegation of $C_m$, but other (proper) delegations exist.

• Rule $S_2$ applies. There are no delegations of $C_m$.

Note that Rule $L_2$ does not hold initially since the monitoring result is not empty. The first case immediately ends with a contradiction while the second and third cases propagate monitoring to other agents. At some point, delegations of $C_m$ cease to exist\(^7\). Let $Z$ be the last agent that delegates $C_m$. Let us review each monitoring case:

• Rule $L_2$ does not apply since $\delta_{out}$ cannot be empty.

• Rule $S_2$ does not apply since $Z$ delegated $C_m$.

• Assume rule $L_1$ applies, and $C_j$ is an improper delegation of $C_i$ (which is a delegation of $C_m$). $M_T$ finds it since $C_m, C_i$ and $C_j$ are all members of $C_T$. Thus, $(C_j, C_m) \in M_T$. We reach a contradiction.

• Assume rule $S_1$ applies, and let the delegatee be $W$. For $W$, let us review each monitoring case:

  – Rule $L_1$ does not apply since $\delta_{imp}$ is empty.
  – Rule $S_1$ does not apply since $\delta_{pro}$ is empty.
  – Rule $S_2$ does not apply since the debtor of the commitment $C_m$ is $W$ itself.
  – Assume rule $L_2$ applies. This results in an empty set for the monitoring result. However, $\delta_{out}$ cannot be empty. We reach a contradiction.

This demonstrates that every possible case of local monitoring, that identifies an exception, leads to a contradiction against the global monitoring process not identifying that exception, thus proving soundness.

\(^7\)Since protocol trace time is fixed, infinite delegations cannot occur.
Proof of Theorem 2

\[ \forall (C_i, C_m) \in \mathcal{M}_T, \exists \text{ an agent } X \in \mathcal{A} \text{ and a derivation } X \triangleright \mathcal{C}_\text{out} C_m \text{ such that } C_i \in \delta_{\text{out}}. \]

**Proof 2.** We prove completeness by contradiction. Let us consider two commitments \( C_i \) and \( C_m \), such that \((C_i, C_m) \in \mathcal{M}_T\) and \[ \forall X, \delta_{\text{out}} X \triangleright \mathcal{C}_\text{out} C_m \text{ and } C_i \notin \delta_{\text{out}}. \] Then, by Definition 17 \( \text{dlg}_{\text{imp}}(C_i, C_m) \). Let debtor(Cm, X). The following cases are possible (Definition 16):

- \( C_i \) is an improper consequent delegation of \( C_m \), or
- \( C_i \) is an improper antecedent delegation of \( C_m \), or
- \( C_i \) is an improper causal delegation of \( C_m \), or
- \[ \exists C_j \in \mathcal{C} \text{ such that } \text{dlg}(C_j, C_m) \text{ and } \text{dlg}_{\text{imp}}(C_i, C_j). \]

In the first three cases, \( X \triangleright_{\mathcal{REC}} (C_m, \delta_{\text{pro}}, \delta_{\text{imp}}) \in \mathcal{C} \text{ and } C_i \in \delta_{\text{imp}}. \text{ Then by rule } L_1, X \triangleright \mathcal{C}_\text{out} C_m \text{ and } C_i \in \delta_{\text{out}}. \text{ Now, assume } X \triangleright \mathcal{C}_\text{out} C_m \text{ where } C_i \notin \delta_{\text{out}}. \text{ If } X \text{ is the debtor of } C_m, \text{ then } C_i \text{ is also a commitment of } X. \text{ When } X \text{ queries the } \mathcal{REC} \text{ reasoner, } < C_m, \delta_{\text{pro}}, \{C_i, \ldots\}> \in \mathcal{C}. \text{ Thus, rule } L_1 \text{ applies with } C_i \in \delta_{\text{out}}. \text{ We reach a contradiction.} \]

In the last case where there is a delegation chain rooted in \( C_m \) and that includes \( C_j \), let \( Y \) be the delegatee or debtor of \( C_j \) and \( Y \neq X \). When \( Y \) is the delegatee, then rule \( S_1 \) applies. When \( Y \) is the debtor, then rule \( S_2 \) applies. Both cases lead to rule \( S_3 \). Rule \( S_3 \) recursively starts a new derivation starting from \( Y \). By iterating the same reasoning, we eventually reach the case where \( C_j \) is a direct delegation of \( C_m \) (since the delegation chain is finite). Thus, rule \( L_1 \) applies as above. We reach a contradiction.

This demonstrates that every possible case of global monitoring, that identifies an exception, leads to a contradiction against the local monitoring process not identifying that exception, thus proving completeness.
Appendix B. Implementation

We have provided an implementation for COMODO using the ComMon tool. Below we explain some important code segments\(^8\) from the case study presented in Section 5. The ComMon tool only needs Java. The simplest way to run the example is to execute \texttt{java -jar ComMon.jar} (or double-click on the ComMon.jar file icon) on a selected agent folder.

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. Then hit the Run, and copy-paste on the right-hand 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.

Now, we describe parts of the REC code. Listing 1 shows the commitment theory that is shared by all the agents. First, the states of the commitments are described. Note that, in addition to the four states described in Section 2, we have detached to describe a conditional commitment that has become active. This is for implementation purposes so that we do not lose track of origin of the active commitment (i.e., the original condition commitment). Then, the rules that describe the state transitions are defined. Following the Event Calculus, in REC, we can express that an event initiates (or terminates) a temporal fluent, by way of initiates(Event, Fluent, Time) relations. A commitment with its state is considered a temporal fluent.

Listing 2 shows the rules that describe the domain model of Amazon. This covers most of the process, and the domain models for other agents are described similarly. First, an exception is described either as a direct improper delegation, or an indirect improper delegation. Then, the rules for fluent manipulation are given in terms of action-consequence relations. For example, a payment from the customer to Amazon initiates the fluent paid at the time of the event. The rules for contract execution are given in terms of commitment create operations. For example an offer from Amazon to the customer creates a conditional commitment between the two agents regarding the Prime delivery scheme. Note that this exact rule is also contained in the customer’s domain model as they share this commitment. However, not all such rules are in the customer’s domain model, e.g., the details of the transaction between Amazon and the office is omitted from the customer.

Here, we also support conjunction of fluents for the consequents of commitments. If the consequent of a commitment is a conjunction of fluents, then we represent it as a Prolog list, which contains all the fluents that are elements of the conjunction. We describe how delegations with conjunctions are handled below.

Listing 3 shows the rules that describe explicit delegation, which is based on the discussion in Section 3.2. Other delegation types are described similarly. Delegations with conjunction of fluents is handled by parsing the list of fluents that make up the conjunction. Note that the deadline intervals are not taken into consideration while describing the delegation similarity relations. The description for improper (causal) delegation is given in Listing 4 by taking into consideration the deadline intervals of the commitments (see Section 3.4).

\(^8\)The complete implementation can be downloaded from \url{http://mas.cmpe.boun.edu.tr/ozgur/code.html}, Section 3.
Listing 1: Commitment theory.

% commitment states
conditional(C, T):- holds_at(status(C, conditional), T).
detached(C, T):- holds_at(status(C, detached), T).
active(C, T):- holds_at(status(C, active), T).
fulfilled(C, T):- holds_at(status(C, fulfilled), T).
violated(C, T):- holds_at(status(C, violated), T).

% create as conditional or active
initiates(E, status(C, conditional), T):- ccreate(E, C, T).
initiates(E, status(C, active), T):- create(E, C, T).

% conditional to active
terminates(E, status(C1, conditional), T):- detach(E, C1, C2, T).
initiates(E, status(C1, detached), T):- detach(E, C1, _, T).
initiates(E, status(C2, active), T):- detach(E, _, C2, T).
detach(E, c(Tc, X, Y, Q, _, P, r(T1,T2)),
     c(Tc, X, Y, true, _, P, a(T3,T4)), T):-
     conditional(c(Tc, X, Y, Q, _, P, r(T1,T2)), T),
     initiates(E, Q, T), T3 is T + T1, T4 is T + T2.

% active to fulfilled
terminates(E, status(C, active), T):- discharge(E, C, T).
initiates(E, status(C, fulfilled), T):- discharge(E, C, T).
discharge(E, c(Tc, X, Y, true, _, P, a(T1,T2)), T):-
     active(c(Tc, X, Y, true, _, P, a(T1,T2)), T),
     T =< T1, T =< T2, initiates(E, P, T).

% active to fulfilled
terminates(E, status(C, active), T):- violate(E, C, T).
initiates(E, status(C, violated), T):- violate(E, C, T).
violets(_, c(Tc, X, Y, true, _, P, a(T1, T2)), T):-
     active(c(Tc, X, Y, true, _, P, a(T1, T2)), T),
     T > T2.
Listing 2: Domain model (Amazon).

% exception model
initiates(\_, exception(C1, C2), T):-
    holds_at(improperDelegation(C1, C2), T).
initiates(\_, exception(C1, C2), T):-
    holds_at(improperDelegation(C1, C), T), active(C2, T), delegation(C, C2).

% fluent manipulation
initiates(exec(pay(customer, amazon, Item)), paid(Item), \_).
initiates(exec(confirm(amazon, office, Item)), confirmed(Item), \_).
initiates(exec(package(office, amazon, Item)), packaged(Item), \_).
initiates(exec(deliver(ups, customer, Item)), delivered(Item), \_).

% contract execution
ccreate(exec(offer(amazon, customer, Item)),
    c(T, amazon, customer, paid(Item), delivered(Item), r(12,36)), T):-
        prime(customer).
ccreate(exec(offer(ups, amazon, Item)),
    c(T, ups, amazon, packaged(Item), delivered(Item), r(6,24)), T).
create(exec(confirm(amazon, office, Item)),
    c(T, office, amazon, true, packaged(Item), rel(6,24)), T).
Listing 3: Delegation model.

% cases of explicit delegation
explicitDelegation(c(Tc1, Z, Y, true, _, P1, _), c(Tc2, X, Y, true, _, P2, _)):-
    partOf(P1, P2), Tc1 > Tc2, X \= Z.

explicitDelegation(c(Tc1, Z, Y, _, _, P1, _), c(Tc2, X, Y, true, _, P2, _)):-
    partOf(P1, P2), Tc1 > Tc2, X \= Z.

explicitDelegation(c(Tc1, Z, Y, true, _, P1, _), c(Tc2, X, Y, _, _, P2, _)):-
    partOf(P1, P2), Tc1 > Tc2, X \= Z.

explicitDelegation(c(Tc1, Z, Y, _, _, P1, _), c(Tc2, X, Y, _, _, P2, _)):-
    partOf(P1, P2), Tc1 > Tc2, X \= Z.

% conjunction
partOf(P, P).
partOf(P, [P|_]).
partOf(P, [ _|L]):- partOf(P, L).
partOf([P|L1], L2):- partOf(P, L2), partOf(L1, L2).

Listing 4: Improper delegation.

% improper causal delegation
initiates(_, improperDelegation(
    c(Tc3, X3, Y3, true, _, P3, a(T5, T6)),
    c(Tc1, X1, Y1, true, _, P1, a(T1, T2))))(T):-
    active(c(Tc1, X1, Y1, true, _, P1, a(T1, T2)), T),
    conditional(c(Tc2, X2, Y2, Q2, _, P2, r(T3, T4)), T),
    active(c(Tc3, X3, Y3, true, _, P3, a(T5, T6)), T),
    implicitDelegation(c(Tc2, X2, Y2, Q2, _, P2, r(T3, T4)),
        c(Tc1, X1, Y1, true, _, P1, a(T1, T2))),
    antecedentDelegation(c(Tc3, X3, Y3, true, _, P3, a(T5, T6)),
        c(Tc2, X2, Y2, Q2, _, P2, r(T3, T4))),
    (T4 + T6) > T2.