AUTOMATED REASONING ON EXCEPTIONS IN COMMITMENT-BASED MULTIAGENT SYSTEMS

by

Remzi Özgür Kafalı

B.S., Computer Engineering, Boğaziçi University, 2004

M.S., Systems & Control Engineering, Boğaziçi University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2012

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Assoc. Prof. Pınar Yolum for her continuous support and guidance on making this thesis come through. I would also like to thank my committee members Prof. H. Levent Akın, Prof. Yaman Barlas, Prof. Oğuz Dikenelli, and Assist. Prof. Alper Şen for their invaluable comments.

I would like to thank Paolo Torroni from Bologna University and Francesca Toni from Imperial College London for their contributions in this thesis. I would also like to thank Marco Montali and Federico Chesani from Bologna University for providing us with a working implementation of $\mathcal{REC}$, which enabled us to run experiments. I would also like to thank my colleagues in the Multiagent Systems research group in Boğaziçi University for their comments in our group discussions.

My dear parents have always been supportive for my choices in life. Although I've spent the second half of my life abroad, their consistent visits helped me feel at home all the time. My brother and I share common interests in life, which is a rare blessing to have. Although we have our differences (e.g., I play Terran, he plays Zerg, and I prefer Les Pauls, he prefers Stratocasters), at the end of the day it is a great feeling to accomplish something in common. I am also very thankful to his wife for putting up with our fierce discussions in our StarCraft sessions.

There are certain people that have kept my spirit up while I was struggling with my research. I would like to thank Tasteless and Artosis for their excellent GSL casts, and MarineKingPrime for his entertaining games and amazing micro skills.

The research done in this thesis is supported by Boğaziçi University Research Fund under grant BAP5694, The Scientific and Technological Research Council of Turkey under Program 2219, and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610.

ABSTRACT

AUTOMATED REASONING ON EXCEPTIONS IN COMMITMENT-BASED MULTIAGENT SYSTEMS

Exceptions constitute a significant portion of people's lives. When things do not go as planned, due to environmental reasons or because one does not bring about his responsibility in a given task, unexpected situations occur. When faced with exceptions, people need to deal with them in a timely fashion in order to restore proper working. However, dealing with exceptions is not an easy task for people to accomplish. First, it requires understanding that something has gone wrong (*detection*). Second, the actual source of the problem needs to be identified (*diagnosis*). Moreover, in some situations, identifying that an exception will possibly occur in the future helps changing the course of previously planned actions in order to avoid the exception (*prediction*). Accordingly, this thesis proposes to use agents for automating the reasoning on exceptions. We model the problem domains with open multiagent systems, and use commitments to formalize agent interactions. We propose automated methods based on computational logic for detecting, predicting, and diagnosing exceptions. We prove that our methods are sound and complete. We study our methods on two domains, online social networks and e-commerce, which exhibit different characteristics for the exceptions that may arise in them. Our specific contributions in this thesis are threefold. First, we extend the scope of detected exceptions in the literature such that an exception is not limited to a commitment violation. Second, we provide a prediction system based on model checking that identifies exceptions before they even occur. Finally, we investigate the temporal relations among commitments in order to diagnose what has gone wrong during an agent's execution.

ÖZET

TAAHHÜT TABANLI ÇOK ETMENLİ SİSTEMLERDE İSTİSNAİ DURUMLAR ÜZERİNE OTOMATİK AKIL YÜRÜTME

İstisnai durumlar insanların hayatında önemli bir yer oluşturmaktadır. Bir kişi göreviyle ilgili üzerine düşeni yapmadığından, veya bulunulan ortamdan kaynaklanan nedenlerden dolayı işler planlandığı gibi gitmediğinde, beklenmedik durumlar oluşur. İnsanların istisnai durumlarla karşılaştıklarında normal çalışmalarına geri dönebilmeleri için bunlarla başa çıkmaları gerekir. Fakat bu, insanlar için başarması kolay bir iş değildir. Öncelikle, bir şeylerin yanlış gittiğini anlamalıdırlar (*tespit*). Daha sonra, sorunun neden kaynaklandığını bulmalıdırlar (*teşhis*). Bunlara ek olarak, bazı durumlarda ileride istisnai bir durum oluşacağını önceden belirlemek bu durumu önlemek için gerekli adımların atılmasına yardımcı olur (*öngörü*). Bunlara dayanarak, bu tezde etmenlerin istisnai durumlar üzerine otomatik olarak akıl yürütmelerini öneriyoruz. Problem alanlarını çok etmenli sistemler olarak modelleyip, etmenler arasındaki etkileşimi taahhütler ile şekillendiriyoruz. İstisnai durumlarla başa çıkabilmek için mantık tabanlı otomatik metodlar sunuyoruz. Bu metodların geçerliliğini ve bütünlüğünü ispatlıyoruz. Bu metodları, oluşabilecek istisnai durumlar açısından farklı karakteristikleri olan sosyal ağlar ve e-ticaret alanlarında inceliyoruz. Bu tezdeki katkılarımız üç tanedir. İlk olarak, literatürde yer alan istisnai durumların kapsamını sadece taahhüt ihlali ile sınırlı kalmayacak şekilde genişletiyoruz. İkinci olarak, istisnai durumları daha oluşmadan fark edebilecek model denetleme tabanlı bir öngörü sistemi sunuyoruz. Son olarak, taahhütler arasındaki zamansal ilişkileri bir etmenin işleyişinde neyin yanlış gittiğini teşhis edebilmek için inceliyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathcal{C}$	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
δ_{pro}	set of proper delegations of a given commitment
δ_{imp}	set of improper delegations of a given commitment
δ_{exc}	commitments to be excluded from a monitoring process
δ_{out}, δ_j, δ_k	output of monitoring processes (sets of improper delegations)

LIST OF ACRONYMS/ABBREVIATIONS

ABA	Assumption-Based Argumentation
ComMon	Commitment Monitoring tool
KGP	Knowledge, Goals and Plans model of agency
$\mathcal{PROTOSS}$	$\mathcal{PR}$ivacy vi$\mathcal{O}$la$\mathcal{T}$ions in $\mathcal{O}$nline $\mathcal{S}$ocial network$\mathcal{S}$
$\mathcal{REC}$	$\mathcal{R}$eactive $\mathcal{E}$vent $\mathcal{C}$alculus

1. INTRODUCTION

Exceptions are part of every day life. Things may go wrong or may not comply with what is expected in a given situation. People face unexpected situations at home while browsing the Internet, when they are driving their cars during the rush hour, or at work while doing business with others. Generally, if things are not going as planned in carrying out a task, one might conclude that there is an exception. Some exceptions can be identified easily. For example, if you cannot access your e-mails, you might conclude that there is a problem with your mail client. However, not all exceptions are identified immediately. If your car breaks in the middle of the road, you may not know for sure what has caused it to stop working.

People need to deal with exceptions in order to continue working towards their goals. In particular, they first need to understand that something is wrong if they sense an interference in their proper working. Understanding that an exception has occurred is important, but not sufficient since one cannot return back to work unless the exception is resolved. Some exceptions are harder to resolve. When many people are involved in a single joint task, an exception gets harder to deal with. Consider a Web-based software application which searches through patients given some search criteria and lists their details. Three people work on the development of this software. A Web developer creates the user interface, a database administrator manages the database, and a software engineer builds the middleware that connects the database and the user interface. Assume that the Web developer senses a delay in the retrieval of search results. Now, the problem might be related to several issues; (i) a network connection issue affecting the software to connect to the database, or (ii) a database query that is not optimized, or (iii) a database table that is not indexed correctly to deal with the query, or lastly (iv) any combination of these. The developers need to collaborate to find out the underlying reason.

In this thesis, we focus on two domains with different characteristics in terms of exceptions that may arise in them. The first domain is online social networks (OSNs).

In an OSN, several users interact with each other, and they share content through the network. Describing which users will access this content determines a user's privacy. Therefore, a privacy exception here corresponds to a content being accessed by a user that it is not explicitly aimed for. The second domain is e-commerce, where several parties do business together to reach their goals, either individual or joint. These business protocols are usually governed by contracts, and exceptions in e-commerce are associated with contracts being violated. Examples 1.1 and 1.2 describe a scenario for each domain, respectively.

Example 1.1. *Consider a document sharing network where users can share private content with their friends. The network operator has a separate privacy agreement with each user stating that their content will remain private unless they share it with a friend. We have three users for this network; Charlie, Sally, and Linus.*

Example 1.2. *Consider an online bookseller that delivers books on demand. A customer, Ali, logs on to the online store of the bookseller, purchases a book of his choice, and enters his delivery information. The bookseller then processes Ali's order, and delivers his book to his specified address using a courier.*

Now, let us see what might go wrong in these settings. First, consider the privacy domain. Assume that Charlie shares his CV with Sally as he needs her help organizing it. If Sally also asks for help from Linus, then Linus will access Charlie's CV even though Charlie has not shared it with Linus. This is an exception for Charlie. How can Charlie detect this exception, i.e., how can he understand that his CV might be visible to Linus through this network?

Second, consider the e-commerce domain. Ali orders a book from the bookseller, and the bookseller informs Ali that the book will be delivered in five days by the courier. If the book does not arrive after five days has passed, then Ali will understand that there is something wrong. But, how can Ali detect the cause of (i.e., diagnose) this exception? That is, how can he know for sure whether it is the bookseller that has delayed the order, or it is the courier that has delayed the delivery?

Considering the above examples, it is a time consuming and tedious task for people to deal with those exceptions. It is better to have software agents to represent humans, and do these tasks on behalf of them. Accordingly, we propose to use agents to automate the reasoning on such exceptions. An agent is an intelligent entity (i.e., a software process) that perceives, reasons, and acts within an environment [1]. It may directly interact with its environment as well as interacting with other agents. In a multiagent system, several agents interact with each other. The type of interactions may change according to the context of the environment; cooperation, competition, or both [2]. Agents are autonomous, e.g., they can decide whom to interact with, and they are heterogeneous, e.g., it is difficult to make assumptions about their present or future behavior.

A multiagent system may be a closed multiagent system or an open multiagent system. For a closed multiagent system, the set of agents is predefined by the controller of the system. An open multiagent system differs from a closed multiagent system since it allows arbitrary external agents to join and leave the system at any time. At any point in time, an agent may not know about the existence of other agents, or their expected behaviors. Thus, it cannot assume how they will act in certain situations, or put a restriction on their actions as in closed systems. The privacy and e-commerce settings above can be modelled as open multiagent systems [3]. In open multiagent systems, it is important to regulate the interactions of agents, so that if something goes wrong during those interactions, the agents can reason using their interaction history.

We use commitments to formalize agent interactions. Commitments are widely used to describe formal agent interaction in multiagent research [4–7]. They provide a flexible way for agents to execute protocols, since they do not constrain the agents in terms of how they should satisfy their obligations. A commitment is formed between a debtor and a creditor agent, in which the debtor is committed to satisfy the commitment's consequent. For Example 1.1, a privacy agreement between the network operator and a user can be represented by a commitment, where the debtor is the network operator, the creditor is the user, and the consequent is to preserve the user's privacy on his shared documents. For Example 1.2, a contract between the cus-

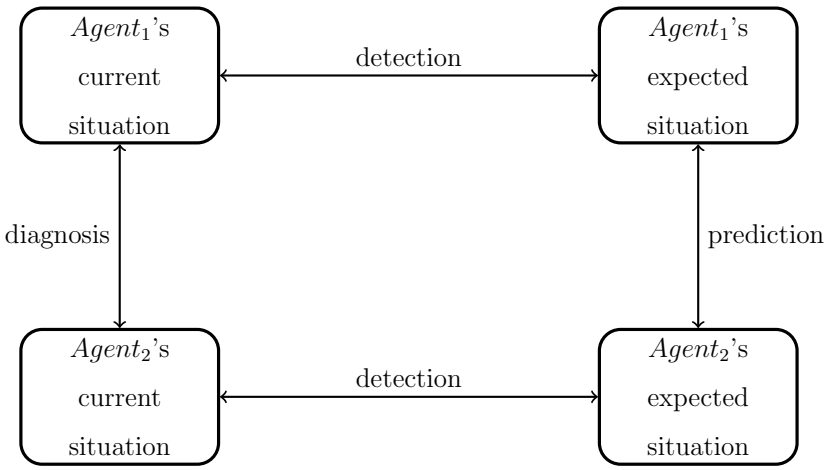


Figure 1.1. Phases of exception handling.

tomer and the bookseller can be represented by a commitment, where the debtor is the bookseller, the creditor is the customer, and the consequent is to deliver the book. Note that this commitment will also have an antecedent to purchase the book. That is, the bookseller will be committed to deliver the book after the customer makes the purchase.

This thesis proposes automated methods that deal with exceptions in commitment-based multiagent systems. Figure 1.1 depicts the phases of exception handling that we focus on. Agents have expectations about the future when they execute in a multiagent system. These expectations may be based on the agent's goals, its relations with other agents (e.g., commitments), or its previous experience on the task to be executed. If the agent faces a situation that does not comply with its expectations, then it should understand that there is an exception. This is the first phase in handling an exception, i.e., *detection of exceptions*. For example, if Ali does not receive the book he has ordered until the specified deadline, then he understands that there is something wrong. In order to detect an exception, the agent compares its current situation with the projection it has previously made, e.g., its expectations. If the agent's current situation is good enough to to meet its projections, then there is nothing wrong with the agent's execution. Otherwise, we say that the agent detects an exception.

One other phase that is as important as detection is to foresee that something might go wrong in the future, i.e., *prediction of exceptions*. For example, if Ali knows

that there will be a bad snow storm, he can understand that the delivery will be delayed. In order to predict an exception, the agent makes use of other agents' projections as well as some assumptions that might be true in the future.

Once the agent detects or predicts an exception, it needs to understand the underlying reason that has caused the exception, i.e., *diagnosis of exceptions*. For example, when Ali sees that the delivery is delayed, he needs to understand why this delay has occurred. He can ask the bookseller about the delay, or he can directly call the courier. In order to diagnose an exception, agents iteratively check their commitments to find out where the exception has initiated from.

The last step in handling an exception is to resolve the exception. That is, the agent needs to take proper action in order to fix the exceptional situation. However, this is a domain-dependent and agent-dependent task. Each agent may act differently in order to resolve an exception based on the environment it executes in. We do not deal with this phase in this thesis.

The contributions of this thesis are the following:

We extend the scope of exceptions to include cases where commitment violation is not the sole cause. With our proposal, an agent can signal an exception even though there is no commitment violation, or it may not signal an exception even if one of its commitment is violated. We provide a sound and complete framework to provide such reasoning based on comparing an agent's current situation with its projections about the future (Chapter 2 [8, 9]).

We apply model checking to predict exceptions based on a set of assumptions on the current environment. The outcome of prediction tells the agent whether its commitments will be violated based on the information it has given. If a violation is predicted, the setting in which violation is possible will also be presented to the agent. This way, an agent can understand whether performing an action will prevent its commitment from being fulfilled. If so, the agent may choose not to perform that

action (Chapter 3 [10]).

We exhaustively investigate the temporal relations among commitments. We use these relations to perform distributed diagnosis that deal with misalignment of commitments and improper commitment delegations. We show that our proposed framework is sound. We implement the framework in $\mathcal{REC}$, a tool for monitoring commitments at run-time (Chapter 4 [11, 12] and Chapter 5 [13, 14]). We also present an application of argumentation techniques and agent dialogues to perform diagnosis of commitment exceptions (Chapter 6 [15, 16]).

The rest of this thesis is structured as follows:

Chapter 2 describes our approach for detection of exceptions, and presents a case study on online social networks. Chapter 3 describes our approach for prediction of exceptions, and extends the case study that is used for detection. Chapter 4 describes our approach for diagnosis of exceptions, and presents a case study on a delivery scenario from e-commerce. Chapter 5 describes our approach for monitoring of commitment delegations, and presents a different case study from e-commerce. Chapter 6 describes an application of argumentation, and extends the case study that is used for diagnosis. Chapter 7 concludes this thesis. First, we review the relevant literature on commitments, distributed diagnosis, and exceptions. We discuss how our work relates to and differs from those work. Then, we review our contributions for this thesis. Finally, we present possible future directions to extend the ideas presented in this thesis.

2. DETECTION OF EXCEPTIONS

Commitments are an important abstraction for understanding and reasoning about agent interactions [4–6, 17, 18]. The key idea in modeling agent interactions with commitments is that messages are given a meaning based on creation and manipulation of commitments. That is, each message among agents either commits an agent to bring about a proposition or alters one of the existing commitments of an agent. Knowing the meaning of each message in the system empowers the agents to decide on their actions on their own.

An execution of interactions is successful when agents fulfill their commitments. In a single-agent environment, if we assume that the environment is fully cooperative, then the agent itself is the only entity that will affect the execution of the system. However, in multiagent systems, even when the environment behaves as expected, taking the correct actions does not guarantee that execution of the system will go as planned since other agents are also involved. Exceptions—deviations from the expected execution—may occur. It is, thus, necessary to monitor how the execution is processing. If the execution is not progressing as expected, it is best for the agent to step in and take further actions to make the execution right again. Specifically, the agent can go back to its commitments to examine what has gone wrong and in principle find ways to correct the execution.

We start investigating such exceptions in Web systems, where preserving the privacy of users is important. The general process of preserving privacy is through privacy agreements. Web systems announce their policies through privacy agreements. Users are expected to use the system only if they are comfortable with the agreement. In settings, where the Web system is a single locus of computation, carrying out privacy dealings through an agreement is reasonable. An example to such a setting is that of an e-mail system. The e-mail system announces to the user (via an agreement), whether it will share his account details or e-mail contents with others. Knowing this, the user can decide whether this is an appropriate e-mail system for his needs.

In online social networks, though, the loci of computation is distributed. The system that provides the social network service (such as Facebook) allows users to see each others' content, make comments, and even share the content with others. In such systems, it is difficult to maintain the privacy of users. Even if the system itself does not share the user information with other systems, other users on the social network can propagate a private content to others, for whom the content was not initially meant for. Or, other applications that benefit from the online social network can use private information for marketing or presentability purposes.

Hence, in systems that provide online social networks, even when a system owner correctly follows the privacy agreement that it announces, the privacy of users can easily be breached through interactions with other users. Consider the following scenario in Example 2.1.

Example 2.1. *The following relations hold among the three users of an online social network (OSN): Charlie and Sally are friends, and Charlie and Linus are colleagues. The OSN operator has two commitments with Charlie: (i) the OSN operator will share Charlie's new location with his friends at most in 15 minutes after he moves to a new location, (ii) the OSN operator will not disclose Charlie's location to any of his colleagues.*

Following this example, a typical understanding of an exception is that if Charlie moves to a new location, and his friend Sally is not updated of his new location in 15 minutes, then an exception occurs. Similarly, if Charlie's location is disclosed to Linus, then another exception occurs. This typical interpretation of an exception corresponds to commitment violation. While this is certainly important, an exception is not always a synonym for violation. Hence, there could be other cases where the commitment is not violated but an exception occurs and vice versa. Let us illustrate these points following Example 2.1.

- Assume that there is a mild earthquake. Charlie predicts that network connection

will be down for some period of time. Hence, even though his location is not shared within 15 minutes (and the commitment is violated), Charlie does not signal an exception and does not take any actions.

- Assume that Charlie is well aware that the OSN operator always shares changes in location in less than 5 minutes. Hence, even though it has been only 10 minutes (and the commitment is not violated), Charlie signals an exception and takes an action to handle the exception, e.g., contacts the OSN operator.
- In either of the above cases, from Sally's point of view, there is no exception, since she does not have a commitment with the OSN operator. She is not aware of the fact that Charlie's location will be shared with her.

The above example demonstrates two crucial facts about exceptions in multiagent systems.

Partial view: Each agent perceives the environment locally and is only aware of a part of the world. This stems from the fact that agents do not share all the information they have with other agents and that some information as well as interactions are private. As a result of this, contrary to centralized systems, where exceptions generally interfere with the working of the entire system, in multiagent systems, an exception may only be visible to a part of the system. That is, a deviation among two agents may have no effect on a third agent; moreover, the third agent may not be aware of the exception. This is an immediate result of the fact that agents have a local view of the environment.

Subjectivity: Each agent has a personalized (or subjective) projection about the future based on its previous interactions with others and capturing its commitments. These projections represent an agent's expectations of the future. Consider the first case described in the above examples: since there is an earthquake, and Charlie learns about this, he makes a projection that his location will not shared on time (i.e., the OSN operator's commitment for sharing his location will be violated). If the commitment is then actually violated, his current state will still satisfy his projection, because he was not expecting a prompt sharing anyway. Hence, even though there is a commitment

violation, Charlie will not immediately take action. Now consider the second case. Even though the commitment is not violated yet, Charlie may decide to step in and nag the OSN operator. Notice that an agent's projections do not necessarily correspond to an agent's goals or desires. As in the example above, Charlie may expect a late sharing of his location, knowing the existence of the earthquake, but certainly that is not his goal.

Each agent in a multiagent system needs to decide with local, private information whether its interactions are progressing as it has projected. It is important to be able to decide this, because if a projected state is not attainable, then an agent would need to step in and take further actions to correct the workings of the system.

Accordingly, we develop a principled, distributed approach to enable agents to specify their commitments, represent their projections, and more importantly compute whether their projections are attainable with the current execution of the system they are in. The specific contributions of this chapter are as follows:

- In order to relate agents' current states with their projections, we propose a *satisfiability* relation. In addition to the usual satisfiability relation in propositional logic, this relation also works on commitments by comparing stateful commitments. If a projected state is not satisfiable by a current state of an agent, we conclude that there is an exception in the system. We prove that our satisfiability relation signals an exception if and only if there is an exception in the system.
- Apart from satisfiability, we also formalize a *recoverability* relation in order to identify whether an exception is recoverable or not. That is, can the agents take proper action to recover from the exception?
  Following the examples above, the second case describes a recoverable exception. Note that Charlie has signaled an exception before the commitment has been violated. Since the commitment is still active, there is still a chance that the OSN operator will honor its commitment, and complete the sharing on time.
- We extend the Reactive Event Calculus ($\mathcal{REC}$) to handle conditional commitments and their states. We further implement our satisfiability and recoverability

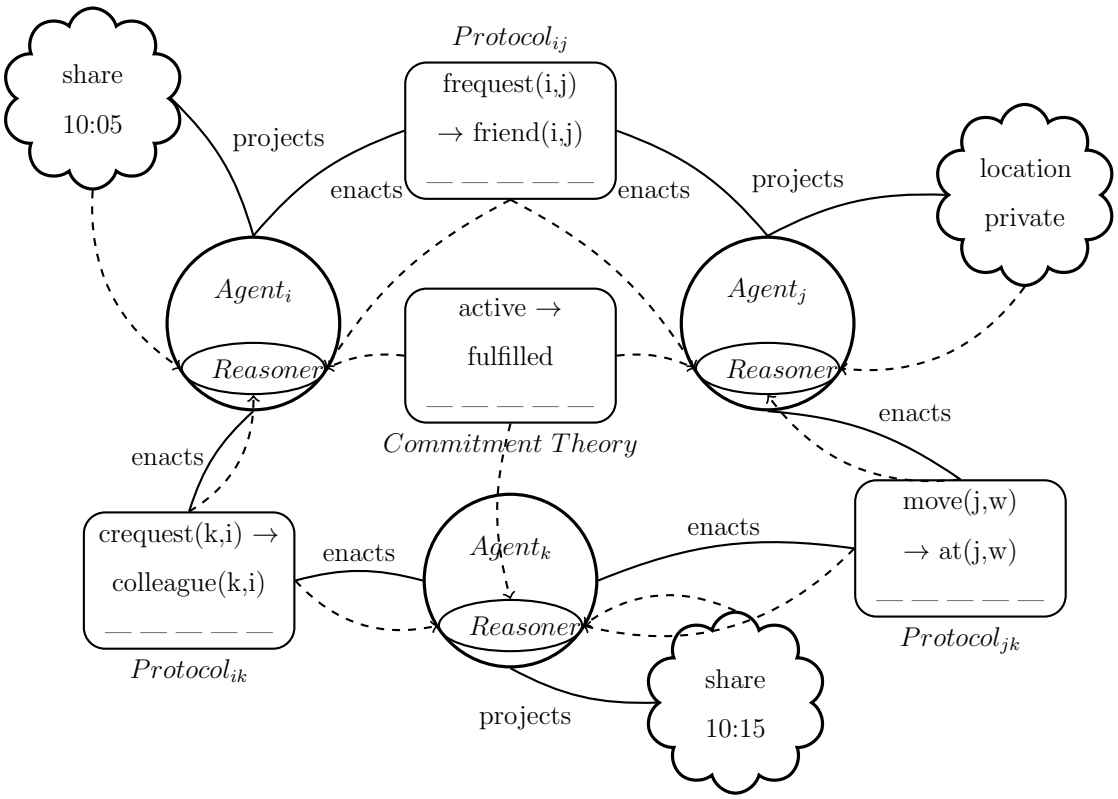


Figure 2.1. Distributed multiagent architecture.

relations in $\mathcal{REC}$, and provide it as a tool to detect exceptions.

- We analyze a case study to show that our tool can successfully catch exceptions, similar to the ones listed above.

2.1. Architecture

We focus on exceptions that are part of contract-based multiagent systems (e.g., e-commerce and e-business applications). In such systems, protocols are described as a set of contracts [19, 20], each describing a certain transaction between some of the participants, e.g., agents. Note that privacy is a key property in business protocols, which is preserved by agents' policies [21]. An effective way of preserving privacy is to enable distributed protocol execution. That is, each agent views its part of the protocol only, and manipulates its own contracts.

Agents may have different projections about their future. By sensing the differences between their projections and the reality they face, they should be able to signal

exceptions, recover from them, and continue proper working. Figure 2.1 shows the distributed multiagent architecture that we propose to detect such exceptions at run-time. Here, in the privacy domain, the agents represent the users of the OSN. Simply, the agents enact a distributed protocol described by a set of interaction and commitment manipulation rules, and periodically check for exceptions throughout their progress. To do so, an agent compares its projection with its current state of the world at a given point in time. In Figure 2.1, the circles denote agents. Each agent has a reasoner inside that enables it to infer whether projected states are reached or not. Each agent enacts a protocol, which is shown with a rounded rectangle in the figure. Note that each agent has access to a portion of the protocol that is relevant for itself. Each agent has a projected state, shown by a cloud in the figure. The lines between the elements show how they are connected. For example, the projection on the top left is only connected to the related agent $Agent_i$, whereas the protocol description $Protocol_{ij}$ is connected to the two related agents $Agent_i$ and $Agent_j$. The dashed arrows show which information is fed into the agent's reasoner, e.g., the rules in the commitment theory are used by the agents in order to reason on their commitments. Next, we briefly describe each element and give a sample execution to demonstrate their usage.

Protocol description: Following a distributed execution, protocol descriptions are shared among related agents, so that an agent only has access to protocol rules concerning itself [21]. In real life, each agent would be aware of the protocol that it is involved in, but would not know the agreements among other entities. In addition, by coupling the protocol descriptions among agents, we ensure that they share the same semantics for the protocol rules (e.g., infer the same consequences for actions), thus providing *interoperability* [22]. An example rule describes how the friend request action is processed, e.g., the agents involved in the action and the consequences of the action.

Commitment theory: The commitment theory defines the rules for creating and manipulating commitments. A commitment is a live object that changes state due to the interactions of the agents [5, 6]. A commitment state captures the current status of a commitment. For example, when a commitment is first created it would be in an active state waiting to be fulfilled. Once the commitment is actually carried out, the

commitment would go into a fulfilled state. We explain the lifecycle of a commitment in more depth when we describe the formal model.

In our architecture, the commitment theory is shared among all the agents. This is required so that all agents can manipulate their commitments in the same way. This is also important to prevent commitment misalignments [12, 23] from occurring, e.g., both the creditor and the debtor infer the same commitment state. An example rule in the commitment theory describes the conditions on how a commitment makes a transition from the active state to the fulfilled state, e.g. when the property of the commitment is satisfied.

Projection: Each agent may have several projections about its future, which describe its expectations. For example, after moving to a new location at $10:00$, Charlie expects his location to be shared with his friends by $10:15$.

Event: Events correspond to the actions of the agents. Each event is associated with a time point that describes at which time the event has occurred. For example, when Charlie goes to his office at time $10:00$, this corresponds to a move event by Charlie to location office at time point $10:00$.

Reasoner: Each agent has a separate reasoner that it uses to check for exceptions. The reasoner is fed with the protocol rules, the commitment theory, the agent's projections, and a sequence of events that are significant to the agent's execution. Accordingly, the reasoner produces an output that tells whether there is an exception in the current state of the agent.

Now, let us see how the following execution of Charlie from Example 2.1 is connected with the above elements:

(i) The commitment ensures sharing of Charlie's new location in 15 minutes (protocol description),
(ii) Charlie moves to his office at $10:00$ (event),

(iii) Upon Charlie's move, the OSN operator becomes committed to Charlie (commitment theory),

(iv) Based on the commitment, Charlie expects his new location to be shared by 10 : 15 (projection),

(v) Charlie checks to see whether his projection is satisfied or not, and finds out it is not since Sally has not been informed that Charlie has come to his office. Based on this finding, Charlie signals an exception (reasoner).

In order to detect such exceptions, we need a reasoner that can interpret our commitment theory and compare agent states to decide if a projected state is satisfiable by the current state. To realize this reasoning, we first develop a formal model of satisfiability in terms of states and commitments. Next, we implement this reasoning using the Reactive Event Calculus.

2.2. Commitments

A commitment represents a contract from a debtor agent towards a creditor agent about a specific property [4]. Definition 35 defines a commitment formally. Below, A_i and A_j denote agents (e.g., agents that enact a business protocol); *Ant* and *Con* are propositions.

Definition 1. *A commitment* $C^{St}_{A_i,A_j}(Ant, Con)$ *denotes the commitment between the agents* A_i *and* A_j*, with St being its state. In particular, four commitment states are meaningful for our work; conditional, active, fulfilled and violated. The above is a conditional commitment; if the antecedent Ant is satisfied (i.e., becomes true), then the debtor* A_i *becomes committed to the creditor* A_j *for satisfying the consequent Con, and the commitment becomes active. If Ant is already True (denoted* $\top$*), then this is an active base-level commitment;* A_i *is committed to* A_j *for satisfying Con unconditionally. Antecedent and consequent are propositions.*

We follow the idea and notation of [23–25] to represent commitments (i.e., every commitment is conditional). A base-level commitment is simply a commitment with

its condition being true. We also separate the agents from the antecedent and the consequent. This way, we can omit the agents from the commitment description whenever they are not significant.

2.3. Formal Model

Our formal model is based on the description of the world through states and the evolution of the states through agents' commitments [4–6].

2.3.1. World Model

Each agent views a part of the world since the execution of the multiagent system is distributed among the agents. Hence, a state is subjective to an agent and captures the agent's view-point at a time point.

Definition 2. *A time point is a discrete measure of time and is totally ordered. We use* t_1, t_2, ..., t_n *to denote time points.*

A state consists of propositions that are known to be true at that time point and commitments that are represented via their states. Propositions tell what has happened in the system so far (i.e., facts) to the agent's perception. We assume that the system is monotonic and so once a proposition becomes true, it cannot become false.

Definition 3. *A proposition can be an atomic proposition, its negation or conjunction of atomic propositions.*

Definition 4. *A term* $\mathcal{T}$ *is either a proposition* ϕ *or a commitment* C*.*

Definition 5. *A state contains terms that hold at a particular time point* T*. To keep processing simple, we partition propositions* Φ *and commitments* $\mathcal{C}$ *in two different sets. Hence,* $\mathcal{S}^T(X) = \langle \Phi, \mathcal{C} \rangle$*.*

We differentiate between three types of states with the variable X:

Table 2.1. Examples of states.

(a) $\mathcal{S}^{10:05}(G) = \langle\{newlocation\}, \{\mathcal{C}^{a}_{osn,charlie}\ (newlocation, shared)\}\rangle$
(b) $\mathcal{S}^{10:05}(charlie) = \langle\{newlocation\}, \{\mathcal{C}^{a}_{osn,charlie}\ (newlocation, shared)\}\rangle$
(c) $\mathcal{S}^{10:05}(sally) = \langle\{\}, \{\}\rangle$
(d) $\mathcal{S}^{10:05}(charlie_p) = \langle\{newlocation, shared\}, \{\}\rangle$

- $\mathcal{S}^{T}(G)$: The world is described by a global state which demonstrates a global view of the multiagent system at time T. The global state is not meant to be known or processed by any of the agents in the multiagent system.
- $\mathcal{S}^{T}(A)$: In contrast with the global state, each agent A has a local state (i.e., its local world model), which is a subset of the real world at time T. In a distributed execution, agents perceive the real world from different view-points. Thus, their states may differ from each other based on their observations.
- $\mathcal{S}^{T}(A_p)$: In order to enable agents about reasoning about the future, it is important to represent how the agent projects the future. That is, based on its current state and its current commitments in this state, how does the agent project the world to evolve at a future time point? We represent this with the notion of a projected state, which represents a projection of A for the world at time T. The projected state is a representation of what the agent expects when time has evolved to that point[1] . Note that the agent always makes the projection at an earlier time point.

[1]In this thesis, we use the term "projection" whenever we refer to the expectations of an agent about a future state of the world.

Table 2.1 shows examples for each type of state for the same time point 10 : 05. In (a), the global state of the world for time 10 : 05 is shown. It contains a single proposition which tells that Charlie has moved to a new location, and an active commitment from the OSN operator to Charlie, which tells that the sharing of Charlie's new location is still in process. In (b), Charlie's state for the same time point 10 : 05 is shown, which is identical to the global state. That is, Charlie fully perceives his environment at this point. However, it is not always the case that the agents can perceive everything in their environment. For example, in (c), Sally, is neither aware of the commitment between Charlie and the OSN operator, nor aware of the fact that Charlie has moved to a new location. In (d), Charlie's projection for time 10 : 05 is shown. One can easily see that Charlie expects sharing of his location to be completed for this time point. However, from the local state of Charlie, we see that the corresponding commitment is still active. Indeed, there is an inconsistency between what Charlie perceives and expects.

A rational agent tries to perform actions that will enable its projected state to be realized. However, note that a projected state may not be realized solely by the agent itself since it may contain propositions or commitments that can only be satisfied by others.

2.3.2. Commitment Model

Commitments are live objects; we always consider a commitment with its state [5, 6]. Next, we describe each commitment state with respect to the corresponding world states and the transitions in between. Assume the following state $\mathcal{S}^T(A) = \langle \Phi, \mathcal{C} \rangle$ and that $C^{St}(Q, P) \in \mathcal{C}$. Let's walk through the different values for St.

Conditional: When the commitment is in conditional state, denoted $C^c(Q, P)$, if Q is brought about, then the debtor will be committed to bring about P. When the state is conditional, then $Q, P \notin \Phi$. In other words, the commitment $C^c(Q, P)$ cannot coexist in the same state with its antecedent or consequent:

- If the commitment's antecedent Q already holds, then the commitment is no longer *conditional* (i.e., its condition is satisfied), and it will become *active*.
- If the commitment's consequent P already holds, then the commitment is no longer *conditional*, and it will become *fulfilled*. Afterward, it is not significant whether the antecedent is also satisfied or not.

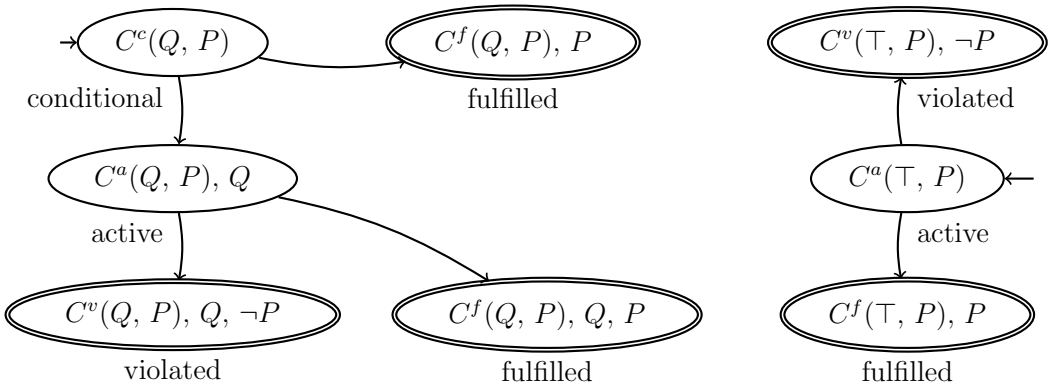
Active: A commitment can be in an active state following two different paths:

- The commitment is initially created without a condition, denoted as $C^a(\top, P)$. We require that $P \notin \Phi$. In other words, the commitment $C^a(\top, P)$ cannot coexist in the same state with its consequent. If the commitment's consequent P already holds, then the commitment is no longer *active*, and it will become *fulfilled*. Once the consequent is satisfied, then the commitment's life-cycle ends.
- Alternatively, a conditional commitment may be detached into an active commitment. That is, if the state that includes the commitment $C^c(Q, P)$ makes a transition to a state where the commitment's antecedent holds, then the commitment will become active, $C^a(Q, P)$.

Fulfilled (after conditional): The commitment is in fulfilled state, denoted $C^f(Q, P)$, when at $\mathcal{S}^{T-1}$, $C^c(Q, P)$ exists and at $\mathcal{S}^T$, $P \in \Phi$. That is, if the state that includes the commitment $C^c(Q, P)$ makes a transition to a state where the commitment's consequent holds, then the commitment will become *fulfilled*. The conditional commitment's life-cycle ends with this state.

Fulfilled (after active): The commitment is in fulfilled state, denoted $C^f(\top, P)$, when at $\mathcal{S}^{T-1}$, $C^a(Q, P)$ exists and at $\mathcal{S}^T$, $P \in \Phi$. That is, if the state that includes the commitment $C^a(\top, P)$ makes a transition to a state where the commitment's consequent holds, then the commitment will be in the fulfilled state. The base-level commitment's life-cycle ends with this state.

Violated: The commitment is in violated state, denoted $C^v(Q, P)$, if it were active and its consequent does not hold. It can either be that there is a time out in



(a) Conditional (b) Base-level

Figure 2.2. Commitment states.

which P has not been realized and thus $\neg P$ holds or $\neg P$ is specifically made true by an agent. Hence, the constraint we require is that if a commitment is in a violated state, then it must have been active and its consequent must be false. That is, when at $\mathcal{S}^{T-1}$, $C^a(Q, P)$ exists and at $\mathcal{S}^T$, $\neg P \in \Phi$. Again, the commitment's life-cycle ends with this state.

Definition 6. *An active state of a commitment can only be followed by a fulfilled or violated state and a conditional state can only be followed by an active or a fulfilled state.*

Figure 2.2 summarizes the commitment states; 2.2(a) for conditional commitments and 2.2(b) for base-level commitments. This flow of states is consistent with existing work [5,6] and represents a review of the state of the art. The transitions that are given in Figure 2.2 are complete in the sense that there are no other ways to change commitment states. Note that double ellipses represent terminal commitment states (i.e., the commitment's life-cycle ends in those states).

Definition 7. *A commitment $C_1^{St_1}(p, q)$ is content-wise identical to a commitment $C_2^{St_2}(r, s)$, denoted $C_1 =_c C_2$, iff $p = r$ and $q = s$.*

Definition 8. *A commitment $C_1^{St_1}(p, q)$ is a successor of a commitment $C_2^{St_2}(r, s)$, denoted $C_2 \gg C_1$, iff $C_1 =_c C_2$ and St_1 follows St_2 as defined in Definition 6.*

2.3.3. Exceptions

We understand exceptions as situations that are worse off for an agent than is expected by the agent. To formalize this, we formalize the benefits of terms for an agent.

Definition 9. *$\mathcal{T}_P(S)$ is the set containing the terms in state S that include proposition P.*

The terms in $\mathcal{T}_P(S)$ can be the following:

- $C^v(Q, P)$ or $C^v(\top, P)$: for simplicity we use $C^v(P)$ to denote both, which represents a violated commitment towards P.
- $C^a(Q, P)$ or $C^a(\top, P)$: similarly, we use $C^a(P)$ to denote both.
- $C^f(Q, P)$ or $C^f(\top, P)$: we use $C^f(P)$ to denote both.
- P: is a proposition.

Definition 10. *The utility of a term X shows how desirable X is in the execution. We assume a function u from terms into real numbers so that each agent can compute and compare the utilities of terms.*

Axiom 1. *From the creditor's point of view, the following is meaningful: $u(C^v(P)) < u(C^a(P)) < u(C^f(P)) = u(P)$. That is, the creditor would benefit most if the commitment is fulfilled. Less beneficial is an active commitment and the worst is a violated commitment. This ordering is compatible with the commitment valuations proposed by Yolum and Singh [26]. Note that $u(C^f(Q, P)) = u(C^f(\top, P))$, so we use the short-hand $u(C^f(P))$ to denote the utility of both.*

Axiom 1 shows how the utility relation is ordered for each possible term that includes proposition P. Note that the utility relation reflects an agent's expectations from a commitment.

Figure 2.3 demonstrates the condition where an exception will occur. The X axis shows expected terms, whereas the Y axis shows the actual terms. When an agent

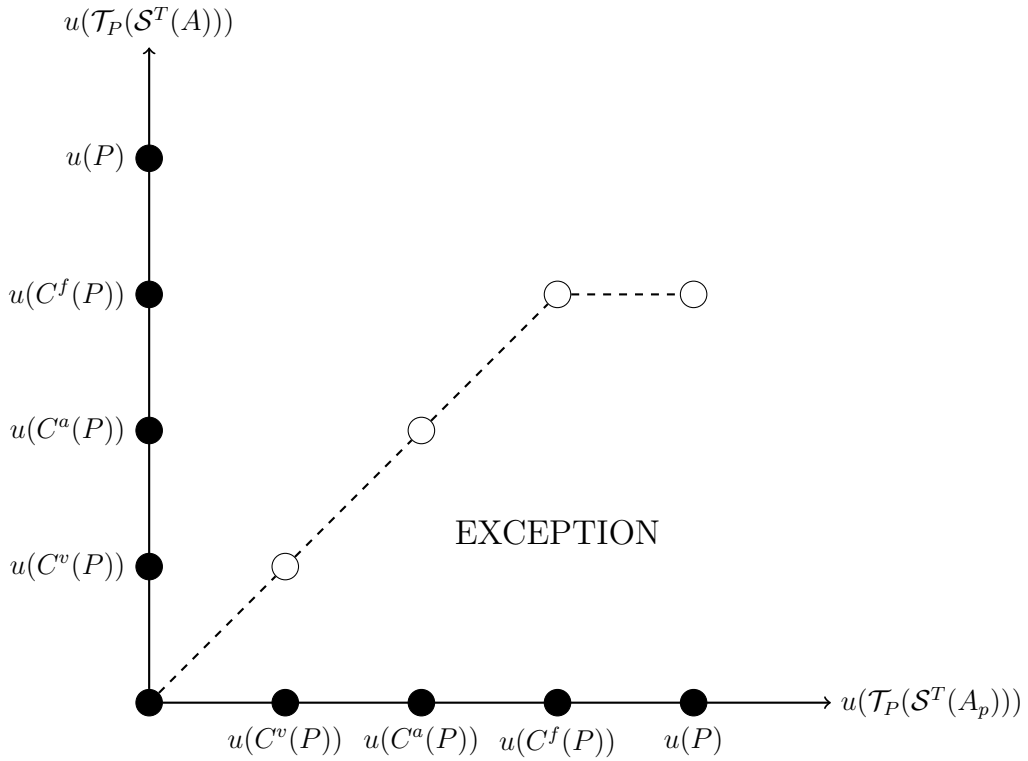


Figure 2.3. Understanding exceptions.

ends up achieving a term that has lower utility than its expected term, then this will yield an exception. For example, if the agent expects $C^f(P)$, but in reality, the world has $C^a(P)$, then it should signal an exception and take an action. The area under the dashed line shows the cases where the agent would signal an exception.

Definition 11 describes exceptions formally based on this intuition. In particular, there are two cases that cause exceptions. In the first case, if there is a term in the projected state and there are no corresponding terms in the real state, then it means that the agent expects something to happen, but there are no events related to that occurring. Note that Charlie expects his location to be shared. If he sees no commitment from the the OSN operator towards sharing his location, then he understands that there is something wrong. In the second case, even though there is a term related to the one in the projected state, the utility of the term in the projected state is higher than the utility of the term in the real state. Again, consider Charlie's case. He expects sharing to be completed. However, assume that the commitment from the OSN operator is still active. Obviously, the utility of the commitment is higher

for Charlie when it is fulfilled, since he is the creditor of the commitment. Thus, he understands that his current situation is not good enough as he has expected. This definition of an exception captures the idea that an exception is subjective in the sense that one agent may identify a particular situation as an exception while another agent may not, depending on its expectations. Notice that our commitment model ensures that only one of the commitment terms can exist in a particular state. Moreover, P can only coexist with $C^f(P)$ in the same state.

Definition 11. *An exception occurs for agent A at time T related to proposition P, denoted exception(A, T, P), iff*

- *the projected state contains a term but the real state does not contain any terms; that is:* $\exists\ x$*:* $x \in \mathcal{T}_P(\mathcal{S}^T(A_p))$ *and* $\mathcal{T}_P(\mathcal{S}^T(A)) = \emptyset$*, or*
- *the projected state contains a term that is higher in utility than the related term in the real state; that is:* $\exists\ x, y$*:* $x \in \mathcal{T}_P(\mathcal{S}^T(A_p))$*,* $y \in \mathcal{T}_P(\mathcal{S}^T(A))$*, and* $u(x) > u(y)$*.*

Note that Definition 11 does not compare the total utility value of the states but of individual terms. The intuition is that we do not want the summation of utilities to cover for the differences in individual differences. For example, assume that there are two terms in the real state as well as in the projected state. It might be the case that when analyzed individually one of the terms in the real state has lower utility than its associated term in the projected state. Once there is a problem related to a single proposition in an agent's state, then there is an exception in that state. Our definition follows this idea and signals an exception. However, if we had looked at the utilities of the states as a whole, it might have been the case that the utility gained through the second term is much higher in the real state, yielding a high utility for the real state than the projected state. Since the utility had been higher in the real state, we would not have signaled an exception, missing a possible exception on the first term. Assume that Charlie moves to his office, and he expects his friends Sally and Alice to be informed of this. In addition, it is more important for Charlie at the time that Alice immediately gets aware of his new location. Now, if Alice is informed of his location, and Sally is not, and if we consider the utilities together, then, we might miss the case

that Sally is not informed of Charlie's location at all.

2.4. Satisfiability

We capture the projections of an agent through projected states. On one hand, a projected state is similar to a temporal achievement goal [27], where the agent plans to reach some properties or be involved in some commitments at a certain time point. On the other hand, projected states do not necessarily represent goals. They also model the agent's expectations about the future. That is, the agent may assume certain properties to hold in the future even though it does not wish so, e.g., Charlie may expect a late sharing if he knows about the earthquake.

It is important that an agent can compare its real and projected states. This is crucial for the agent to understand if everything is progressing as it has expected. Basically, there are two outcomes of this comparison. If the agent finds out that the the situation observed in its real state is not good enough to support its projection, then it understands that something has gone wrong. If not, then its execution is fine. For comparing the real and projected states, we propose a *satisfiability relation*. It compares two states and tells if one satisfies the other. This is a strong relation in the sense that once a state satisfies another, then the two states are either equal or the former can replace the latter. This captures our intuition that if the current state of the world is equivalent to or better than the projected world state, the execution is in order and no action needs to be taken. However, if the comparison yields that the current state does not satisfy a projected state, then there is an exception and it should be handled by the agent.

Our understanding of satisfiability is inspired from satisfiability in propositional logic. However, in this case, we do not only have propositions but also commitments with states. Hence, we need a relation that can also take care of comparisons of stateful commitments. Our satisfiability relation is denoted by $X \Vdash Y$, and is read as "Y is satisfiable by X". Since the proposed relation will be used to compare the current state of an agent with a projected state, X will correspond to the real state and Y

will correspond to the projected state of the agent. The satisfiability relation we need should have these properties:

- Reflexive: If the current and projected states are identical, we expect our satisfiability relation to hold.
- Non-symmetric: If the current state is worse than a projected state, the current state will not satisfy the expectation, but the other direction will hold. Hence, the satisfiability relation should be non-symmetric.
- Transitive: If a current state satisfies a projected state, which in turn satisfies a third state, we want to conclude that the first state would satisfy the third state as well.

We formalize the intuition of satisfiability via the following axioms. Figure 2.4 is an aid in following the axioms. Given two atomic propositions *newlocation* and *shared*, Figure 2.4 depicts some possible terms that can be derived from them and draws the satisfiability relation between them. Note that the proposition *newlocation* represents that Charlie has moved to a new location, and the proposition *shared* represents that his new location is shared among his friends. Arrows show the direction of satisfiability. To reduce clutter, we remove the agent names. In order to understand satisfiability between states, we start with studying satisfiability between terms.

Axiom 2. *A proposition ϕ_j is satisfiable by a proposition ϕ_i, denoted $\phi_i \Vdash\!\vdash \phi_j$, iff $\phi_i \vdash \phi_j$.*

Axiom 2 follows directly from logic entailment in propositional logic. When commitments are involved, we need to go beyond basic logic entailment. Accordingly, Axiom 3 states that if a proposition entails a commitment's consequent, then the commitment is satisfiable independently of its state.

Axiom 3. *A base-level commitment $\mathcal{C}^S_{A_i,A_j}$ ($\top$, Con) is satisfiable by a proposition ϕ, denoted $\phi \Vdash\!\vdash \mathcal{C}^S_{A_i,A_j}$ ($\top$, Con), iff $\phi \vdash Con$.*

We can study Axiom 3 as three cases that correspond to the different commitment

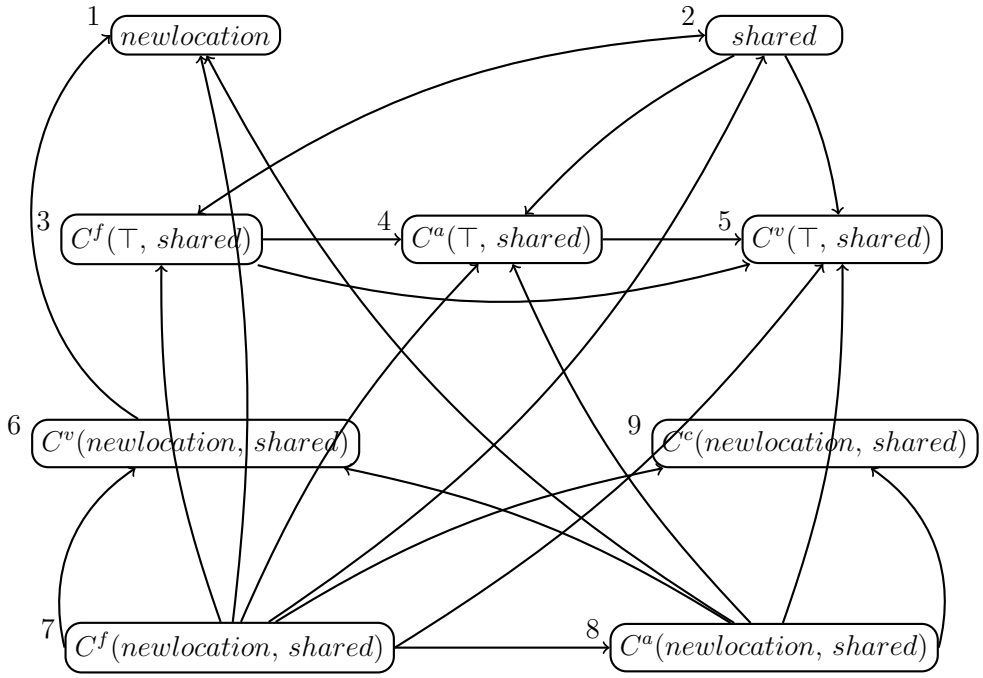


Figure 2.4. A partial view of the satisfiability network.

states $\{f,\ a,\ v\}$.

- A proposition entailing a fulfilled commitment's consequent posesses the same outcome as the commitment itself. For example, in Figure 2.4, nodes 2 and 3 ($shared \Vdash C^f(\top,\ shared)$) exhibit this relation. The interpretation is that if Charlie has a projection that his commitment towards sharing his location will be fulfilled and if he perceives that the sharing is performed, then his projection is satisfied. Hence, the proposition satisfies the fulfilled commitment.
- A proposition entailing an active commitment's consequent is more beneficial than the commitment itself. For example, in Figure 2.4, nodes 2 and 4 ($shared \Vdash C^a(\top,\ shared)$) exhibit this relation. That is, if Charlie has a projection that his commitment will be active but perceives that the sharing is actually already performed, then he is indeed in a better situation. Hence, the proposition satisfies the active commitment.
- A proposition entailing a violated commitment's consequent is more beneficial than the commitment itself. For example, in Figure 2.4, nodes 2 and 5 ($shared \Vdash C^v(\top,\ shared)$) exhibit this relation. Now, if Charlie expects that his commitment will be violated, then he will be satisfied when he actually sees that the

sharing is performed. Hence, the proposition satisfies the violated commitment.

Axiom 4. *A conditional commitment* $\mathcal{C}^{S}_{A_i,A_j}$ *(Ant, Con) is satisfiable by a proposition* ϕ*, denoted* $\phi \Vdash \mathcal{C}^{S}_{A_i,A_j}$ *(Ant, Con) , iff*

- $\phi \vdash Con$*, or*
- $\phi \vdash Ant$ *and* $S \in \{v\}$*.*

Axiom 4 describes the cases when a proposition would satisfy a conditional commitment. If the proposition already entails the consequent of the conditional commitment, then no matter what the state of the conditional commitment is the proposition satisfies the conditional commitment. The intuition is that by carrying out the conditional commitment at different stages, the creditor will at most get the consequent. If the consequent is already there, then this yields the same output. Otherwise, if the proposition entails the antecedent, the only way for the proposition to satisfy the conditional commitment is if the conditional commitment is violated. That is, the antecedent has been realized but the consequent is not going to be realized.

Axiom 5. *A proposition* ϕ *is satisfiable by a base-level commitment* $\mathcal{C}^{S}_{A_i,A_j}$ *(*$\top$*, Con), denoted* $\mathcal{C}^{S}_{A_i,A_j}$ *(*$\top$*, Con)* $\Vdash \phi$*, iff* $Con \vdash \phi$ *and* $S \in \{f\}$*.*

Similar to Axiom 3, Axiom 5 states that if the commitment's consequent entails the proposition, then the proposition is only satisfiable if the commitment is fulfilled. This time, we are again considering nodes 2 and 3 but computing whether node 3 satisfies node 2 ($C^{f}(\top, \mathit{shared}) \Vdash \mathit{shared}$). A fulfilled commitment towards sharing means that sharing has occurred. Indeed, this is equivalent to the proposition *shared*.

Axiom 6. *A base-level commitment* $\mathcal{C}^{S_2}_{A_k,A_l}$ *(*$\top$*,* Con_2*) is satisfiable by a base-level commitment* $\mathcal{C}^{S_1}_{A_i,A_j}$ *(*$\top$*,* Con_1*), denoted* $\mathcal{C}^{S_1}_{A_i,A_j}$ *(*$\top$*,* Con_1*)* $\Vdash \mathcal{C}^{S_2}_{A_k,A_l}$ *(*$\top$*,* Con_2*), iff*

- $Con_1 \vdash Con_2$ *and* $S_1 \in \{f\}$*, or*
- $Con_1 \vdash Con_2$*,* $S_1 \in \{a\}$*, and* $S_2 \in \{a, v\}$*, or*

- $Con_1 \vdash Con_2$, $S_1 \in \{v\}$, *and* $S_2 \in \{v\}$.

Axiom 6 uses the ideas in Axioms 3 and 5 for comparing base-level commitments. Let us review each case:

- A projection of a commitment in any state can be satisfied by a fulfilled commitment as long as its consequent is entailed. For example, in Figure 2.4, nodes 3 and 4 ($C^f(\top, shared) \Vdash C^a(\top, shared)$) exhibit this relation, since the fulfilled commitment is better than a projection where the commitment is only active.
- A projection of an active or violated commitment can also be satisfied by an active commitment. For example, in Figure 2.4, nodes 4 and 5 ($C^a(\top, shared) \Vdash C^v(\top, shared)$) exhibit this relation.
- A projection of a violated commitment can also be satisfied by another violated commitment. That is, if the expectation was that the commitment would have been violated, nothing worse can happen concerning the commitment. Hence, any commitment state is as good as the violated state.

Axiom 7. *A conditional commitment* $\mathcal{C}^{S_2}_{A_k,A_l}(Ant, Con_2)$ *is satisfiable by a base-level commitment* $\mathcal{C}^{S_1}_{A_i,A_j}(\top, Con_1)$, *denoted* $\mathcal{C}^{S_1}_{A_i,A_j}(\top, Con_1) \Vdash \mathcal{C}^{S_2}_{A_k,A_l}(Ant, Con_2)$, *iff*

- $Con_1 \vdash Con_2$ *and* $S_1 \in \{f\}$, *or*
- $Con_1 \vdash Con_2$, $S_1 \in \{a\}$, *and* $S_2 \in \{a, v, c\}$, *or*
- $Con_1 \vdash Con_2$, $S_1 \in \{v\}$, *and* $S_2 \in \{v\}$, *or*
- $Con_1 \vdash Ant$, $S_1 \in \{f\}$, *and* $S_2 \in \{v\}$.

Axiom 7 describes when a base-level commitment satisfies a conditional commitment. First, let's look at the cases where the consequent of the base-level commitment entails the consequent of the conditional commitment. If the base-level commitment is fulfilled, then this satisfies any state of the conditional commitment since this is the best that can result from the conditional commitment. The active state of the base-level commitment satisfies all states of the conditional commitment except the

fulfilled state, since the base-level commitment is in active state, the consequent does not hold. In the case of violated state, since neither consequent is going to be realized, the base-level commitment satisfies the conditional commitment. If instead the consequent of the base-level commitment satisfies the antecedent of the conditional commitment, there is only one state combination that yields satisfaction: when the base-level is fulfilled and the conditional commitment is violated.

Axiom 8. *A proposition ϕ is satisfiable by a conditional commitment $\mathcal{C}^{S}_{A_i,A_j}$ (Ant, Con), denoted $\mathcal{C}^{S}_{A_i,A_j}$ (Ant, Con) $\Vdash\!\vdash \phi$, iff*

- *Ant $\vdash \phi$ and $S \in \{a, v\}$, or*
- *Con $\vdash \phi$ and $S \in \{f\}$.*

Axiom 8 describes the two ways that a conditional commitment satisfies a proposition. The first case states that if a conditional commitment reaches an active state, then it satisfies its antecedent. This is straightforward since the only way to reach an active conditional commitment is by having the antecedent hold. Further, if the consequent never becomes true (i.e., the commitment is violated), the commitment would still satisfy the antecedent; hence for example, in Figure 2.4, node 6 would satisfy node 1 ($C^{v}(newlocation, shared) \Vdash\!\vdash newlocation$). Note that the fulfilled state is not included in this case, because the conditional commitment may have been fulfilled directly from the conditional state leaving us with no information about the antecedent itself. The second case states that a fulfilled conditional commitment satisfies its consequent. In Figure 2.4, a similar example would be among nodes 7 and 2 ($C^{f}(newlocation, shared) \Vdash\!\vdash shared$).

Axiom 9. *A base-level commitment $\mathcal{C}^{S_2}_{A_k,A_l}$ ($\top$, Con_2) is satisfiable by a conditional commitment $\mathcal{C}^{S_1}_{A_i,A_j}$ (Ant, Con_1), denoted $\mathcal{C}^{S_1}_{A_i,A_j}$ (Ant, Con_1) $\Vdash\!\vdash \mathcal{C}^{S_2}_{A_k,A_l}$ ($\top$, Con_2), iff*

- *$Con_1 \vdash Con_2$ and $S_1 \in \{f\}$, or*
- *$Con_1 \vdash Con_2$, $S_1 \in \{a\}$, and $S_2 \in \{a, v\}$, or*
- *$Con_1 \vdash Con_2$, $S_1 \in \{v\}$, and $S_2 \in \{v\}$, or*

- $Ant \vdash Con_2$ *and* $S_1 \in \{a, v\}$.

Axiom 9 describes the conditions in which a conditional commitment satisfies a base-level commitment. If we consider the consequents of the commitments, then the relation between the two commitments contain the relations between two base-level commitments (Axiom 6). The only extra case here is when the antecedent of the conditional commitment can entail the consequent of the base-level commitment. In this case, even if the consequent of the conditional commitment does not hold, the conditional commitment will satisfy the base-level commitment. Note that a conditional commitment that is in a conditional state cannot satisfy a base-level commitment.

Axiom 10. *A conditional commitment* $\mathcal{C}^{S_2}_{A_k,A_l}$ *(*Ant_2*,* Con_2*) is satisfiable by a conditional commitment* $\mathcal{C}^{S_1}_{A_i,A_j}$ *(*Ant_1*,* Con_1*), denoted* $\mathcal{C}^{S_1}_{A_i,A_j}$ *(*Ant_1*,* Con_1*)* $\Vdash\!\!\!\Vdash \mathcal{C}^{S_2}_{A_k,A_l}$ *(*Ant_2*,* Con_2*), iff*

- $Ant_2 \vdash Ant_1$, $Con_1 \vdash Con_2$, $S_1 \in \{f\}$, *or*
- $Ant_2 \vdash Ant_1$, $Con_1 \vdash Con_2$, $S_1 \in \{a\}$, *and* $S_2 \in \{a, v, c\}$, *or*
- $Ant_2 \vdash Ant_1$, $Con_1 \vdash Con_2$, $S_1 \in \{v\}$, *and* $S_2 \in \{v\}$, *or*
- $Ant_2 \vdash Ant_1$, $Con_1 \vdash Con_2$, $S_1 \in \{c\}$, *and* $S_2 \in \{c\}$, *or*
- $Ant_1 \vdash Con_2$ *and* $S_1 \in \{a, v\}$.

Axiom 10 applies a similar reasoning to Axiom 6 for conditional commitments. For example, $C^f(newlocation, shared) \Vdash\!\!\!\Vdash C^f(\top, shared)$ (denoted with nodes 7 and 3 in Figure 2.4). Here both commitments are fulfilled with the same consequent. Moreover, $C^f(newlocation, shared) \Vdash\!\!\!\Vdash C^a(newlocation, shared)$. Here, the fulfilled commitment entails *shared*, thus satisfying the active commitment towards *shared*.

The above axioms cover the possible satisfiability relations among terms.

Theorem 1. $\Vdash\!\!\!\Vdash$*relation is reflexive, non-symmetric, and transitive.*

The above development shows how a term satisfies another term. Starting from

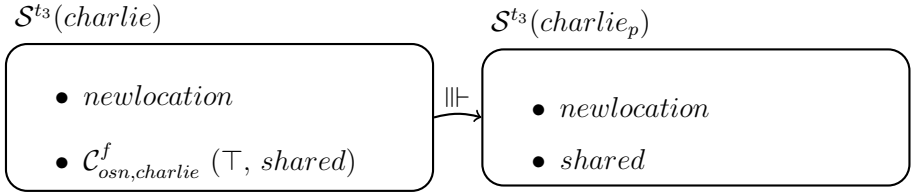


Figure 2.5. State satisfiability.

that, we need to show how a state satisfies another state. As a motivating example, consider the states in Figure 2.5. To decide whether the state on the right is satisfiable by the state on the left, we need to figure out whether there exists a term on the left state for each term on the right state such that the term on the left satisfies the term on the right. That is, the term *newlocation* in state $\mathcal{S}^{t_3}(charlie_p)$ is satisfiable by the term *newlocation* in state $\mathcal{S}^{t_3}(charlie)$. Similarly, the term *shared* in state $\mathcal{S}^{t_3}(charlie_p)$ is satisfiable by the term $\mathcal{C}^f_{osn,charlie}$ ($\top$, *shared*) in state $\mathcal{S}^{t_3}(charlie)$.

Axiom 11. *A term $\mathcal{T}_j$ is satisfiable by a state $\mathcal{S}^T(X) = \langle \Phi, \mathcal{C} \rangle$, denoted $\mathcal{S}^T(X) \Vdash \mathcal{T}_j$, iff $\exists\ \mathcal{T}_i$: $\mathcal{T}_i \in \Phi \cup \mathcal{C}$ and $\mathcal{T}_i \Vdash \mathcal{T}_j$.*

According to Axiom 11, a term is satisfiable by a state if there is a term in the state that satisfies the former term.

Definition 12. *A state $\mathcal{S}^{T_n}(A_j)$ is satisfiable by a state $\mathcal{S}^{T_m}(A_i)$, denoted $\mathcal{S}^{T_m}(A_i) \Vdash \mathcal{S}^{T_n}(A_j)$, iff $\forall\ \mathcal{T} \in \mathcal{S}^{T_n}(A_j)$: $\mathcal{S}^{T_m}(A_i) \Vdash \mathcal{T}$.*

According to Definition 12, a state is satisfiable by another state if every term in the former state is satisfiable by the latter.

By computing the satisfiability relation as explained above, an agent can catch an exception and take an action as it sees fit. We now show that our satisfiability relation is sound and complete. That is, our satisfiability relation signals an exception only when there is an exception (soundness) and if there is an exception it always signals it (completeness).

Theorem 2. *For an agent A, at time T, given a proposition P, $\mathcal{S}^T(A) \nVdash \mathcal{S}^T(A_p)$ iff exception(A, T, P).*

Table 2.2. Examples of recoverability.

(a) $\mathcal{C}^{a}_{osn,charlie}\ (\top,\ shared)\} \Vdash_1 shared$
(b) $\mathcal{C}^{c}_{osn,charlie}\ (newlocation,\ shared)\} \Vdash_2 shared$
(c) $\mathcal{C}^{v}_{osn,charlie}\ (newlocation,\ shared)\} \not\Vdash shared$

2.5. Recoverability

The satisfiability relation enables us to compute exceptions at run time. If there is an exception, a second question comes up. Is the exception recoverable? That is, is it possible to take some steps to put the execution back into the projected state? In order to decide if an exception is recoverable or not, we use a second relation called *recoverability*. This relation is denoted by $X \Vdash Y$, which is read as "Y is recoverable from X".

Recoverability is again first defined among terms and then among states. We define the following levels of recoverability:

K-recoverability: Y is k-recoverable from X, denoted $X \Vdash_k Y$, iff it takes k commitment operations (i.e., state changes) for X to satisfy Y. Definition 13 describes k-recoverability recursively. In the base case, if Y is satisfiable by X, we mean that Y has already been recovered. We call this case zero-recoverable. In the recursive case, we are saying that Y is k-recoverable from X, if there exists X' such that X' is recoverable from X by a single commitment operation (Definition 8) and Y is $k-1$ recoverable from X'.

Definition 13. *Y is k-recoverable by X, iff either Y is satisfiable by X, or there is X'*

such that X' is a successor of X, and Y is (k-1)-recoverable by X. Formally,

- $X \Vdash_0 Y$ *iff* $X \Vdash\!\!\vdash Y$.
- $X \Vdash_k Y$ *iff* $\exists\ X'$*:* $X \gg X'$ *and* $X' \Vdash_{k-1} Y$.

Table 2.2a shows an example of one-recoverability; when the sharing of Charlie's location occurs, the commitment's state changes from active to fulfilled, thus the proposition *shared* will be satisfied. Table 2.2b shows an example of two-recoverability; first Charlie goes to his office, which brings the commitment to the active state (*newlocation* is satisfied). Then, the commitment is fulfilled (*shared* is satisfied).

Non-recoverability: Y is not recoverable by X, denoted $X \nVdash Y$, iff there is no way of going from X to Y. That is, there is no way of realizing the term Y after the term X holds. Definition 14 describes non-recoverability formally.

Definition 14. *Y is not recoverable by X, iff there is no such k that Y is k-recoverable by X. Formally, $X \nVdash Y$ iff $\nexists\ k$: $X \Vdash_k Y$.*

Table 2.2c shows an intuitive example for non-recoverability; once a commitment is violated, then its consequent cannot be reached. This can be verified by inspecting Figure 2.2; the violated commitment state is a terminal state meaning that once a commitment is violated, then the commitment's life-cycle ends (i.e., it stays violated). That is, even if the proposition of the commitment becomes true after the commitment becomes violated, we do not modify the state of the commitment anymore. Thus, a fulfilled commitment is not recoverable from its violated state.

One can easily verify that the recoverability relation is also reflexive, non-symmetric, and transitive.

State recoverability: Next, we describe state recoverability in terms of k-recoverability.

Axiom 12. *A term $\mathcal{T}_j$ is k-recoverable from a state $\mathcal{S}^T$, denoted $\mathcal{S}^T \Vdash_k \mathcal{T}_j$, iff $\exists\ \mathcal{T}_i$: $\mathcal{T}_i \in \mathcal{S}^T$ and $\mathcal{T}_i \Vdash_k \mathcal{T}_j$.*

Similar to term satisfiability, a term is recoverable by a state if there is a term in the state that the former term is recoverable by it.

Definition 15. *A state* $\mathcal{S}^{T_n}(A_j)$ *is k-recoverable from state* $\mathcal{S}^{T_m}(A_i)$*, denoted* $\mathcal{S}^{T_m}(A_i) \Vdash_k \mathcal{S}^{T_n}(A_j)$*, iff* $\forall\ \mathcal{T} \in \mathcal{S}^{T_n}(A_j)$*: either* $\mathcal{S}^{T_m}(A_i) \Vdash\!\!\!\Vdash \mathcal{T}$ *or* $\mathcal{S}^{T_m}(A_i) \Vdash_k \mathcal{T}$*, and* $\exists\ \mathcal{T}_r \in \mathcal{S}^{T_n}(A_j)$*:* $\mathcal{S}^{T_m}(A_i) \nVdash\!\!\!\Vdash \mathcal{T}_r$.

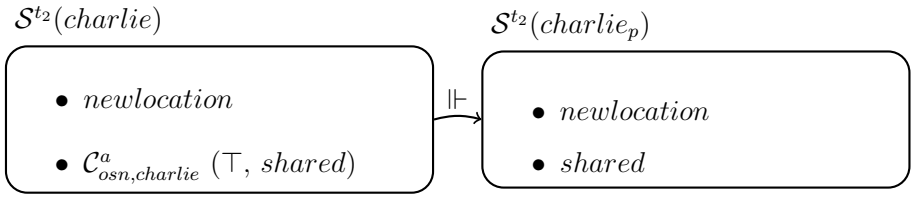


Figure 2.6. State recoverability.

According to Definition 15, a state is recoverable by another state if every term in the former state is either satisfiable or recoverable by the latter, and at least one term is not satisfiable. Figure 2.6 shows an example: the term *newlocation* in state $\mathcal{S}^{t_2}(charlie_p)$ is satisfiable by the term *newlocation* in state $\mathcal{S}^{t_2}(charlie)$. Moreover, the term *shared* in state $\mathcal{S}^{t_2}(charlie_p)$ is recoverable (but not satisfiable) by the term $\mathcal{C}^{a}_{osn,charlie}$ ($\top$, *shared*) in state $\mathcal{S}^{t_2}(charlie)$. Thus, $\mathcal{S}^{t_2}(charlie) \Vdash \mathcal{S}^{t_2}(charlie_p)$.

Now, we describe how to utilize the recoverability relation in order to understand whether an exception is recoverable or not.

Definition 16. *An exception is recoverable for agent A at time T only if* $\mathcal{S}^{T}(A) \Vdash_k \mathcal{S}^{T}(A_p)$.

It is important for the agent to understand whether the exception is recoverable or not. If recoverable, the agent can take proper action to recover from the exception. For example, if Charlie thinks that the sharing should have been completed, and there is a commitment from the OSN operator towards sharing his location that is still active, then the exception signaled by Charlie can be recovered by the OSN operator, via fulfillment of the commitment. In addition, we associate a parameter k with the recoverability relation, to provide a measure of recoverability. As an example, consider

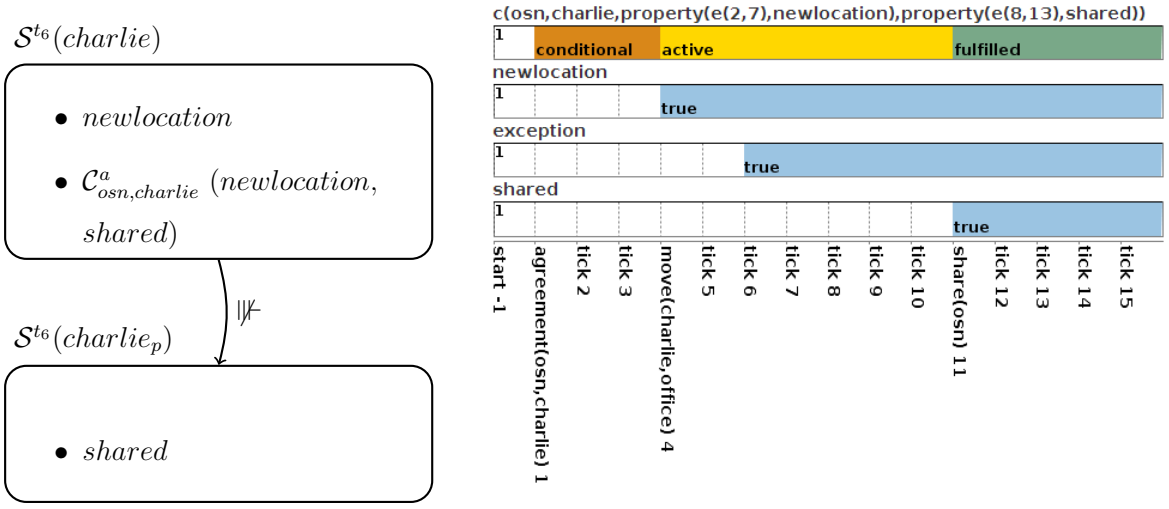
again the above case; the OSN operator needs to perform a single action to satisfy sharing (one-recoverable). This is indeed more desirable than any case where the OSN operator (or another party) has to perform several actions to satisfy Charlie's projection (k-recoverable where $k > 1$). So, in a sense the parameter k acts as a measure to calculate how bad the exception is.

2.6. $\mathcal{REC}$

The Reactive Event Calculus ($\mathcal{REC}$) extends the Event Calculus, which is based on Prolog. $\mathcal{REC}$ has been developed to monitor commitments in run-time. We employ $\mathcal{REC}$ for the reasoner component in Figure 2.1. Note that each agent has a separate $\mathcal{REC}$ engine that it can run at any time throughout the execution. When used for commitment tracking (e.g., monitoring the states of commitments), the $\mathcal{REC}$ engine is generally fed with three types of input:

- *Commitment theory* contains the rules on how commitments are manipulated. This is a shared rule-base for all the agents.
- *Protocol description* contains the protocol rules that describe the consequences of the agents' actions as well as domain facts. This is an agent-dependent as well as domain-dependent rule-base as each agent has a separate protocol description that covers its own view.
- *Event trace* contains the actions that the agents perform throughout time. Like the protocol description, the event traces are also agent-dependent. That is, each agent is aware of the events that are related to it, but does not see the events that might take place among other agents.

Once the $\mathcal{REC}$ engine is run with above input, it produces an outcome that demonstrates the fluents the agent is aware of through time (e.g., states of commitments). This process is often called commitment tracking [28]. However, here, we do not use $\mathcal{REC}$ only for monitoring commitment states, but also to detect exceptions at run-time. Thus, we further integrate the agent's *projections* and an *exception theory*



(a) States (b) j-$\mathcal{REC}$ output

Figure 2.7. Exception: Charlie expects early sharing of his location.

into $\mathcal{REC}$ in order to do so.

Here, we assume some basic knowledge of Prolog. 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)* (or *terminates(Event, Fluent, Time)*) relations. Negation is denoted with \+. A detailed explanation of how $\mathcal{REC}$ manipulates commitment states can be found in [29].

We have implemented the framework in j-$\mathcal{REC}$; a Java-based version of $\mathcal{REC}$ [7,28]. The full implementation can be downloaded from `http://mas.cmpe.boun.edu.tr/ozgur/code.html`, under Section "2. Experiments for Monitoring Interactions". We illustrate the important points in Appendix B.

2.7. Case Study

Let us review several cases to demonstrate the usage of the satisfiability relation for detecting exceptions.

Exception for Charlie: Figure 2.7(a) demonstrates an intuitive case. The left box shows Charlie's state at time t_6, while the right box shows his projected state for the

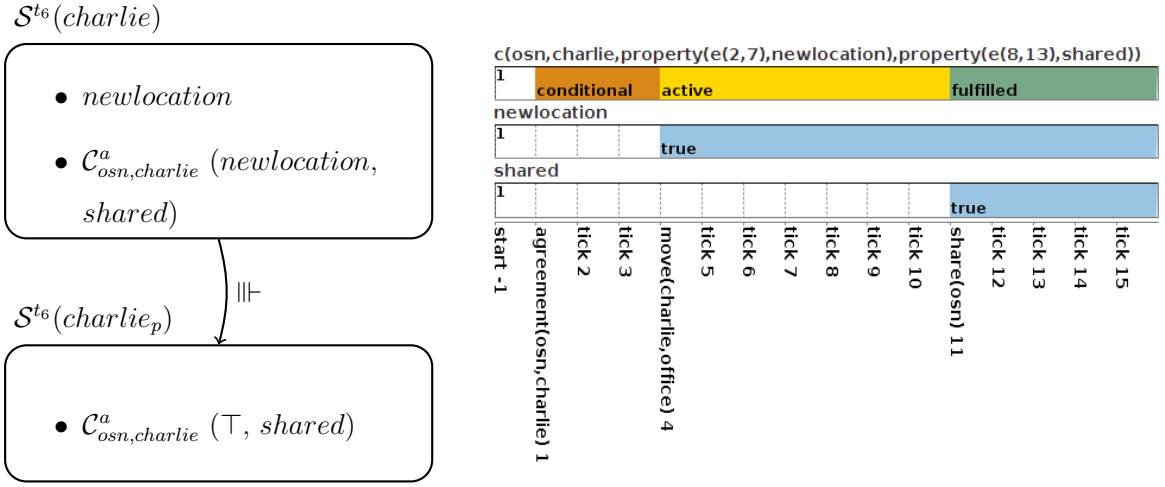
same time point. The following $\mathcal{REC}$ rule describes an exception according to Charlie's projected state:

```
initiates(_, exception, 6):-
        \+ satisfiable(exp(shared, 6)).
```

The *satisfiable(exp(Fluent, Time))* predicate in the rule describes the expectation of the *Fluent* to be satisfied at time point *Time*. So, the complete rule states that if the sharing of Charlie's location is not performed by time t_6 (note the negation sign before the *satisfiable* predicate), then Charlie will signal an exception following the *initiates* relation. Note that the placeholder for the *Event* here is empty since the exception is not caused by the occurrence of a specific event, but rather the unsatisfiability of an expectation. At time t_6, the OSN operator currently has an active commitment to Charlie for sharing his location. However, Charlie expects sharing to be completed. Recall that for a state to be satisfiable, all its terms should be satisfiable (Definition 12). There is only one term in state $\mathcal{S}^{t_6}(charlie_p)$, which is the proposition *shared*. However, none of the Axioms for satisfiability can be applied to satisfy *shared* from the terms in state $\mathcal{S}^{t_6}(ali)$. Thus, Charlie's projected state is not satisfiable. This causes an exception for Charlie according to Definition 11. Figure 2.7(b) shows the output of j-$\mathcal{REC}$ for this case. The horizontal axis shows the timeline of events that have occurred during Charlie's execution. Notice a *tick* event is associated with every non-occupied (i.e., no protocol events) discrete time-point. This is required for $\mathcal{REC}$ to process properly, since it is event-driven, e.g., a new event triggers $\mathcal{REC}$ to process further. The fluents are positioned vertically, and their truth values (and the corresponding states for commitments) are computed according to the events. You can see from the figure that at t_6, *newlocation* is true and the commitment is active, which corresponds to Charlie's state in Figure 2.7(a)[2] . When $\mathcal{REC}$ processes at t_6 with the exception rule above, the fluent *exception* becomes true as Charlie's state fails to satisfy his projection.

No exception for Charlie: Figure 2.8(a) demonstrates a slightly different case. Charlie's state is the same as before. However, this time, he does not expect sharing to

[2]Note that we omit from the state the fluents that represent the stock values of items.



(a) States (b) j-$\mathcal{REC}$ output

Figure 2.8. No exception: Charlie does not expect sharing of his location.

be fulfilled at time t_6. His exception condition can be described by the following $\mathcal{REC}$ rule:

```
initiates(_, exception, 6):-
        \+ satisfiable(exp(active(c(_, _, true, property(e(_, _), shared))), 6)).
```

According to Axiom 9, $\mathcal{C}^a_{osn,charlie}$ (⊤, *shared*) is satisfiable by $\mathcal{C}^a_{osn,charlie}$ (*newlocation*, *shared*). Thus, no exception occurs for Charlie. Figure 2.8(b) shows the output of j-$\mathcal{REC}$ for this case. Notice that the *exception* fluent does not hold since Charlie's projected state is attainable.

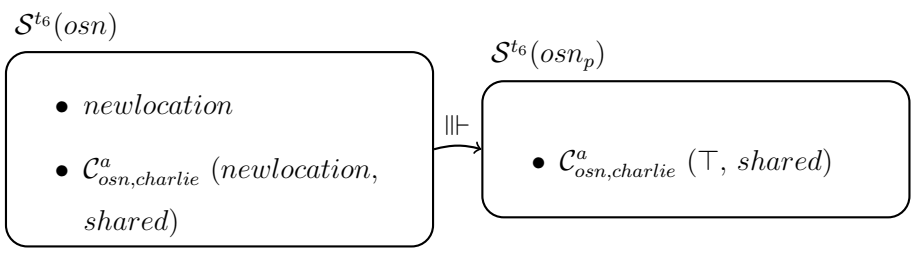


Figure 2.9. Charlie and the OSN operator's projections do not match.

Exception for Charlie, no exception for the OSN operator: Consider Figures 2.7(a) and 2.9 together. There is an exception for Charlie as he expects sharing at time t_6. However, the OSN operator's projection for t_6 is that the commitment to share Charlie's location is still active. The projected state of the OSN operator is satisfiable

by the actual state. Thus, there is no exception for the OSN operator. Note that, in a centralized environment, this would never happen. That is, a central monitor either signals an exception or not. Here, on the other hand, we allow autonomous agents to have their own projections about the future. Accordingly, an exception for one agent might just be an expected situation for another.

Next, we will see some examples of recoverable vs. non-recoverable exceptions.

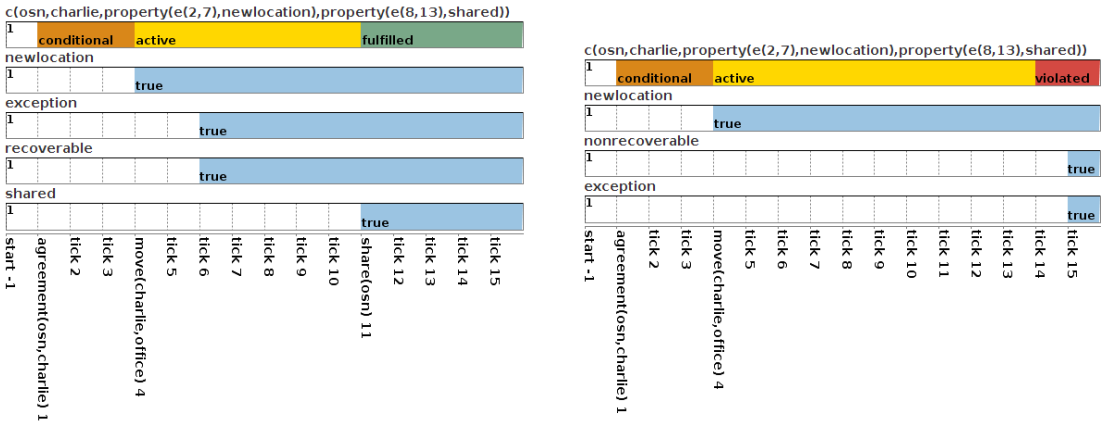
Recoverable exception for Charlie: Consider again Figure 2.7(a). Charlie expects sharing at time t_6, however the commitment towards sharing is still active. Now, the following $\mathcal{REC}$ rules describe a recoverable exception according to Charlie's projected state:

```
initiates(_, exception, 6):-
        \+ satisfiable(exp(shared, 6)).

initiates(E, recoverable, 6):-
        initiates(E, exception, 6),
        kRecoverable(exp(shared, 6)).
```

The *kRecoverable(exp(Fluent, Time))* predicate, which is similar to the *satisfiable* predicate, describes the expectation of the *Fluent* to be k-recoverable at time point *Time*. So, if there is an exception but the projected state is still k-recoverable from the actual state, then Charlie will understand that the exception is recoverable. Indeed, although the commitment is not fulfilled and thus causes an exception, there is still chance of recovery since the commitment is active. That is, the OSN operator can still perform the sharing. Figure 2.10(a) shows the output of j-$\mathcal{REC}$ for this case. Note that the fluents *exception* and *recoverable* both become true at time t_6.

Non-recoverable exception for Charlie: Assume that the situation in Figure 2.7(a) has been slightly changed; Charlie expects sharing at time t_{15}, however sharing does not occur. The following $\mathcal{REC}$ rules describe a nonrecoverable exception according to Charlie's projected state:



(a) Recoverable: j-$\mathcal{REC}$ output (b) Non-recoverable: j-$\mathcal{REC}$ output

Figure 2.10. Recoverable vs. non-recoverable exception.

```
initiates(_, exception, 15):-
        \+ satisfiable(exp(shared, 15)).

initiates(E, nonrecoverable, 15):-
        initiates(E, exception, 15),
        nRecoverable(exp(shared, 15)).
```

Here, the *nRecoverable(exp(Fluent, Time))* predicate describes the non-recoverability of the *Fluent* at time point *Time*. The commitment is violated at time t_{14}. When Charlie checks for an exception, he sees that there is no chance of sharing anymore. This is because the violated commitment state is terminal; it cannot be fulfilled later. Figure 2.10(b) shows the output of j-$\mathcal{REC}$ for this case. Note that the fluents *exception* and *nonrecoverable* both become true at time t_{15} as the commitment is violated.

In this chapter, we have proposed a satisfiability relation that can be used to compare agents' states. When used to compare an agent's state with its projected state, the outcome tells whether there is an exception for the agent or not. That is, if the agent's projected state is not satisfiable by its current state, then the agent signals an exception. In addition, the agent may verify its compliance to the protocol it is executing by consistently comparing its current state to its projected states. This way, the agent can identify at which point of the protocol there has been a problem.

We extend the concept of exceptions that are described in the literature. Often, an

exception is considered identical to a contract violation. While we accommodate that perspective, we also take into account the agent's expectations from their contracts. Note that we would normally signal an exception when a commitment is still active when it should be fulfilled. However, by letting the projected state to be constructed according to the agent's projections, an insignificant exception is avoided, e.g., when Charlie is already aware that the sharing of his location will be delayed due to the earthquake.

3. PREDICTION OF EXCEPTIONS

So far, we have seen a method that allows agents to detect exceptions, by comparing their current states with their projections. While detecting that something has gone wrong based on the actual events that took place in the world is important, identifying that an exception might occur if the agent takes a certain course of action could be as important. While the former allows agents to deal with the exceptions that have taken place, the latter enables agents to possibly avoid exceptions before they even take place. Following the scenarios in the previous chapter, assume that Charlie had known that if he were to be with Sally, then Linus would have identified his location. Then, Charlie would have cancelled his meeting with Sally, if he would not want his location to be disclosed. This gives Charlie more control over his actions. He could choose a safer path, e.g., where his commitments would not be violated. In this chapter, we describe how exceptions can be predicted beforehand, either by taking into account others' projections, or by introducing some extra information into the reasoning (e.g., assumptions about other interactions in the environment).

First, we propose a method for the agent to understand that there is an exception even though its current state satisfies its projection. That is, the agent compares its projection with others for the same situation in order to find out a conflict. We use model checking to generate such projections. Then, we extend this method by relaxing the agent's model of its environment, enabling it to put in some assumptions regarding possible interactions of other agents in the system. This way, the agent can predict possible future states of the system, and identify whether there will be a contract violation in any of them.

3.1. Comparing Projections

Even though the agent's current state satisfies its projection, there may still be exceptions because the agent has misinterpreted its environment. This may be due to any missing information the agent has, which in turn affects the projection it makes.

To predict such exceptions, it is better that the agent compares its projection with others, and sees whether they are expecting the same outcome.

In many situations, other agents might have extra information regarding the agent's interactions indirectly. This extra information may enable them to make better judgements on the outcomes of the agent's commitments. That is, they may project a future state of the environment that cannot be projected by the agent given the information it has. If we continue on the privacy domain, this may correspond to a scenario, where a relation does not include the agent itself, but it may still affect the privacy of the agent. Consider a document sharing scenario where only friends have access to shared documents. Assume Charlie and Sally are friends, but Charlie and Linus are not. Thus, when Charlie shares a document, then Sally will be able to access it while Linus cannot. Now, if Sally and Linus are also friends, then Linus can access the document through Sally. However, as far as Charlie is concerned, his document is only visible to Sally. Here, the OSN operator has more information on the current state of the system than Charlie himself. Thus, it can make a better projection on whether his commitment will be violated or not. Considering the above example, Charlie thinks that Linus cannot access his document. However, when the OSN operator makes a projection regarding Charlie's state, it finds out that Linus is also able to access the document.

Next, we briefly describe model checking. Then, we propose a method for generating projections based on model checking that identifies such privacy violations.

3.2. Model Checking

Model checking is a computational method to automatically verify whether a given property holds for a system [30, 31]. The system under consideration is modeled as a state transition graph in some formal language and the property that is aimed to be verified is represented as a logic formula in a suitable language, such as linear temporal logic (LTL) or computation tree logic (CTL) [32]. Given the system model and the logic formula of the investigated property, a model checking algorithm checks

whether the system model satisfies the desired property.

In our work we use NuSMV, a state of the art model checker based on binary decision diagrams [33]. A NuSMV model defines a set of variables and how these variables evolve according to possible executions of the modeled system. For instance, a variable may represent that two users are friends in a social network and evolution of this variable can be modeled according to the operations provided on this relation by the modeled social network. In other words, a NuSMV model defines the underlying operational mechanism of the considered system. Once there is such a model, NuSMV can be used to verify certain properties of the model. For instance, a property of the social network about privacy could be that the location of a user is not revealed to users that are not friends.

NuSMV uses CTL to represent properties that are aimed to be verified. CTL is a branching time logic, where the future is modeled as a tree structure in which each branch corresponds to a possible different future. CTL formulas are built up from a set of propositional variables, the usual logic connectives and a set of temporal modal operators. The first type of temporal operators are A and E, which quantify over paths. A stands for *all* and means that the quantified formula has to hold on all paths. E stands for *exists* and means that the quantified formula has to hold at least on one path. The other four temporal operators X, F, G and U are specific to a single path. X stands for *next* and means that its bounded formula has to hold at the next state of the given path. F stands for *eventually* and means that the bounded formula has to hold eventually at some future state(s) of the given path. G stands for *globally* and means that the bounded formula has to hold at all future states of the given path. Finally, U stands for *until* and it is the only binary operator. It means that the first formula bounded to U has to hold until the second formula starts to hold.

3.3. Generating Projections

It is most important that the agent makes a projection as accurate as possible, in order to be able to identify exceptions. Here, we propose a method for agents to gen-

erate projections based on a given state of the world. These projections tell the agent what to expect if it reaches such a state during execution. Thus, generating accurate projections will help the agent identify future states that may lead to exceptions.

Accordingly, we have developed $\mathcal{PROTOSS}$ [10], a run time tool to generate projections for the agent, based on its current state of the world. The main technique underlying our approach is model checking, which given a model of the system checks whether certain properties hold. For the privacy domain, we use $\mathcal{PROTOSS}$ to detect possible privacy breaches in online social networks. Our system model represents the privacy agreements of a system with users as well as the relations of users formally. Using this model, we can check interesting properties such as whether a certain user's content will ever reach a certain individual even when that individual is not an acquaintance or whether the relations among individuals can lead to unwanted information to be revealed to certain individuals for which the content was not intended. We show that $\mathcal{PROTOSS}$ can detect subtle information leakages that are not easy to detect in conventional online social networks. We demonstrate these over scenarios.

3.3.1. Privacy-Aware OSN Architecture

We are interested in online social networks (OSNs) that are administered by an OSN operator. Each user can post content as it sees fit. The content could vary. One can post personal information such as her location, the people she is with, and so on as well as links to news, jokes and so forth. Our primary aim in this work is the first set of information since we are interested to see how private information can float in the system.

Since it is a social network, users are related to each other. As in newer social networks, users can be related to each other through different relations. For example, Charlie could be a friend of Linus but a colleague of Sally. These relations identify how much and of what type of content would be shared with other users. For example, Charlie would share his whereabouts with his friends, but may not want to share this with colleagues. Essentially, the OSN operator is responsible for ensuring that these

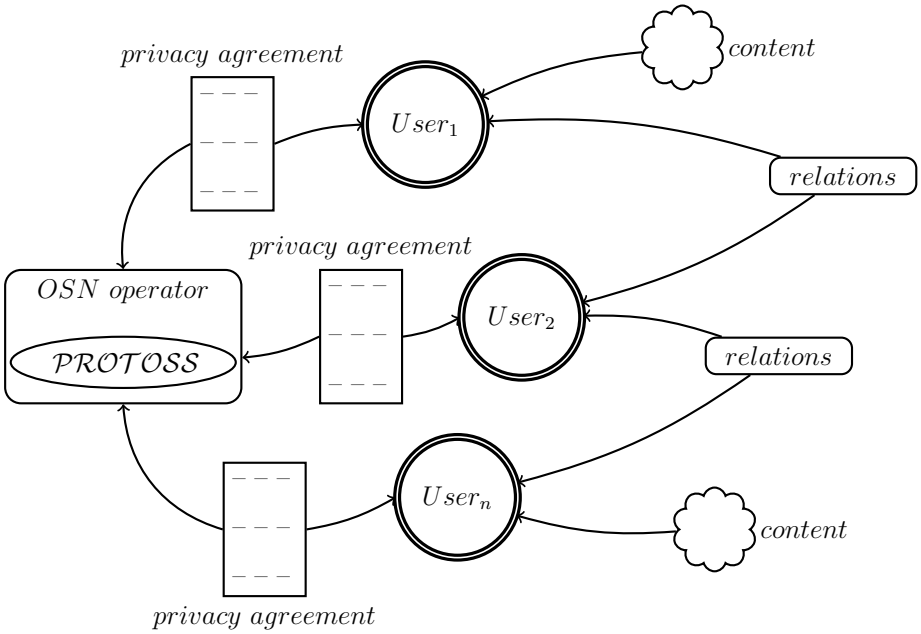


Figure 3.1. Privacy-aware OSN architecture.

expectations are met. That is, the OSN operator is supposed to ensure that only the users with the right privileges are shown private content.

Figure 3.1 demonstrates the architecture that we use to verify privacy agreements. The circles represent the users. The users are connected to each other through relations and they provide content.

Among each user and the OSN operator, there exists a privacy agreement. This is an agreement that contains clauses about which relations are entitled to which privileges. For example, an agreement between Charlie and the OSN operator can state that all friends of Charlie are entitled to see his location. This is not a static agreement. That is, as Charlie creates more relations with other users, this privacy agreement is updated accordingly. Since the OSN operator is responsible for realizing the clauses in the privacy agreement, it needs a mechanism to check whether it can honor the agreement. We call this the privacy checker ($\mathcal{PROTOSS}$).

$\mathcal{PROTOSS}$ uses the network information as well as the agreement information to decide whether the agreements can be honored in the system. For example, if Charlie does not want any of his colleagues to see his location and if he has identified Linus

both as a colleague and a friend, then OSN will end up disclosing the location to a colleague. In this case, OSN should let Charlie know that a colleague will hear of his location and maybe let him decide what to do.

3.3.2. Running Example

Consider the following social network with specific components as outlined above:

(i) Users: We have three users of the OSN; *charlie*, *sally*, and *linus*.

(ii) Relations: The users of the system can initiate relations among themselves. This is typical of online social networks [34]. We assume that the following relations exist:

- $colleague(X, Y)$: Users X and Y are colleagues.
- $friend(X, Y)$: Users X and Y are friends.

We instantiate them as follows: $friend(sally, linus)$ and $colleague(charlie, linus)$. We assume that if a relation is not instantiated, then it doesn't hold. Hence, for example, one can conclude that *sally* and *charlie* are not friends.

(iii) Content: To complement the relations of the system, we have two types of content. These are:

- $location(X, W)$: User X is at location W.
- $with(X, Y)$: User X is with user Y.

(iv) OSN operator: There is a single OSN operator in the system. It is responsible for displaying appropriate information to the users based on its own policies; i.e., decides what will be visible to each user. For example, $visible$ predicate below describes whom to grant access to:

- R_1: $visible(with(X, Y), Z) \leftarrow friend(X, Z) \vee friend(Y, Z)$: This rules states that any friend of the user X or Y knows who X or Y is with.
- R_2: $visible(location(X, W), Y) \leftarrow friend(X, Y)$: This rule states that any friend of the user X knows the location of X.

(v) Privacy agreements: Privacy agreements contain the clauses for disclosing information. We essentially represent these through commitments among the OSN operator and a particular user. The commitments capture how visibility will be released to other parties. For each of the scenarios below, we use a subset of the following commitments:

- C_1: $C(osn, sally, friend(sally, X), visible(with(sally, Y), X))$: The OSN operator commits to the user *sally* that her friends will be able to see who she is with.
- C_2: $C(osn, sally, friend(sally, X), visible(location(sally, W), X))$: The OSN operator commits to *sally* that her friends will be able to see where she is.
- C_3: $C(osn, charlie, colleague(charlie, X), \neg visible(location(charlie, W), X))$: The third commitment is slightly different from the first two. Here, the OSN operator commits to the user *charlie* that his colleagues will not see where he is.

Next, we describe several scenarios that might occur in such a system. The OSN operator, three users, two relations, and two contents all exist in the below scenarios.

First, we begin with a setting in which no location information is passed among the users. Example 3.1 demonstrates this case.

Example 3.1. *Consider a setting where rule R_1, and commitments C_1 and C_3 exist. In this setting, friends know who each other is with, but they do not know where they are. Hence, if Charlie is in Montreal for a conference, then Linus should not be shown this information.*

Next, we extend the first setting where we allow location information to be passed

among users. Example 3.2 demonstrates this case.

Example 3.2. *Consider a setting where rules R_1 and R_2, and commitments C_1, C_2, and C_3 exist. In this setting, friends both know who each other is with, and where they are. Hence, if Charlie is in Montreal, then Linus will have access to this information, if they are friends.*

Now, we will look at settings where all the rules and commitments exist, and different combinations of relations among the users may result in different outcomes for the commitments. Examples 3.3 and 3.4 demonstrate such cases.

Example 3.3. *Consider a setting where charlie and linus are colleagues, sally and linus are friends, and charlie is with sally. Without knowing the rest of the network, we can infer the following from this setting:*

- *linus should not know where charlie is since they are colleagues,*
- *linus knows where sally is and who she is with since they are friends.*

Example 3.4. *Consider a setting where only charlie and linus are colleagues, and charlie is with sally. There are no other relations among the users or any content information. In this setting, charlie's privacy agreement states that linus cannot know where charlie is.*

3.3.3. *PROTOSS*

The reasoning necessary for the examples above are done within the privacy checker of the OSN operator (Figure 3.2). *PROTOSS* is a tool to realize this reasoning. As input, it uses the privacy agreements of the users, user relations, the content they upload as well as some inference rules. Using this input state, it decides whether a certain privacy property that is of concern will be violated. This corresponds to the projection of the agent that is generated for this input state. Using this projection, the agent identifies whether there will be an exception.

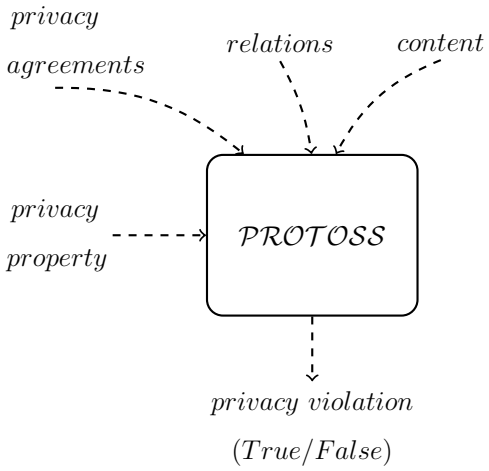


Figure 3.2. $\mathcal{PROTOSS}$.

Figure 3.3 shows a screenshot of the $\mathcal{PROTOSS}$ interface, which is built in Java[3] . From the interface, we can create the social network (together with its relations) according to the number of agents set or we can simply upload an existing social network specification, similar to the one in the running example. Once the model of the social netwoek is created (on the left pane), we can check wheter the properties of interest are satisfied by the model. After the execution is completed, the output of the check is shown with relevant performance statistics (on the right pane). This output specifies whether the property of interest (e.g., whether OSN's commitment to hide a user's location) can be violated or not in a given social network. A user can then use this output to decide its actions.

The $\mathcal{PROTOSS}$ engine uses NuSMV model checker as a core component. However, NuSMV is not by itself capable of checking models with commitments in them. Hence, we have first introduced a commitment module into the NuSMV model checker, based on Telang and Singh's work [35].

Figure 3.4 shows the commitments module that we use to describe commitments. We have modeled two types of commitments in order to account for both

- regular consequents that the commitment is violated when the consequent is

[3]The full implementation can be downloaded from `http://mas.cmpe.boun.edu.tr/ozgur/code.html`, under Section "4. Experiments for Model Checking Privacy Agreements".

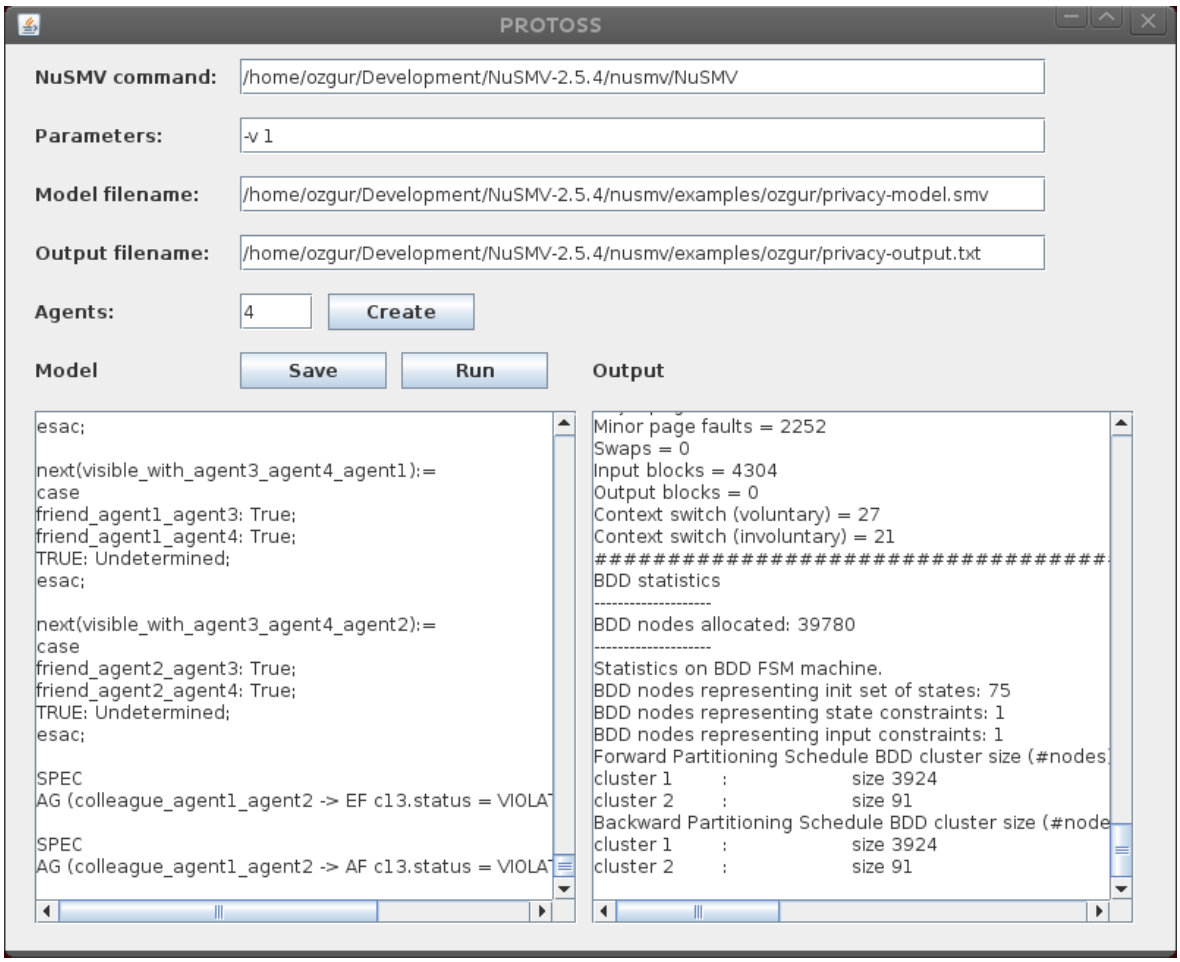


Figure 3.3. *PROTOSS* interface.

unsatisfied, e.g., the OSN operator commits to make sure friends can see where each other is, and

- negated consequents that the commitment is violated when the consequent is satisfied, e.g., the OSN operator commits to make sure colleagues cannot see where each other is.

Below are three instances of the commitments module, that represent the commitments given in the running example.

```
c11: commitment(friend_sally_linus, visible_with_charlie_sally_linus);

c5: commitment(friend_sally_linus, visible_location_sally_linus);

c15: neg_commitment(colleague_charlie_linus, visible_location_charlie_linus);
```

```
MODULE commitment(ant, cons)

CONSTANTS CONDITIONAL, ACTIVE, FULFILLED, VIOLATED;
DEFINE
 status:=
  case
   !ant: CONDITIONAL;
   ant & cons = Undetermined: ACTIVE;
   ant & cons = True: FULFILLED;
   ant & cons = False: VIOLATED;
  esac;

MODULE neg_commitment(ant, cons)
CONSTANTS CONDITIONAL, ACTIVE, FULFILLED, VIOLATED;
DEFINE
 status:=
  case
   !ant: CONDITIONAL;
   ant & cons = Undetermined: ACTIVE;
   ant & cons = False: FULFILLED;
   ant & cons = True: VIOLATED;
  esac;
```

Figure 3.4. Commitments module.

Inference rules are an important aspect of $\mathcal{PROTOSS}$. An example inference rule is the following:

```
visible(location(Y, W1), Z) ←
        visible(with(X, Y), Z) ∧ visible(location(X, W2), Z)
```

This rule states that if X and Y are together and this fact is visible to Z, then when Z knows the location of X, he will also know the location of Y. These rules define semantic relations among concepts in the real world so that $\mathcal{PROTOSS}$ can make further inferences beyond its privacy agreement.

Given the above as input, $\mathcal{PROTOSS}$ can then check whether a privacy condition of concern takes place or not. Below we give two example properties. Both of these properties are phrased from *charlie*'s point of view. The first property (P_1) checks whether there is a chance that the OSN operator's commitment to *charlie* is

violated at some point after it has become active. Remember that this commitment states that *charlie*'s location is not going to be visible to his colleagues. A violation of this commitment means that *charlie*'s privacy may be jeopardized. The second property (P_2), however, checks whether the commitment will be violated every time it has become active. That is, no matter what happens during run-time, *charlie*'s privacy will be violated.

```
SPEC
 AG (colleague_charlie_linus -> EF c15.status = VIOLATED);

SPEC
 AG (colleague_charlie_linus -> AF c15.status = VIOLATED);
```

3.3.4. Case Study

Here, we simulate the scenarios described in the running example from the point of view of the user *charlie*. Then, based on the outcome produced by $\mathcal{PROTOSS}$, we comment on how these settings have an effect on the privacy of *charlie*. Table 3.1 shows the results of our execution. False means that the property does not hold for that example, while true means the property holds. Since we are checking whether a commitment is violated, true corresponds to a violation and thus breach of privacy.

Table 3.1. Outcome of experiments.

Experiment Setting	Property P_1	Property P_2
Example 3.1	False	False
Example 3.2	True	False
Example 3.3	True	True
Example 3.4	False	False

When we run $\mathcal{PROTOSS}$ on setting described in Example 3.1, we see that both formulas return false (Table 3.1). That is, the commitment C_3 (c15 in the NuSMV code) will not be violated in this setting, meaning that the location of *charlie* will not be visible to a colleague of his. This is an expected outcome, since there is no way of

knowing where another user is (since rule R_2 does not exist). This is an example of compile-time verification where the OSN operator knows that its commitments are safe with this set of relations. The actions of the users will not jeopardize the agreements.

When we run $\mathcal{PROTOSS}$ on the setting described in Example 3.2, we see that the first formula returns true and the second formula returns false. That is, while the OSN operator's commitment C_3 will not be violated every time, there is indeed a progression of events that will lead to the violation of C_3. This means that C_3 is no longer completely safe for the OSN operator. A simple case that depicts the violation is the following: Assume *charlie* is a colleague of *linus*. If they are also friends, then *charlie*'s location will be visible to *linus*. Thus, the C_3 will be violated.

```
next(colleague_charlie_linus):= TRUE;

next(friend_sally_linus):= TRUE;

next(with_charlie_sally):= TRUE;

next(...):= {TRUE, FALSE};
```

The above NuSMV code segment represents the setting described in Example 3.3. Note that *next(...)* is just a shortcut for every other variable being randomly assigned during state progression. This setting corresponds to the second case described above for Scenario 2. When we run $\mathcal{PROTOSS}$ on this setting, we see that both formulas return true. This means that this combination of relations and contents lead to the violation of commitment C_3 no matter how other relations are formed. This time the violation can be detected based on relations and inference rules. A case that describes this situation is the following: Assume *charlie* is a colleague of *linus*, and *linus* is *sally*'s friend. Thus, *linus* knows where *sally* is. Now, if *charlie* is with *sally*, then *linus* will also know where *charlie* is (according to the inference rule), which again violates C_3.

Note that if we were only checking the privacy agreements among the users and

the OSN, we would conclude that the location of *charlie* is not being revealed, since the OSN operator does not explicitly make this information public. However, with the help of inference rules, we can reason on the concept of privacy and decide that the location will be revealed through other means.

```
next(colleague_charlie_linus):= TRUE;

next(with_charlie_sally):= TRUE;

next(...):= {FALSE};
```

The above NuSMV code segment represents the setting described in Example 3.4. When we run $\mathcal{PROTOSS}$ on this example, we see that both formulas return false. This means that commitment C_3 will not be violated as long as the relations and content other than *charlie* and *linus* being colleagues and *charlie* being with *sally* remain false.

Note that while we check the above two properties as examples, the CTL language allows a richer variety of privacy properties to be specified. In addition to these, we could specify properties that check whether a commitment is going to be violated immediately in the next state of execution or whether the location will not be revealed until a particular proposition or relation starts to hold in the system. Such properties can be useful when initiating a new relation in the system.

Performance Results: Now, we briefly study the performance results related to the workings of NuSMV on our privacy models. Table 3.2 shows the performance of NuSMV for Example 3.4 on an Intel Core 2 Duo 2.4 GHz computer with 4 GB of memory running Ubuntu 11.10 32-bit OS. The results demonstrate the number of states, the memory used, and the time consumed based on 3 to 15 agent models. Note that the time and memory consumption increases exponentially since the NuSMV model grows based on the number of agents, i.e., the number of possible relations among the agents increase exponentially with the number of agents. For small networks, the computation times lie within reasonable amounts. However, with large networks, the

Table 3.2. Performance results for Example 3.4.

#Agents	#States	Memory	Time
3	4.4 K	2.1 MB	0.01 s
4	27.8 K	2.9 MB	0.03 s
5	109.8 K	20.6 MB	0.11 s
6	172.7 K	30.7 MB	0.15 s
7	374.9 K	36.7 MB	0.32 s
8	879.1 K	54.4 MB	0.68 s
9	1.4 M	72.5 MB	1.93 s
10	2.8 M	80.9 MB	13.11 s
13	5.7 M	173.7 MB	63.23 s
15	39.8 M	732.1 MB	228.1 s

checking time can become intolerable for quick decisions. Hence, it is important to optimize these results for large networks by pruning the state space appropriately so that only a relevant part of the OSN is investigated for each decision.

3.4. Predicting Exceptions

Now, we are going to change the way that $\mathcal{PROTOSS}$ can be used, in order to enable agents to predict privacy violations at an earlier state. Given a current state of the system, the agent adds some assumptions on top of the relations that are known to it. These assumptions are about the relations of other agents that the agent thinks are either true or false. The following list describes what can be assumed by the agent about its environment:

- The relations among other agents, e.g., X and Y are friends, or X is a colleague of Y.
- The content related to an agent, e.g., X is in location W, or X is with Y.

Every other relation that is unknown to the agent are set as free variables that can be instantiated to either true or false in different runs during the model checking process. Once the agent has created a state by describing known, assumed, and unknown relations, this state is verified for commitment violations. Note that since some relations are set to unknown, not only this particular state is checked, but several variations of it are also verified.

Consider the setting given in Example 3.3 from the privacy domain. Charlie knows its own relations with others. Thus, he sets $colleague(charlie, linus)$ to true. Now, if Charlie thinks that *sally* and *linus* are friends, then he also sets the relation $friend(sally, linus)$ to true. Charlie set every other relation that either he is not involved in, or he does not know about, as free variables. For example, $with(sally, linus)$ is not set to true or false, but left for $\mathcal{PROTOSS}$ to be decided during execution. Given this setting, $\mathcal{PROTOSS}$ verifies that Charlie's commitment is violated. In addition, it tells Charlie the state that leads to the violation. In this case, the additional information that Charlie has not given to $\mathcal{PROTOSS}$, and has lead to the violation is $with(charlie, sally)$. This state is also a projection of Charlie, not only based on the current state, but also depending on some assumptions.

Accordingly, we provide a user interface where prediction can be performed using different settings. Once the prediction program is run, it checks the possible future states generated through the free variables, and tells whether there will be a commitment violation. Moreover, whenever a violation is possible, the program interface presents the relations that will lead to the violation.

3.4.1. Counter Example Processing

Figure 3.5 shows a screenshot of the prediction interface that extends $\mathcal{PROTOSS}$. It is built in Java, and once a NuSMV model is selected, it shows the relations and the formula to be verified. The user can alter the relations, creating a prediction setting. This setting will then be checked for violations according to the commitment given in the formula. The output shows the combination of relations that will lead to the

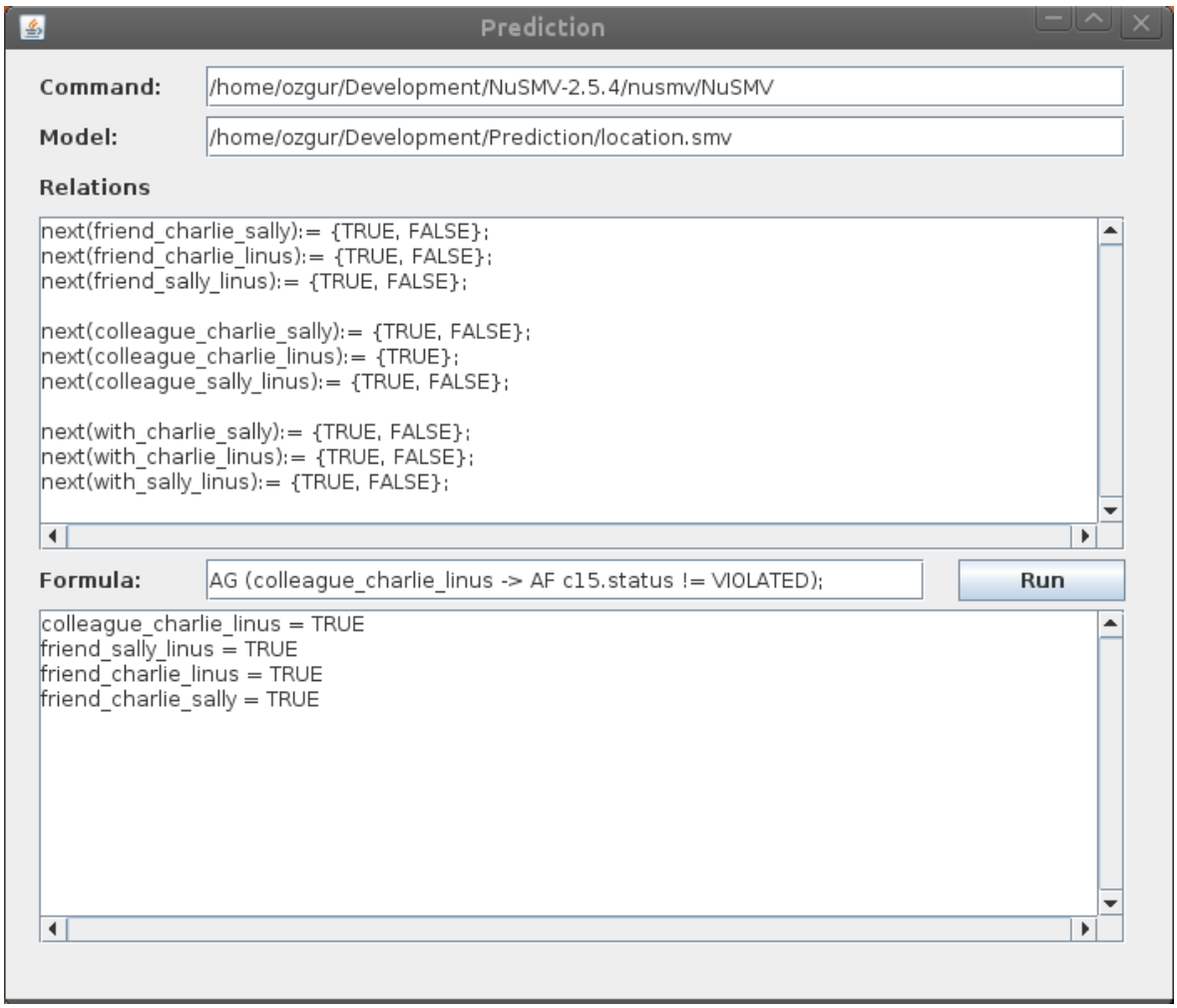


Figure 3.5. Prediction interface.

violation, if any violation is to occur.

3.4.2. Case Study

Now, we describe the workings of the prediction interface on two different settings.

Disclosed location scenario: Figure 3.6 shows the output of the prediction program that is run on a setting described for the disclosed location scenario. Now, when providing the setting for prediction, we specifically set all the relations of *charlie* to either true or false. Because, it is reasonable to assume that *charlie* knows about all the relations that he is involved in. So, *charlie* and *linus* are colleagues, but they are not friends. In addition, *charlie* and *sally* are friends, but they are not colleagues.

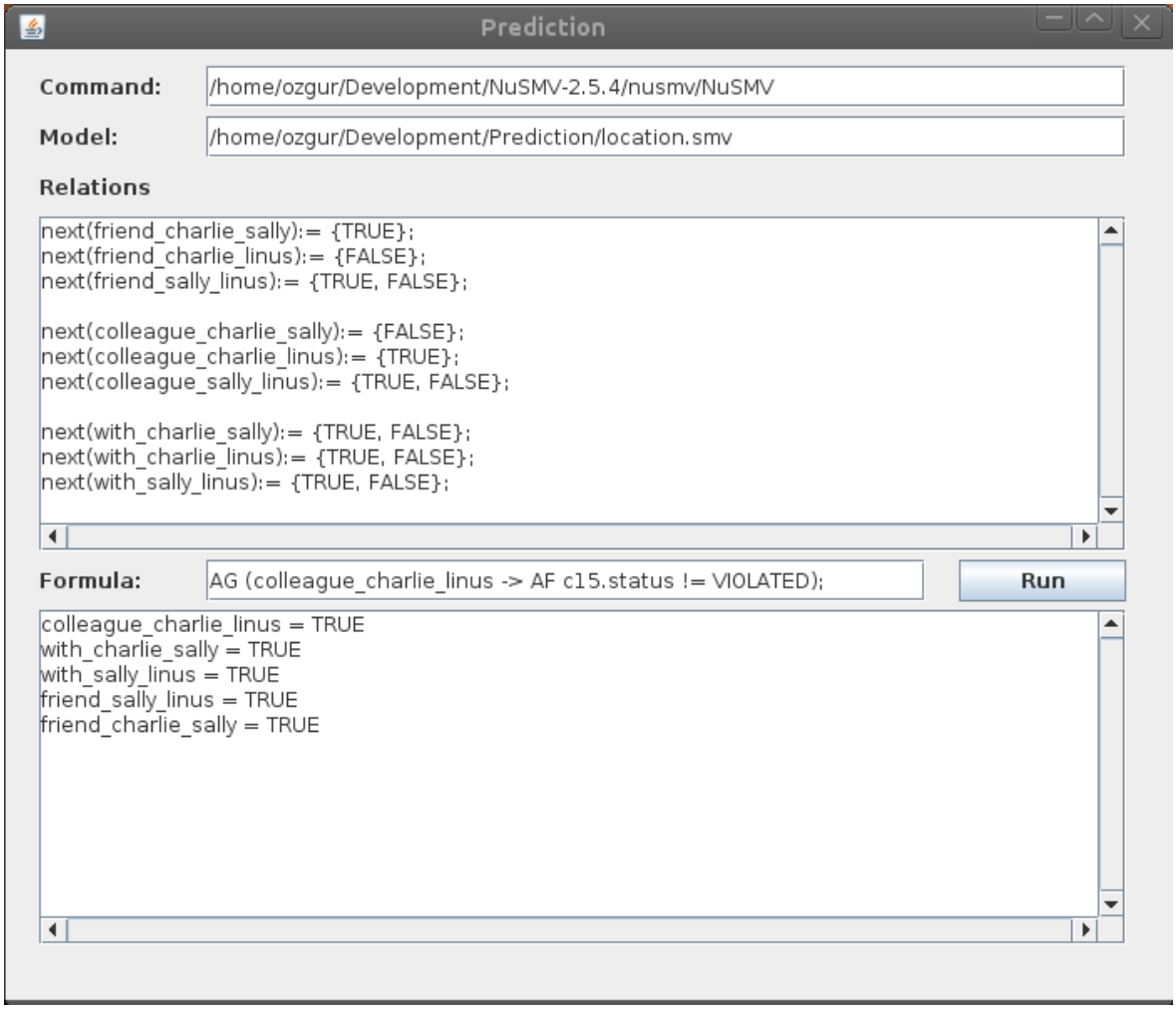


Figure 3.6. Prediction for disclosed location scenario.

This describes the actual state of *charlie*. Here, *charlie* does not give any additional information about the environment (e.g., assumptions). The other relations that are unknown to *charlie* are set as free variables, thus, $\mathcal{PROTOSS}$ sets them to either true or false in different runs. When we run the prediction program with this setting, it generates a possible scenario that will lead to the violation of *charlie*'s commitment. According the scenario, if *sally* and *linus* are friends, and *charlie* is with *sally*, then *charlie*'s location will be visible to *linus*. This corresponds to the setting given in Example 3.3. However, note that here we do not give $\mathcal{PROTOSS}$ the two relations that cause the violation, $friend(sally, linus)$ and $with(charlie, sally)$. Rather, $\mathcal{PROTOSS}$ tells us the relations.

Document sharing scenario: Now, we introduce another setting where prediction

is useful. Figure 3.7 summarizes the NuSMV model for the friend-only document sharing scenario. There are four users; x, y, z, and w. The only relation available in the system is the friend relation. The network provider commits to the user x, that if x and z are not friends, then z cannot access x's documents.

The inference rules that are given in the ASSIGN block of the NuSMV code segment describe how a document of x will be visible to z. There are four rules that describe how the variable $visible\_X\_Z$ is evaluated after every state change in the system, each depending on the relations among other agents as well as x's own relations.

The formula basically aims at identifying whether it is possible for x's commitment to be violated when x and z are not friends. If the commitment is violated, then it means that z can access x's shared document even though they are not friends. Since the formula is written in such a way that it asks whether the commitment will not be violated in any execution, the prediction program will search for all possible executions, and try to prove that the commitment will not be violated in any of those executions. If it fails to prove so, it will present a counter-example to the fact that the commitment is not violated. That is, it will give a trace of an execution in which the commitment is violated. Looking at this trace, an agent can understand which relations can cause the violation of the commitment.

Figure 3.8 shows the output of the prediction program that is run on a setting described for the document sharing scenario. Here, x and w are friends, and x does not know about any other relations in the system. When we run this setting with the prediction program, we see that if w and y are friends, and y and z are friends, then x's document will be visible to z even though they are not friends.

In this chapter, we have proposed a method to allow an agent to further detect exceptions even when it does not expects anything wrong from its point of view. The method takes into account other agents' projections for the same situation, since they may have a wider perception of the environment. In addition, we proposed another

```
VAR
 friend_X_Y: boolean;
 friend_X_Z: boolean;
 friend_X_W: boolean;
 friend_Y_Z: boolean;
 friend_Y_W: boolean;
 friend_Z_W: boolean;

 visible_X_Y: {Undetermined, True, False};
 visible_X_Z: {Undetermined, True, False};
 visible_X_W: {Undetermined, True, False};

 c: neg_commitment(!friend_X_Z, visible_X_Z);

ASSIGN
 next(visible_X_Z):=
  case
   friend_X_Y & friend_Y_Z & !friend_X_Z: True;
   friend_X_W & friend_Z_W & !friend_X_Z: True;
   friend_X_Y & friend_Y_W & friend_Z_W & !friend_X_Z: True;
   friend_X_W & friend_Y_W & friend_Y_Z & !friend_X_Z: True;
   TRUE: Undetermined;
  esac;

SPEC
 AG (!friend_X_Z -> AF c.status != VIOLATED);
```

Figure 3.7. Friends model.

method, that extends the former, to enable agents to predict possible exceptions before they occur.

Predicting an exception beforehand gives the agent control over its actions. For example, in the privacy domain, if the agent already knows that creating a relation with another agent will jeopardize its privacy, then it will probably choose not to have that relation. Moreover, the agent may choose not to get involved in a commitment, if it knows that there is a strong chance of the commitment being violated.

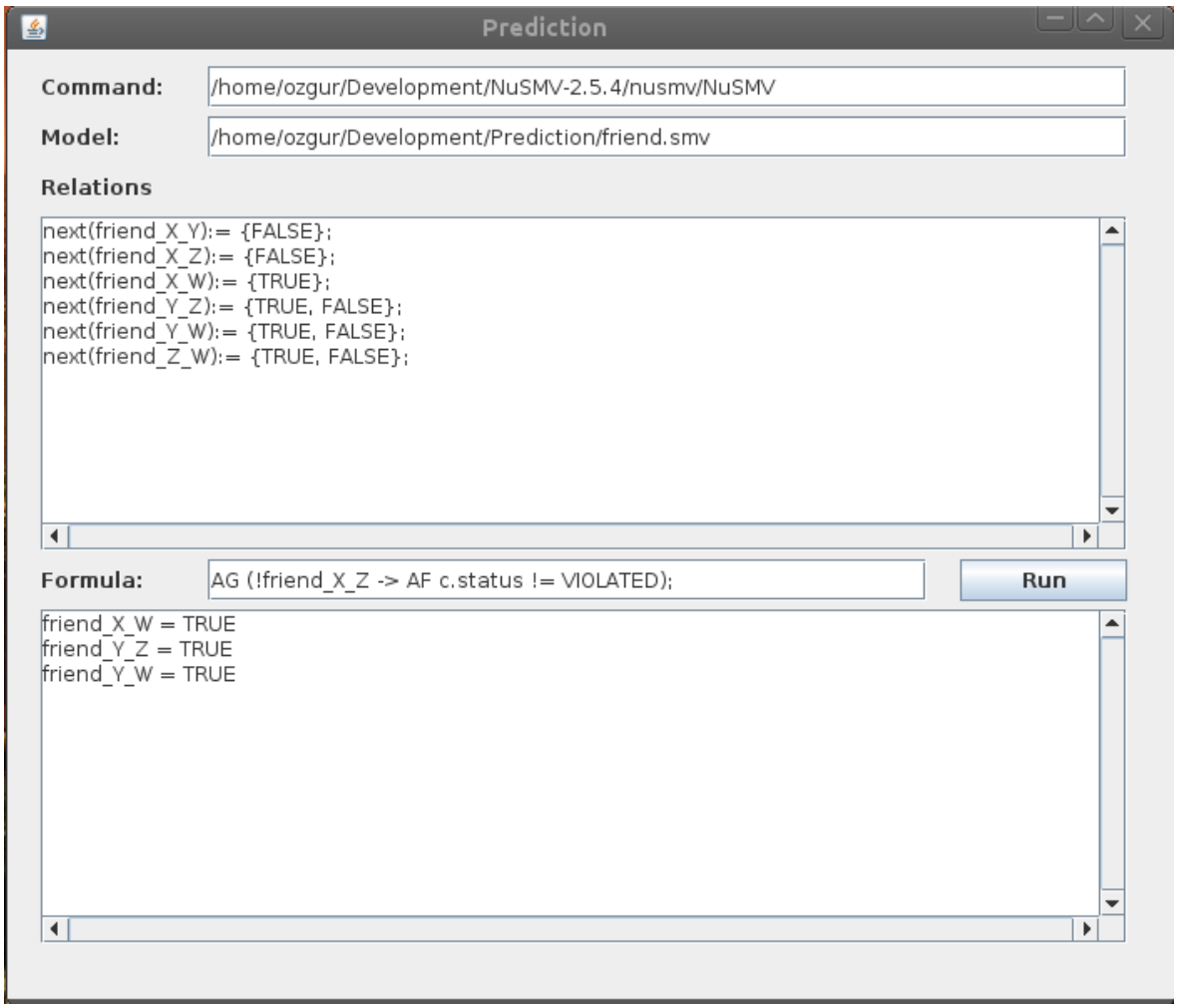


Figure 3.8. Prediction for document sharing scenario.

4. DIAGNOSIS OF EXCEPTIONS

The next phase in handling an exception, after it is detected or predicted, is to diagnose what has gone wrong. That is, once an agent has identified that there is something wrong with its execution, it needs to find out what has caused the problem, in order to take proper action towards fixing that problem. So far, we have seen how exceptions can be detected in privacy agreements. In such systems, once the privacy statement is agreed upon, the contract is binding for the involved parties for an unlimited period of time, or until it is cancelled by either party. However, not all contracts have such properties. Now, we move to a domain where contracts have deadlines. That is, the property associated with the contract has to be brought about within a specific period of time.

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 [36]. Like in privacy agreements, we represent e-commerce contracts with social commitments. *Commitments* [4] are an increasingly common way of representing contracts among software agents. In realistic e-commerce environments, a commitment is associated with a deadline, telling that its property has to be brought about within that time bound [7]. Otherwise, a violation occurs regarding that commitment. A violation of a commitment means an *exception* for its creditor.

Contracts are manipulated in two phases; contract negotiation and contract execution [36, 37]. Contract negotiation is out of the scope of this thesis. 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 monitoring on the environment, and verify their contracts accord-

ingly. 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 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 4.1 presents such a scenario from a real-life delivery process.

Example 4.1. *The customer, the bank, the store and its courier: a book's online purchase. Let us consider a scenario with a customer, Ali, who wishes to buy a copy of a book from an online store. The transaction needs two additional parties: a bank to carry out the payment and a courier to deliver the book to Ali.*

The process begins with Ali paying for the book using the bank on Monday. The contract between Ali and the store states that the book will be delivered in five business days as of the time the bank verifies Ali's payment. Assume that the bank verifies Ali's payment on Wednesday. Now, Ali expects a delivery by the following Wednesday. But what if the bank does not notify the store about the verification of Ali's payment until Friday? When the store receives the notification, it infers the deadline for delivery as the following Friday (two days later than Ali has previously inferred!). When Ali does not receive the book on Wednesday, he contacts the store to ask the reason of the delay. The store tells Ali that there are two more days until the deadline. At this point, Ali 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 the store's, by changing his deadline, and wait a bit longer.

One key point in Example 4.1 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 Ali becomes aware of a mismatch between the store's understanding of the commitment about the book 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 [23].

Another possible mismatch can be due to the debtor's *misbehavior*, e.g., the store delegates its commitment to the courier but then gives a wrong deadline. Finally, a mismatch may be caused by a simple *misunderstanding* among agents, e.g., Ali receives another book instead of the book. Schroeder and Schweimeier [38] 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[4] .

Among the few related works, Chopra and Singh [23] 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

[4]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 thesis.

an exception occurs. Besides, Chopra and Singh do not formalize commitment deadlines. We instead accommodate commitments with temporal constraints. The main differences 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 chapter.
- 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.

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 4.1, Ali makes a diagnosis request to the store about his violated commitment. The diagnosis process may involve a larger set of agents, e.g., the store may have delegated its commitment to the courier. In the end, such a process results in one of the following outcomes:

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

There are two cases of misalignment: (i) the one described by Chopra and Singh [23] where the creditor infers the commitment, but the debtor does not, and (ii) 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 thesis, 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. 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. Therefore, we safely 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.

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 4.1 by presenting a case study that leads to both misalignment and misbehavior diagnosis outcomes. We formalize the agents' interactions in $\mathcal{REC}$.

4.1. Commitments with Temporal Constraints

It is more realistic to consider commitments with time [7]. That is, the debtor is committed to satisfy the property for the creditor within a predefined deadline. Here, we use the Reactive Event Calculus ($\mathcal{REC}$) [28] to model *time-aware* commitments (i.e., commitments with temporal constraints). $\mathcal{REC}$ models two types of temporal constraints on commitments: (i) an existential temporal constraint where the property of the commitment has to be brought about inside a time interval, and (ii) a universal temporal constraint where the property of the commitment has to be maintained valid along a time interval. In this chapter, we focus on base-level commitments with existential temporal constraints.

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

$$s(c(x,\, y,\, \mathit{property}(e(t_1,\, t_2),\, p))),$$

where:

- s is a label identifying the state of the commitment at a specific time point. It can be *active*, *fulfilled*, or *violated*[5] ;
- 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, $t_1 \leq t \leq t_2$.

Note that the state, the debtor and creditor agents are the same as our previous commitment definition (Definition 35). Here, we only use the consequent since we deal with base-level commitments. However, the consequent is now attached with a temporal property as described above.

When the commitment is first created by the *create* operation [5], 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 [29].

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 Figure 4.6 below). We use the following syntax to represent a happened event:

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

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.

[5]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.

4.2. Running Example

Figure 4.1 shows the delivery process introduced in Example 4.1. 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*). Figures 4.2 through 4.6 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.

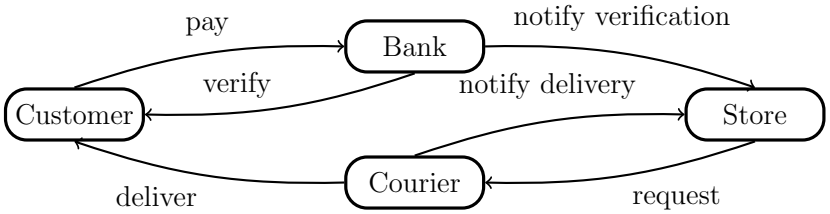


Figure 4.1. Delivery process.

Figure 4.2 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 [5, 7]. 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 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.

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

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

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

Figure 4.2. Coupled knowledge-base of the customer and the bank.

Figure 4.3 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[6] .

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

Figure 4.3. Coupled knowledge-base of the bank and the store.

Figure 4.4 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[7] .

[6]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" [39].

[7]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.

```
% 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).
```

Figure 4.4. Coupled knowledge-base of the customer and the store.

Figure 4.5 shows the $\mathcal{REC}$ rules shared among the store and the courier agents.

```
% 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.
```

Figure 4.5. Coupled knowledge-base of the store and the courier.

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.

Figure 4.6 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.

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

Figure 4.6. Coupled knowledge-base of the courier and the customer.

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

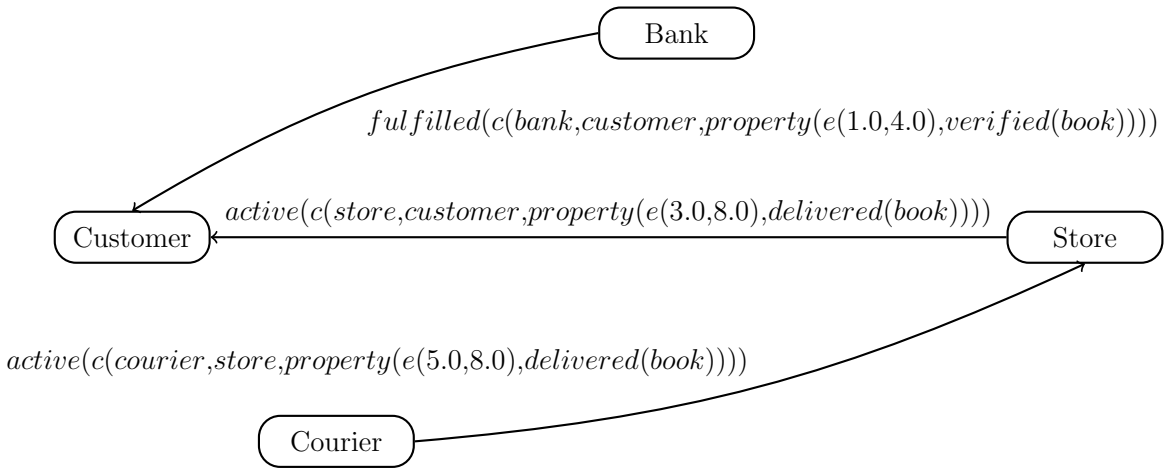


Figure 4.7. Commitments in the delivery process at time 6.0.

name. Such definitions could be written modularly in $\mathcal{REC}$, by resorting to the concept of role [40].

Figure 4.7 demonstrates a snapshot of the agents' commitments in the delivery process. The snapshot is taken at time 6.0. One 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.

Implementation: We have implemented the diagnosis framework in the j-$\mathcal{REC}$ tool for run-time monitoring which embeds a tuProlog reasoner[8] . We could thus run experiments in a simulated environment[9] . 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 in each of 4 agents (ali, bank, store, and courier). The simplest way to

[8] `http://sourceforge.net/projects/tuprolog`

[9] Here, we only show some excerpts of the whole code. A running prototype with a full implementation can be downloaded from `http://mas.cmpe.boun.edu.tr/ozgur/code.html`, under Section 1. Experiments for Diagnosis of Misalignment vs. Misbehavior.

run the example is to execute `java -jar CMon.jar` (or double-click on the `CMon.jar` file icon) on a selected agent folder (see Appendix C for details).

4.3. Commitment Similarity

Chopra and Singh [23] 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 17. *(Relevance) Commitment* $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p)))$ *is relevant to commitment* $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p)))$.

Example 4.2. *Table 4.1a 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 18. *(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 19. *(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 4.3. *Table 4.1b 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*

Table 4.1. Similarity relations: relevance and cover.

(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) $c_3 = active(c(store, customer, property(e(5.0,8.0), delivered(book))))$ $c_4 = active(c(store, customer, property(e(3.0,10.0), delivered(book))))$

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 20. *(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* $t_2 > t_4$*.*

Definition 21. *(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 4.4. *Table 4.2a 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 4.3 is a backward-shift of c_5 from Example 4.4, 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.*

Table 4.2. Similarity relations: shift and delegation.

(a) $c_5 = active(c(store, customer, property(e(5.0,10.0), delivered(book))))$ $c_6 = violated(c(store, customer, property(e(3.0,8.0), delivered(book))))$
(b) $c_7 = active(c(courier, store, property(e(5.0,8.0), delivered(book))))$ $c_8 = active(c(store, customer, property(e(3.0,8.0), delivered(book))))$

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 18-21. 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.*

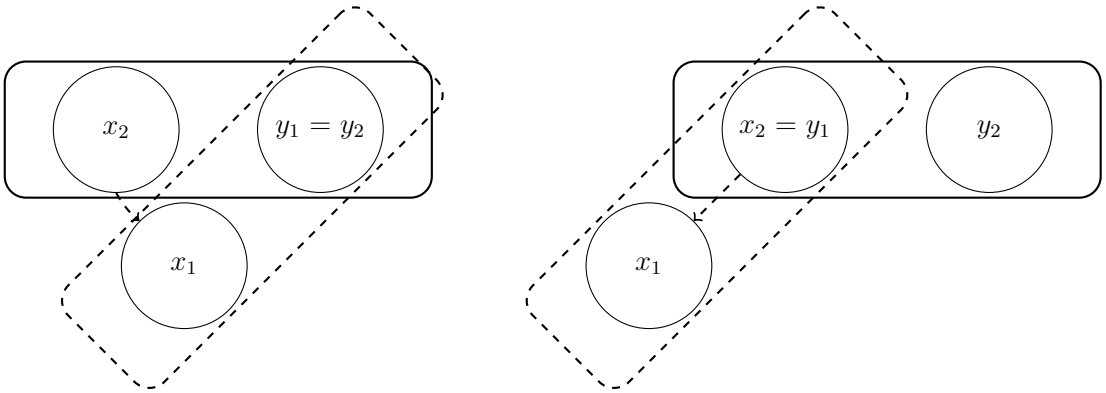
(each option excludes the other ones).

Proof. There are 13 possible (exclusive) relations between two time intervals according to Allen [41]. 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.

(i) $t_2 < t_3$: c_i is a backward-shift of c_j,

(ii) $t_1 > t_4$: c_i is a forward-shift of c_j,

(iii) $t_1 = t_3$ and $t_2 = t_4$: c_i covers c_j,



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

Figure 4.8. Delegatee.

(iv) $t_2 = t_3$: c_i is a backward-shift of c_j,

(v) $t_1 = t_4$: c_i is a forward-shift of c_j,

(vi) $t_1 > t_3$ and $t_2 < t_4$: c_i covers c_j,

(vii) $t_1 < t_3$ and $t_2 > t_4$: c_i is an extension of c_j,

(viii) $t_1 < t_3 < t_2 < t_4$: c_i is a backward-shift of c_j,

(ix) $t_3 < t_1 < t_4 < t_2$: c_i is a forward-shift of c_j,

(x) $t_1 = t_3$ and $t_2 < t_4$: c_i covers c_j,

(xi) $t_1 = t_3$ and $t_2 > t_4$: c_i is a forward-shift of c_j,

(xii) $t_1 > t_3$ and $t_2 = t_4$: c_i covers c_j,

(xiii) $t_1 < t_3$ 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 from 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 22. *(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 4.8 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.

Example 4.5. *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 [5, 23] 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 22 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 23. *(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 23 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 4.6. *Table 4.2b 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 24. *(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 25. *(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 4.7. *Table 4.3a 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 26. *(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 4.8. *Table 4.3b 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}.*

Table 4.3. Similarity relations: shift delegation.

(a) $c_9 = active(c(courier, store, property(e(7.0,10.0), delivered(book))))$ $c_{10} = violated(c(store, customer, property(e(3.0, 8.0), delivered(book))))$
(b) $c_{11} = active(c(courier, store, property(e(4.0,7.0), delivered(book))))$ $c_{12} = active(c(store, customer, property(e(5.0, 10.0), delivered(book))))$

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 23-26. 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 23-26. □

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.

```
% 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 \= X2.
delegatee((X1,Y),(Y,Y2)):- X1 \= Y2.

% 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)).
```

Figure 4.9. Commitment similarity in $\mathcal{RECC}$ (excerpt).

Figure 4.9 shows some of the $\mathcal{RECC}$ rules for the similarity relations.

4.4. 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 17, $\mathcal{C}_A^C \subseteq \mathcal{C}_A$ 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 23 belong to $\mathcal{C}_A^{CX}$; those that are a forward-shift of C by Definition 20 belong to $\mathcal{C}_A^{Cf}$, and those that are a forward-shift delegation of C by Definition 24 belong to $\mathcal{C}_A^{CfX}$. 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

Table 4.4. 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

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 4.4, and graphically illustrate the relations among commitments in Figure 4.10. Note that each set of commitments may contain violated, active and fulfilled commitments.

Definition 27 below formally defines the concept of diagnosis. A diagnosis *process* starts from a violated commitment C, and aims to identify the reason behind that

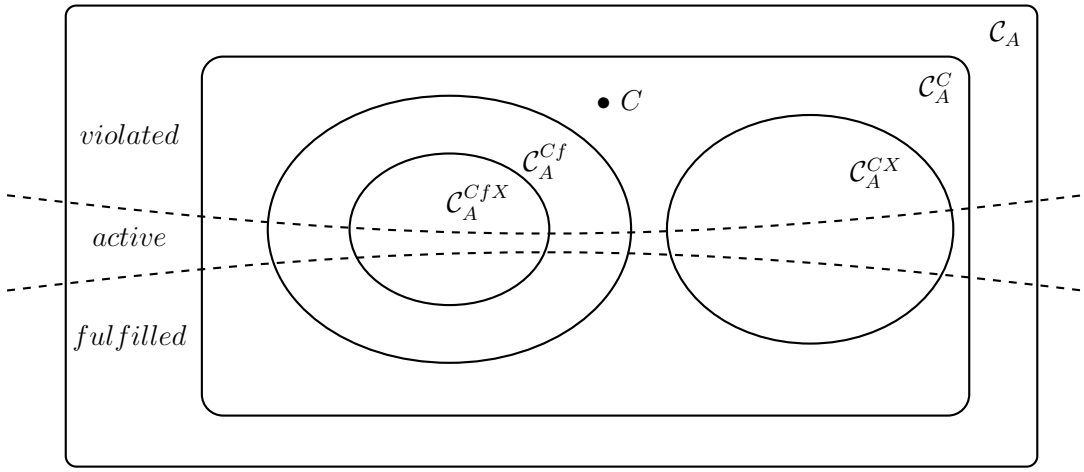


Figure 4.10. Commitment relations.

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

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 27 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:

(i) *Misalignment*: When we are in the presence of a (correct) misalignment diagnosis, one of the following applies:

- there is a commitment that is relevant to C, such that *its creditor infers the commitment but its debtor does not*,
- 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 the first case, 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 the second case, 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.

(ii) *Misbehavior*: When we are in the presence of a (correct) misbehavior diagnosis, one of the following applies:

- 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,
- 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 the first case, 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 the second case, 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 28 describes correct diagnosis formally taking into account the above cases.

Definition 28. *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:*

(i) if δ = misalignment($\emptyset$), then

$\exists x, y \in \mathcal{A}$, c = violated(c(x, y, 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);*

if δ = misalignment(c), $c \in \mathcal{C}$, then

$\exists x, y \in \mathcal{A}$, c = violated(c(x, y, property(e(t_1, t_2), p))),

c' = s(c(x, y, property(e(t_3, t_4), p))) $\in \mathcal{C}^C_{\mathcal{A}}$, such that

- *$s \in \{active, fulfilled\}$,*
- *$c \in \mathcal{C}_y$ (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);*

(ii) if δ = misbehavior(x), then

$\exists x, y \in \mathcal{A}$, c = violated(c(x, y, 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

- *$\mathcal{C}^{cY}_x = \emptyset$ for all $Y \in \mathcal{A}$ (the debtor, x, has not delegated c to anyone), or*
- *$\mathcal{C}^{cfY}_x \cup \mathcal{C}^{ceY}_x \neq \emptyset$ for some $Y \in \mathcal{A}$ (x has delegated c using wrong deadlines).*

Note that Definition 28 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 4.11 describes the architecture used to perform such distributed diagnosis. Agents are equipped with local and private knowledge-bases. Note that this is consistent with the distributed detection architecture described before. 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-$\mathcal{REC}$ (that is also used for detection), 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 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 28 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 28. 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 29-34 specify the diagnosis process declaratively.

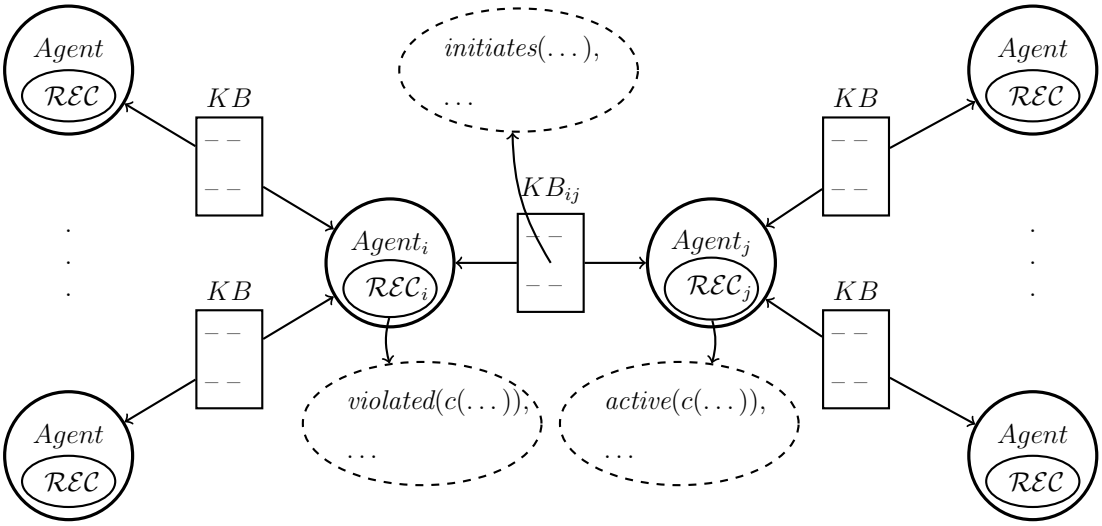


Figure 4.11. Diagnosis architecture.

Definition 29. *(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 28. 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 30. *(misalignment diagnosis following forward-shift)*

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

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 28.

Definition 31. *(misalignment diagnosis following wrong delegation)*

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

Similar to Definition 30, 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 32. *(misbehavior diagnosis following failure to meet deadline)*

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

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 28.

Definition 33. *(misbehavior diagnosis following wrong delegation)*

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

[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.

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 28.

Definition 34. *(propagation of diagnosis following successful local check)*

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

Definition 34 covers the 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.

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 29-34. The following properties hold for $\mathcal{M}_1$:

Property 1: $\mathcal{M}_1$ *terminates.* We consider two cases for termination: (i) there does not exist any circular chain of delegations, and (ii) 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 = c(x, y, ...), c_2 = c(z, x, ...), ..., c_{n-1} = c(w, u, ...), c_n = 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 34).

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 28. 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 29-34 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 29-34 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 4.9. *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 4.9 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.

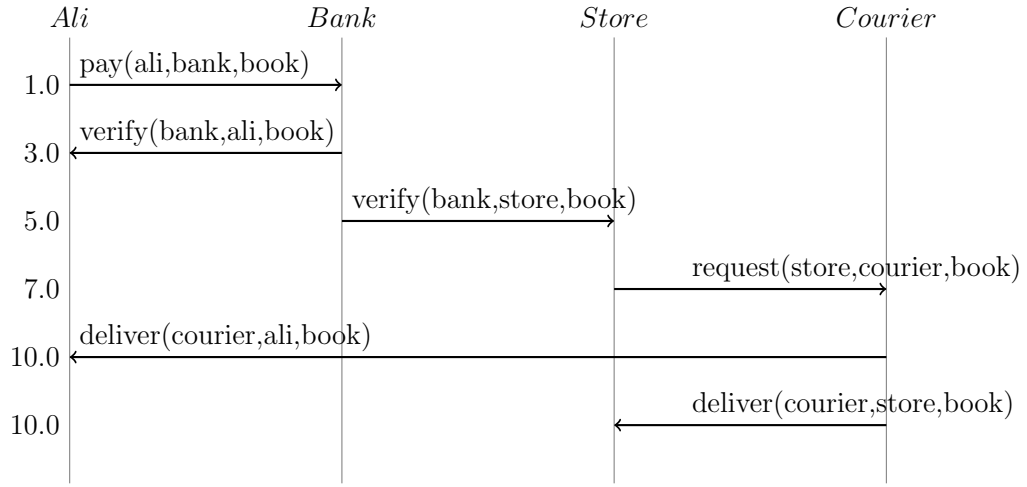


Figure 4.12. Trace of events for Case I.

4.5. 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.

4.5.1. Case I: Misalignment

Figure 4.12 shows the trace of events for the first case. This case represents a misalignment of commitments between the customer (Ali) and store agents. The misalignment is caused by the late notification of the bank agent. Let us review the trace of events. Ali sends the payment to the bank at day 1, regarding its purchase of the book from the store. At day 3, the bank verifies Ali's payment. However, the bank sends the notification of Ali's payment to the store at day 5. This is where the misalignment occurs between Ali and the store. Recall that their commitment is triggered by the bank's confirmation of payment. Then, the store requests the delivery of the book from the courier agent at day 7. Finally, the courier delivers the book to Ali at day 10. At the same time, the courier notifies the store about the delivery.

Let us now track the commitments of agents in time. At day 3, Ali infers c_1 = *active* (*c*(*store*, *ali*, *property*(*e*(3.0, 8.0), *delivered* (*book*)))). At day 5, the store

infers $c_2 = active\ (c(store,\ ali,\ property(\ e(5.0,\ 10.0),\ delivered\ (book))))$. Notice the deadline shift between those two commitments. At day 7, both the store and the courier infer $c_3 = active\ (c(courier,\ store,\ property(\ e(7.0,\ 10.0),\ delivered\ (book))))$. At day 9, when Ali runs its $\mathcal{REC}$ engine, it detects that c_1 has been violated. Figure C.1 in Appendix C shows the $\mathcal{REC}$ output for this instance.

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

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

According to the output of its monitoring facility, the store finds c_2, which is a (still active) forward-shift of c_1. Things could still be OK for the store if the store had delegated the task with a correct deadline. Unfortunately, this is not the case. Based on its records, the store verifies that an active commitment, c_3, is a forward-shift delegation of c_1. Thus, the store will immediately inform Ali of a misalignment following wrong delegation (see Definition 31): $\delta = misalignment(c_3)$. Ali 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 the courier makes the delivery.

4.5.2. Case II: Misbehavior

Figure 4.13 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. Ali sends the payment to the bank at day 1, regarding its purchase of the book from the store. At day 3, the bank verifies Ali's payment, and sends the notification to the store. Then, the store requests the delivery of the book from the courier at day 5. The courier delivers the book to Ali at day 10, and notifies the store about the delivery. This time, however, the courier should have delivered earlier. Thus, this is where the courier violates its commitment by disrespecting its deadline.

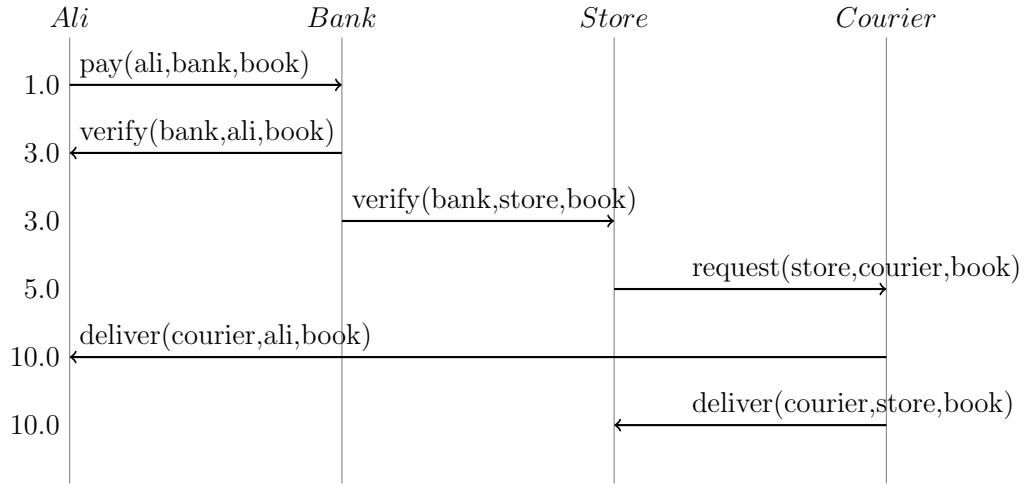


Figure 4.13. Trace of events for Case II.

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

Ali initiates the diagnosis process with the diagnosis request

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

Accordingly, the store 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, the store redirects the diagnosis request to the courier (see Definition 34):

$$courier \overset{d(c_2)}{\vDash_{\{c_1\}}} \delta.$$

the courier finds that c_2 is violated. Since there is no delegation of c_2, the courier is the cause of the exception (by Definition 32). The answer $\delta = misbehavior(courier)$ is passed from the courier to the store, and from the store back to Ali. Figure C.2 in Appendix C shows the $\mathcal{REC}$ output for this instance.

In this chapter, 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 [23, 38, 42], 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. 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.

Agents' goals are also important when considering the commitments in a protocol [25, 27]. 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 [27]. Here, 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 [43, 44] 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

Table 4.5. Levels of forward-shift.

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

involved in the mismatch, it is not possible to diagnose cases of agent death [45].

In order to demonstrate how our approach works, we have extended a delivery process description by involving temporal constraints, and formalized it in $\mathcal{REC}$ [7, 28]. 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 4.5 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 [46].

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 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.

5. MONITORING OF COMMITMENT DELEGATIONS

The use of commitments to model agent interaction has been advocated especially in heterogeneous and open settings where autonomous agents must flexibly interact, e.g., to handle exceptions and exploit opportunities [5]. One reason why commitment-based approaches are more flexible than traditional approaches is that they enable the stakeholders to delegate their commitments. Delegation may be desirable fore several reasons. One reason is that an agent may not be capable of satisfying the properties he committed to bringing about. This is very normal in e-commerce scenarios. For example, a merchant may delegate a delivery task to a courier. In that case, a new commitment is formed C(courier, merchant, $\top$, delivered) as a delegation of C(merchant, customer, paid, delivered) from the merchant (*delegator*) to the courier (*delegatee*), about the delivery task (*delegandum*).

In order to safeguard protocol flexibility and agent autonomy, it makes sense to leave a certain degree of freedom in the delegation process. For example, a delegator wants to autonomously decide when to set deadlines to his delegatees, based on a number of criteria. More or less flexible deadlines can result, e.g. from contractual negotiations, normative constraints, or reputation-based considerations. Whatever the reason for delegation, it is always important to understand whether the flexibility that comes together with such a mechanism introduces any weakness in the system, which may lead to undesirable global states.

Usually, commitments formed between different agents are connected to each other; either explicitly (by delegation), or implicitly (other dependencies). The delegation operation extends the set of agents related with a commitment. In the merchant-courier example, the delivery commitment between the merchant and customer agents is extended with the courier agent. 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 the courier, it can be identified.

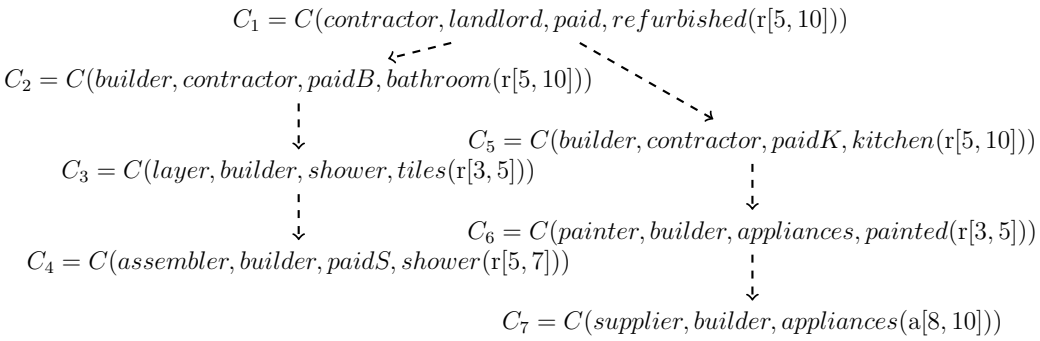


Figure 5.1. Refurbish house.

Let us look at a more complicated case. Consider the commitments and their delegations in Figure 5.1. The landlord wants to refurbish his house, so he makes a commitment with the contractor (C_1). In order to refurbish the house, the contractor makes two commitments with the builder; one for the bathroom (C_2) and one for the kitchen (C_5). Building the bathroom requires first the shower (C_4), then the tiles (C_3). Similarly, building the kitchen requires first the appliances (C_7), then the painting of the walls (C_6).

When there are many commitments in the system at hand (like in Figure 5.1), 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 down individual commitment states is called commitment monitoring [7, 28]. We extend monitoring to enable run-time tracking of exceptions via the links between agents' commitments. To this end, we define a simple language for commitments and deadlines, inspired from [7], 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 deadlines that can be absolute or relative. No deadlines are associated with the antecedent, and the property language does not accommodate negation nor disjunction. This simplification makes it possible to efficiently and safely monitor the states of all possible delegations that can be referred to any given commitment. In this way, an agent can make reasonable assumptions about whether such a commitment is likely to be fulfilled or not.

We define a complete set of possible rational delegation schemes, identifying for each combination of (possibly nested) delegations what critical situations may lead to improper delegation and possibly to commitment violation. We give an exhaustive account of all possible improper delegations, and show a sound and complete distributed reasoning procedure that is able to find all improper delegations of a given commitment.

5.1. Commitments with Extended Temporal Constraints

We now extend our commitment description with extended temporal properties.

Definition 35. *$C(X, Y, Q, P(\gamma[t_1, t_2]))$ represents a commitment, where the debtor X commits to the creditor Y for satisfying the consequent P when the antecedent Q holds. If Q is $\top$[11] , then X is committed to Y unconditionally. P has to be satisfied within the interval $\gamma[t_1, t_2]$, where $\gamma[t_1, t_2]$ can be one of the following:*

- a$[t_1, t_2]$ *defines an absolute deadline, where P has to be brought about at some point between t_1 and t_2;*
- r$[t_1, t_2]$ *defines a relative deadline, where P has to be satisfied between t_1 and t_2 time units as of the time t Q gets satisfied, i.e., P has to be brought about at some point between $t + t_1$ and $t + t_2$. A relative deadline is only defined when the antecedent is not $\top$.*

In the remainder of this chapter, we may use the term *base-level* to refer to commitments whose antecedent is $\top$, and *conditional* to refer to commitments whose antecedent is not $\top$ [5]. When we discuss commitments independently of the temporal dimension, we may also use the simplified notation $C(X, Y, Q, P)$.

In Definition 35, X, Y are agents, and Q, P are (conjunctions of) atomic propositions. We do not support negation or disjunction of propositions, nor nested commitments. When P is a conjunction of propositions, we assume that all the atomic propositions in P have the same deadline.

[11] $\top$ is a constant symbol indicating a fictitious property that does not need to be satisfied because it is already *true*.

Example 5.1. *C(merchant, customer, paid, packaged ∧ delivered(*r*[*4, 7*]))* tells that the packaging and delivery should take place between four and seven time units after the payment is completed, since r*[*4, 7*]* represents a relative deadline, and it applies to all properties in the consequent. Thus, if the payment is done at time 3, then this commitment will become *C(merchant, customer, ⊤, packaged ∧ delivered(*a*[*7, 10*]))*, which is in the active state and tells that the merchant has to package and deliver the merchandise between times 7 and 10 (absolute deadline).

5.2. 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. Chopra and Singh [23, 42] compare commitments via a *strength* relation using the commitments' properties, Kafalı *et al.* [12] 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 36. *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 part of the antecedent Q (when Q is not ⊤) or part of the consequent P. The common property between primary and delegation is called delegandum.*

In the sequel, we use the notation debtor(C_m, X) to indicate that X is C_m's debtor and delegatee(C_m, C_j, Y) to indicate that Y is the delegatee of C_m's delegation C_j.

Six types of delegation are particularly meaningful. Let us define and illustrate them one by one, considering variations of *C(merchant, customer, paid, packaged ∧ delivered)* as our *primary*.

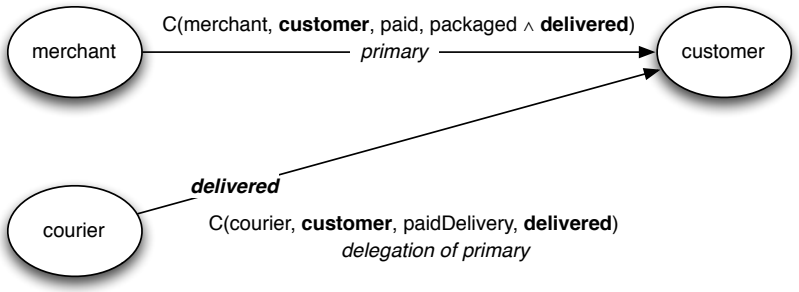


Figure 5.2. Explicit delegation.

Definition 37. *Commitment C(Z, Y, Q′, P′) is an explicit delegation of commitment C(X, Y, Q, P) iff P ⊨ P′*[12] .

This type of delegation was proposed by Yolum and Singh [5] 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 [5]. An explicit delegation is shown in Figure 5.2. The new debtor *courier* replaces the old debtor *merchant*.

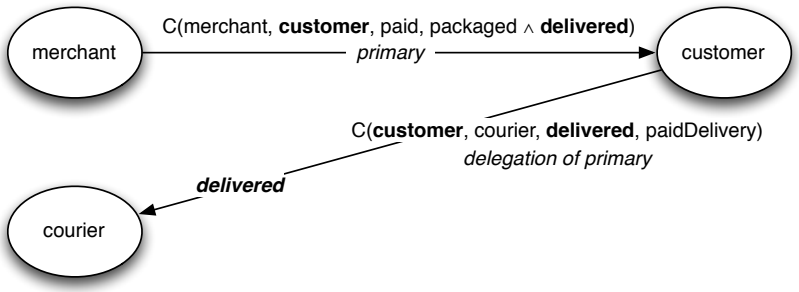


Figure 5.3. Weak explicit delegation.

Definition 38. *Commitment C(Y, Z, Q′, P′) is a weak explicit delegation of commitment C(X, Y, Q, P) iff P ⊨ Q′.*

[12]The semantics of ⊨ will depend on the language of the antecedent/consequent properties. Here for simplicity, we consider properties to be conjunctions of propositions, therefore $P \models P' \Leftrightarrow (P = P') \vee (P = P' \wedge P'')$, where P, P', P'' are all (conjunctions of) propositions. Since we use these definitions to evaluate the relevance/similarity of commitments to one another, ⊨ does not consider temporal aspects.

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 a weak delegation to achieve P since there is no obligation for Z to satisfy P, unless Z needs Q satisfied. The concept of weak delegation was introduced by Kafalı and Torroni [13], inspired by Chopra *et al.* 's work [25]. A weak explicit delegation is shown in Figure 5.3. Note that the roles of creditor and debtor are reversed, and accordingly also antecedent and consequent are reversed.

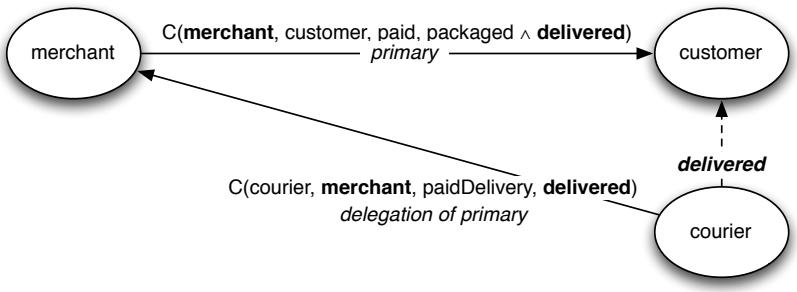


Figure 5.4. Implicit delegation.

Definition 39. *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, in a chain-like fashion. This type of delegation chain was proposed by Kafalı *et al.* [12]. An implicit delegation is shown in Figure 5.4. Note that the creditor is the *merchant*, which is the primary's debtor. The primary is not canceled, because a commitment must be kept to the initial creditor (the *customer*).

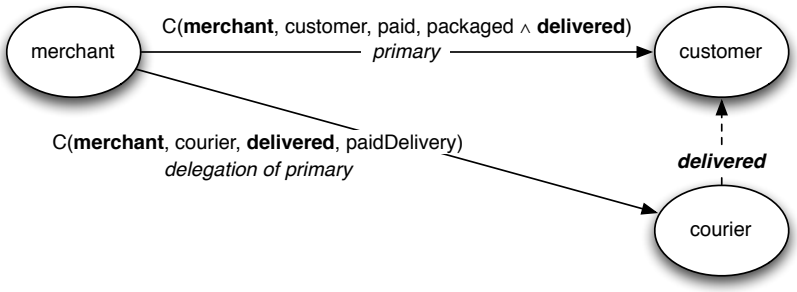


Figure 5.5. Weak implicit delegation.

Definition 40. *Commitment $C(X, Z, Q', P')$ is a weak implicit delegation of commitment $C(X, Y, Q, P)$ iff $P \models Q'$.*

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 (antecedent and weak antecedent delegation), were introduced by Kafalı and Torroni [13]. A weak implicit delegation is shown in Figure 5.5.

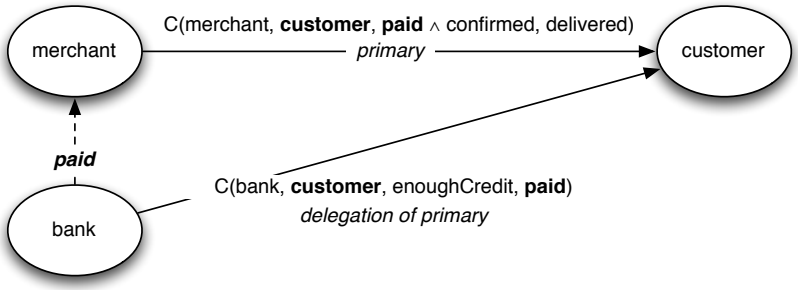


Figure 5.6. Antecedent delegation.

Definition 41. *Commitment $C(Z, Y, Q', P')$ is an antecedent delegation of commitment $C(X, Y, Q, P)$ iff Q is not $\top$ and $Q \models P'$.*

The creditor of the primary also becomes the creditor of a new commitment for (part of) the antecedent of the primary. An antecedent delegation shown in Figure 5.6. Because the initial consequent (*delivered*) does not appear in the antecedent delegation, in order to maintain a commitment about the initial consequent, the primary is not canceled. Antecedent delegations and implicit delegations can be combined together and bind multiple commitments into causal similarity relations.

The last type of delegation we consider is the weak variant of antecedent delegation.

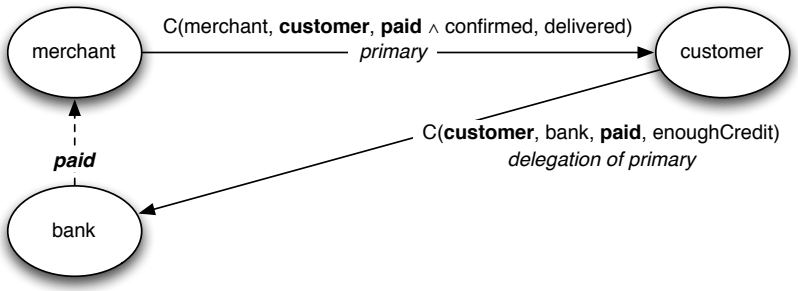


Figure 5.7. Weak antecedent delegation.

Definition 42. *Commitment $C(Y, Z, Q', P')$ is a weak antecedent delegation of commitment $C(X, Y, Q, P)$ iff Q is not $\top$ and $Q \models Q'$.*

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. A weak antecedent delegation of the primary is shown in Figure 5.7.

Figure 5.8 summarizes the various types of commitment delegation. The figure shows only the key aspects that distinguish each type of delegation. Each agent is represented by a circle, with the agent's name inside (X, X', Y, Y'). Each commitment is represented by an arrow from the debtor to the creditor, e.g., the first commitment is from X to Y. The delegating agent is displayed with a thicker line. Only the delegandum is displayed. The antecedent is represented at the creditor end of the arrow, whereas the consequent is represented near the debtor end of the arrow. If a delegation cancels the primary, the primary is represented by a dashed line arrow, to emphasize that the primary is no longer in place.

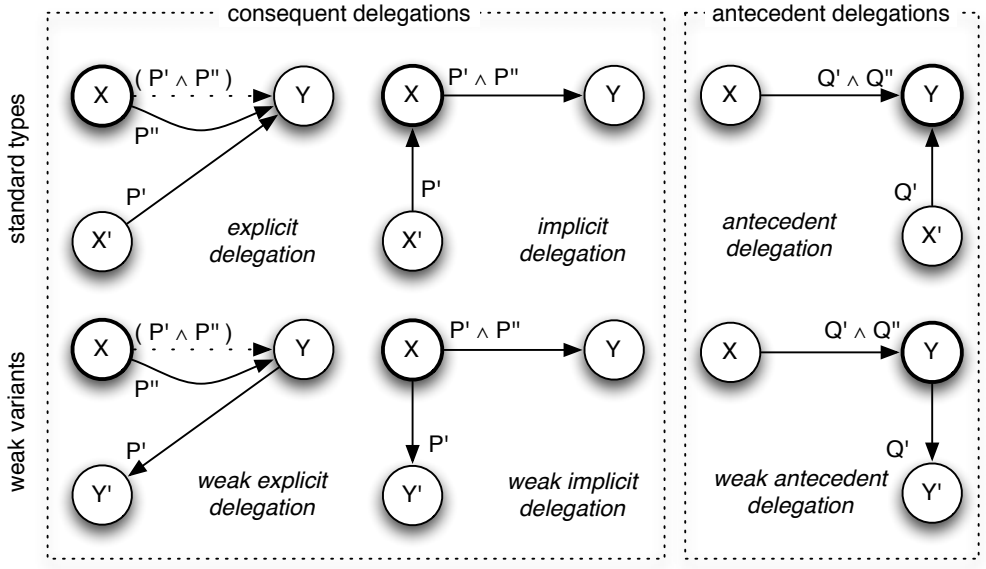


Figure 5.8. Types of delegation.

Definitions 37 through 42 give an exhaustive account of how a commitment can be *rationally* delegated, i.e., by preserving the responsibilities of roles in relation with the primary's properties [13].

5.3. Similarity relations

We will now shift the focus to commitments that are linked to each other *via* other commitments. To this end, in [14] we introduced the concept of commitment similarity, which we explore here in greater detail, and extend to capture the notion of chains of delegations. We then analyse the temporal properties of chains of delegations, and isolate cases of improper delegation in the chain, in order to identify exceptions as they occur.

5.3.1. Chains of delegations

Let us introduce the idea of commitment similarity with an example.

Example 5.2. *Consider set* $\mathcal{C}_{S5.2} = \{C_{S5.2.1}, C_{S5.2.2}, C_{S5.2.3}\}$ *(see Figure 5.9), where:*

$$C_{S5.2.1} = C(\textit{merchant, customer, paid, delivered})$$
$$C_{S5.2.3} = C(\textit{courier, merchant, confirmed} \wedge \textit{paidDelivery, delivered})$$
$$C_{S5.2.2} = C(\textit{bank, merchant, enoughCredit, paidDelivery})$$

According to $C_{S5.2.1}$*, once the customer pays, the merchant will have the merchandise delivered. Now, the merchant delegates the delivery to the courier via* $C_{S5.2.3}$*. However, in order to deliver, the courier needs the delivery to be paid. Thus, the merchant makes another delegation to the bank via* $C_{S5.2.2}$*. As long as the merchant has enough credits in his account, the bank will send the payment for him.*

Example 5.2 illustrates a *causal relation* between commitments here: in order to satisfy $C_{S5.2.1}$'s delegation $C_{S5.2.3}$, a new commitment $C_{S5.2.2}$ is in place. In particular, $C_{S5.2.3}$ is the link between two otherwise seemingly unrelated commitments. We will say that $C_{S5.2.1}$ and $C_{S5.2.2}$ are causal-delegation similar via $C_{S5.2.3}$. To put it formally:

Definition 43. *Commitment* $C_1 = C(X_1, Y_1, Q_1, P_1)$ *is causal-delegation similar to 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*

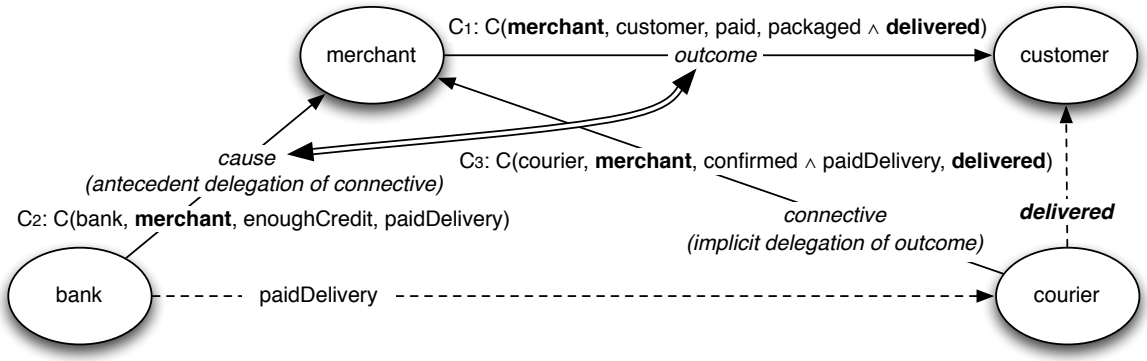


Figure 5.9. Causal delegation similarity.

(i) $P_1 \models P_3$ *and* $X_1 = Y_3$ *(implicit delegation), and*

(ii) $Q_3 \models P_2$ *and* $Y_3 = Y_2$ *(antecedent delegation).*

We call C_1 *outcome,* C_2 *cause, and* C_3 *connective.*

Definition 43 connects two commitments through two delegations; one consequent (implicit) and one antecedent delegation. Here, C_1 and C_2 can also be base-level commitments since Q_1 and Q_2 are not part of the relation. However, C_3 cannot be a base-level commitment since Q_3 is part of the relation. Definition 43 allows us to trace chained commitments where the property P changes along the delegation chain.

Causal delegation similarity is a very special case of delegation chains. Let us see what we can tell in general about such chains. Now, consider the delegations that do not cancel the primary: (weak) implicit and (weak) antecedent delegations. If we start from a primary $C(X, Y, Q' \wedge Q'', P' \wedge P'')$ and apply two delegations in a row, we obtain 16 different cases. These are illustrated in Figure 5.10.

The two bottom rows of Figure 5.10 show a (weak) antecedent delegation followed by other delegations. Because we are not considering temporal constraints (e.g., time intervals to represent deadlines) associated with the antecedent, these cases are not interesting for monitoring. There are 8 cases left, of which only 4 involve more than two commitments in the scope of a single agent (X). One such case is implicit delegation followed by antecedent delegation, corresponding to a causal delegation (Figure

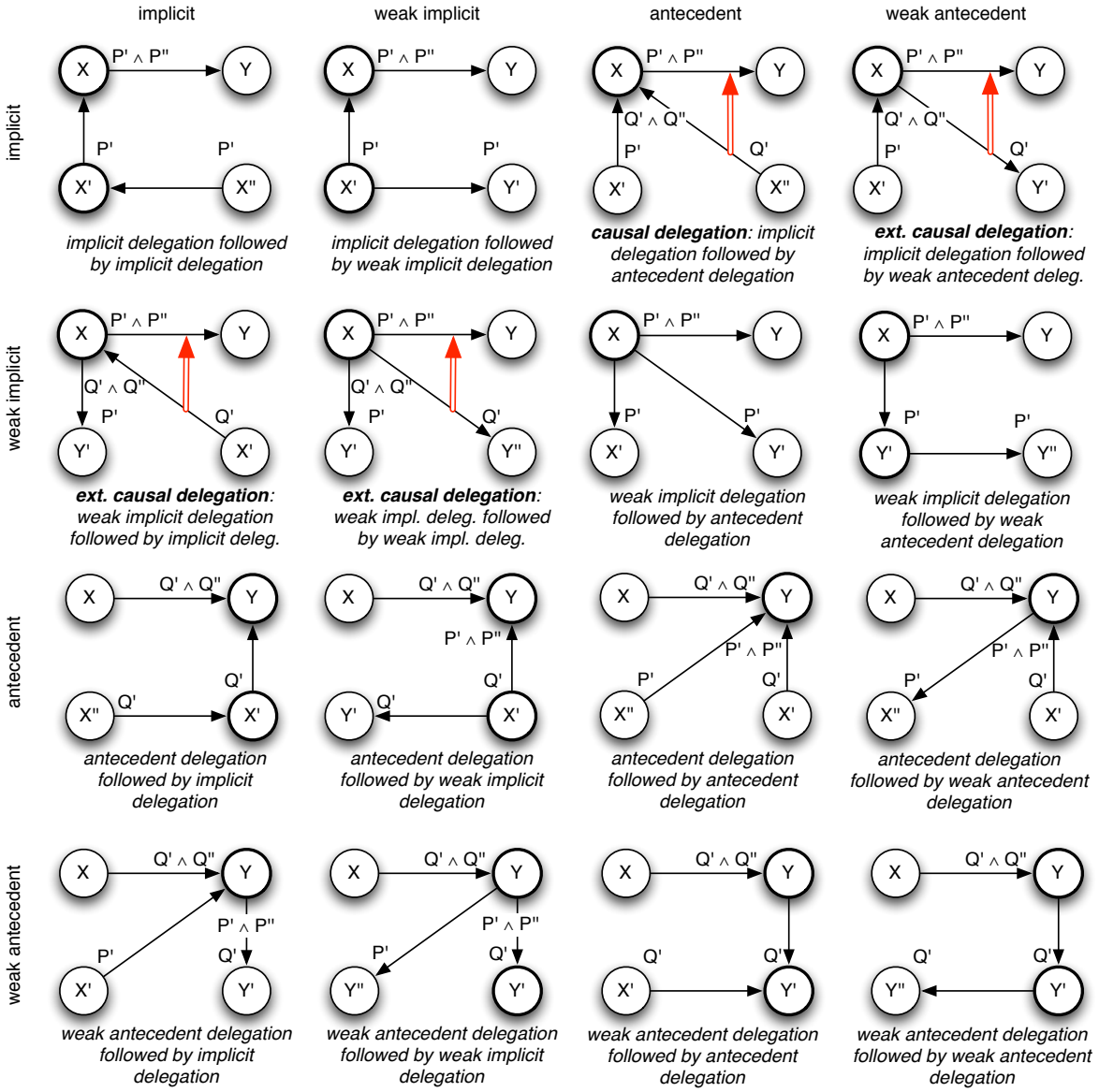


Figure 5.10. Sequential delegations.

43). In fact, all 4 cases introduce a causal dependency between the first and the last commitment in the delegation chain, via a third commitment. Thus we can identify outcome, connective, and cause for each of them. Figure 5.10 displays a double-line arrow from cause to outcome in each case of *causal delegation* similarity. We can then revise Figure 43:

Definition 44 (extends Definition 43)**.** *Commitment* $C_1 = C(X_1, Y_1, Q_1, P_1)$ *is causal-delegation similar to commitment* $C_2 = C(X_2, Y_2, Q_2, P_2)$ *via commitment* $C_3 = C(X_3, Y_3, Q_3, P_3)$ *iff*

- $P_1 \models P_3$, $Q_3 \models P_2$, *and* $X_1 = Y_2 = Y_3$ *(implicit delegation followed by antecedent delegation), or*
- $P_1 \models P_3$, $Q_3 \models Q_2$, *and* $X_1 = X_2 = Y_3$ *(implicit delegation followed by weak antecedent delegation), or*
- $P_1 \models Q_3$, $P_3 \models P_2$, *and* $X_1 = Y_2 = X_3$ *(weak implicit delegation followed by implicit delegation), or*
- $P_1 \models Q_3$, $P_3 \models Q_2$, *and* $X_1 = X_2 = X_3$ *(weak implicit delegation followed by weak implicit delegation).*

We call C_1 *outcome,* C_2 *cause, and* C_3 *connective.*

We can further extend this definition to the case of N delegations. With the help of Figure 5.10 we can understand how chains of delegations can be constructed, starting from either of the 4 cases above. For example, after implicit delegation followed by antecedent delegation, we can have another antecedent delegation from X's side, falling in the case shown at the intersection between the third row and the third column of Figure 5.10 (antecedent delegation followed by antecedent delegation), and so on. Not all possibilities are open. Figure 5.11 shows how chains of causal delegations can be obtained, all with X as a stakeholder (debtor or creditor).

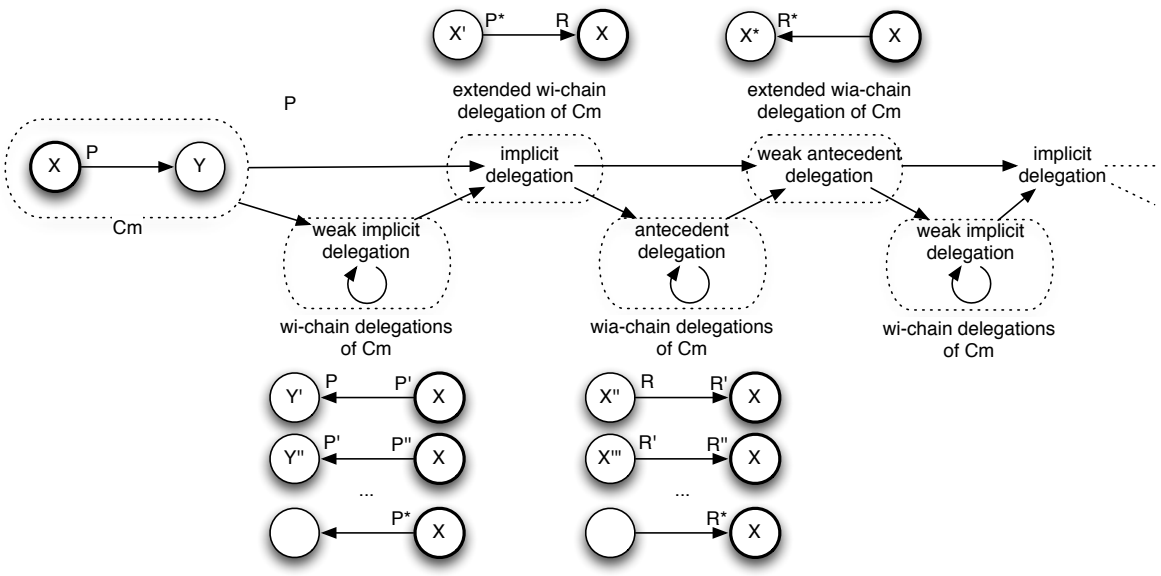


Figure 5.11. Causal delegation chains.

We shall now formally define a causal delegation chain. We will do it incrementally.

Definition 45. *A commitment C_j is a* **wi-chain delegation** *of a commitment C_m iff*

- *C_j is a weak implicit delegation of C_m, or*
- *$\exists$ C_k s.t. C_k is a wi-chain delegation of C_m, and C_j is a weak implicit delegation of C_k, or*
- *$\exists$ C_k s.t. C_k is an extended wia-chain delegation of C_m (see Definition 48), and C_j is a weak implicit delegation of C_k*

Definition 46. *A commitment C_j is an* **extended wi-chain delegation** *of a commitment C_m iff*

- *C_j is an implicit delegation of C_m, or*
- *$\exists$ C_k s.t. C_k is a wi-chain delegation of C_m, and C_j is an implicit delegation of C_k, or*
- *$\exists$ C_k s.t. C_k is an extended wia-chain delegation of C_m (see Definition 48), and C_j is an implicit delegation of C_k*

Definition 47. *A commitment C_j is a* **wia-chain delegation** *of a commitment C_m iff*

- *$\exists$ C_k s.t. C_k is an extended wi-chain delegation of C_m, and C_j is an antecedent delegation of C_k, or*
- *$\exists$ C_k s.t. C_k is a wia-chain delegation of C_m, and C_j is an antecedent delegation of C_k*

Definition 48. *A commitment C_j is an* **extended wia-chain delegation** *of a commitment C_m iff*

- *$\exists$ C_k s.t. C_k is an extended wi-chain delegation of C_m, and C_j is a weak antecedent delegation of C_k, or*

- $\exists\ C_k$ *s.t.* C_k *is a wia-chain delegation of* C_m*, and* C_j *is a weak antecedent delegation of* C_k

All these are cases of indirect delegations.

Definition 49. *Commitment* C_j *is an indirect delegation of commitment* C_m*, denoted* $\overset{\sim\sim\sim}{\text{deleg}}(C_j, C_m)$*, iff*

- C_j *is a wi-chain delegation of* C_m*, or*
- C_j *is an extended wi-chain delegation of* C_m*, or*
- C_j *is a wia-chain delegation of* C_m*, or*
- C_j *is an extended wia-chain delegation of* C_m*.*

Definition 50. *A* **causal delegation chain** *from a commitment* C_m *(outcome) to a commitment* C_k *(cause) is a sequence* σ *of commitments* $\sigma = [C_0, C_1, \ldots, C_k]$*,* $(k \geq 2)$*, such that*

(i) $C_0 = C_m$*, and*
(ii) $\forall i, 1 \leq i \leq k, C_i$ *is a delegation of* C_{i-1}*, and*
(iii) $\forall i, 1 \leq i \leq k$, $\overset{\sim\sim\sim}{\text{deleg}}(C_j, C_m)$*.*

We will conclude this section by extending the definition of delegation, in the light of these new notions.

Definition 51. *Commitment* C_j *is a direct delegation of commitment* C_m*, denoted* $\overrightarrow{\text{deleg}}(C_j, C_m)$*, iff*

- C_j *is an explicit delegation of* C_m*, or*
- C_j *is a weak explicit delegation of* C_m*, or*
- C_j *is an implicit delegation of* C_m*, or*
- C_j *is a weak implicit delegation of* C_m*, or*
- C_j *is an antecedent delegation of* C_m*, or*
- C_j *is a weak antecedent delegation of* C_m*.*

Definition 52. *[extends Definition 36] Let C_m, C_j be two commitments, $C_m, C_j \in \mathcal{C}$. We say that C_j is a delegation of a commitment C_m, denoted* $\mathrm{deleg}(C_j, C_m)$*, iff*

- $\overrightarrow{\mathrm{deleg}}(C_j, C_m)$*, or*
- $\overset{\sim\sim\sim}{\mathrm{deleg}}(C_j, C_m)$*.*

Definition 52 accounts for all paths between a given commitment (primary) and its direct and indirect delegations. 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.

5.3.2. 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 behavior. To this end we will define cases of delegations in which the deadline of the primary is not properly propagated onto the delegation. We will use the term *improper* to label a delegation that exceeds the deadline of the primary commitment. We will consider the overall system as it is observed at a specific time, *time of observation*.

Definition 53 describes how we compare two deadline intervals.

Definition 53. *Let t be the time of observation. A deadline interval $\mathcal{I}_j = \gamma[t_1, t_2]$ exceeds a deadline interval $\mathcal{I}_k = \gamma'[t_3, t_4]$, iff either of the following holds:*

- *$\mathcal{I}_j$ is an absolute deadline, $\mathcal{I}_k$ is an absolute deadline, and $t_1 < t_3$ or $t_2 > t_4$;*
- *$\mathcal{I}_j$ is an absolute deadline, $\mathcal{I}_k$ is a relative deadline, and $t_1 < t_3 + t$ or $t_2 > t_4 + t$;*
- *$\mathcal{I}_j$ is a relative deadline, $\mathcal{I}_k$ is an absolute deadline, and $t + t_1 < t_3$ or $t + t_2 > t_4$;*
- *$\mathcal{I}_j$ is a relative deadline, $\mathcal{I}_k$ is a relative deadline, and $t_1 < t_3$ or $t_2 > t_4$.*

We use the notion of exceeding deadline intervals to define improper delegations. In general, if the deadline interval of a delegation's consequent exceeds that of the primary consequent, we might have a problem regarding an improper delegation.

The first thing one should notice is that we cannot have an improper (weak) antecedent delegation, since our commitment language does not allow us to specify deadline intervals for the antecedent of a commitment. Besides, as we discussed earlier, we cannot track down improper delegations of (weak) explicit delegations, because explicit delegations cancel the primary, therefore we cannot compare the two deadline intervals any more[13] .

Definition 54. *Let C_m, C_j be two commitments, such that C_j is a (weak) implicit delegation of C_k. We say that C_j is an improper consequent delegation of C_m iff C_j's interval exceeds C_m's interval.*

Example 5.3. *Consider for instance set $\mathcal{C}_{S5.3} = \{C_{S5.3.1}, C_{S5.3.2}, C_{S5.3.3}\}$, where:*

$$C_{S5.3.3} = C(\textit{merchant, customer, } \top\textit{, delivered} \wedge \textit{invoiced(a[10, 12]))}$$
$$C_{S5.3.1} = C(\textit{courier, merchant, } \top\textit{, delivered(a[10, 12]))}$$
$$C_{S5.3.2} = C(\textit{accountant, merchant, } \top\textit{, invoiced(a[11, 14]))}$$

Assume that the current time is 8*. Now, this is an improper delegation, because the deadline of $C_{S5.3.2}$ is greater than that of $C_{S5.3.3}$. Note that the occurrence of an exception, although likely, is not inevitable since the accountant may still satisfy invoiced at time* 12*.*

The case of causal delegation chains is trickier.

Definition 55. *Let $C_m, C_j \in \mathcal{C}$,* deleg*(C_j, C_m), and σ=[C_m, C_1, ..., C_j] be a causal delegation chain from C_m to C_j in $\mathcal{C}$. Let $\mathcal{I}_i$=γ[$t_{i,start}$, $t_{i,end}$] be C_i's interval, for each*

[13] We could assume that agents always keep track of canceled commitments, but this assumption would interfere with agent autonomy, and it would be in contrast with our agent architecture-agnostic approach. Thus on the agent side we do not assume any knowledge other than the minimum needed for commitment-based interaction to take place, i.e., that an agent knows what *active* or *conditional* commitments are in place, of which he is either the debtor or the creditor.

C_i in σ, $0 \leq i \leq j$ (C_m=C_0). Let t be the time of observation. We say that C_j is an improper causal delegation of C_m iff either of the following holds:

(i) all $\mathcal{I}_i$ are relative, and $t + \Sigma_{i=0}^{j} t_{i,start} < t_{0,start}$ or $t + \Sigma_{i=0}^{j} t_{i,end} > t_{0,end}$;

(ii) let $\mathcal{I}_k$ be the last absolute deadline in σ[14] :

- *$\mathcal{I}_m$ is an absolute deadline and*
  $t_k + \Sigma_{i=k+1}^{j} t_i < t_{0,start}$ or $t_k + \Sigma_{i=k+1}^{j} t_i > t_{0,end}$, or
- *$\mathcal{I}_m$ is a relative deadline and*
  $t_k + \Sigma_{i=k+1}^{j} t_i < t + t_{0,start}$ or $t_k + \Sigma_{i=k+1}^{j} t_i > t + t_{0,end}$.

Example 5.4. *Consider set $\mathcal{C}_{S5.4} = \{C_{S5.4.1}, C_{S5.4.2}, C_{S5.4.3}\}$, where:*

$$C_{S5.4.1} = C(\textit{bank, client, requested, delivered}(\mathrm{r}[5, 7]))$$
$$C_{S5.4.3} = C(\textit{courier, bank, printed, delivered}(\mathrm{r}[3, 5]))$$
$$C_{S5.4.2} = C(\textit{office, bank, confirmed, printed}(\mathrm{r}[2, 3]))$$

When investigated properly, one can see the problem with these conditional commitments. In order for the card to be delivered, it has to be printed first, and the time requirements for those two processes exceed the time limit that the bank has towards the client. The bank should have gotten into other commitments that would have lead the fulfillment of its primary commitment towards the client [25]. However, note that $C_{S5.4.2}$ and $C_{S5.4.3}$ may still fulfill $C_{S5.4.1}$ since the debtors of those commitments may satisfy the consequents before the deadlines.

Definition 56. *Let $C_m, C_j \in \mathcal{C}$, $\mathrm{deleg}(C_j, C_m)$. C_j is an improper delegation of C_m, denoted $\mathrm{deleg}_{imp}(C_j, C_m)$, iff*

- *C_j is an improper consequent delegation of C_m, or*
- *C_j is an improper causal delegation of C_m, or*
- *$\exists\ C_k \in \mathcal{C}$ such that $\mathrm{deleg}(C_k, C_m)$ and $\mathrm{deleg}_{imp}(C_j, C_k)$.*

[14] In other words, $\mathcal{I}_k \in \sigma$ is an absolute deadline, $\nexists k' > k$ s.t. $\mathcal{I}_{k'} \in \sigma$ is an absolute deadline.

A delegation which is not improper is called a *proper delegation* and denoted by $\text{deleg}_{prop}(C_i, C_m)$ (meaning that C_i is a proper delegation of C_m).

This concludes the temporal analysis of direct and indirect delegations, which allows us to identify what we need to look for, in order to detect and possibly prevent faulty activity executions. In the next section, we describe a distributed, collaborative procedure to detect improper delegations.

5.4. Commitment monitoring

In abstract terms, at a given time of observation t, a monitoring process $\mathcal{M}$ identifies all the improper delegations $\mathcal{M}_t$ that occurred up to t.

Definition 57. *A monitoring process $\mathcal{M}$ 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 $\mathcal{M}_t = \{(C_i, C_j) | \text{deleg}_{imp}(C_i, C_j)\}$.

We use $\mathcal{M}$ to denote the *monitoring process* itself. $\mathcal{M}$ is a collaborative, distributed process that consistently checks for improper delegations during the protocol's execution. Similar to diagnosis, which looks for assumptions over executions of activities that classify these executions either as correct or faulty [47], monitoring seeks to detect and possibly prevent faulty executions. It is important that monitoring is carried out at runtime, in reaction to events that bring about properties characterising a possibly faulty state. $\mathcal{M}$ is an abstract concept, as we cannot assume that there is always an agent who has complete global knowledge.

A monitoring process can be initiated by an agent $X \in \mathcal{A}$ that detects an anomaly. If, for instance, the anomaly is associated with a property P that has not been brought about within a deadline due to a commitment C_m, and X had delegated C_m to another agent Y, X will ask Y's collaboration in understanding what went wrong. Accordingly, Y will run a 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. In the end, a monitoring task around C_m produces a set $\mathcal{M}_t(C_m) = \{(C_i, C_m) | \text{deleg}_{imp}(C_i, C_m)\}$, $\mathcal{M}_t(C_m) \subseteq \mathcal{M}_t$. If $\mathcal{M}_t(C_m) \neq \emptyset$, an exception is raised.

In a concrete implementation, answering to a monitoring request could be implemented as a background agent behavior, 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" behavior. A possible concrete architecture for distributed monitoring is described in [12], 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 has a separate Reactive Event Calculus ($\mathcal{REC}$) [48] engine running in background, providing the core reasoning ability required to implement the "answering to a monitoring request" behavior and check that commitments are properly fulfilled, or, in case of exception, to raise a flag, and possibly involve other agents in the monitoring process.

5.4.1. Distributed monitoring procedure

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 5.1 summarizes the notation.

Given an agent $X \in \mathcal{A}$ and a commitment $C_m \in \mathcal{C}$, a derivation $X \rhd^{\emptyset}_{\delta_{out}} C_m$ starts when X decides to monitor one of its commitments C_m. The $\emptyset$ symbol (which is an

Table 5.1. Notation used for the monitoring procedure.

Symbol	Description
δ_{pro}	set of proper delegations of a given commitment
δ_{imp}	set of improper delegations of a given commitment
$\mathcal{C} = \{\langle C, \delta_{pro}, \delta_{imp}\rangle, \dots\}$	for each known commitment C, a triplet consisting of: C, its proper delegations δ_{pro}, and its improper delegations, δ_{imp}
$X \bigtriangledown_{\mathcal{REC}} \mathcal{C}$	$\mathcal{C}$ contains all (locally) known information about X's commitments, extracted by a $\mathcal{REC}$ reasoner such as ComMon [49]
δ_{exc}	commitments to be excluded from a monitoring process
δ_{out}, δ_j, δ_k	output of monitoring processes (sets of improper delegations)
$X \rhd_{\delta_{out}}^{\delta_{exc}} C_m$	the result of a monitoring process *issued* by X about C_m and excluding δ_{exc} is the set of improper delegations δ_{out}
$X \rhd_{\delta_{out}}^{\delta_{exc}} Y \gg C_m$	the result of a monitoring process *requested* by X to agent Y about C_m and excluding δ_{exc} is the set of improper delegations δ_{out}
debtor(C_m, X)	agent X is the debtor of commitment C_m
delegatee(C_m, C_j, Y)	agent Y is the delegatee in the delegation C_j of C_m

input to the derivation) signifies that no commitment is initially excluded from the monitoring process, because no commitment has been analysed yet. The output δ_{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 δ_{exc} when performing further derivation. In this way, we prevent agents from analysing the same commitment more than once. In the rules for collaborative reasoning ("social monitoring"), we use the notation $\gg$, whose semantics is given in S_3, to indicate a request for monitoring.

Each agent involved will only use its local knowledge of commitments to contribute to the derivation by applying local and social 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 $\mathcal{REC}$[15] 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.

$$L_1) \quad \frac{X \bigtriangledown_{\mathcal{REC}} \mathcal{C} \ \wedge \ \langle C_m, \delta_{pro}, \delta_{imp} \rangle \in \mathcal{C} \ \wedge \ \delta_{out} = \delta_{imp} \setminus \delta_{exc} \ \wedge \ \delta_{out} \neq \emptyset}{X \rhd_{\delta_{out}}^{\delta_{exc}} C_m}$$

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 δ_{exc} (the set containing the commitments that are already processed during monitoring), then they are added to the output (δ_{out}) of the monitoring process. Consider the commitments in Example 5.3: let X be the merchant, C_m be $C_{S5.3.3}$, and $\delta_{exc} = \emptyset$. Now, when the merchant queries his $\mathcal{REC}$ engine, he will find out that $\delta_{pro} = \{C_{S5.3.1}\}$ and $\delta_{imp} = \{C_{S5.3.2}\}$. Thus, $\delta_{out} = \{C_{S5.3.2}\}$ which contains the only improper delegation of $C_{S5.3.3}$.

[15] $\mathcal{REC}$ ($\mathcal{R}$eactive $\mathcal{E}$vent $\mathcal{C}$alculus) is an event calculus-based language and reasoning framework [48, 50]. ComMon is an award-winning $\mathcal{REC}$-based monitoring engine [49]. ComMon can be downloaded from `http://ai.unibo.it/projects/comMon`.

$$L_2) \quad \frac{X \bigtriangledown_{\mathcal{RECC}} \mathcal{C} \ \wedge \ \langle C_m, \delta_{pro}, \delta_{imp} \rangle \in \mathcal{C} \quad \wedge \ (\delta_{pro} \cup \delta_{imp}) \setminus \delta_{exc} = \emptyset \ \wedge \ \text{debtor}(C_m, X)}{X \rhd_{\emptyset}^{\delta_{exc}} C_m}$$

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 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.

$$S_1) \quad \frac{\begin{array}{c} X \bigtriangledown_{\mathcal{RECC}} \mathcal{C} \ \wedge \ \langle C_m, \delta_{pro}, \delta_{imp} \rangle \in \mathcal{C} \ \wedge \ C_j \in \delta_{pro} \setminus \delta_{exc} \\ \wedge \ \text{delegatee}(C_m, C_j, Y) \ \wedge \ X \rhd_{\delta_j}^{\delta_{exc}} Y \gg C_j \\ \wedge \ X \rhd_{\delta_k}^{\delta_{exc} \cup \{C_j\}} C_m \ \wedge \ \delta_{out} = \delta_j \cup \delta_k \end{array}}{X \rhd_{\delta_{out}}^{\delta_{exc}} C_m}$$

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 δ_j. X will then continue monitoring its other delegations, excluding C_j from the process, thereby obtaining a result δ_k. The final result δ_{out} is the union of the two partial results, $\delta_j \cup \delta_k$.

$$S_2) \quad \frac{X \bigtriangledown_{\mathcal{REC}} \mathcal{C} \ \wedge \ \langle C_m, \delta_{pro}, \delta_{imp} \rangle \in \mathcal{C} \ \wedge \ (\delta_{pro} \cup \delta_{imp}) \setminus \delta_{exc} = \emptyset \quad \wedge \text{debtor}(C_m, Y) \ \wedge \ X \neq Y \ \wedge \ X \rhd^{\delta_{exc}}_{\delta_{out}} Y \gg C_m}{X \rhd^{\delta_{exc}}_{\delta_{out}} C_m}$$

By rule S_2, if there is no locally known delegation of the monitored commitment, and X is the creditor of that commitment, the result δ_{exc} is that provided by C_m's debtor Y as Y answers X's request for monitoring.

$$S_3) \quad \frac{Y \rhd^{\delta_{exc}}_{\delta_{out}} C_m}{X \rhd^{\delta_{exc}}_{\delta_{out}} Y \gg C_m}$$

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 $\mathcal{M}$. Indeed, under the assumption that the $\mathcal{REC}$ reasoner provides sound and complete results (one such reasoner is described in [48]), we can prove the following theorems:

Theorem 3 (Soundness)**.** *Given a commitment $C_m \in \mathcal{C}_T$, and an agent $X \in \mathcal{A}$, if $X \rhd^{\emptyset}_{\delta_{out}} C_m, C_i \in \delta_{out}$, then ($C_i$, C_m) $\in \mathcal{M}_T$.*

By Theorem 3, 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 57).

Theorem 4 (Completeness)**.** *$\forall$ (C_i, C_m) $\in \mathcal{M}_T$, $\exists$ an agent $X \in \mathcal{A}$ and a derivation*

$X \rhd^{\emptyset}_{\delta_{out}} C_m$ *such that* $C_i \in \delta_{out}$.

By Theorem 4, for any two given commitments C_i, $C_m \in \mathcal{C}_T$, 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.

5.4.2. Implementation

We implemented a proof-of-concept prototype. We wrote specifications in the $\mathcal{REC}$ language, and relied on ComMon [49] for monitoring of commitments. More details on the implementation can be found in Appendix D.

5.5. Case Study

Let us now build two different settings to demonstrate how our approach works.

5.5.1. Request Credit Card

This setting builds on Example 5.4. We have the following three commitments to represent the process for the client to request a credit card from the bank:

- $C_1 = C($*bank*, *ali*, *requested*, *delivered*(r[4, 7])): The bank must deliver the credit card within 7 days of the client's request;
- $C_2 = C($*courier*, *bank*, *printed*, *delivered*(r[2, 3])): When the card is requested, the bank notifies the office for printing the card;
- $C_3 = C($*office*, *bank*, *confirmed*, *printed*(r[2, 3])): Then, the courier delivers the card to the client.

The client only has access to commitment C_1^t, and he is aware of two actions, for requesting and getting the card delivered.

- request(*ali*, *bank*) $\rightarrow$ requested.
- deliver(*courier*, *ali*) $\rightarrow$ 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. Now, consider now the following trace:

4	request(*ali*, *bank*)	(client requests card from bank on day 4)
7	confirm(*bank*, *office*)	(bank confirms request)
10	print(*office*, *courier*)	(office produces card and gives it to courier)

The following commitments are in place at time 11:

$C_1 = C(bank, ali, \top, delivered(\mathrm{a}[8, 11]))$
$C_2 = C(courier, bank, printed, delivered(\mathrm{a}[12, 13]))$
$C_3 = C(office, bank, \top, printed(\mathrm{a}[9, 10]))$

Notice the pattern among these three commitments; C_2 is an implicit delegation of C_1 (Definition 39), and C_3 is an antecedent delegation of C_2 (Definition 41). Then C_3 is delegation-similar to C_1 via C_2.

First, we look at the global monitoring result considering all the commitments in the system. Assume that no delivery has occurred until time 12. C_1 is indeed violated since its deadline has passed. Because of the similarity relation, C_2 and C_3's deadlines together affect C_1. Even though the printing of the card is completed at day 10, the courier has 3 more days for delivery, which will eventually exceed C_1's deadline. When the delivery is completed at time 13, the commitment of the courier to the bank (C_2) is fulfilled. However, the commitment of the bank to Ali (C_1) is violated. Here, the bank should have confirmed the client's request earlier, and notified the office accordingly.

Next, we look at the agents' local reasoning:

- Ali: $\mathcal{C} = \{\langle C_1, \{\}, \{\}\rangle\}$
  - (i) Rules L_1, L_2, and S_1 do not apply (see Figure E.1 in Appendix E for $\mathcal{REC}$ output),
  - (ii) Rule S_2 delegates to the bank.
- Bank: $\mathcal{C} = \{\langle C_1, \{\}, \{C_2, C_3\}\rangle, ...\}$
  - (i) Rule L_1 applies, and finds an exception (see Figure E.2 in Appendix E for $\mathcal{REC}$ output),
  - (ii) Rule S_3 propagates the result to Ali.

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

4	request(*ali*, *bank*)
5	confirm(*bank*, *office*)
7	print(*office*, *courier*)

Figures E.3 and E.4 in Appendix E show the output of $\mathcal{REC}$ for this case. Note that there is no fluent for *improperDelegation* or *exception*.

5.5.2. Refurbish House

Let us now focus on another setting, which is described earlier. Consider again the commitments and delegations in Figure 5.1. Recall that the landlord wants to refurbish his house, thus makes a commitment with the contractor, who in turn makes commitments with the builder. Note that we also have conjunction of fluents for the consequents of some commitments (C_1, C_2, C_5). The following rules show how the fluents are described as conjunctions[16] .

- refurbished $\leftarrow$ kitchen $\wedge$ bathroom.
- bathroom $\leftarrow$ shower $\wedge$ tiles.

[16] Appendix D describes how conjunction is handled in $\mathcal{REC}$.

- kitchen ← appliances ∧ painted.

Now, consider the trace of events for this set of commitments and fluents:

4	pay(*landord*, *contractor*)
4	payKitchen(*contractor*, *builder*)
4	payBathroom(*contractor*, *builder*)
7	payShower(*builder*, *assembler*)
9	supplyAppliances(*supplier*, *builder*)
14	assembleShower(*assembler*, *builder*)

Let us analyze what the improper delegations are according to this trace of events. When the landlord pays to the contractor at time 4, C_1 becomes C(contractor, landlord, true, refurbished(a[9, 14])). Following that, the contractor pays for the kitchen at the same time point, which makes C_5 C(builder, contractor, true, kitchen(a[9, 14])). Now, note that C_5 is an implicit delegation of C_1, C_6 is an implicit delegation of C_5, and C_7 is an antecedent delegation of C_6. Thus, we have a *wia-chain* delegation from C_7 to C_1 (see Definition 47). If we look at the deadline for C_7, it can be fulfilled in the interval a[8, 10]. This is an improper delegation, since when combined with C_6's deadline interval r[3, 5], it exceeds the deadline of C_5 (and C_1).

Similarly, let us take a look at the delegations of C_2, which is an implicit delegation of C_1. C_3 is an implicit delegation of C_2, and C_4 is an antecedent delegation of C_3. Again, we have a *wia-chain* delegation from C_4 to C_1. This is also an improper delegation, since the combined deadlines of C_3 and C_4 exceed that of C_2 (and C_1). Figure E.5 in Appendix E shows the output of $\mathcal{REC}$ for the builder.

In this chapter, we have made an extensive analysis of the temporal relations for commitment delegations. The delegation mechanism gives great flexibility to commitment protocols. However, it also lay itself open to misuse and may induce possible mismatches amongst agent beliefs about deadlines associated with properties. Improper delegation may 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. The distributed procedure assumes that agents are able to send each other requests for monitoring a commitment, and gather local information about such a commitment, by applying the definitions we gave in this chapter, and possibly by collaborating with other agents that are part of the chain of delegation. To gather information about commitments and more in general about the state of properties (fluents) in a local environment, agents may use a facility such as ComMon [49]. This is a $\mathcal{REC}$-based [17, 28, 50] commitment monitoring tool, proven to be sound and complete [48]. Because there is a sound a complete method to reason locally, distributed reasoning is also sound and complete.

Besides this theoretical and practical result, our work contributes to research in commitment-based agent interaction by investigating what agents should look at in order to enable commitment monitoring, e.g. for e-commerce applications. The applicability of our method is quite broad. In practical applications, "properties" could be for example indicators of accomplishment of specific tasks. Because our language accommodates conjunctive properties, it can be used for modeling complex contracts. Thus our work has a theoretical and practical significance. We do not claim that all agents must be $\mathcal{REC}$-enabled or that they must use ComMon for monitoring their commitments, but we identify how agents should explore a search space of commitments, so to say, in order to enable monitoring. Concrete implementations of our monitoring procedure could be based standardized agent languages. We did not cover this aspect here. However, an important result is that there are off-the-shelf tools that could be used to implement core functionalities used by our framework. We did not formally study the complexity of the reasoning procedure, but we conjecture that it is linear in the size of the delegation tree. That depends on the applications, but we can assume that in typical e-commerce scenarios it would be linear or at worst polynomial in the number of agents (it definitely is linear in the number of commitments, as a commitment cannot be a direct delegation of more than one commitment).

6. APPLYING ARGUMENTATION TO DIAGNOSIS

In this chapter, we show an application of Assumption-Based Argumentation (ABA) [51–54] for reasoning about exceptions in e-commerce contracts. In open multi-agent systems, agents need to deal with missing or conflicting information throughout their protocol execution. Consider the delivery protocol that is used for diagnosis and monitoring. Assume that the customer wants his package to be delivered to his home. After the delivery deadline has passed, he senses an exception since he has not received the package. When he asks the courier, he finds out that the courier has delivered the package to the door keeper. Note that the customer and the courier has conflicting information about the outcome of the delivery (two different arguments), for which one of them has a valid justification. Since the courier has the door keeper's signature on the delivery chart, his argument wins over the customer's, and the exception is resolved. We show that argumentation techniques are well suited to carry out such reasoning tasks and to support dialogues for collaborative diagnosis of multiagent contract exceptions. The dialogues provide the information exchange among the agents to enable diagnostic activities to step from agent to agent until the reason of the exception is found. Reasoning is based on the ABA argumentation framework. Thanks to its grounding on consolidated argumentation theories, we are able to describe the diagnosis process in a high-level, declarative way. We can enable agents to construct hypotheses (arguments) about what went wrong and exchange such hypotheses between them, and we can ensure that the overall process is deterministic.

6.1. Argumentation Architecture

We will use *c(X,Y,P(O))* and *cc(X,Y,Q(O),P(O))* as commitment templates, for base-level and conditional commitments, respectively. Here, we use O as a variable standing for an object, e.g., a certain *book*, *delivered(book)*. All variables are implicitly universally quantified, and the actual commitments are instances of these templates. In the case of conditional commitments, when Y satisfies *Q(O)* with a specific substitute o for O, then X becomes committed to Y for satisfying *P(o)*.

We will assume that commitments are represented with respect to an underlying *commitment language* $\mathcal{L}_c$ shared amongst all agents. In the remainder of the chapter we will use the following conventions: as earlier in this section, upper-case letters denote variables; variables X, Y, Z are used to represent agents; P is used to represent an atomic property; Q is used to represent a property in the form of a literal (of the form P or $\neg P$); R is used to represent a conjunction of (literal) properties. As standard, $\neg\neg P$ is P. As in Prolog, _ represents an anonymous variable.

For simplicity, we will assume that there is at most one *cc(x, y, q(o), p(o))* for a given *x*, *y*, and *p(o)*. In other words, we do not consider those cases in which an agent x conditionally commits to the same agent y to bring about the very same property *p(o)* using two different contracts, which would unnecessarily complicate the diagnosis process. Such cases are left as an extension for future work. Also, note that, for the sake of simplicity and because we focus on diagnosis procedures, we will ignore (and not explicitly model) the *active* commitment state.

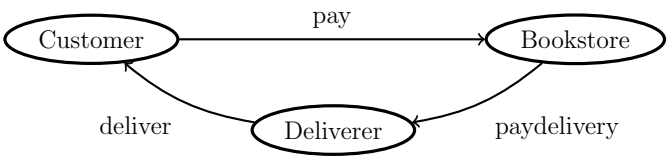


Figure 6.1. Delivery process.

To describe the delivery process shown in Figure 6.1 using contracts, we use two commitments. One tells that if *customer* pays for an *item*, then *bookstore* will be committed to deliver that item, *cc*(*bookstore*, *customer*, *paid*(*book*), *delivered*(*book*)). The second one tells that if *bookstore* pays for the delivery of an *item*, then *deliverer* will be committed to deliver that item, *cc*(*deliverer*, *bookstore*, *paid_delivery*(*book*), *delivered*(*book*)).

As we have shown before, it may happen that the book does not get delivered. Usually, the customer is the first one to realize. We say that the customer detects an exception. Let us say we are now the customer. What shall we do? In order to recover the book, we should first understand what went wrong. We will consider several options: the bookstore may not have correctly place the order with the deliverer; the deliverer may be late; the bookstore may not have yet received the payment; and so

on. A diagnosis process will tell us which of these possibilities truly explains what went wrong. The diagnosis process is initiated by the agent detecting an exception. During this process agents collect evidence from the environment and exchange information with each other.

Following the KGP agent model [55], we will assume that actions may be "physical" (e.g. *pay*), communicative (e.g. *justify*), or sensing actions on the environment (i.e. *question*). In the delivery process, we will consider the following physical actions:

- *pay(customer,bookstore,item):* a customer agent pays a bookstore agent for an item of interest.
- *paydelivery(bookstore,deliverer,item):* the bookstore pays a deliverer for the delivery of an item of interest.
- *deliver(deliverer,customer,item):* a deliverer delivers an item of interest to a customer.

Note that these physical actions are consistent with the events (e.g., pay, deliver) that we have used in previous chapters where we have discussed agent reasoning with $\mathcal{REC}$.

The sensing actions amount to a particular type of database lookup for evidence gathering, that we call *evidence request exchange*, between an agent and some trustworthy element of the environment that can produce evidence. By doing so, we abstract away from the specific ways agents interact with the environment. Note that each agent has a partial view of the overall environment. Thus we call E_X the environment accessible to agent X. We use two types of sensing actions:

- *question*(X, E_X, P): agent X looks up E_X to check whether property P holds or not.
- *answer*(X, E_X, Q): agent X gets to know from E_X that Q holds.

Inter-agent dialogues are instead based on the following utterances (communicative actions):

- *explain*(X, Y, P): agent X sends a diagnosis request to Y, asking for a justification for a given property P.
- *justify*(X, Y, Q, P): agent X provides agent Y with a justification Q to why P holds.
- *rebut*(X, Y, Q, $\neg P$): agent X provides agent Y with a justification Q to why P does *not* hold.

6.2. Reasoning

We show how agents can reason about commitment exceptions based on assumption-based argumentation (ABA) [51–54], which we first briefly review below. We choose this framework because it can deal with inconsistent and incomplete information in general and in support of decision-making, it can generate justifications that can be communicated across agents, and because of its strong theoretical properties and the fact that it is equipped with provably correct computational mechanisms, that will support any future deployment, e,g, using the publicly available CaSAPI ABA framework [56][17] .

6.2.1. Assumption-Based Argumentation (ABA)

ABA is a general-purpose argumentation framework where arguments and attacks between them are built from *ABA frameworks*, which are tuples $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ where:

- $(\mathcal{L}, \mathcal{R})$ is a *deductive system*, with $\mathcal{L}$ a language and $\mathcal{R}$ a set of inference rules,
- $\mathcal{A} \subseteq \mathcal{L}$, referred to as the set of *assumptions*,
- $\overline{\ }$ is a (total) mapping from $\mathcal{A}$ into $\mathcal{L}$, where $\overline{x}$ is referred to as the *contrary* of x.

[17] `http://www.doc.ic.ac.uk/~dg00/casapi.html`

Here, we assume that inference rules have the syntax $s_0 \leftarrow s_1, \ldots, s_n$ (for $n \geq 0$) where $s_i \in \mathcal{L}$. We refer to $s_1, \ldots, s_n$ as the *premises* and to s_0 as the *head* of the rule. If $n = 0$, we represent a rule simply by its head and we call the rule a *fact*. As in [52], we will restrict attention to *flat* ABA frameworks, such that no assumption occurs in the head of a rule.

Rules may be domain-dependent or not, and some of the premises of rules may be assumptions. These can be used to render the rules defeasible. In this setting, contraries of assumptions can be seen as representing "defeaters". Assumptions can also be used to fill gaps in incomplete knowledge/beliefs, and in this setting contraries are reasons for not making some gap-filling choices. Also, assumptions can be used to resolve inconsistencies (by making these depend upon assumptions that can be defeated).

An *(ABA) argument* in favour of a sentence $c \in \mathcal{L}$ supported by a set of assumptions $A \subseteq \mathcal{A}$ is a proof of c from A and (some of) the rules in $\mathcal{R}$. This proof can be understood as a tree (with root the claim and leaves the assumptions), as in [54], as a backward deduction, as in [52, 53], or as a forward deduction, as in [51], equivalently. For the purposes of this chapter, we will use the notation $A \vdash_R c$ to stand for an argument for c supported by A by means of rules $R \subseteq \mathcal{R}$. When the rules can be ignored, we write an argument $A \vdash_R c$ simply as $A \vdash c$. An argument $A \vdash c$ *attacks* an argument $A' \vdash c'$ if and only if $c = \overline{\alpha}$ for some $\alpha \in A'$.

Several "semantics" for ABA have been defined in terms of sets of assumptions fulfilling a number of conditions. These are expressed in terms of a notion of attack between sets of assumptions, where $A \subseteq \mathcal{A}$ attacks $A' \subseteq \mathcal{A}$ if and only if there is an argument $B \vdash c$, with $B \subseteq A$, attacking an argument $B' \vdash c'$, with $B' \subseteq A'$.

We will focus on the following notions:

- $A \subseteq \mathcal{A}$ is *conflict-free* if and only if A does not attack itself
- $A \subseteq \mathcal{A}$ is *admissible* if and only if A is conflict-free and attacks every $B \subseteq \mathcal{A}$ that

attacks A

- $A \subseteq \mathcal{A}$ is *preferred* if and only if A is (subset) maximally admissible.

Note that these notions can be equivalently expressed in terms of arguments, rather than assumptions, as shown in [53].

Given an ABA framework $\mathcal{F}=\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ and a (conflict-free, admissible or preferred) set of assumptions $A \subseteq \mathcal{A}$ in $\mathcal{F}$, the (conflict-free, admissible or preferred, respectively) *extension* $\mathcal{E}_{\mathcal{F}}(A)$ is the set of all sentences supported by arguments with support a set of assumptions $B \subseteq A$:

$$\mathcal{E}_{\mathcal{F}}(A) = \{s \in \mathcal{L} | \exists B \vdash s \text{ with } B \subseteq A\}.$$

Note that conflict-free, admissible and preferred extensions are guaranteed to exist, for any ABA framework.

In the remainder of this section, we give an ABA framework supporting the reasoning of our agents. We will assume that each agent is equipped with an ABA framework $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ } \rangle$ such that the commitment language, $\mathcal{L}_c$, is contained in the internal language $\mathcal{L}$ underlying the ABA framework, namely $\mathcal{L}_c \subseteq \mathcal{L}$. We will leave this $\mathcal{L}$ implicit, and focus on rules, assumptions and contraries. Indeed, $\mathcal{L}$ will always consist of the set of all sentences occurring in all the given rules, as well as the given assumptions and contraries. We will give rules/assumptions/contraries as schemata, standing for all their ground instances over appropriate universes (for agents and properties). Until the case study, we will not focus on any specific agent and define rules, assumptions and contraries for a generic agent X. Assumptions will be of the form $asm(\_)$. The contrary of $asm(a)$ will be of the form $c\_asm(a)$, for any a, formally: $\overline{asm(a)} = c\_asm(a)$.

6.2.2. Domain-Dependent Rules

These depend on the domain of application. In our example, these rules are related to the delivery process and include:

(F_1) $by\_contract(cc(bookstore, customer, paid(book), delivered(book)))$.

(F_2) $by\_contract(cc(deliverer, bookstore, paid\_delivery(book), delivered(book)))$.

(F_3) $effect(pay(customer, bookstore, book), paid(book))$.

(F_4) $effect(paydelivery(bookstore, deliverer, book), paid\_delivery(book))$.

(F_5) $effect(deliver(deliverer, customer, book), delivered(book))$.

(R_1) $justification(\neg paid\_delivery(book), \neg delivered(book)) \leftarrow$
$\neg paid\_delivery(book), \neg delivered(book)$.

The first two facts model the conditional commitment between the *customer* and *bookstore* agents (F_1) and between the *bookstore* and *deliverer* agents (F_2). The other facts (F_3–F_5) describe the action-consequence relations. The rule R_1 represents that a problem in the delivery payment may be the reason for no delivery.

We will assume that the domain-dependent rules only define predicates *by_contract*(_), *effect*(_, _), and *justification*(_, _) as well as the predicates *believe*(_), *answer*(_, _), and *executed*(_) used below.

6.2.3. General-Purpose Reasoning Rules

These rules are held by agent X, independently of the specific scenario for exception diagnosis. They consist of belief rules (BR), commitment rules (CR) and action rules (AR).

Belief rules: These allow to "internalise" beliefs drawn from observations and expected effects of actions, unless there are reasons not to do so. They are required to avoid epistemic inconsistencies to arise, such as $\mathit{believe}(\mathit{paid}(\mathit{book}))$ and $\mathit{believe}(\neg\mathit{paid}(\mathit{book}))$.

(BR_1) $P \leftarrow \mathit{believe}(P), \mathit{asm}(P).$

(BR_2) $\neg P \leftarrow \mathit{believe}(\neg P), \mathit{asm}(\neg P).$

(BR_3) $\mathit{believe}(\neg P) \leftarrow \mathit{asm}(\mathit{believe}(\neg P)).$

(BR_4) $P \leftarrow \mathit{answer}(X, E_X, P).$

(BR_5) $\neg P \leftarrow \mathit{answer}(X, E_X, \neg P).$

(BR_6) $\mathit{believe}(Q) \leftarrow \mathit{executed}(A), \mathit{effect}(A, Q).$

Belief rules BR_1–BR_3 are defeasible, as represented by the presence of assumptions in their premises. The following rules for the contraries of the assumptions are their defeaters:

(BR_7) $c\_asm(P) \leftarrow \neg P.$

(BR_8) $c\_asm(\neg P) \leftarrow P.$

(BR_9) $c\_asm(believe(\neg P)) \leftarrow believe(P).$

Note that rules BR_1–BR_2 could be combined within a single rule

$$Q \leftarrow believe(Q), asm(Q).$$

Similarly for BR_4–BR_5 and BR_7–BR_8. However, we prefer to leave them separate for clarity, and to better underline the asymmetry in our treatment of positive and negative literals (properties). In particular, note that rules BR_3 and BR_9, together, model a form of closed-world assumption/negation-as-failure over properties, where $believe(\neg(\neg P))$ can be interpreted as the negation-as-failure of P.

Rules BR_4–BR_9 are strict, as there are no assumptions in their premises. Rules BR_4–BR_5 allow to turn observations (that a specific answer has been obtained after consulting the agent's environment E_X) into (internalised) beliefs. The fact that these rules are strict represents that we consider the environment as beyond doubt.

Note however that, should this not be the case, we could turn BR_4–BR_5 into defeasible rules by adding suitable assumptions and definitions for their contraries. Rule BR_6 allows to introduce (non-internalised) beliefs about the effects of executed actions. These beliefs may then become internalised by applying rules BR_1–BR_2. Specific definitions of effects of actions belong to the domain-dependent part of the beliefs.

As an illustration of the use of these rules extended with domain-dependent rules and under the notion of admissible sets of assumptions/arguments, consider the fol-

lowing cases:

- *believe*(p) is the only domain-dependent rule;
  then $\{asm(p)\}$ is the only (non-empty) admissible set of assumptions, supporting argument $\{asm(p)\} \vdash p$ in favour of p;
- *believe*($\neg p$) is the only domain-dependent rule;
  then $\{asm(\neg p)\}$ and $\{asm(believe(\neg p))\}$ and their union are the only (non-empty) admissible sets of assumptions, all supporting arguments in favour of $\neg p$;
- *believe*(p) and *believe*($\neg p$) are the only domain-dependent rules;
  then $\{asm(p)\}$ and $\{asm(\neg p)\}$ are the only (non-empty) admissible sets of assumptions, representing alternative choices for resolving the given inconsistency; the agent can choose whichever alternative;
- there are no domain-dependent rules;
  then $\{asm(\neg p),\ asm(believe(\neg p))\}$ is the only (non-empty) admissible set of assumptions, supporting an argument in favour of $\neg p$.

Note that (i) in no case an agent can assume both $asm(p)$ and $asm(\neg p)$, (ii) it can only derive p if it has some evidence for p, and (iii) it can only assume $asm(believe(\neg p))$ if it cannot derive *believe*(p). Thus, the following result holds:

Property 1. *Given an ABA framework with rules $BR_1 - BR_9$ and any set of domain-dependent rules*

(i) every preferred extension $\mathcal{E}$ is consistent*, namely there does not exist any property P such that both P and $\neg P$ belong to $\mathcal{E}$;*

(ii) every preferred extension $\mathcal{E}$ is total*, namely for every property P either P or $\neg P$ belongs to $\mathcal{E}$.*

This result directly follows from our definition of belief rules and from the definition of preferred extensions in ABA.

Commitment rules: These model the evolution of commitments (*by_contract(. . .)*)

and commitment states (*fulfilled*(...) and *violated*(...)) [18] during the agent's life-cycle.

(CR_1) *fulfilled*$(c(X,Y,P)) \leftarrow$ *by_contract*$(c(X,Y,P)), P,$ *asm*(*fulfilled*$(c(X,Y,P)))$.

(CR_2) *by_contract*$(c(X,Y,P)) \leftarrow$ *by_contract*$(cc(X,Y,Q,P)), Q,$
asm(*by_contract*$(c(X,Y,P)))$.

(CR_3) *violated*$(c(X,Y,P)) \leftarrow$ *by_contract*$(c(X,Y,P)), \neg P,$ *asm*(*violated*$(c(X,Y,P)))$.

(CR_4) *c_asm*(*fulfilled*$(c(X,Y,P))) \leftarrow \neg P$.

(CR_5) *c_asm*(*by_contract*$(c(X,Y,P))) \leftarrow$ *by_contract*$(cc(X,Y,Q,P)), \neg Q$.

(CR_6) *c_asm*(*violated*$(c(X,Y,P))) \leftarrow P$.

Note that, like belief rules, commitment rules may also be defeasible, since commitments change during the agent's life-cycle. That is, if a commitment has become active from conditional based on an assumption that a proposition holds, then if we have a defeater that defeats that assumption, the commitment becomes conditional again. In particular, $CR_1 - CR_3$ above are defeasible.

Property 2. *Given an ABA framework with rules* $CR_1 - CR_6$, $BR_1 - BR_9$ *and any set of domain-dependent rules, for any preferred extension* $\mathcal{E}$, *for any agent* Y:

- *there does not exist any property* P *such that fulfilled*$(c(X,Y,P))$ *and violated*$(c(X,Y,P))$ *belong to* $\mathcal{E}$;
- *for every property* P *such that by_contract*$(cc(X,Y,P))$ *belongs to* $\mathcal{E}$, *either*

[18] As discussed before, we ignore the *active* commitment state.

fulfilled$(c(X,Y,P))$ *or violated*$(c(X,Y,P))$ *belongs to* $\mathcal{E}$.

Part 1 follows from Part 1 of Property 1, since P and $\neg P$ are respectively conditions of CR_1 and CR_3. Part 2 follows from Part 2 of Property 1.

Action rules: These are of two types: for determining whether and how to consult the environment (action *question*) or for determining whether and how to conduct a *request explanation* dialogue. The first type of rules are:

(AR_1) $question(X, E_X, \neg P) \leftarrow violated(c(Y,X,P)), asm(question(X, E_X, \neg P))$.

(AR_2) $question(X, E_X, \neg Q) \leftarrow violated(c(Y,X,P)), by\_contract(cc(Y,X,Q,P)),$
$asm(question(X, E_X, \neg Q))$.

where contraries of assumptions are defined by rules:

(AR_3) $c\_asm(question(X, E_X, \neg P)) \leftarrow by\_contract(c(Y,X,P)), answer(X, E_X, \neg P)$.

(AR_4) $c\_asm(question(X, E_X, \neg Q)) \leftarrow by\_contract(cc(Y,X,Q,P)),$
$answer(X, E_X, \neg P), asm(question(X, E_X, \neg P))$.

(AR_5) $c\_asm(question(X, E_X, \neg Q)) \leftarrow by\_contract(cc(Y,X,Q,P)),$
$answer(X, E_X, \neg Q)$.

AR_3 and AR_5 prevent a *question* by X to its environment on a violated property or a commitment condition if the question has already been answered. AR_4 forces

a preference of a question about a violated commitment over a question about the condition of that commitment.

The second type of action rules regulate dialogues between agents. These are as follows:

$$(DR_1)\ \mathit{explain}(X, Y, \neg P) \leftarrow \mathit{violated}(c(Y, X, P)), \mathit{by\_contract}(cc(Y, X, Q, P)), \\ \mathit{answer}(E_X, X, \neg P), \mathit{answer}(E_X, X, Q), \mathit{asm}(\mathit{explain}(X, Y, \neg P)).$$

$$(DR_2)\ \mathit{c\_asm}(\mathit{explain}(X, Y, \neg P)) \leftarrow \mathit{violated}(c(Y, X, P)), \mathit{violated}(c(Z, Y, P)).$$

Namely, in case of a violation over P by X towards Y, and after having already checked within its environment, X asks Y for an explanation (DR_1), unless Y itself is object of a violation on P by some other agent Z (DR_2).

$$(DR_3)\ \mathit{justify}(X, Y, R, \neg P) \leftarrow \mathit{explain}(Y, X, \neg P), \mathit{justification}(R, \neg P), \\ \mathit{asm}(\mathit{justification}(R, \neg P)).$$

$$(DR_4)\ \mathit{rebut}(X, Y, R, P) \leftarrow \mathit{explain}(Y, X, \neg P), \mathit{justification}(R, P), \\ \mathit{asm}(\mathit{justification}(R, P)).$$

$$(DR_5)\ \mathit{c\_asm}(\mathit{justification}(R, X)) \leftarrow \mathit{justification}(R, \neg X).$$

6.3. Case Study

We present here two case studies of the diagnosis process, in the delivery scenario we have used throughout this chapter.

6.3.1. Customer's Fault

This case presents the trace of the diagnosis process on the exception that is caused by the customer (the customer has not paid for the item correctly, although he thinks he did).

The customer's domain-dependent rules initially consist solely of $F_1, F_3 - F_5$. The general rules are all domain-independent rules. Let us refer to the resulting ABA framework as $\mathcal{F}$. After the customer pays for the book, one fact is added to $\mathcal{F}$: (F_6) *executed*(*pay*(*customer*, *bookstore*, *book*)) resulting in a new ABA framework $\mathcal{F}'$.

- By F_6, F_3, BR_6 and BR_1, the property *paid*(*book*) becomes supported in the unique preferred extension P of $\mathcal{F}'$, with argument (where we use p to stand for the property)

$$\{asm(p)\} \vdash_{\{F_6,F_3,BR_6,BR_1\}} p.$$

- Moreover, using additionally CR_2 and F_1, a contract

$$by\_contract(bookstore, customer, delivered(book))$$

  can also be derived in the context of P, supported by the argument (where we use c to stand for the contract and $\mathcal{R}_1$ to stand for $\{F_6, F_3, BR_6, BR_1\}$)

$$\{asm(p), asm(c)\} \vdash_{\mathcal{R}_1 \cup \{CR_2,F_1\}} c.$$

- Then, the customer realizes that the book has not been delivered, supported by argument (where we use $\neg d$ to stand for $\neg delivered(book)$)

$$\{asm(\neg d)\} \vdash_{\{BR_3\}} \neg d.$$

- This causes an exception, since an argument for a violation can be supported in P using, additionally, CR_3 ($\mathcal{R}_2 = \mathcal{R}_1 \cup \{CR_2, F_1\} \cup \{BR_3\}$ and v stands for *violated*(*c*(*bookstore*, *customer*,*delivered*(*book*))))

$$\left\{ \; asm(p), asm(c), asm(\neg d), asm(v) \; \right\} \vdash_{\mathcal{R}_2 \cup \{CR_3\}} v.$$

- When the time comes for the agent to think about the possible reasons of the exception, the action rules give support to a possible question for the environment of the customer (E_c). Namely, the following argument is supported (where q stands for *question*(*customer*, E_c, $\neg$*paid*(*book*) and $\mathcal{R}_3 = \mathcal{R}_2 \cup \{CR_3\}$)

$$\left\{ \; asm(p), asm(c), asm(\neg d), asm(v), asm(q) \; \right\} \vdash_{\mathcal{R}_3 \cup \{AR_1\}} q.$$

  Thus, since $q \in P \cap Actions$, the agent cycle selects to perform action q.
- Assume, in return, the customer learns that he did not pay, namely

$$(F_7) \; answer(E_c, \neg paid(book))$$

  is observed and added to $\mathcal{F}'$, resulting in a new ABA framework $\mathcal{F}''$. Let P' be the preferred extension of $\mathcal{F}''$. P' supports the conclusion that the customer did not actually pay for the book, namely

$$\{\} \vdash_{\{F_7, BR_5\}} \neg p.$$

  as well as the argument

$$\{\} \vdash_{\{F_7, BR_5, BR_7\}} c\_asm(p).$$

  This attacks all arguments including $asm(p)$ in their support. Thus, v does not belong to P' and therefore the customer is aware that no commitment is violated (even though no delivery has occurred). Moreover, the customer knows that it

has not paid for the book correctly.

At the end of this process, the customer has been able to remove the exception.

6.3.2. Bookstore's Fault

Now imagine that the bookstore has not paid for the delivery of the item correctly.

As in the previous case, the customer's domain-dependent rules initially are $F_1, F_3 - F_5$. The general rules are all domain-independent rules. Let us refer to the resulting ABA framework as $\mathcal{F}_c$. The bookstore's domain-dependent rules initially consist of $F_1 - F_5$ and R_1. Again, the general rules are all domain-independent rules. Let us refer to the resulting ABA framework as $\mathcal{F}_b$. The customer's reasoning starts and proceeds as before.

However, now, the answer

$$(F_8)\ answer(E_c, paid(book))$$

is observed instead and added to the ABA framework of the customer. Then the dialogue rules lead the customer to ask the bookstore to explain why the book has not been delivered.

- An argument exists in favour of

$$explain(customer, bookstore, \neg delivered(book))$$

using rule such as DR_1 to construct an argument for

$$violated(c(bookstore, customer, delivered(book))$$

, supported by

$$asm(explain(bookstore, customer, delivered(book)))$$

as well as all assumptions for the argument in the previous case.

- Upon receipt of such an explanation request, the bookstore also derives the conclusion that the book has not been delivered, similarly to the previous case (but now the bookstore is constructing the arguments). Then, again similarly to the previous case, a violation is detected (by the bookstore now), that

$$violated(c(bookstore, customer, delivered(book)))$$

and, in addition, a further violation

$$violated(c(deliverer, bookstore, delivered(book)))$$

using rule F_2 instead of F_1.

- The bookstore's ABA framework at this point supports an argument in favour of a new action of querying the environment of the bookstore (E_b):

$$question(bookstore, E_b, \neg paid\_delivery(book))$$

- On observation of

$$answer(E_b, bookstore, \neg paid\_delivery(book))$$

the bookstore knows that he has not paid for delivery, therefore the set of conclusions supported by his ABA framework change. In particular,

$$violated(c(deliverer, bookstore, delivered(book)))$$

is not supported any more, while a *justify* for the failed delivery is now supported.

- Using rule DR_4 and R_1, the bookstore can now answer the customer's request for explanation:

$$justify(bookstore, customer, \neg paid\_delivery(book), \neg delivered(book))$$

In this chapter, we have shown that argumentation techniques, in particular ABA, can be used as a principled way to model agent reasoning about commitment exceptions, and as a basis to run diagnosis dialogues. In spite of the many works on argumentation-based negotiation [57], to the best of our knowledge, this is the first attempt to use argumentation theories in the context of multiagent contracts and exception handling. Dialogues for diagnosis are also another novel contribution, as diagnosis does not seem to fit in any of the purposes of dialogues identified by Walton & Krabbe [58] and in other relevant literature. Dialogues help resolve conflicts among agents that are caused by different arguments created according to the agent's own perception. We have discussed temporal conflicts before when we deal with misalignment of commitments. There, the agents have reasoned only on the outcomes of those conflicts, i.e., by detecting misalignments via the commitment similarity relations. Here, on the other hand, we deal with the conflicts themselves and resolve them using the *justify* or *rebut* dialogue responses.

7. DISCUSSION

In this chapter, we first review relevant literature and discuss the similarities and differences between them and our research. Then, we review our contributions in this thesis. Finally, we discuss possible future directions.

7.1. Related Work

Exceptions are of importance in many computational systems including distributed systems and multiagent systems. Detecting and diagnosing that an exception takes place in a distributed system allows entities to deal with the exception in a timely manner. Hence, instead of a failed execution, the entities, either individually or cooperatively, can work to solve the problem.

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 is a giant knowledge source on business processes which aims at combining the efforts developed so far by separate disciplines [59]. It provides a taxonomy of business processes, and proposes dependencies between those; (i) a *flow* represents that a process consumes the resource produced by another process in order to perform, (ii) a *fit* represents that two processes together produce a resource, and (iii) a *sharing* represents that two processes consume the same resource. In this thesis for e-commerce domain, 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 protocols for each business party, and reason on exceptions with partial knowledge. We mainly focus on flow type dependencies for the delivery process.

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 [60]. 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 a predefined exception handling process that is used to recover from the exception. The MIT Exception Repository is a good resource for identifying exceptions at compile time. In contrast to the static recovery schemes, in this thesis we do not assume that the possible exceptions are static and that they are known to participants ahead of time. Instead, we deal with run-time exceptions that need to be detected and diagnosed during execution.

7.1.1. Commitments

Commitments are proven to be effective in modeling multiagent interactions [4–6, 17, 19]. Commitments can be shared with only relevant parties and thus allow specification of interactions locally. A variety of commitment types and relations among them have been proposed in the literature [12, 23, 61], as well as obligations and prohibitions that can also be related with social commitments [62]. Singh [24] discusses two commitment types: a dialectical commitment represents the word for an agent towards the truth of a certain fact. This is suitable for negotiation or argumentation [15,63,64], where agents base their reasoning on previously committed arguments. A practical commitment, on the other hand, represents an obligation to perform an action in the sense that an agent promises another towards the successful completion of a duty. This is a common way of regulating interactions in e-commerce, and we also employ these commitments in this thesis. Singh further gives a formal syntax and semantics for these two types of commitments based on an extension of linear-time temporal logic.

We make use of the following four commitment states in this thesis: conditional, active, fulfilled, and active. There are, however, other commitment states that have been used in the literature [5, 6]. One of these states is the "expired" commitment state, which describes that a commitment is no longer in effect. One way to formalize this state is to associate temporal constraints to the antecedent of the commitment as

well. If the deadline represented by these temporal constraints is violated, then the commitment becomes expired. Consider the following commitment: $c(store, customer, property(e(3, 10), pay), property(e(15, 30), discount))$. This represents an offer from the store to the customer that if the customer pays between the time points 3 and 10, then he gets a discount. Note that this commitment gets expired after time point 10, and the discount offer becomes invalid.

Commitments account for the constitutive specification of a protocol [42, 65]. They provide flexible execution for the agents as long as their commitments are satisfied. When an exception occurs (e.g., a commitment is violated), there may be several reasons behind it. One such reason for exceptions in constitutively regulated protocols is the misalignment of agents' commitments. That is, the debtor and the creditor have different understandings for the same commitment [12, 23]. In order for agents to interoperate correctly [42] in a distributed protocol, their commitments should be aligned. For example, if the customer expects delivery today, and the merchant thinks the payment is not processed yet (assume that payment is required prior to delivery), then there is obviously a problem between the two agents.

Two key properties of multi-party business processes are the conformance and interoperability of agents. First, the agent's design should conform to the protocol's standards [25, 66]. In other words, the agent should have the necessary capabilities to satisfy its commitments. Moreover, the agent should be interoperable with other agents [22,42]. That is, all agents should infer the same semantics for the protocol rules. Consider the following intuitive example: the customer wishes to pay using his credit card, however the merchant only accepts cash as payment. Obviously, these two agents are not interoperable, since payment by credit card is not defined in the merchant's rule-base. Assume that they have a commitment which tells if payment is processed, then delivery will be done. Now, when the customer makes the payment using his credit card, he thinks that the merchant will deliver. However, looking from the merchant's point of view, the merchant is not committed to deliver since the payment is not processed (until it is in cash). In this thesis, we assume that both agents conform to the protocol they are executing and are interoperable with each other. Even then,

exceptions might occur. We investigate such cases.

There has been past work in the literature on comparing agents' states. One related work is that of Mallya and Singh [67], which focuses on similarity relations for protocol runs. Similar to our work, a state is described as a set of propositions and commitments, and several similarity relations are given to compare states. The idea is to compare protocol runs in terms of states. One major difference between their similarity relations and our satisfiability relation is that their relations are equivalence relations (e.g., supports symmetry) while ours is not. This is mainly related to the motivation behind using those relations. That is, they aim to merge smaller protocols into larger ones by comparing protocol states. We, on the other hand, provide a one-way relation (e.g., not necessarily symmetric) to compare states. Since our main motivation is to ensure that an agent's state complies with its projections, the inverse direction is not significant at all (e.g., whether expectations support actual states). Moreover, it is not necessary to ensure that both states are equivalent.

Social expectations are a mechanism to formalize the outcomes of agent interactions [68]. Similar to commitments, they are used to model open MAS. Moreover, they somewhat capture the unknown, e.g., model unobserved events or private knowledge. As opposed to the state-oriented perspective of commitments that we also employ in this thesis, expectations take a rule-based approach without a specified state [17]. Usually, expectations are inferred from the past interactions of agents. Here, we do not employ rule-based social expectations, but have a notion of expectation (projection) of the future. This expectation is created according to the states of the agent's commitments with others.

Cranefield and Winikoff use model checking on paths to verify social expectations [69]. They formulate expectations as temporal logic rules, and use a model checker to verify whether those expectations are fulfilled or violated over a path. Although the context of their work is similar, our focus in this thesis is different. They use model checking to track the progression of expectations over paths, while we are not interested in the progression as such but whether such expectations are attainable.

Their system is neither agent-based, nor distributed. Thus, the model checker verifies the system properties as a central authority. We, on the other hand, consider a distributed execution, where each agent can only access a portion of the protocol. Given this, an agent can verify properties that are considered relevant, e.g., the customer can check whether the delivery is performed on time. Moreover, an exception for one agent may just be an ordinary case for another. Since both the protocol specification and the states of the execution are distributed, the use of a central model checker is not appropriate for our purposes. From a system's point of view, there is no central node that can access all the data in the system. Agents would be reluctant to share information with each other. For example, if the model checker was running at the courier agent, the customer may not provide all the details about its workings. From a theoretical point of view, a model checker would need the entire model and a property to decide whether the property holds or not. Since we are considering exceptions at run time with only a partial snapshot of the execution, we would not have an exact model of the system to apply model checking on.

In central monitoring systems, tracking the states of individual commitments is an effective way to detect protocol exceptions [28], since all the interactions of agents are observable. Such tracking of commitments can be done with the Event Calculus, which is a logic used for representing events and their outcomes [70]. 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 [7, 28], 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 [68]. It models the agents' interactions through expectations rather than commitments. However, commitments can also be modeled in SCIFF using expectations [17]. Once commitments and deadlines are created, the tracking procedure just runs a monitoring process based on the happened events [7].

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 such exceptions for the involved agents [23, 38, 42] as we have described in this thesis.

Model checking has been used to identify a variety of failures in commitment-based systems. Bentahar *et al.* consider social commitments and its operations in a Kripke-like world model [18]. The commitment semantics and related operations are formalized with CTL using that world model, and some properties are justified regarding desired execution of commitment protocols. Upon that model built with such semantics, model checking can be utilized in verifying the compliance of agents to their commitments. In [71], two properties of commitments are considered for verification; fulfillment and violation. By extending the MCMAS model checker [72], several properties (e.g., safety and liveness) of commitment protocols are verified. In [35], Telang and Singh model several business patterns as commitment interactions and map them onto CTL specifications. Then, the aim is to use model checking to verify whether the underlying operational model (built with commitment semantics and its operations) supports the business specifications. Both in the works of Bentahar *et al.* and Telang and Singh, the aim is to analyze the system as design time. While this is certainly useful, many times exceptions may not be detected at design time. Hence, our development here is towards creating methods to detect exceptions at run time. There are major differences between these works and our work. First of all, we are not only interested in checking whether commitments have been violated or fulfilled through agents' actions but also through the relations that bind the agents. This means that even if an agent doesn't violate a commitment through explicit actions, its relations with other agents can make the commitment violated. Second, both in the works of Bentahar *et al.* and Telang and Singh, the aim is to analyze the system at design time. This is helpful to see if any commitments are going to be violated.

Normative multiagent systems are an alternative to commitment-based proto-

cols, where artificial institutions and organizations are modeled via norms rather than commitments [73–75]. Similar to commitments, norms represent obligations for agents to follow, but they also possess additional properties like power, which is needed to represent the hierarchical behavior in organizations, e.g., whether an agent possessing a certain role can enforce a norm. Norms can also be violated since there are no guarantees on an agent's behavior in an open environment. Sanctions can be specified in the case of norm violations. In this thesis, we do not consider power or the hierarchy among agents when creating or delegating commitments. In fact, in most cases, we start with a system that already includes the commitments among agents. Because, our focus is on run-time exceptions related to commitments, we do not deal with specifying commitments at design time.

In this thesis, we assume that the agents share a common commitment theory, so that when they look at the same commitment, they understand exactly the same things. Thus, we do not deal with misunderstandings that may arise from agents interpreting the same commitment differently. Note that the misalignment of commitments is different from this, where the debtor and the creditor have two different versions of the commitment.

7.1.2. Distributed Diagnosis

Failure detection and diagnosis in distributed systems goes hand to hand with exception handling in multiagent systems. The approaches vary from from logic-based diagnosis [76] and model-based diagnosis [77], to multiagent diagnosis [78–81] and distributed systems diagnosis [82]. If we assume that the protocols in a multiagent system are known to all and that there could exist agents that can monitor the entire set of interactions, then one can develop specific agents to detect exceptions. Poutakidis *et al.* [83] have developed one of the early systems for debugging problems in a multiagent system. Their proposed debugging environment depends on a single (central) debugging agent, which is responsible for monitoring all the interactions among other agents, and signal an error if there is an inconsistency (i.e., the observed interaction does not meet the protocol specification). In the same spirit, in [84], a central agent is

responsible for monitoring the system. The monitoring framework makes use of LTL, and verifies whether a formula holds in the system given a sequence of events that occurred in that system. These approaches would be difficult to use in systems where there is a concern of privacy and that no single agent can see the entire execution of the system, such as the delivery process that we use in this thesis.

One major area of interest for diagnosis is the medical domain. Simply, medical diagnosis is the process of identifying a disease based on the symptoms that are observed on a patient [85]. Several different diagnostic models have been proposed in the literature [86–89] to automate the medical diagnosis process, and to help physicians with their decisions. These approaches range from rule-based systems to temporal logic reasoning based on observed symptoms, and to qualitative methods which allow flexibility in describing incomplete medical knowledge.

There are two fundamentally different approaches to diagnostic reasoning (e.g., where intelligent entities reason on their environment to diagnose faults or errors): heuristic approaches, such as *fault-based diagnosis* (FBD) and diagnosis from the first principles or *model-based diagnosis* (MBD) [90]. 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 modeled instead. Conversely, MBD starts from a model of the structure (components and their connections), a function (or behavior) of the system, and a set of observations indicating a normal behavior. A system is considered faulty if the observed behavior of the system contradicts the behavior predicted by assuming that all of its components are correct [77].

As pointed out by Kalech and Kaminka following Micalizio *et al.* [91], fault-based techniques [92–95], in which faults are modeled in advance, cannot be used for multiagent diagnosis. The reason is that the interactions in multiagent systems are unpredictable. For this reason, multiagent diagnosis is typically model-based, where each possible fault of the system is not described explicitly. This is especially true in open systems, where the idea of 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 [96].

Thus in recent years, the MBD approach has been applied to MAS diagnosis by several research groups, including Console *et al.* [97] with applications in the automotive industry [98], Roos *et al.* [99, 100] for plan diagnosis, and Kalech and Kaminka [43, 44] for coordination failures in agent teams. These are in general closed systems, where the agents or components included in the system are fixed during execution. For example, in [43], 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 [101]. 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 [43].

MBD has also been used by Ardissono *et al.* to enable Web services with diagnostic capabilities [102]. 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, multiagent interactions are not orchestrated (as in BPEL programs), but emerge as a result of agents' autonomous actions. Diagnosis exception in such an open and flexible setting calls for new approaches, such as the ones we present in this thesis.

Another approach to detecting exceptions is to divide the system into small parts that can be monitored by different agents and then to combine the results to decide if the execution is going as expected. This is generally called multiagent diagnosis [78,79]. While its primary application focuses on global diagnosis which gives a system-wide solution, it can also be applied in a distributed manner. That is, each agent finds a local diagnosis in its territory. Then, the local diagnoses are combined into a global

diagnosis. Simply, a diagnosis is a collection of fault modes for each component of the system (one for each component). However, in most e-commerce scenarios (like the delivery process described in this thesis), an exception is usually related to more than one agent, which requires the joint effort of agents to be resolved. Consider the delivery scenario we have used for argumentation. Only when the customer and the courier combine their knowledge, they can find out that the package is received by the door keeper. Thus, a collaborative diagnosis process is required to capture such exceptions.

One general assumption with divide-and-conquer methods is that a situation is identified as an exception unanimously. That is, each agent has a fixed notion of what an exception is. However, again in e-commerce processes, many times one agent sees a situation as problematic while another agent might be happy about the current status. Hence, we need mechanisms that enable agents to detect exceptions locally rather than globally, such as the detection mechanism we propose in this thesis.

In this thesis, we are interested in monitoring interactions when agents have local views of the environment and their ideas of exceptions vary. Hence, the above approaches for diagnosing exceptions are not applicable for our purposes. Instead, we need to (i) treat multiagent systems as embodying interaction protocols that are only partially shared and (ii) analyze exceptions as cases raising through discrepancies in agent's expectations of the system.

Our multiagent architecture can be considered an open system in the sense that agents can enter or leave the system at any time during execution. However, there are some assumptions (or restrictions) on the agents when they enter the system. The agents should have access to the common commitment theory, and use it when manipulating their commitments. Similarly, when they start to do business with other agents, they are supposed to use the protocol rules described for that business. For now, we do not provide a means to distribute this information to the agents that are just entering the system, and assume that they start their execution with this information at hand. In addition, the protocol rules are fixed at design-time, and

cannot be changed during a protocol's execution. However, an interesting extension to this would be to allow agents to dynamically agree on new protocol rules, and add them into their knowledge bases.

7.1.3. Exceptions

Multiagent systems can be designed to model complicated organizations using agent oriented software development methodologies; such as Tropos [46], or Gaia [103]. The concepts in an organization (i.e., a social community, a business dealing, or a complicated workflow process) can be described in the agent domain as follows; (i) an *actor* has strategic goals and intentions, and possesses a role in the organization (i.e., an agent in a multiagent system), (ii) a *goal* is a strategic interest of an actor that it needs to satisfy, (iii) a *plan* is a path for the actor to satisfy its goal, (iv) a *resource* is an entity, either physical or informational, that an actor needs, and another actor can provide, (v) a *capability* is the ability of an actor to perform certain actions, and (vi) a *belief* represents an actor's knowledge about its social environment.

If we consider an open multiagent system as a social community, exceptions may occur for several different reasons; (i) bad citizen (i.e., member of the society) behavior, (ii) incapability of the citizens for certain tasks, or (iii) coordination problems between the citizens. When citizens are embedded with individual exception handling routines, the burden on the them increase significantly [45]. This distributed approach, called the *survivalist*, requires the agents to include complex and most possibly domain-dependent behaviors. In contrast, the *citizen* approach proposes that the exception handling service is extracted from the problem solving agents (i.e, citizens), and served separately as in real societies (i.e., by the government).

Workflows provide a good basis for agent-based exception handling systems [104]. Higher level exceptions may arise from organizational or coordination problems such as the invalid structure of the workflow or the unavailability of roles at some points during the execution of the workflow. In order for the agents to reason efficiently in exceptional cases, they need significant amount of domain knowledge about the system they are

executing in. In addition, role assignment is an important challenge when considering complicated workflows [105]. A role can be thought of as an abstract representation of a service in an organizational model. In order to enact roles, agents may alter their behavior in the society. Since agents continuously enter and exist the system, roles are also switched between the agents that enact them. In addition, the agents in a system are usually heterogeneous (i.e., possess different skills and capabilities), thus each agent cannot fit into each role successfully. One interesting problem occurs when an agent's goals conflict with the goals of a role it wishes to enact, which often increases the possibility that exceptions occur in such complicated systems.

In systems that provide online social networks, even when a system owner correctly follows the privacy agreement that it announces, the privacy of users can easily be breached through interactions with other users. The above discussion clearly shows that it is not enough to check the privacy agreement of the system. In addition to this, the user's relations with other users and what these relations enable the other users to do should be systematically analyzed to reach a conclusion.

Krishnamurthy and Wills study the leakage of personal identifiable information in social networks [106]. They consider personal identifiable information as any piece of information that can by itself or when combined with other information help decipher a person's identity. They depict different ways that such information can leak to external applications. The types of leakages they are concerned with are generally based on HTTP side effects that allow information to appear in URLs or cookies that can be used by other applications. While those identified leakages are important, the types of leakages we are concerned here are more high-level in the sense that even when such system level details are fixed can still exist.

Fang and LeFevre point out that following a privacy agreement for a novice user is difficult and there is a need for easy-to-use privacy specification tools for such users [107]. To address this, they develop a learning-based privacy wizard that asks for some example cases from a user and learns with whom a user wants to share information and what kind. Based on this learning, the wizard decides on the privacy settings of

the user. This is certainly an important aspect of privacy. In our work, we assume that privacy agreements can be represented and processed formally so that we can focus on the interplay between privacy agreements and user relations.

Akcora, Carminati, and Ferrari measure how risky an individual in a social network is in terms of privacy [108]. They develop a method in which a user's friends' friends are analyzed in terms of their potential for learning and misusing personal information. The approach is based on active learning and hence the risk is decided by the community itself. They have adapted their approach to Facebook in order to detect people that can potentially violate privacy of a user. While their aim is to identify risky individuals, our aim in this thesis is to help a system decide on potential privacy violations and inform the user appropriately.

In this thesis, we have focused on two domains, e-commerce and privacy in online social networks, to investigate the underlying exceptions that might occur. However, exceptions are also possible in other fields or applications. We believe that the automated methods we have proposed in this thesis are applicable to not only those two domains, but also to others where intelligent agents can be deployed with the reasoning capabilities we have presented here.

7.2. Conclusions

Our contributions in this thesis are the following:

- We have extended the scope of exceptions to include cases where commitment violation is not the sole cause. That is, an agent may signal an exception even though neither of its commitments are violated, if it observes a situation different than what it expects. Similarly, the agent may not signal an exception even if one of its commitments is violated, because he foresees the violation beforehand. We have provided a systematic way to capture such exceptions.
- We have applied model checking to predict exceptions based on a set of assumptions on the current environment. This way, an agent can understand whether

performing an action will prevent its commitment from being fulfilled. If so, the agent may choose not to perform that action.

- We have exhaustively investigated the temporal relations among commitments. We have used these relations to build distributed diagnosis procedures that deal with misalignment of commitments and improper commitment delegations. We have showed that these procedures are sound, and implemented them in $\mathcal{REC}$, a tool for monitoring commitments at run-time.

Next, we review the individual contributions for each phase in exception handling.

Detection: We proposed a satisfiability relation that can be used to compare agents' states. When used to compare an agent's state with its projected state, the satisfiability relation tells whether there is an exception for the agent or not. That is, if the agent's projected state is not satisfiable by its current state, then the agent signals an exception. By consistently checking its current state, the agent can identify at which point of the protocol there has been a problem.

Moreover, we extended the concept of exceptions that are described in the literature. Often, an exception is considered identical to a contract violation. While we accommodate that perspective, we also take into account the agent's expectations from their contracts.

Prediction: We proposed a method to allow agents to further detect and predict exceptions even when they do not expect anything wrong by themselves. Our method takes into account other agents' projections for the same situation, based on the assumption that they may have a wider perception of the environment. Predicting an exception beforehand gives the agent control over its actions. If the agent already knows that taking an action will cause an exception, then it will probably choose not to take that action.

Diagnosis & Monitoring: We studied diagnosis of exceptions when the commitments of agents are misaligned with each other. In particular, we focused on the

temporal aspects. That is, we aimed at fixing misalignments that are caused by conflicts in the commitments' deadlines. We proposed commitment similarity relations that can be used to verify if two commitments are aligned in time. In the case of misalignment, the agents can update their commitments based on the alignment policy we proposed.

In addition, we made an extensive analysis of the temporal relations for commitment delegation. While delegation allows flexibility, it may also produce possible mismatches amongst the deadlines of agents' commitments. Such improper delegations may eventually drive the system into a state of violation. 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.

Moreover, we showed an application of assumption-based argumentation, that is used as a principled way to extend agent reasoning about commitment exceptions, and as a basis to run diagnosis dialogues. This reasoning enables agents to resolve conflicting cases based on justification of arguments.

7.3. Future Directions

There are several in which the ideas presented in this thesis can be extended. In particular, we plan to investigate the following topics for future work:

- Throughout the thesis, we have a used a language for commitments that currently supports conjunction of properties only. This commitment language can be extended to include negation, disjunction, and even nested commitments, i.e., other commitments in the properties (antecedent or consequent) of a commitment. This will enrich our commitment language, and allow us to represent more complex scenarios.
- We have not tested the prediction scenarios on a real social network. It will be interesting to see how our model behaves in a real system with several users and relations among them. In addition, the performance of model checking should

be investigated to see how it scales for bigger systems. Our preliminary analysis shows that there is an exponential increase in the size of relations among users related to the number of users in the system. When verifying properties in such big systems, techniques like pruning the model can be useful.

- Another way to extend our prediction module is to associate probabilities with the prediction outcomes. The agents can determine how likely their assumptions are to occur, and accordingly, the model checking process can present several future states, each associated with a probability showing how likely it is to have a commitment violation in that state.
- Recently, teleo-reactive systems have gained some popularity in describing logic based systems [109, 110]. A teleo-reactive system is a variety of production systems with access to only the current state of the environment. It combines goal-driven behavior (i.e., proactive), with reactive behavior, which allows sensitivity to the changing environment. It will be interesting to apply agent reasoning based on teleo-reactive rules. In such an environment, agents can reactively respond to exceptions following basic logic rules. In addition, they can take advantage of situations, where some exceptions are resolved by the environment itself, or by other agents.
- We have experienced extensive use of temporal reasoning in this thesis to capture commitment deadlines. One way to extend this temporal reasoning is to investigate some basic relations (e.g., before, overlapping, contains) on temporal intervals [41, 86]. This type of reasoning is used in the medical domain to diagnose certain types of diseases based on temporal relations among the different phases of the disease. This is also applicable to commitment exceptions, where we can explore certain phases of commitments (e.g., commitment states) to identify which type of exception we are dealing with.

APPENDIX A: Proofs

Proof of Theorem 1

$\Vdash\!\vdash$relation is reflexive, non-symmetric, and transitive.

Proof. It is trivial that $\Vdash\!\vdash$is reflexive and non-symmetric. For atomic propositions, reflexivity follows from Axiom 2 and for commitments it follows from Axioms 6 and 10. For showing non-symmetry, any unidirectional edge in Figure 2.4 serves as a counter-example for symmetry. Consider the edge between nodes 2 and 5. The proposition *shared* satisfies the violated base-level commitment $C^v(\top, \mathit{shared})$, but the other direction is not true. The edges between nodes 6–7 and 1–6 show similar properties.

For transitivity, we need to show that if $\alpha \Vdash\!\vdash \beta$ and $\beta \Vdash\!\vdash \gamma$ then $\alpha \Vdash\!\vdash \gamma$. Assume α is a proposition P, β and γ are both base-level commitments with consequents P. By Axiom 3, $\alpha \Vdash\!\vdash \beta$, independently of the state of β. Now, when (i) the state of β is fulfilled, by Axiom 6, $\beta \Vdash\!\vdash \gamma$, independently of the state of γ. Then, $\alpha \Vdash\!\vdash \gamma$, by Axiom 3. When (ii) the state of β is active, by Axiom 6, $\beta \Vdash\!\vdash \gamma$, if the state of γ is active or violated. Then, again $\alpha \Vdash\!\vdash \gamma$, by Axiom 3. When (iii) the state of β is violated, by Axiom 6, $\beta \Vdash\!\vdash \gamma$, if the state of γ is also violated. Then, again $\alpha \Vdash\!\vdash \gamma$, by Axiom 3. Thus, $\Vdash\!\vdash$is transitive. Other combinations of α, β and γ follow similarly (see Table A.1).

□

Proof of Theorem 2

For an agent A, at time T, given a proposition P, $\mathcal{S}^T(A) \not\Vdash\!\vdash \mathcal{S}^T(A_p)$ iff $\mathit{exception}(A, T, P)$.

Proof. We prove both directions of Theorem 2 by contradiction.

(i) Forward direction (Completeness): If $exception(A, T, P)$, then $\mathcal{S}^T(A) \nVdash \mathcal{S}^T(A_p)$. Let $\exists P$: $exception(A, T, P)$. Then, by Definition 11, either $\mathcal{T}_P(\mathcal{S}^T(A)) = \emptyset$, or $\exists\, x$: $x \in \mathcal{T}_P(\mathcal{S}^T(A))$.

Consider the former case. Assume $\mathcal{S}^T(A) \Vdash \mathcal{S}^T(A_p)$. This will occur only if $\mathcal{T}_P(\mathcal{S}^T(A_p)) = \emptyset$ also. Otherwise, a term that includes P cannot be satisfied according to the satisfiability axioms. This is a contradiction, because, by Definition 11, $\mathcal{T}_P(\mathcal{S}^T(A_p))$ cannot be empty.

Now, consider the latter case. Again, assume $\mathcal{S}^T(A) \Vdash \mathcal{S}^T(A_p)$. There are three possibilities:

(i) $\mathcal{S}^T(A_p) = \langle \emptyset, \emptyset \rangle$. Then, every state will satisfy the empty state since there are no terms in it to satisfy (Definition 12). This is a contradiction, because, by Definition 11, $\mathcal{T}_P(\mathcal{S}^T(A_p))$ cannot be empty.

(ii) $\mathcal{T}_P(\mathcal{S}^T(A_p)) = \emptyset$. Then, there is no need to satisfy a term that includes P. This is a contradiction, because, again by Definition 11, $\mathcal{T}_P(\mathcal{S}^T(A_p))$ cannot be empty.

(iii) $\exists\, y$: $y \in \mathcal{T}_P(\mathcal{S}^T(A_p))$. Then, x should satisfy y (since both are the terms that include P). There are three possibilities:

- $x = P$ or $x = C^f(P)$. Then, according to the satisfiability axioms, $x \Vdash y$ for any y. This is a contradiction, because, by Definition 11, $u(x)$ cannot be greater than or equal to $u(y)$.
- $x = C^a(P)$ and ($y = C^a(P)$ or $y = C^v(P)$). Then, by Axioms 6, 9, and 10, $x \Vdash y$. This is a contradiction, because, by Definition 11, $u(x)$ cannot be greater than or equal to $u(y)$.
- $x = C^v(P)$ and $y = C^v(P)$. Then, by Axioms 6, 9, and 10, $x \Vdash y$. This is a contradiction, because, by Definition 11, $u(x)$ cannot be equal to $u(y)$.

Hence, we conclude that whenever there is an exception, $\mathcal{S}^T(A) \nVdash \mathcal{S}^T(A_p)$.

(ii) Backward direction (Soundness): If $\mathcal{S}^T(A) \nVdash \mathcal{S}^T(A_p)$, then $\exists P$: $exception(A,$

T, P).

Let $\mathcal{S}^T(A) \nVdash \mathcal{S}^T(A_p)$. Then, by Definition 12, $\exists\ y$: $y \in \mathcal{S}^T(A_p)$ and $\mathcal{S}^T(A) \nVdash y$. That is, there is at least one term that is not satisfiable. Let $y \in \mathcal{T}_P(\mathcal{S}^T(A_p))$. That is, P be the proposition related to the term y (either y itself or the consequent of y in case y is a commitment). Now, assume *exception*(A, T, P) does not hold. Then, $\exists\ x$: $x \in \mathcal{S}^T(A)$ and $u(x) \geq u(y)$. Otherwise, *exception*(A, T, P) would hold. There are three possibilities:

(i) $x = P$ or $x = C^f(P)$. Then, $u(x) \geq u(y)$ for any y, and no exception occurs. This is a contradiction, because, according to the satisfiability axioms x would satisfy any such y.

(ii) $x = C^a(P)$ and ($y = C^a(P)$ or $y = C^v(P)$). Then, $u(x) \geq u(y)$, and no exception occurs. This is a contradiction, because, by Axioms 6, 9, and 10, x would satisfy y.

(iii) $x = C^v(P)$ and $y = C^v(P)$. Then, $u(x) = u(y)$, and no exception occurs. This is a contradiction, because, by Axioms 6, 9, and 10, x would satisfy y.

□

Proof of Theorem 3

Given a commitment $C_m \in \mathcal{C}_T$, and an agent $X \in \mathcal{A}$, if $X \rhd^{\emptyset}_{\delta_{out}} C_m, C_i \in \delta_{out}$, then ($C_i$, C_m) $\in \mathcal{M}_T$.

Proof. We prove soundness by contradiction. Given an agent X and two commitments C_i, C_m, assume that $X \rhd_{\delta_{out}} C_m$, $C_i \in \delta_{out}$, and (C_i, C_m) $\notin \mathcal{M}_T$.

We have three possibilities:

(i) By Rule L_1; C_i is an improper delegation of C_m, and M_T finds it since both C_m and C_i are members of $\mathcal{C}_T$ (the set of commitments M_T uses for monitoring). Thus, M_1 holds. We reach a contradiction.

(ii) By Rule S_1; there is no improper delegation of C_m. M_1 does not hold. However, by Rule S_3; $Y \rhd^{\delta_{exc}}_{\delta_j} C_j$ should hold.

(iii) By Rule S_2; there is no delegation of C_m. Again, M_1 does not hold. However, by Rule S_3; $Y \rhd^{\delta_{exc}}_{\delta_{out}} C_m$ should hold.

Note that Rule L_2 does not hold initially since the monitoring result is not empty. The second and third cases propagate monitoring to other agents. At some point, delegations of C_m cease to exist. That is, there are a finite number of delegations for C_m[19] . Let Z be the last agent that delegates C_m. The following cases are to be considered:

(i) Rule S_2 does not apply; Z has delegated C_m.

(ii) Assume Rule S_1 applies, and let the delegatee be W. For W, again the following three cases are considered:

- Rule L_1 does not apply; δ_{imp} is empty.
- Rule S_1 does not apply; δ_{pro} is empty.
- Rule S_2 does not apply; the debtor of the commitment C_m is W itself.
- Rule L_2 applies for W; however this results in an empty set for the monitoring result (δ_{out} cannot be empty). We reach a contradiction.

(iii) Rule L_2 does not apply; δ_{out} cannot be empty.

(iv) Rule L_1 should apply, which means that M_2 holds. We reach a contradiction.

This demonstrates that every possible case of local monitoring identifying an exception leads to a contradiction against the fact that global monitoring cannot find that exception, thus proving soundness. □

Proof of Theorem 4

$\forall$ $(C_i, C_m) \in \mathcal{M}_T$, $\exists$ an agent $X \in \mathcal{A}$ and a derivation $X \rhd^{\emptyset}_{\delta_{out}} C_m$ such that $C_i \in \delta_{out}$.

[19]Since protocol trace time is fixed, infinite delegations cannot occur.

Proof. We prove completeness by contradiction.

Let us consider two commitments C_i, C_m, such that $(C_i, C_m) \in \mathcal{M}_T$ and $\forall X, \delta_{out}$ such that $X \rhd^{\emptyset}_{\delta_{out}} C_m, C_i \notin \delta_{out}$. Now, let it be debtor($C_m$, X). Then, if $(C_i, C_m) \in \mathcal{M}_T$, by Definition 57 $\text{deleg}_{imp}(C_i, C_m)$. There are three cases (Definition 56):

- C_j is an improper consequent delegation of C_m, or
- C_j is an improper causal delegation of C_m, or
- $\exists\ C_k \in \mathcal{C}$ such that deleg(C_k, C_m) and $\text{deleg}_{imp}(C_j, C_k)$.

In the first two cases, $X \bigtriangledown_{\mathcal{RECC}} \mathcal{C} \wedge\ < C_m, \delta_{pro}, \delta_{imp} > \in \mathcal{C}\ \wedge\ C_i \in \delta_{imp}$, then by L_1 $X \rhd^{\emptyset}_{\delta_{out}} C_m, C_i \in \delta_{out}$. Assume $X \rhd^{\emptyset}_{\delta_{out}} C_m$ where $C_i \notin \delta_{out}$. Now, let C_m be a commitment of X, i.e., X is the debtor or the creditor. Then, C_i is also a commitment of X, because improper delegations only occur among commitments of the same agent (Definitions 51 and 56). When X queries the $\mathcal{RECC}$ reasoner, $< C_m, \delta_{pro}, \{C_i, ...\} > \in \mathcal{C}$. Thus, Rule L_1 applies with C_i *in* δ_{out}. We reach a contradiction.

If $\exists\ C_k \in \mathcal{C}$ such that deleg(C_k, C_m) and $\text{deleg}_{imp}(C_j, C_k)$, by Definition 52, either $\overrightarrow{\text{deleg}}(C_j, C_m)$, or $\exists$ a causal delegation chain σ from C_m to C_j, whose elements are in $\mathcal{C}$, whereby again L_1 would apply, or $\exists\ C_k \in \mathcal{C}$ such that $\overrightarrow{\text{deleg}}(C_j, C_k)$ and deleg(C_k, C_m). If $\text{deleg}_{imp}(C_j, C_m)$ again L_1 applies. Otherwise, let Y be the delegatee or debtor of C_k, $Y \neq X$. In the first case (delegatee), S_1 applies, and therefore S_3. In the second case (debtor), S_2 applies, and therefore also S_3. S_3 recursively starts a new derivation, starting from Y. By iterating the same reasoning, we eventually reach the case where $\overrightarrow{\text{deleg}}(C_j, C_m)$ is the only option, because a delegation tree if finite. Thus L_1 applies. We reach a contradiction.

This demonstrates that every possible case of global monitoring identifying an exception leads to a contradiction against the fact that local monitoring cannot find that exception, thus proving completeness. □

Table A.1. Combination of terms for Theorem 1.

α	β	γ	Axioms used for proof
Proposition	Proposition	Proposition	2
Proposition	Proposition	Base-level com.	2 & 3
Proposition	Proposition	Conditional com.	2 & 4
Proposition	Base-level com.	Proposition	3 & 5 & 2
Proposition	Base-level com.	Base-level com.	3 & 6
Proposition	Base-level com.	Conditional com.	3 & 7 & 4
Proposition	Conditional com.	Proposition	4 & 8 & 2
Proposition	Conditional com.	Base-level com.	4 & 9 & 3
Proposition	Conditional com.	Conditional com.	4 & 10
Base-level com.	Proposition	Proposition	5 & 2
Base-level com.	Proposition	Base-level com.	5 & 3 & 6
Base-level com.	Proposition	Conditional com.	5 & 4 & 7
Base-level com.	Base-level com.	Proposition	6 & 5
Base-level com.	Base-level com.	Base-level com.	6
Base-level com.	Base-level com.	Conditional com.	6 & 7
Base-level com.	Conditional com.	Proposition	7 & 8 & 5
Base-level com.	Conditional com.	Base-level com.	7 & 9 & 6
Base-level com.	Conditional com.	Conditional com.	7 & 10
Conditional com.	Proposition	Proposition	8 & 2
Conditional com.	Proposition	Base-level com.	8 & 3 & 9
Conditional com.	Proposition	Conditional com.	8 & 4 & 10
Conditional com.	Base-level com.	Proposition	9 & 5 & 8
Conditional com.	Base-level com.	Base-level com.	9 & 6
Conditional com.	Base-level com.	Conditional com.	9 & 7 & 10
Conditional com.	Conditional com.	Proposition	10 & 8
Conditional com.	Conditional com.	Base-level com.	10 & 9
Conditional com.	Conditional com.	Conditional com.	10

APPENDIX B: $\mathcal{REC}$ Implementation for Detection

B.1. Commitment Theory

The following code shows the $\mathcal{REC}$ rules for modeling the states of commitments. The commitment formalization is compatible with existing work [7, 28], in which rules on manipulating base-level commitments have already been given. We extend this formalization with conditional commitments. That is, we provide additional rules for manipulating a commitment when the antecedent is also considered.

When there is an event E that creates a conditional commitment C at time T, then C will be in the conditional state from T onwards (C_1)[20] . In $\mathcal{REC}$, 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 also a temporal fluent. A *ccreate* clause creates a commitment in conditional state (C_1). Note that the event E will be associated with a domain action in the domain model (e.g., an offer from the merchant to the customer). Similarly, a base-level commitment will be in the active state if there is an event that creates it (C_2). A commitment can also make a transition from the conditional state to the active state via the detach operation. That is, if there is an event that initiates the antecedent of the commitment, then the commitment will be active (C_3).

Notice the temporal interval $e(T1, T2)$ in the *detach* clause that is associated with the commitment's antecedent Q. Even though we do not explicitly talk about the temporal constraints here, they are required for $\mathcal{REC}$ to process the state changes. An existential temporal constraint for the antecedent Q of the commitment means that if Q is initiated between $T1$ and $T2$, then the commitment's condition is satisfied, and thus the commitment will be active. This is indeed what the *detach* clause in rule C_3 tells.

[20]Note that lines starting with % are comments.

```
% C1: create as conditional
initiates(E, status(C, conditional), T):-
        ccreate(E, C, T).

% C2: create as active
initiates(E, status(C, active), T):-
        create(E, C, T).

% C3: conditional to active
terminates(E, status(C, conditional), T):-
        detach(E, C, T).

initiates(E, status(C, active), T):-
        detach(E, C, T).

detach(E, c(X, Y, property(e(T1, T2), Q), P), T):-
        conditional(c(X, Y, property(e(T1, T2), Q), P), T),
        T >= T1, T =< T2, initiates(E, Q, T).

% C4: conditional or active to fulfilled
terminates(E, status(C, conditional), T):-
        discharge(E, C, T).

terminates(E, status(C, active), T):-
        discharge(E, C, T).

initiates(E, status(C, fulfilled), T):-
        discharge(E, C, T).

discharge(E, c(X, Y, Q, property(e(T1, T2), P)), T):-
        conditional(c(X, Y, Q, property(e(T1, T2), P)), T),
        T >= T1, T =< T2, initiates(E, P, T).

discharge(E, c(X, Y, Q, property(e(T1, T2), P)), T):-
        active(c(X, Y, Q, property(e(T1, T2), P)), T),
        T >= T1, T =< T2, initiates(E, P, T).

% C5: active to violated
terminates(E, status(C, active), T):-
        violate(E, C, T).

initiates(E, status(C, violated), T):-
        violate(E, C, T).

violate(_, c(X, Y, Q, property(e(T1, T2), P)), T):-
        active(c(X, Y, Q, property(e(T1, T2), P)), T),  T > T2.
```

From the active state, the commitment can either be fulfilled via the discharge operation if its consequent has been initiated within the deadline (C_4), or be violated otherwise (C_5). Note that a conditional commitment can also be fulfilled directly without having its antecedent being satisfied.

B.2. Protocol Description

The following code shows the $\mathcal{REC}$ rules for modeling the privacy domain from the point of view of Charlie. Charlie has a single action to perform, which is the movement event for changing location (D_1). Moreover, Charlie knows about the sharing action, which will allow his location to be shared among his friends (D_2). These two rules describe how the truth values of the fluents change in time according to events.

Apart from the fluents, commitments are also manipulated via the *ccreate* and *create* rules. For example, an agreement from the OSN operator to Charlie creates a conditional commitment (D_3). Notice the commitment has a temporal constraint for its consequent. For example, when the offer is made by the OSN operator at time 4, then the temporal constraint for sharing will be from 11 to 16.

In addition to the rules, the customer has a domain ontology where certain facts about the domain are stored (D_4). For example, Charlie knows that *osn* is the OSN operator and $office$ is a location.

```
% D1: move event
initiates(exec(move(User, Location)), newlocation, _):-
        isUser(User),
        isLocation(Location).

% D2: share event
initiates(exec(share(Osn)), shared, _):-
        isOsn(Osn).

% D3: agreement event
ccreate(exec(agreement(Osn, User)),
        c(Osn, User,
                property(e(Tsq, Teq), newlocation),
                property(e(Tsp, Tep), shared)),
        T):-
        isOsn(Osn),
        isUser(User),
        Tsq is T + 1,
        Teq is Tsq + 5,
        Tsp is Teq + 1,
        Tep is Tsp + 5.

% D5: domain ontology
isUser(charlie).
isUser(sally).
isUser(linus).
isOsn(osn).
isLocation(home).
isLocation(office).
```

B.3. Satisfiability

The following code segments show the $\mathcal{REC}$ rules for describing satisfiability. The rules follow the Axioms 2 - 10. Note that *exp* stands for expected, such that an expected fluent is satisfiable once the premises of the rule hold.

```
% Axiom 2
satisfiable(exp(P, T)):-
        holds_at(P, T).

% Axiom 3
satisfiable(exp(active(c(_, _, true, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(fulfilled(c(_, _, true, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), P))), T)):-
        holds_at(P, T).

% Axiom 4
satisfiable(exp(fulfilled(c(_, _, _, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(active(c(_, _, _, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(conditional(c(_, _, _, property(e(_, _), P))), T)):-
        holds_at(P, T).

satisfiable(exp(violated(c(_, _, property(e(_, _), Q), _)), T)):-
        holds_at(Q, T).

% Axioms 5 & 8
satisfiable(exp(P, T)):-
        fulfilled(c(_, _, _, property(e(_, _), P)), T).
```

```
% Axioms 6 & 9 (a)
satisfiable(exp(fulfilled(c(_, _, true, property(e(_, _), P))), T):-
        fulfilled(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(active(c(_, _, true, property(e(_, _), P))), T):-
        active(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(active(c(_, _, true, property(e(_, _), P))), T):-
        fulfilled(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), P))), T):-
        violated(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), P))), T):-
        active(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), P))), T):-
        fulfilled(c(_, _, _, property(e(_, _), P)), T).

satisfiable(exp(fulfilled(c(_, _, true, property(e(_, _), Q))), T):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(fulfilled(c(_, _, true, property(e(_, _), Q))), T):-
        violated(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(active(c(_, _, true, property(e(_, _), Q))), T):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(active(c(_, _, true, property(e(_, _), Q))), T):-
        violated(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), Q))), T):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(violated(c(_, _, true, property(e(_, _), Q))), T):-
        violated(c(_, _, property(e(_, _), Q), _), T).
```

```
% Axiom 7
satisfiable(exp(fulfilled(c(_, _, _, property(e(_, _), P))), T)):-
        fulfilled(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(active(c(_, _, _, property(e(_, _), P))), T)):-
        fulfilled(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), P))), T)):-
        fulfilled(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(conditional(c(_, _, _, property(e(_, _), P))), T)):-
        fulfilled(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(active(c(_, _, _, property(e(_, _), P))), T)):-
        active(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), P))), T)):-
        active(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(conditional(c(_, _, _, property(e(_, _), P))), T)):-
        active(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), P))), T)):-
        violated(c(_, _, true, property(e(_, _), P)), T).

satisfiable(exp(violated(c(_, _, property(e(_, _), Q), _)), T)):-
        fulfilled(c(_, _, true, property(e(_, _), Q)), T).

% Axiom 8
satisfiable(exp(Q, T)):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(Q, T)):-
        violated(c(_, _, property(e(_, _), Q), _), T).
```

```
% Axiom 10 (a)
satisfiable(exp(
        fulfilled(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        fulfilled(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        active(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        fulfilled(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        violated(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        fulfilled(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        conditional(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        fulfilled(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(
        exp(active(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        active(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        violated(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        active(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        conditional(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        active(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        violated(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        violated(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).

satisfiable(exp(
        conditional(c(_, _, property(e(_, _), Q), property(e(_, _), P))), T)):-
        conditional(c(_, _, property(e(_, _), Q), property(e(_, _), P)), T).
```

```
% Axiom 10 (b)
satisfiable(exp(fulfilled(c(_, _, _, property(e(_, _), Q))), T)):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(fulfilled(c(_, _, _, property(e(_, _), Q))), T)):-
        violated(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(active(c(_, _, _, property(e(_, _), Q))), T)):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(active(c(_, _, _, property(e(_, _), Q))), T)):-
        violated(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), Q))), T)):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(violated(c(_, _, _, property(e(_, _), Q))), T)):-
        violated(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(conditional(c(_, _, _, property(e(_, _), Q))), T)):-
        active(c(_, _, property(e(_, _), Q), _), T).

satisfiable(exp(conditional(c(_, _, _, property(e(_, _), Q))), T)):-
        violated(c(_, _, property(e(_, _), Q), _), T).
```

B.4. Recoverability

The following code shows the $\mathcal{REC}$ rules for describing recoverability. The rules follow Definitions 13 & 14.

```
% k-recoverable
oneRecoverable(exp(P, T)):-
        active(c(_, _, _, property(e(_, _), P)), T).

oneRecoverable(exp(Q, T)):-
        conditional(c(_, _, property(e(_, _), Q), _), T).

twoRecoverable(exp(P, T)):-
        conditional(c(_, _, _, property(e(_, _), P)), T).

kRecoverable(exp(P, T)):-
        oneRecoverable(exp(P, T)).

kRecoverable(exp(P, T)):-
        twoRecoverable(exp(P, T)).

% non-recoverable
nRecoverable(exp(P, T)):-
        violated(c(_, _, _, property(e(_, _), P)), T).
```

APPENDIX C: $\mathcal{REC}$ Output for Diagnosis

We show the outputs of $\mathcal{REC}$ for the two cases described in Chapter 4; misalignment and misbehavior.

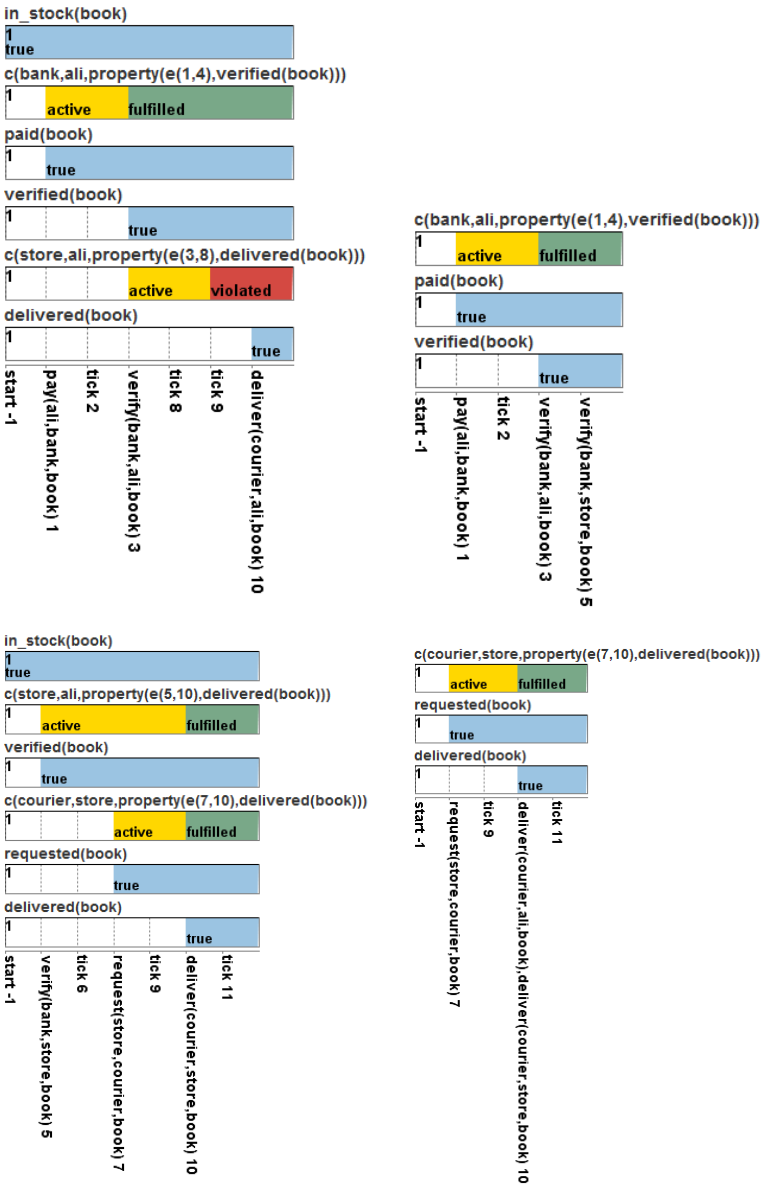


Figure C.1. *Misalignment*: Top (Customer, Bank) - Bottom (Store, Courier).

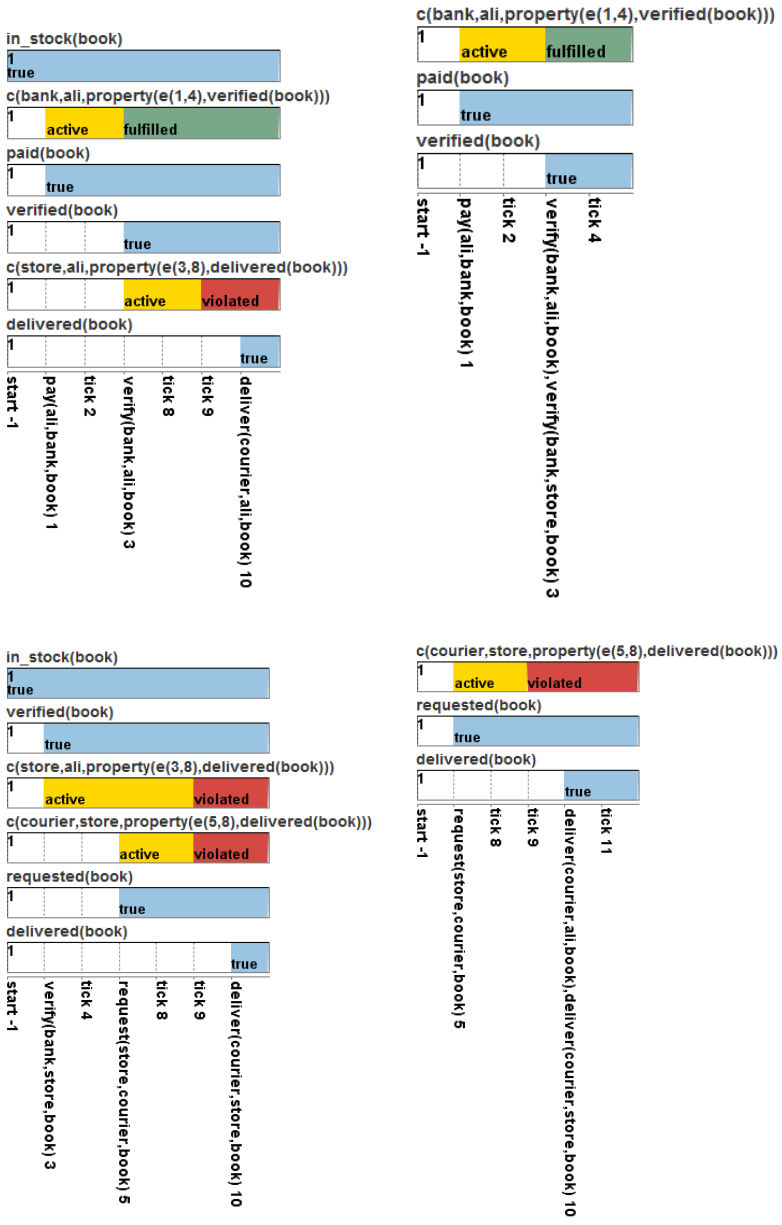


Figure C.2. *Misbehavior*: Top (Customer, Bank) - Bottom (Store, Courier).

APPENDIX D: $\mathcal{REC}$ Implementation for Delegation Monitoring

We implemented a proof-of-concept monitoring framework prototype using ComMon [49]. The input to the ComMon $\mathcal{REC}$-based reasoner is the following:

- a *commitments model* 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 demonstrates the agents' fluents through time. This is used to monitor the individual states of the commitments at run-time. Besides, we defined a subset of the commitment relations introduced in this thesis in the $\mathcal{REC}$ language, thus extending the commitment model with a similarity model and an exception model, in order to accommodate local reasoning.

A running prototype can be downloaded from `http://mas.cmpe.boun.edu.tr/ozgur/code.html`, Section 3. Below we explain some important code segments from the case study presented here, using ComMon Tool. The ComMon tool only needs Java. The simplest way to run the example is to execute `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 $\mathcal{REC}$ code. The following code 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 usual four states of commitments, 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 $\mathcal{REC}$, 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.

```
% 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, rel(T1, T2)),
        c(Tc, X, Y, true, P, abs(T3, T4)), T):-
        conditional(c(Tc, X, Y, Q, P, rel(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, abs(T1, T2)), T):-
        active(c(Tc, X, Y, true, P, abs(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).
violate(_, c(Tc, X, Y, true, P, abs(T1, T2)), T):-
        active(c(Tc, X, Y, true, P, abs(T1, T2)), T), T > T2.
```

The following code shows the rules that describe the domain of the bank for *Request Credit Card*. The domain for the bank covers most of the process, and the domains 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 client to the bank 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 the bank to client creates a conditional commitment between the two agents with a relative deadline of 4 to 7 time units. This means that the delivery should occur some time between 4 to 7 time units after the payment is done. Note that this rule is also contained in the client's domain model. However, not all such rules are in the client's domain model, e.g., the details of the transaction between the bank and the office is omitted from the client.

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

% fluent manipulation
initiates(exec(pay(Client, Bank, Card)), paid(Card), _):-
        isClient(Client), isBank(Bank), isCard(Card).

initiates(exec(confirm(Bank, Office, Card)), confirmed(Card), _):-
        isBank(Bank), isOffice(Office), isCard(Card).

initiates(exec(print(Office, Courier, Card)), printed(Card), _):-
        isOffice(Office), isCourier(Courier), isCard(Card).

initiates(exec(deliver(Courier, Client, Card)), delivered(Card), _):-
        isCourier(Courier), isClient(Client), isCard(Card).

% contract execution
ccreate(exec(offer(Bank, Client, Card)),
        c(T, Bank, Client, paid(Card), delivered(Card), rel(4, 7)), T):-
        isBank(Bank), isClient(Client), isCard(Card).

create(exec(confirm(Bank, Office, Card)),
        c(T, Office, Bank, true, printed(Card), abs(T1, T2)), T):-
        isBank(Bank), isOffice(Office), isCard(Card),
        T1 is T + 2, T2 is T + 3.

ccreate(exec(offer(Courier, Bank, Card)),
        c(T, Courier, Bank, printed(Card), delivered(Card), rel(2, 3)), T):-
        isBank(Bank), isOffice(Office),
        isCourier(Courier), isCard(Card).

% domain ontology
isClient(ali).
isBank(bank).
isOffice(office).
isCourier(courier).
isCard(card).
```

The following code shows the rules that describe the domain of the builder for *Refurbish House*. Here, we have conjunction of fluents for the consequents of commitments. Note that we only show the part of code related to handling conjunction as

others parts are similar to the bank's domain model. 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.

```
% contract execution
ccreate(exec(offer(builder, contractor)),
        c(T, builder, contractor, paidK, [appliances, painted], rel(5, 10)), T).

ccreate(exec(offer(builder, contractor)),
        c(T, builder, contractor, paidB, [shower, tiles], rel(5, 10)), T).
```

The following code shows the rules that describe explicit delegation. 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.

```
% 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).
```

The description for improper (causal) delegation is given in the following code by taking into consideration the deadline intervals of the commitments.

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

APPENDIX E: $\mathcal{REC}$ Output for Delegation Monitoring

Figures E.1 - E.5 display the $\mathcal{REC}$ outputs for the case studies presented in Chapter 5.

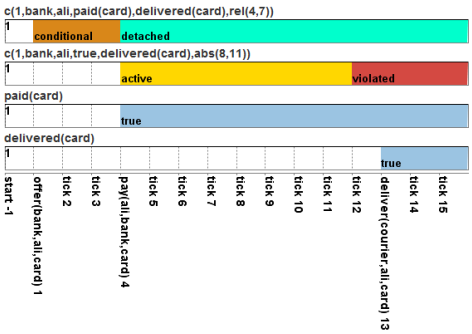


Figure E.1. Client in *Request Credit Card* (exception).

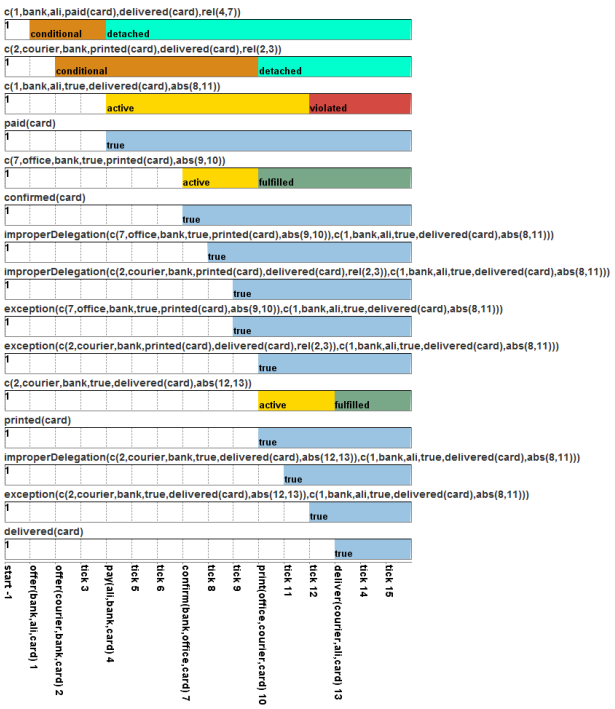


Figure E.2. Bank in *Request Credit Card* (exception).

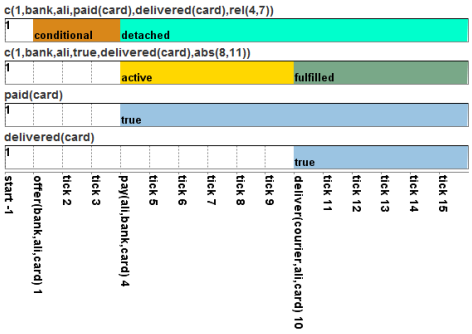


Figure E.3. Client in *Request Credit Card* (no exception).

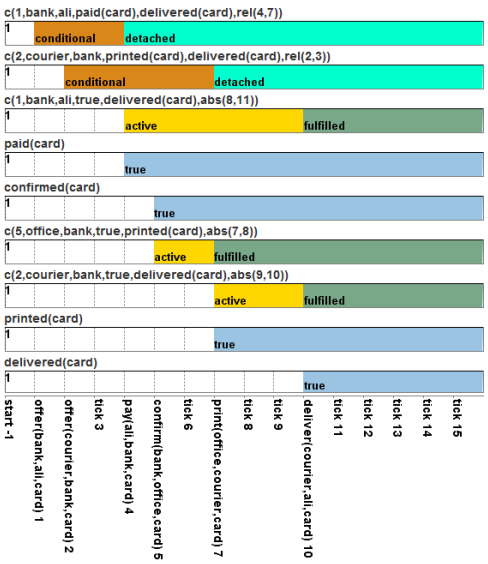


Figure E.4. Bank in *Request Credit Card* (no exception).

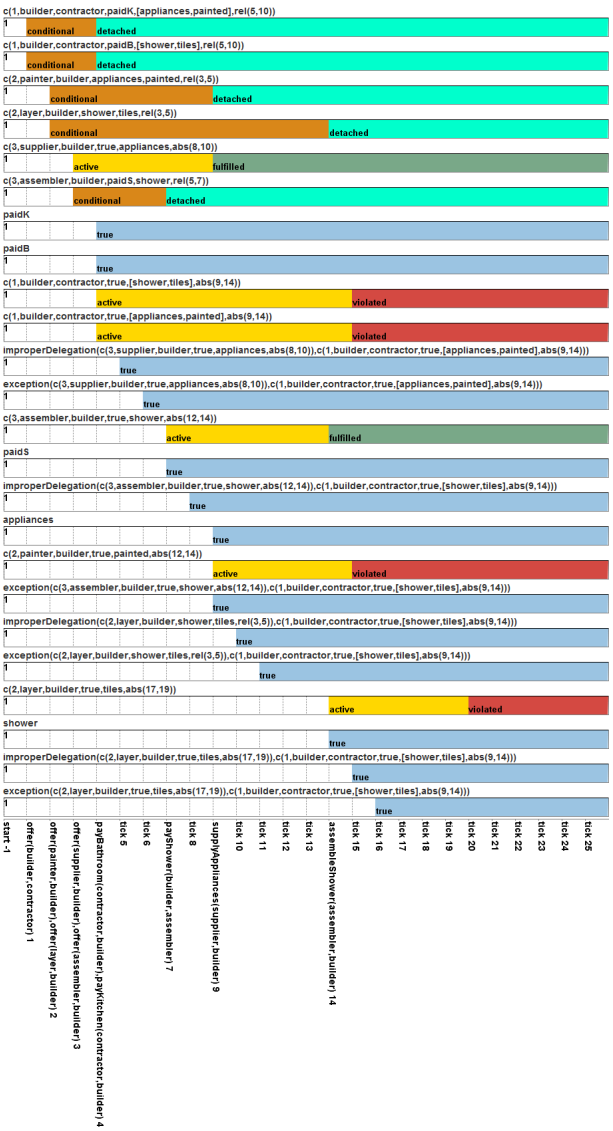


Figure E.5. Builder in *Refurbish House*.

REFERENCES

1. Fisher, M. and M. Wooldridge, "On the Formal Specification and Verification of Multi-Agent Systems", *International Journal of Cooperative Information Systems*, pp. 37–66, 1997.

2. Huhns, M. N. and M. P. Singh, "Agents and Multiagent Systems: Themes, Approaches, and Challenges", M. N. Huhns and M. P. Singh (Editors), *Readings in Agents*, pp. 1–23, Morgan Kaufmann, San Francisco, 1998.

3. Nwana, H. S., J. Rosenschein, T. Sandholm, C. Sierra, P. Maes and R. Guttmann, "Agent- Mediated Electronic Commerce: Issues, Challenges and Some Viewpoints", *Proceedings of the Second International Conference on Autonomous Agents*, AGENTS '98, pp. 189–196, 1998.

4. Singh, M. P., "An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts", *Artificial Intelligence and Law*, Vol. 7, pp. 97–113, 1999.

5. Yolum, P. and M. P. Singh, "Flexible Protocol Specification and Execution: Applying Event Calculus Planning Using Commitments", *Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 527–534, 2002.

6. Fornara, N. and M. Colombetti, "Defining Interaction Protocols Using a Commitment-Based Agent Communication Language", *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 520–527, 2003.

7. Torroni, P., F. Chesani, P. Mello and M. Montali, "Social Commitments in Time: Satisfied or Compensated", *Declarative Agent Languages and Technologies*, Vol. 5948 of *Lecture Notes in Computer Science*, pp. 228–243, Springer, 2009.

8. Kafalı, Ö. and P. Yolum, "A Distributed Treatment of Exceptions in Multiagent Contracts (Preliminary Report)", *Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies (DALT)*, 2011.

9. Kafalı, Ö. and P. Yolum, "Detecting Exceptions in Commitment Protocols: Discovering Hidden States", *Languages, Methodologies and Development Tools for Multi-Agent Systems*, Vol. 6039 of *LNCS*, pp. 112–127, 2010.

10. Kafalı, Ö., A. Günay and P. Yolum, "$\mathcal{PROTOSS}$: A Run Time Tool for Detecting $\mathcal{PR}$ivacy vi$\mathcal{O}$la$\mathcal{T}$ions in $\mathcal{O}$nline $\mathcal{S}$ocial network$\mathcal{S}$ (Short Paper)", *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2012.

11. Kafalı, Ö. and P. Torroni, "Exception Diagnosis in Multiagent Contract Executions", *Annals of Mathematics and Artificial Intelligence*, Vol. 64, No. 1, pp. 73–107, 2012.

12. Kafalı, Ö., F. Chesani and P. Torroni, "What Happened to My Commitment? Exception Diagnosis Among Misalignment and Misbehavior", *Computational Logic in Multi-Agent Systems (CLIMA XI)*, Vol. 6245 of *LNCS*, pp. 82–98, 2010.

13. Kafalı, Ö. and P. Torroni, "Diagnosing Commitments: Delegation Revisited (Extended Abstract)", *AAMAS 2011: 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1175–1176, 2011.

14. Kafalı, Ö. and P. Torroni, "Social Commitment Delegation and Monitoring", *Computational Logic in Multi-Agent Systems (CLIMA XII)*, Vol. 6814 of *LNCS*, pp. 171–189, 2011.

15. Kafalı, Ö., F. Toni and P. Torroni, "Collaborative Diagnosis of Exceptions to Contracts (Extended Abstract)", *AAMAS 2011: 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1167–1168, 2011.

16. Kafalı, Ö., F. Toni and P. Torroni, "Reasoning About Exceptions to Contracts", *Computational Logic in Multi-Agent Systems (CLIMA XII)*, Vol. 6814 of *LNCS*, pp. 225–242, 2011.

17. Torroni, P., P. Yolum, M. P. Singh, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma and P. Mello, "Modelling Interactions via Commitments and Expectations", *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pp. 263–284, 2009.

18. Bentahar, J., M. El-Menshawy and R. Dssouli, "An Integrated Semantics of Social Commitments and Associated Operations", *Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops, Turin, Italy, September 7-10, 2009*, Vol. 494 of *CEUR Workshop Proceedings*, 2009.

19. Jakob, M., M. Pěchouček, S. Miles and M. Luck, "Case studies for Contract-Based Systems", *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS): Industrial Track*, pp. 55–62, 2008.

20. Artikis, A., "Dynamic Protocols for Open Agent Systems", *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 97–104, 2009.

21. Desai, N., A. U. Mallya, A. K. Chopra and M. P. Singh, "Processes = Protocols + Policies: A Methodology for Business Process Development", *Technical report, NC State University, TR2004-34*, 2004.

22. Chopra, A. K. and M. P. Singh, "Producing Compliant Interactions: Conformance, Coverage, and Interoperability", *Proceedings of the 4th International Workshop on Declarative Agent Languages and Technologies (DALT)*, pp. 1–15, 2006.

23. Chopra, A. K. and M. P. Singh, "Multiagent Commitment Alignment", *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent*

Systems (AAMAS), pp. 937–944, 2009.

24. Singh, M. P., "Semantical Considerations on Dialectical and Practical Commitments", *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI)*, pp. 176–181, 2008.

25. Chopra, A. K., F. Dalpiaz, P. Giorgini and J. Mylopoulos, "Reasoning about Agents and Protocols via Goals and Commitments", *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 457–464, 2010.

26. Yolum, P. and M. P. Singh, "Enacting Protocols by Commitment Concession", *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 116–123, 2007.

27. van Riemsdijk, M. B., M. Dastani and M. Winikoff, "Goals in Agent Systems: A Unifying framework", *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 713–720, 2008.

28. Chesani, F., P. Mello, M. Montali and P. Torroni, "Commitment Tracking via the Reactive Event Calculus", *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, pp. 91–96, 2009.

29. Chesani, F., P. Mello, M. Montali and P. Torroni, "Monitoring Time-Aware Social Commitments with Reactive Event Calculus", *20th European Meeting on Cybernetics and Systems Research, 7th International Symposium "From Agent Theory to Agent Implementation" (AT2AI-7)*, pp. 447–452, 2010.

30. Clarke, E. M. and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic", *Logic of Programs, Workshop*, pp. 52–71, Springer-Verlag, London, UK, 1982.

31. Huth, M. and M. Ryan, *Logic in Computer Science: Modelling and Reasoning*

about Systems, Cambridge University Press, New York, NY, USA, 2004.

32. Emerson, E. A., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, MIT Press, Cambridge, MA, USA, 1990.

33. Cimatti, A., E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking", *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, Vol. 2404 of *LNCS*, Springer, Copenhagen, Denmark, July 2002.

34. Carminati, B. and E. Ferrari, "Privacy-Aware Access Control in Social Networks: Issues and Solutions", *Privacy and Anonymity in Information Management Systems*, chap. 9, pp. 181–195, Springer Verlag, 2010.

35. Telang, P. and M. Singh, "Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach", *IEEE Transactions on Services Computing*, Vol. 99, No. PrePrints, 2011.

36. Kollingbaum, M. and T. Norman, "A Contract Management Framework for Supervised Interaction", *UK Multi-Agent Systems (UKMAS) Annual Conference, Liverpool, UK*, 2002.

37. Jennings, N. R., P. Faratin, T. J. Norman, P. O'Brien and B. Odgers, "Autonomous Agents for Business Process Management", *International Journal of Applied Artificial Intelligence*, Vol. 14, No. 2, pp. 145–189, 2000.

38. Schroeder, M. and R. Schweimeier, "Arguments and Misunderstandings: Fuzzy Unification for Negotiating Agents", *Electronic Notes in Theoretical Computer Science*, Vol. 70, No. 5, pp. 1 – 19, 2002.

39. Chittaro, L. and A. Montanari, "Temporal Representation and Reasoning in Artificial Intelligence: Issues and approaches", *Annals of Mathematics and Artificial*

Intelligence, Vol. 28, No. 1-4, pp. 47–106, 2000.

40. Chesani, F., P. Mello, M. Montali and P. Torroni, "Role Monitoring in Open Agent Societies", *Agent and Multi-Agent Systems: Technologies and Applications, 4th KES International Symposium*, pp. 112–121, 2010.

41. Allen, J. F., "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, Vol. 26, pp. 832–843, 1983.

42. Chopra, A. K. and M. P. Singh, "Constitutive Interoperability", *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 797–804, 2008.

43. Kalech, M. and G. A. Kaminka, "On the Design of Social Diagnosis Algorithms for Multi-Agent Teams", *IJCAI 2003: 18th International Joint Conference on Artificial Intelligence*, pp. 370–375, 2003.

44. Kalech, M. and G. A. Kaminka, "Towards Model-Based Diagnosis of Coordination Failures", *AAAI 2005: 20th National Conference on Artificial intelligence*, pp. 102–107, 2005.

45. Klein, M., J. Rodriguez-Aguilar and C. Dellarocas, "Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-agent Systems: the Case of Agent Death", *Journal of Autonomous Agents and Multi-Agent Systems*, pp. 179–189, 2003.

46. Bresciani, P., A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology", *Autonomous Agents and Multi-Agent Systems*, Vol. 8, pp. 203–236, 2004.

47. Friedrich, G., "Repair of Service-Based Processes - An Application Area for Logic Programming", The ALP Newsletter, 2010.

48. Chesani, F., P. Mello, M. Montali and P. Torroni, "A Logic-Based, Reactive Calculus of Events", *Fundamenta Informaticae*, Vol. 105, No. 1-2, pp. 135–161, 2010.

49. Chesani, F., M. Montali, P. Mello and P. Torroni, "Monitoring Time-Aware Social Commitments with Reactive Event Calculus", *Proceedings of the 7th International Symposium ''From Agent Theory to Agent Implementation'' (AT2AI-7)*, 2010.

50. Torroni, P., F. Chesani, P. Mello and M. Montali, "A Retrospective on the Reactive Event Calculus and Commitment Modeling Language", C. Sakama, S. Sardina, W. Vasconcelos and M. Winikoff (Editors), *Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies (DALT 2011)*, Vol. 7169 of *Lecture Notes in Computer Science*, pp. 120–127, Springer, 2012.

51. Bondarenko, A., P. Dung, R. Kowalski and F. Toni, "An Abstract, Argumentation-Theoretic Approach to Default Reasoning", *Artificial Intelligence*, Vol. 93, No. 1-2, pp. 63–101, 1997.

52. Dung, P., R. Kowalski and F. Toni, "Dialectic Proof Procedures for Assumption-Based, Admissible Argumentation", *Artificial Intelligence*, Vol. 170, No. 2, pp. 114–159, 2006.

53. Dung, P., P. Mancarella and F. Toni, "Computing Ideal Sceptical Argumentation", *Artificial Intelligence*, Vol. 171, No. 10–15, pp. 642–674, 2007.

54. Dung, P., R. Kowalski and F. Toni, "Assumption-Based Argumentation", I. Rahwan and G. Simari (Editors), *Argumentation in AI*, pp. 199–218, Springer, 2009.

55. Kakas, A. C., P. Mancarella, F. Sadri, K. Stathis and F. Toni, "Computational Logic Foundations of KGP Agents", *Journal of Artificial Intelligence Research*, Vol. 33, pp. 285–348, 2008.

56. Gaertner, D. and F. Toni, "Computing Arguments and Attacks in Assumption-

Based Argumentation", *IEEE Intelligent Systems*, Vol. 22, No. 6, pp. 24–33, 2007.

57. Rahwan, I., S. D. Ramchurn, N. R. Jennings, P. Mcburney, S. Parsons and L. Sonenberg, "Argumentation-Based Negotiation", *Knowledge Engineering Review*, Vol. 18, pp. 343–375, 2003.

58. Walton, D. N. and E. C. W. Krabbe, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, State University of New York Press, Albany, NY, USA.

59. Klein, M. and C. Dellarocas, "A Systematic Repository of Knowledge About Handling Exceptions in Business Processes", *ASES Working Report. MIT*, 2000.

60. Klein, M. and C. Dellarocas, "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems", *Computer Supported Cooperative Work*, Vol. 9, No. 3/4, pp. 399–412, 2000.

61. Letia, I. A. and A. Groza, "Agreeing on Defeasible Commitments", *Proceedings of the 4th International Workshop on Declarative Agent Languages and Technologies (DALT)*, pp. 156–173, 2006.

62. Fornara, N. and M. Colombetti, "Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology", *Proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT)*, pp. 101–118, 2009.

63. McBurney, P. and S. Parsons, "Dialogue Games for Agent Argumentation", *Argumentation in Artificial Intelligence*, pp. 261–280, Springer, 2009.

64. McBurney, P. and S. Parsons, "Locutions for Argumentation in Agent Interaction Protocols", *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1240–1241, 2004.

65. Singh, M. P. and A. K. Chopra, "Correctness Properties for Multiagent Systems",

Prooceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT), pp. 192–207, 2009.

66. Giordano, L. and A. Martelli, "Computational Logic in Multi-Agent Systems", chap. Verifying Agents' Conformance with Multiparty Protocols, pp. 17–36, Springer-Verlag, Berlin, Heidelberg, 2009.

67. Mallya, A. U. and M. P. Singh, "An Algebra for Commitment Protocols", *Autonomous Agents and Multi-Agent Systems*, Vol. 14, No. 2, pp. 143–163, 2007.

68. Alberti, M., F. Chesani, M. Gavanelli, E. Lamma, P. Mello and P. Torroni, "Verifiable Agent Interaction in Abductive Logic Programming: The SCIFF Framework", *ACM Transactions on Computational Logic*, Vol. 9, No. 4, pp. 1–43, 2008.

69. Cranefield, S. and M. Winikoff, "Verifying Social Expectations by Model Checking Truncated Paths", *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pp. 204–219, 2008.

70. Kowalski, R. and M. Sergot, "A Logic-Based Calculus of Events", *New Generation Computing*, Vol. 4, No. 1, pp. 67–95, 1986.

71. El Menshawy, M., J. Benthar, H. Qu and R. Dssouli, "On the Verification of Social Commitments and Time", *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 483–490, 2011.

72. Lomuscio, A., H. Qu and F. Raimondi, "MCMAS: A Model Checker for the Verification of Multi-Agent Systems", *Computer Aided Verification*, Vol. 5643 of *Lecture Notes in Computer Science*, pp. 682–688, 2009.

73. Hoek, W. V. D., J. A. Rodríguez-aguilar, C. Sierra and M. Wooldridge, "On the Logic of Normative Systems", *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI07*, pp. 1175–1180, 2007.

74. Boella, G. and L. van der Torre, "Regulative and Constitutive Norms in Normative Multiagent Systems", *Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 255–265, AAAI Press, 2004.

75. Fornara, N. and M. Colombetti, "Specifying and Enforcing Norms in Artificial Institutions", *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '08, pp. 1481–1484, 2008.

76. Poole, D., "Normality and Faults in Logic-Based Diagnosis", *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI) - Volume 2*, pp. 1304–1310, 1989.

77. Console, L. and O. Dressler, "Model-based Diagnosis in the Real World: Lessons Learned and Challenges Remaining", *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1393–1400, 1999.

78. Roos, N., A. ten Teije and C. Witteveen, "A Protocol for Multi-agent Diagnosis with Spatially Distributed Knowledge", *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 655–661, 2003.

79. Jonge, F. D., N. Roos and C. Witteveen, "Diagnosis of Multi-Agent Plan Execution", *Multiagent System Technologies: MATES 2006, LNCS 4196*, pp. 86–97, Springer, 2006.

80. Kaminka, G. A., "Detecting Disagreements in Large-Scale Multi-Agent Teams", *Autonomous Agents and Multi-Agent Systems*, Vol. 18, No. 3, pp. 501–525, 2009.

81. Kalech, M. and G. A. Kaminka, "Diagnosing a Team of Agents: Scaling-up", *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 249–255, 2005.

82. Subbiah, A. and D. M. Blough, "Distributed Diagnosis in Dynamic Fault Environments", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, pp. 453–467, 2004.

83. Poutakidis, D., L. Padgham and M. Winikoff, "Debugging Multi-Agent Systems Using Design Artifacts: The Case of Interaction Protocols", *Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 960–967, 2002.

84. Barringer, H., A. Goldberg, K. Havelund and K. Sen, "Program Monitoring with LTL in EAGLE", *International Parallel and Distributed Processing Symposium*, Vol. 17, p. 264, 2004.

85. Ledley, R. and L. Lusted, "Reasoning Foundations of Medical Diagnosis", *Science*, Vol. 130, No. 3366, pp. 9–21, 1959.

86. Gamper, J. and W. Nejdl, "Abstract Temporal Diagnosis in Medical Domains", *Artificial Intelligence in Medicine*, Vol. 10, pp. 209–234, 1997.

87. Buchanan, B. G. and E. H. Shortliffe, *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley Series in Artificial Intelligence)*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

88. Console, L. and P. Torasso, "On the Co-operation between Abductive and Temporal Reasoning in Medical Diagnosis", *Artificial Intelligence in Medicine*, Vol. 3, No. 6, pp. 291–311, Dec. 1991.

89. Ironi, L., M. Stefanelli and G. Lanzola, "Qualitative Models in Medical Diagnosis", *Artificial Intelligence in Medicine*, Vol. 2, No. 2, pp. 85 – 101, 1990.

90. Lucas, P. J. F., "Analysis of Notions of Diagnosis", *Artificial Intelligence*, Vol. 105, No. 1-2, pp. 295–343, 1998.

91. Micalizio, R., P. Torasso and G. Torta, "On-line Monitoring and Diagnosis of a Team of Service Robots: A Model-Based Approach", *AI Communications*, Vol. 19, pp. 313–340, 2006.

92. Horling, B., B. Benyo and V. R. Lesser, "Using Self-Diagnosis to Adapt Organizational Structures", *Agents 2001: 5th International Conference on Autonomous Agents*, pp. 529–536, 2001.

93. Pencole, Y., M.-O. Cordier and L. Roze, "Incremental Decentralized Diagnosis Approach for the Supervision of a Telecommunication Network", *IEEE Conference on Decision and Control*, pp. 435–440, 2002.

94. Lamperti, G. and M. Zanella, "EDEN: An Intelligent Software Environment for Diagnosis of Discrete-Event Systems.", *Applied Intelligence*, pp. 55–77, 2003.

95. Dellarocas, C., M. Klein and J. A. Rodríguez-Aguilar, "An Exception-Handling Architecture for Open Electronic Marketplaces of Contract Net Software Agents", *ACM Conference on Electronic Commerce*, pp. 225–232, 2000.

96. Singh, M. P., "Agent Communication Languages: Rethinking the Principles", *IEEE Computer*, Vol. 31, pp. 40–47, 1998.

97. Console, L., D. T. Dupré and P. Torasso, "Towards the Integration of Different Knowledge Sources in Model-Based Diagnosis", *Trends in Artificial Intelligence, 2nd Congress of the Italian Association for Artificial Intelligence, AI*IA*, Vol. 549 of *LNCS*, pp. 177–186, 1991.

98. Picardi, C., R. Bray, F. Cascio, L. Console, P. Dague, D. Millet, B. Rehfus, P. Struss and C. Vallée, "IDD: Integrating Diagnosis in the Design of Automotive Systems", *ECAI 2002: 15th European Conference on Artificial Intelligence*, pp. 628–632, IOS Press, 2002.

99. Witteveen, C., N. Roos, R. van der Krogt and M. de Weerdt, "Diagnosis of Single

and Multi-Agent Plans", *AAMAS 2005: 4th International Joint Conference on Autonomous Agents*, pp. 805–812, ACM, 2005.

100. Roos, N. and C. Witteveen, "Models and Methods for Plan Diagnosis", *Autonomous Agents and Multi-Agent Systems*, Vol. 19, No. 1, pp. 30–52, 2009.

101. Kaminka, G. A. and M. Tambe, "Robust Agent Teams via Socially-Attentive Monitoring", *Journal of Artificial Intelligence Research*, Vol. 12, pp. 105–147, 2000.

102. Ardissono, L., L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan and D. T. Dupré, "Enhancing Web Services with Diagnostic Capabilities", *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS 2005)*, pp. 182–191, IEEE Computer Society, 2005.

103. Wooldridge, M., N. R. Jennings and D. Kinny, "The Gaia Methodology For Agent-Oriented Analysis And Design", *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, pp. 285–312, 2000.

104. Lam, J. S.-C., F. Guerin, W. Vasconcelos and T. J. Norman, "Engineering Societies in the Agents World IX", chap. Coping with Exceptions in Agent-Based Workflow Enactments, pp. 154–170, 2009.

105. Dastani, M., V. Dignum and F. Dignum, "Role Assignment in Open Agent Societies", *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 489–496, 2003.

106. Krishnamurthy, B. and C. E. Wills, "On the Leakage of Personally Identifiable Information via Online Social Networks", *Computer Communication Review*, Vol. 40, No. 1, pp. 112–117, 2010.

107. Fang, L. and K. LeFevre, "Privacy Wizards for Social Networking Sites", *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pp.

351–360, 2010.

108. Akcora, C. G., B. Carminati and E. Ferrari, "Privacy in Social Networks: How Risky is Your Social Graph?", A. Kementsietsidis and M. A. V. Salles (Editors), *Proceedings of the 28th International Conference on Data Engineering (ICDE)*, pp. 9–19, IEEE Computer Society, 2012.

109. Kowalski, R. A. and F. Sadri, "Logic Programs, Norms and Action", chap. Teleo-Reactive Abductive Logic Programs, pp. 12–32, 2012.

110. Marinovic, S., K. Twidle and N. Dulay, "Teleo-Reactive Workflows for Pervasive Healthcare", *Pervasive Computing and Communications Workshops*, pp. 316 – 321, 2010.