Abstract

Open and dynamic multiagent systems are prone to exceptions due to unpredictable behavior. Since some of these exceptions may not be as serious as others, each exception need not be diagnosed in the first place. In order to determine which exceptions to diagnose, the first step involved is to define exception categories, and to classify exceptions into those categories according to their immediate causes. That is, an agent facing an exception should decide which category the exception belongs to based on its protocol knowledge. We propose such classification using commitment protocols. We provide a case study using a delivery process from e-commerce.

1. Introduction

Open multiagent systems are characterized by the fact that agents can enter and leave on will. There is usually no central authority that regulates the actions of the agents. Hence, an agent’s behavior can neither be controlled nor predicted before the system’s run time execution (Fisher & Wooldridge, 1994). When one or more agents behave unexpectedly in such a system, exceptions may occur, leading to the improper workings of the entire system. While diagnosing an exception is important, it is not feasible to deal with each exception separately. Thus, when an exception occurs, the agent facing the exception should determine its significance and decide whether to diagnose it further.

Accordingly, we provide categories of exceptions and propose to classify exceptions based on their immediate causes. Several work has been noted before in effort to classify exceptions (Klein & Dellarocas, 2000; Sadiq & Orlowska, 2000). Some of these work focus on exceptions that can be anticipated before. That is, the exception itself and its preconditions are known prior to its occurrence. Those exceptions can therefore be added to the system’s design before execution. However, most exceptions of that type are domain-specific. Thus, it is hard to make a general judgment based on that classification.

In this work, we deal with exceptions that are not anticipated at design-time, but rather encountered during runtime. So, the classification we propose can be applied to any domain (e.g., e-commerce) as long as a formal model for action descriptions and agent interactions are provided. Here, we use commitments to model agent interactions and predicate logic to represent outcomes of agent actions. We give a case study from a delivery process to exemplify its applicability. The next section gives necessary background on the concepts that are key to this work. Then, we present our classification algorithm, and the case study. Finally, we conclude with possible future directions.

2. Background

2.1 Commitments

Commitments are formed between two agents and roughly correspond to obligations (Singh, 1999). The debtor of a commitment is the agent that is committed to bring about a condition. The creditor benefits from the commitment. A base-level commitment \( c(x, y, p(z)) \) represents an obligation from debtor \( x \) to creditor \( y \) to bring about proposition \( p(z) \). A conditional commitment \( cc(x, y, q(z), p(z)) \) represents a contract between debtor \( x \) and creditor \( y \) with condition \( q(z) \) and proposition \( p(z) \). When \( q(z) \) is satisfied, \( x \) will become committed to \( y \) for satisfying \( p(z) \). We allow commitments to have three states; (1) active commitments are created and are in charge, (2) fulfilled commitments are no longer in charge due to their propositions being satisfied, (3) violated commitments are also not in charge but their propositions have not been satisfied.

2.2 Ontologies

An ontology is a data model used for representing a domain, and it defines a set of concepts and the relations between those concepts (Guarino, 1998). As is customary in multiagent systems, we use ontologies to provide domain knowledge to agents. The agents can then use this to reason on exceptions, and deal with them.
2.3 Protocols

A process is represented by a set of protocols (Papazoglou, 2003). Here, we are concerned with distributed settings where agents are only aware of part of their environment. Thus, it is important to specify protocols from the viewpoint of the roles that the agents are enacting. This enables us to specify the exact information that is available to each role.

Definition 1 A protocol template \( \mathcal{P}_r = (\mathcal{M}, \mathcal{P}, \mathcal{R}) \) is a description of what role \( r \) is aware of in order to act in its environment. \( \mathcal{M} \) is a set of messages that role \( r \) can send. It simply corresponds to the actions that role \( r \) can perform (with particular effects on commitments) while executing the protocol \( \mathcal{P}_r \). \( \mathcal{P} \) is a set of predicates that role \( r \) is aware of and \( \mathcal{R} \) is a set of roles that role \( r \) is aware of.

Definition 2 A protocol state \( S \) is a set containing predicates that hold at a particular time point and the status of commitments at that time point.

Definition 3 A goal state \( G_i \) is a desired state for agent \( i \) that it wishes to reach during execution of the protocol.

2.4 Exceptions

Exceptions have been classified before in the literature according to several criteria (Klein & Dellarocas, 2000; Sadiq & Orlowska, 2000). In the light of those, we classify exceptions into three distinct categories, each covering separate cases that may arise during protocol execution.

- **Violation**: A violation occurs when a predicate or commitment that is included in the agent’s goal state is violated, e.g., a bookstore violating its commitment with a customer for delivering its book.

- **Bad-Fulfillment**: There may be situations that although the goal is satisfied, an extra event obstructs its complete fulfillment, e.g., a CD comes damaged.

- **Bonus**: A bonus is an unexpected situation which is not necessarily a bad thing for the agent, e.g., the book comes with a CD. Although the agent may benefit from the situation, it is still considered an exception, and the agent may benefit from inspecting it further. Usually, protocols are extended with such opportunities. That is, the protocol execution that covers the bonus is added to the protocol itself.

The second and third classifications both represent a monotonic increase in the agent’s goal state (i.e., extra predicates). Thus, they point to unexpected situations even though the goal seems to be satisfied. The second represents an unwanted case and the agent’s goal is not fully satisfied since the extra predicate obstructs the goal itself. The third represents a beneficial case which the agent may benefit from. In that case, the extra predicate has no relation with the goal whatsoever.

3. Proposed Method

During protocol execution, agents change state due to performed actions or occurred events. When entered a new state, an agent compares the state with its goal state, checking for exceptions.

3.1 Classification

**Algorithm 1** describes how an agent classifies an exception based on its current state and its goal state. The first two cases cover violation based exceptions (lines 1-10). The third case covers situations where the goal of the agent is obstructed by an extra event (lines 11-15). The object of the extra predicate provides the relation with the goal. The last case covers bonus situations where the extra event has no relation with the goal (lines 16-20). Further classification can be done within these categories using a domain ontology (e.g., determine the level of violation, how badly the goal is fulfilled, or how significant the bonus is).

3.2 Case Study

Figure 1 describes the delivery process inspired from the *MIT Process Handbook* (Klein & Dellarocas, 2000). It contains three business parties; customer, bookstore, and
deliverer. In a normal execution, the customer purchases an item from the bookstore. The bookstore pays for the delivery of the item. The deliverer delivers the item.

\[ \text{Customer} \xrightarrow{\text{purchase}} \text{Bookstore} \xrightarrow{\text{deliver}} \text{Deliverer} \xrightarrow{\text{paydelivery}} \]

**Figure 1. Delivery Process**

We use following commitments to represent the contracts between the agents (i.e., parties): \( cc(\text{bookstore, customer, purchase}(z), \text{deliver}(z)) \) tells that if the customer purchases an item, the bookstore delivers that item. \( cc(\text{deliverer, bookstore, paydelivery}(z), \text{deliver}(z)) \) tells that if the bookstore pays for the delivery of an item, the deliverer delivers that item. The goal state of the customer is \( \{G_{\text{customer}} = \{\text{purchase}(z), \text{deliver}(z)\}\} \). Once the customer purchases an item, it wishes to get that item delivered. Let us now consider three cases from the delivery process.

- **Violation:** Assume the customer’s state to be \( G_{\text{customer}} = \{\text{purchase}(\text{book1})\} \). The customer has purchased a book, but the book does not arrive. This is a violation of the bookstore’s commitment to the customer, and it is classified as a violation.

- **Bad-Fulfillment:** Assume the customer’s state to be \( G_{\text{customer}} = \{\text{purchase}(\text{cd1}), \text{deliver}(\text{cd1}), \text{damaged}(\text{cd1})\} \). The customer has received the CD, but it is damaged. This situation obstructs the customer’s goal, and it is classified as a bad-fulfillment as the extra predicate is related with the goal.

- **Bonus:** Assume the customer’s state to be \( G_{\text{customer}} = \{\text{purchase}(\text{book1}), \text{deliver}(\text{book1}), \text{deliver}(\text{book2})\} \). So, the customer has received the book, in addition another book is also delivered. This corresponds to a bonus situation which is classified as the extra predicate’s object is not included in the customer’s goal.

Consider the following change in the bonus situation above; \( G_{\text{customer}} = \{\text{purchase}(\text{book1}), \text{deliver}(\text{book2})\} \). So, the customer has purchased \( \text{book1} \), but \( \text{book2} \) comes while there is still time for \( \text{book1} \) to be delivered. This is again classified as a bonus. However, \( \text{book2} \) may be wrongly delivered instead of \( \text{book1} \) (i.e., violation). This case actually corresponds to a deviation from normal protocol execution, and should be considered separately as a new category. In addition, the significance of an exception may vary within a category. Consider the customer’s state \( G_{\text{customer}} = \{\text{purchase}(\text{cd1}), \text{deliver}(\text{cd1}), \neg\text{invoice}(\text{cd1})\} \), meaning that an invoice has not been sent with the delivery. This is also considered a bad-fulfillment. However, it may not be as significant as the CD being damaged. The current classification does not support such level of detail. Domain knowledge is required to provide priorities within categories.

### 4. Discussion

We have previously considered identifying exceptions that occur in business protocols (Kafalı & Yolum, 2009). As a single agent task, identifying an exception’s actual cause is hard, if not impossible (due to limited knowledge). When considered as a collaborative task, diagnosing exceptions is even harder and requires considerable amount of computation. Thus, we have provided a classification based on exception categories. This classification is a forward step in determining the significance of each exception, and selecting which ones to diagnose further. Further classification can be done based on domain knowledge. This will provide extra accuracy while determining the diagnosis effort of an agent when faced with several exceptions with strict time considerations. In addition, other types of processes exist in e-commerce scenarios such as fit or sharing processes rather than flow processes as we have seen in our case study (Klein & Dellarocas, 2000). Although not for violation based exceptions, we believe that classification can be harder in those process types.

### References


