Handling Mitigating Circumstances for Electronic Contracts

Simon Miles[1], Paul Groth[2], Nir Oren[1], Michael Luck[1]

[1] Department of Computer Science, Kings College London, UK
[2] Vrije Universiteit, Amsterdam, Netherlands

Abstract. Electronic contracts are a means of representing agreed responsibilities and expected behaviour of autonomous agents acting on behalf of businesses. They can be used to regulate behaviour by providing negative consequences, penalties, where the responsibilities and expectations are not met, i.e. the contract is violated. However, long-term business relationships require some flexibility in the face of circumstances that do not conform to the assumptions of the contract: *mitigating circumstances*. In this paper, we describe how contract parties can represent and enact policies on mitigating circumstances. As part of this, we require records of what has occurred within the system leading up to a violation: the *provenance* of the violation. We therefore bring together contract-based and provenance systems to address the issue of mitigating circumstances.

1 Introduction

Commitments between business parties are generally regulated through *contracts*. These documents allocate responsibility for particular outcomes, allow parties to know what to expect of each other and provide a basis for redress should those responsibilities and expectations not be met. In many contexts, autonomous software agents can be used to advantageously represent businesses' interests in an automated way, including preparing, agreeing on, reasoning over, acting on and enforcing contracts in an electronic form. Much research has been conducted on how best to instantiate *contract-based systems* [3, 16, 17].

For the purposes of this paper[3], we consider a contract to be a set of *clauses*, each of which specifies some responsibility of an agent. A clause may specify an *obligation*, a *prohibition* or a *permission*, i.e. what should, should not or may be done respectively. The set of agents to which clauses apply are called the *contract parties*. One crucial aspect of an autonomous approach to electronic contracting is the handling of *violations* of a contract clause, when the stated responsibilities have not been fulfilled. There are different ways that a violation could be dealt with. For example, most contract-based systems will include a notion of payments, and so violations may automatically incur financial penalties.

However, relationships in business are important (as espoused by relational contract theory, all transactions occur in the context of a relationship [9]) and a company

[3] First presented at the Symposium on Behaviour Regulation in Multi-Agent Systems (BRMAS'08)

that handled all violations of a contract clause equally could damage its long-term relationships with partners. In situations in which unexpected circumstances have led a contract party to be unable to fulfil their responsibilities, other parties may act more leniently than they are contractually allowed to do, in order to maintain the long-term business relationship. Such circumstances are called *mitigating circumstances*.

In current electronic contracting approaches, mitigating circumstances are addressed (if at all) by passing the decision on how to handle a violation up to a human. However, organisations often have standard, if not publicised, policies for handling mitigating circumstances, and so automation is certainly possible. We would like to extend contract-based systems to allow agents to autonomously consider, and react appropriately to, mitigating circumstances. Note that a related factor influencing how forgiving a party may be to violation, not considered in this paper but addressed elsewhere [13], is the *intention* behind the actions leading to violation.

A pre-requisite to providing this extended functionality is the ability to determine whether there were, or may have been, mitigating circumstances for a violation, which requires reliable documentation of what has occurred and how that *caused* the violation. It is only through such documentation that mitigating circumstances will be evident. The problem of obtaining the relevant documentation of a violation's causes is exacerbated by the fact that violations may only be dealt with some time after they occurred, for instance where it is only through the accumulation of multiple failures over time that a contract clause is violated.

In this paper, we describe how recording and reasoning over the causes of violations can help to better manage the behaviour of parties in the system. This allows contracting parties to handle problems more flexibly, and encourage better coordination. Specifically, the technical contributions of this paper are as follows.

- An algorithm for handling mitigating circumstances in contract violation based on technologies for electronic contracting and for determining the *provenance* of violations, i.e. what caused them to occur.
- A re-usable model for expressing mitigating cirumstance policies.
- Application of this algorithm and model to an aerospace scenario.

The rest of the paper is structured as follows. Section 2 describes a motivating example application in the aerospace domain. Section 3 introduces our electronic contracting approach, and discusses the use of *provenance* to determine the cause of violations. Section 4 then details our algorithm, which is applied to the example application in Section 5. We finish the paper with a discussion of related work in Section 6 and conclusions in Section 7.

2 Example Scenario

Our example scenario is based on the aftercare market for aircraft engines. It is an extended version of that considered by Lost Wax's Aerogility application [10].

2.1 Contract

In this scenario, aircraft operators (e.g. airlines) establish contracts with engine manufacturers whereby the manufactures are obliged to ensure the aircraft have engines in

working order. To achieve this, an engine manufacturer regularly removes an engine from an aicraft for which it is responsible and replaces it with an already serviced engine to allow the aircraft to continue flying. This replacement must be performed in a timely fashion, so that the aircraft remains usable. As well as regular servicing, the engine manufacturer must respond to possible faults in an engine by similarly replacing it. Once removed, an engine is serviced and then returned to the pool of engines available for swapping into other aircraft.

The core contract between aircraft operator and engine manufacturer specifies the following:

- An engine requires servicing after every X flights, as well as when its health data indicates a possible fault.
- When an aircraft's engine requires servicing, the engine manufacturer must remove the engine and replace it with a serviced one.
- The aircraft operator is permitted to penalise the engine manufacturer if an aircraft is left on the ground for more than Y hours due to an engine not being available.

The core contract may be extended by extra constraints on the engine manufacturer in particular cases.

- Engines are ultimately composed of parts supplied by part suppliers. An aircraft operator may constrain an engine manufacturer only to use parts from given named suppliers in engines used in their aircraft.
- For best use of resources in fulfilling multiple contracts, an engine manufacturer will often take and service an engine from one operator's aircraft and put it into the aircraft of another operator. In some cases, one operator may not trust another. An operator may therefore constrain an engine manufacturer never to put engines into their aircraft that have previously been used by a particular other operator's aircraft.

2.2 Mitigating Circumstances

Where an aircraft has been grounded due to lack of a working engine, an aircraft operator will want to recoup their costs by penalising the manufacturer. However, the two companies wish to retain a good working relationship, and particular mitigating circumstances may be considered. Whether the operator makes these circumstances clear to the manufacturer in advance is a choice of the individual business.

In this scenario, we consider two mitigating circumstances.

Late Health Data A manufacturer is aware of a potential fault in an engine through analysing the engine's health data. This is recorded in the aircraft, and so the health data must be supplied by the operator. If supplied late, the manufacturer is delayed in servicing the engine.

Part Supplier Late If an operator restricts the manufacturer as to where it can source engine parts, and the required part supplier was late in supplying parts, then this can affect the manufacturer's ability to provide a working engine on time.

2.3 Managing Violations

The way that violations of the contract are handled should depend on circumstances. In the scenario, one or more of the following broad actions can be performed by the aircraft operator given a violation (e.g. aircraft remaining on the ground too long).

Full Penalty Operator deducts 30% from the monthly payment to the engine manufacturer.

Reduced Penalty Operator sends a formal notice reprimanding the manufacturer but acknowledging mitigating circumstances.

Reconsider Policy Operator starts reconsideration of its constraining policies in the contract.

The choice of a specific action is entirely based on the goals of the business, and is out of scope of this paper. For convenience, we assume that a reduced penalty will be the action taken in all subsequent examples. We now discuss the two primary technologies that our algorithm for handling mitigating circumstances depends upon.

3 Contracts and Provenance

Our approach brings together two technologies, described in detail below. The application is based on *contract-based systems* to support regulation of agents' behaviour through explicit contracts agreed between agents. The policies for mitigating circumstances use *provenance systems* to record documentation on what occurs within a system and use this to determine why a particular violation occurred.

3.1 Contract-Based Systems

A contract-based system is one in which agents agree to documents that specify requirements (obligations and prohibitions) or permissions on their behaviour. For our purposes, we define a *contract* to be an assignment of *clauses* to agents that have agreed to fulfil them.

For agents, acting on behalf of multiple organisations to be able to set up and rely on contracts for mutual benefit, we require a supporting infrastructure. This can be expressed in terms of agents playing administrative roles, such as storing contracts to ensure access to them and preserving their integrity over their lifetime. It may also include *monitor* roles, which require the agents playing them to check that clauses are being fulfilled and, where they are not, to notify the *enforcer* of that contract clause, i.e. the agent responsible for handling that clause's violation.

Current work, such as that conducted in the CONTRACT project [14, 15, 17], has begun to bring together: existing technologies to specify contract languages; frameworks for defining contract-based applications; administrative architectures containing those infrastructural roles needed to manage the contracts; and model checking techniques for verifying that agents in an application are able to fulfil their contractual responsibilities. In this paper, we assume the presence of a contract language and administrative infrastructure. In general, we will not refer to these further, as they are out

of scope of the work. However, the monitoring of the fulfilment of contract clauses is a vital part of understanding the context in which a violation occurs. For explanatory purposes, we will assume single agents playing monitor and enforcer roles for checking and ensuring the fulfilment of all contract clauses. In reality, it is often the case that many such agents need to exist for an application, as monitoring may use and/or produce information private to individual contract parties.

3.2 Provenance and Causation

As previously stated, reliable documentation is necessary in order to determine whether the causes of a violation are sufficient to mitigate it. Thus, we need to be able to determine the *provenance* of a particular event (e.g. violation), that is what *caused* it to occur as it did. In the study of art, the provenance of an artwork can include the artist, the materials used in creating it, the restoration done over time, the different locations where it has been stored or exhibited, and so on. All of these ultimately caused the artwork to be as it is now.

In prior research, we studied provenance in the context of a wide range of sciences [11], where it was clear that knowing the provenance of results is important in science experiments for many purposes, e.g. peer reviewers determining if an experiment was rigorous and sound, understanding where an error may have occurred that affected results, re-use of configuration of successful experiments, etc. Many of these same requirements are present in business situations, in particular the ability to verify the correct performance of business workflows. Thus, we can apply many of the same techniques that we used for the scientific domain to our business use case.

With regard to the violation of a contract clause, causes can include the actions (or absence of actions) of the responsible party, but may also include actions of other agents and occurrences more widely within the application environment.

Determining the provenance of an occurrence therefore requires data on its causes. As we often do not know in advance that something particular will occur, agents must *record* both what occurs *and* causal connections between occurrences around the time that they happen. The documentation forms a *causal graph*, depicting where A was caused by B, which was caused by C, etc. Below, we denote that A was caused by B (A is effect, B is cause) as $\boxed{A} \rightarrow \boxed{B}$

In order to ensure the availability of such a causal graph for our algorithm, we only consider applications that have been made *provenance-aware*, which entails that most, if not all, software agents are designed or adapted to record what they do and what caused them to do it (messages received, their goals etc.) [13]. In a contract-based environment, this includes both the contract parties and the infrastructure agents, such as the monitor. When it is not possible to record the causal connections between occurrences, it is often possible to infer that they exist from what has been recorded. The engineering exercise of making an application provenance-aware is out of scope here, but explained elsewhere [12].

To illustrate how provenance provides better understanding of an occurrence, we describe the provenance of an engine being made available after servicing in Figure 1. We begin at the top of the figure. Originally, it is determined that an engine, E, requires

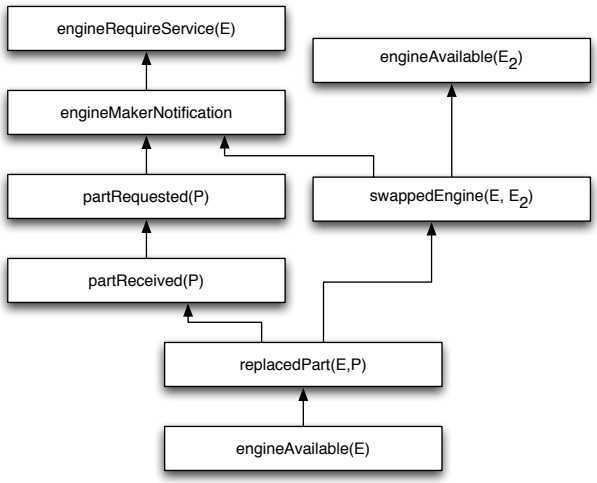

Fig. 1. Provenance of an engine as a causal graph

servicing. This occurrence causes the engine manufacturer to be notified. The engine manufacturer then requests and receives a part, P. In parallel to the engine manufacturer being notified, another engine becomes available (the relative times of the events occurring are omitted for brevity but each event could potentially occur long before the other or near simulataneously). Together, the occurrences of a engine becoming available and the engine manufacturer being notified, cause the engine, E, to be swapped out for engine E_2. Once engine E is taken out of the airplane, its defective part is replaced with the part ordered by the engine manufacturer. This replacement of parts causes E to be made available once again for use in other aircraft.

4 Handling Mitigating Circumstances

In this section, we bring together the contract-based and provenance technologies described above to give an algorithm for handling mitigating circumstances when a violation is detected. We summarise the algorithm below, and then describe each step in more detail.

1. **Violation Detection** From checking the environment, the monitor determines that a clause was violated, and informs the relevant enforcer.
2. **Cause Determination** The enforcer uses heuristics to infer possible undocumented causes of the violation.
3. **Mitigating Circumstances** The enforcer uses policies to determine, from the recorded and inferred causes of the violation, whether there were mitigating circumstances.
4. **Remedy** If mitigating circumstances were found, then the enforcer acts to remedy the situation and ensure that violations are less likely to occur in future. The action is again determined by its policy.

Remember that enforcer is a role, so may be performed by one of the contract parties (e.g. the aircraft operator) or an appropriate third party.

4.1 Violation Detection

In order to detect the violation, the monitor must observe its environment on the basis of what is expected from fulfilling the contract clause. When the violation occurs, it notifies the enforcer of that clause (the enforcer is often a party to the contract, i.e. the agent that gains from the clause's fulfilment).

Being provenance-aware, the monitor records several pieces of documentation about what it does: the clause-related observation of the environment, the signalling of a violation, and the causal connection between the two (the former causes the latter).

4.2 Cause Determination

When a violation has occurred, the enforcer first checks whether there are undocumented causes it can infer from the available documentation. This provides a more complete picture from which it can then determine whether there were mitigating circumstances. In particular, many violations of contractual clauses occur because something *did not* happen (e.g. a supply contract is violated when goods are *not* delivered). In these casess, there will be no documentation because there was no occurrence to document. However, that absence of occurrence is directly related to the violation and so needs to be made explicit. Inference is achieved by applying *inference rules*.

An inference rule expresses a heuristic by which the enforcer determines new causal connections from existing facts. Its antecedent is an expression composed of parametrised predicates, and its consequent takes the form of causal graph edges between occurrences. The antecedent's predicates are facts from one of four sources:

Domain Knowledge Timeless knowledge about the domain available to the enforcer.
Contract Clauses Clauses of the contract that has been violated.
Contract Party Documentation Documentation recorded by the contract parties of what they know to have occurred.
Monitor Documentation Documentation recorded by the monitor of what it knows to have occurred.

The antecedent may also contain mathematical expressions resolving to true or false based on the predicate variables, e.g. $A > B$. The consequent of an inference rule contains a set of causal connections between predicates from the antecedent (i.e. known occurrences).

An example of a whole rule is given below. In this rule, the antecedent is a conjunction of two facts documented by agents in the system and a relationship between them. The facts are that an engine's health data was received at time T_1 and that the aircraft with that engine was unserviced at time T_2 (determined by the monitor because the contract states it *should* have been serviced by this time). The relationship expresses that T_2 was less than 10 hours after T_1. The consequent of the rule, i.e. that implied by any pattern of occurrences matching the former facts, is that the fact that the engine was

not serviced at T_2 was caused by the health data being received at T_1. This rule derives from a heuristic that waiting beyond 10 hours for the health data can be expected to affect the ability to service the aircraft on time. Whenever the documented facts of a violation match the antecedent, the consequential causal connection will be added to the facts from which mitigating circumstances will be assessed.

Antecedent
$receivedHealthData(E, T_1) \wedge$
$unserviced(A, E, T_2) \wedge$
$T_2 < T_1 + 10$
Consequent
$\boxed{unserviced(A, E, T_2)} \rightarrow \boxed{receivedHealthData(E, T_1)}$

Of course, we cannot know from the rule above that the lack of health data was the only or 'primary' cause of violation. Instead, the rule should be read as a heuristic for determining likely mitigating circumstances.

4.3 Mitigating Circumstances

Determination of mitigating circumstances is achieved by a *policy* setting out where the causes of a violation suggest mitigating circumstances, and what action to take in each such case. Such a policy could be included as part of a contract document, in which case other parties may use it to reason about what they can get away with, or may be private to the owning contract party, if they prefer to keep the mitigating circumstances considerations secret. There are likely to be several different kinds of mitigating circumstance, such as the two given for the example in Section 2.2. For each kind, there is a pre-condition and a remedy.

The pre-condition is a causal graph between occurrences, in the form of a tree with a violation occurrence as its root. All occurrences in the tree can be parametrised with variables. As a whole, the pre-condition graph acts as a template for chains of causes of a particular form leading to a violation. The template graph is then matched against the documentation recorded and inferred. If they match, mitigating circumstances have been found, and the remedy enacted.

An example of a mitigating circumstances policy statement is given below. The precondition is a tree of causal connections from the violation of a clause concerning a particular aircraft. The precondition is a template which can be matched against documented, or inferred, facts. In this case, the violation must have been caused by the aircraft's engine not being serviced at a given time, which in turn must have been caused by the engine health data being received at a given time. If this pattern is found within the documentation, then the pre-condition is matched, the policy applies, and the appropriate action is taken: reduced penalty, in this case.

Precondition	**Remedy**
$\boxed{violation(A)} \rightarrow$	
$\boxed{unserviced(A, E, T_2)} \rightarrow$	Reduced Penalty
$\boxed{receivedHealthData(E, T_1)}$	

4.4 Remedy

For the purposes of this paper, we consider the remedy to be a simple action by the enforcer, such as reducing the penalty that would otherwise be placed on the violating agent. In future work, we will consider more sophisticated mechanisms, such as negotiating to adjust the contract to more realistically suit the working environment.

4.5 Algorithm for Handling Mitigating Circumstances

The algorithm can be expressed in pseudo-code as follows. First, we define the variables referred to in the algorithm.

- C: the set of contract clauses of which to detect violations
- R: the set of inference rules
- G: a graph (V, E) where vertexes represent occurrences and edges causal relationships between them
- G_{KB}: the union of the knowledge sources (domain, contract, contract party, monitor)
- P: a mapping from violation types to sets of policies concerning those types
- A: an array of actions to be taken indexed by a policy

Next, we describe the functions used as part of the algorithm.

- RETRIEVEVIOLATION() - retrieves a violation from the monitior
- RETRIEVEOBSERVATION(v) - retrieves the observation that caused a violation
- CONSEQUENTOF(r) - retrieves the edge representing the consequent of a rule
- APPLYRULE(r, G) - apply an inference rule, r to the graph G
- UNION(G_1, G_2) - perform a union between the two graphs (i.e. combine their edges and vertexes)
- EXTRACTSUBGRAPH(v, G) - given a vertex extract the subgraph beginning at that vertex
- TEMPLATEISOMORPHISM(G_1, G_2) - determine whether the two graphs are isomorphic, implementations may define isomorphism in terms of attributes associated with vertexes and edges
- EXECUTE(a) - execute a given action, a

The algorithm itself is then shown below, as a sequence of four steps corresponding to those outlined above.

VIOLATIONDETECTION(C)

1 **if** the monitor detects a violation of clause, $cl \in C$
2 **then** $v \leftarrow$ RETRIEVEVIOLATION()
3 $c_v \leftarrow$ RETRIEVEOBSERVATION(v)
4 **return** new graph edge (v, c_v)

CAUSEDETERMINATION(c_v, R, G_{KB})

1 $G_{KB}^{new} = \emptyset$
2 **for** each inference rule $r \in R$
3 **do** $(e, c) \leftarrow$ CONSEQUENTOF(r)
4 **if** $e = c_v$
5 **then** $G \leftarrow$ APPLYRULE(r, G_{KB})
6 $G_{KB}^{new} \leftarrow$ UNION(G_{KB}^{new}, G)
7 **return** G_{KB}^{new}

MITIGATINGCIRCUMSTANCES(v, P, G_{KB})

1 **for** each policy $p \in P[v]$
2 $G_{c_v} \leftarrow$ EXTRACTSUBGRAPH(v, G_{KB})
3 **if** TEMPLATEISOMORPHISM(G_v, p)
4 **then return** p

REMEDY(p, A)

1 $a \leftarrow A[p]$
2 EXECUTE(a)

5 Applying to the Case Study

In this section, we apply our algorithm to the scenario presented in Section 2. We start by defining the facts that may be documented or inferred by an aircraft operator agent in the scenario. These are expressed using predicate logic and described in Table 1. Using statements of this form, we can construct propositions about what is documented or believed at any one time. In this and subsequent schemas, we use the convention of lower case letters (a) for constants and upper case letters (A) for variables.

We now describe two use cases in which there are mitigating circumstances that the aircraft operator, acting in the role of enforcer, takes into account, matching those described in Section 2.2. In both use cases below, the monitoring mechanism discovers that an engine has not been serviced at a given time, even though it contractually should have been. In each case, a different mitigating circumstance has occurred, and so a reduced penalty is applied. For each use case, we show how the algorithm in the previous section is applied.

5.1 Late Health Data

The following operation of the contract parties is documented.

– The engine manufacturer, as part of its operation, receives the health data for an engine at a given time: $receivedHealthData(e, t_1)$. This is the engine of an aircraft, a, which has earlier been recorded as requiring servicing.

Violation Detection The monitor determines that an aircraft requires servicing but has not been serviced at this moment: $unserviced(a, e, t_2)$. It further determines that, contractually, it should have been serviced before now. It therefore reports a violation:

Predicate	Description
Domain Knowledge	
$owns(A, O)$	Operator O owns aircraft A
Contract Clauses	
$constrainedPartSupplier(S, P)$	Contractual obligation to use supplier S for part P
$disallowedPriorUse(O)$	Contractual prohibition from using engines previously used by operator O
Contract Party Documentation	
$engineAvailable(E)$	Engine E is serviced and available for use
$partReceived(S, P, T)$	Manufacturer received part P from supplier S at time T
$partRequested(S, P, T)$	Manufacturer requested part P from supplier S at time T
$receivedHealthData(E, T)$	Manufacturer received health data about engine E at time T
$replacedPart(E, P, T)$	Part P was replaced in engine E at time T
$swappedEngine(A, E_1, E_2, T)$	Engine E_1 was removed, engine E_2 inserted into aircraft A at time T
Monitor Documentation	
$unserviced(A, E, T)$	Engine E of aircraft A requires but has not received servicing at time T
$violation(A)$	Violation of the contractual obligation regarding servicing aircraft A

Table 1. Example knowledge predicates

$violation(a)$. The causal connection between these two occurrences is documented: $\boxed{violation(a)} \rightarrow \boxed{unserviced(a, e, t_2)}$.

Cause Determination The operator believes that the health data should have been received at least 10 hours in advance for the manufacturer to be able to complete the job in time. This belief implies a causal connection between an engine not being serviced and that engine's health data being received late (the former was due to the latter). The operator first infers the causal connection from the available data using the following inference rule.

Antecedent
$receivedHealthData(E, T_1) \wedge$
$unserviced(A, E, T_2) \wedge$
$T_2 < T_1 + 10$
Consequent
$\boxed{unserviced(A, E, T_2)} \rightarrow \boxed{receivedHealthData(E, T_1)}$

Mitigating Circumstances From this, we then have a sequence leading to a violation matching the pre-condition of the following rule: receiving engine health data at a given time (later than expected) caused the engine not to be serviced, which caused a violation.

Precondition	Remedy
$\boxed{violation(A)} \rightarrow$ $\boxed{unserviced(A, E, T_2)} \rightarrow$ $\boxed{receivedHealthData(E, T_1)}$	Reduced Penalty

5.2 Part Supplier Late

The following operation of the contract parties is documented.

- The contract constrains the manufacturer to use a given part supplier for parts of a particular type: $constrainedPartSupplier(s, p)$.
- At some point, the manufacturer requires a part of this type and orders it from the supplier: $partRequested(s, p, t_1)$.
- The supplier eventually provides the part: $partReceived(s, p, t_2)$.
- The engine manufacturer is required to service an engine that requires a part of the above type. When the part is available, the manufacturer puts the new part into the engine: $replacedPart(e_2, p, t_3)$.
- This repaired engine is later used to swap into an aircraft requiring a service: $swappedEngine(a, e_1, e_2, t_4)$.
- A causal chain is recorded: the engine swap required the replacement of the part, which required the part to be received, which required the part to be requested from the supplier, which was made to that supplier because of the contract clause:

$\boxed{swappedEngine(a, e_1, e_2, t_4)} \rightarrow \boxed{replacedPart(e_2, p, t_3)} \rightarrow \boxed{partReceived(s, p, t_2)} \rightarrow$
$\boxed{partRequested(s, p, t_1)} \rightarrow \boxed{constrainedPartSupplier(s, p)}$

Violation Detection The monitor determines that an aircraft requires servicing but has not been serviced at this moment: $unserviced(a, e_2, t_5)$. It further determines that, contractually, it should have been determined before now. It therefore reports a violation: $violation(a)$. The causal connection between these two occurrences is documented: $\boxed{violation(a)} \rightarrow \boxed{unserviced(a, e_2, t_5)}$.

Cause Determination The operator believes that if the part supplier supplied a part late, this can lead to problems servicing aircraft (specifically, the part is supplied less than 48 hours before servicing is due). This belief expresses a causal connection between the engine not being serviced and the part being received late. It first infers the causal connection from available data using the following rule.

Antecedent
$partReceived(S, P, T_2) \wedge$ $swappedEngine(A, E_1, E_2, T_4) \wedge$ $replacedPart(E_2, P, T_3) \wedge$ $unserviced(A, E_2, T_5) \wedge$ $T_5 < T_2 + 48$
Consequent
$\boxed{unserviced(A, E_2, T_5)} \rightarrow \boxed{partReceived(S, P, T_2)}$

Mitigating Circumstances From this, we have a sequence leading to a violation matching the pre-condition of the rule below: a constraint on the part supplier caused a supplier to be used in requesting a part which caused the part to be delivered at a particular time (late) which caused the engine not to be serviced, which caused a violation.

Precondition	**Remedy**
$violation(A)$ $\rightarrow$	
$unserviced(A, E_2, T_5)$ $\rightarrow$	
$partReceived(S, P, T_2)$ $\rightarrow$	Reduced Penalty
$partRequested(S, P, T_1$ $\rightarrow$	
$constrainedPartSupplier(S, P)$	

6 Related Work

In recent work on normative systems and agreement in service-oriented architectures, *norms* specifying patterns of behaviour for agents, *contract clauses* as concrete representations of dynamic norms, management or enforcement of norms itself being a norm, are all already established in the literature [1, 3, 8, 16]. Such work has focused on the infrastructure needed to support such systems and handling of violations is often through the mechanism of immediately issuing contractually fixed penalties. There are notable exceptions, however. For example, Xu et al. [18] use commitment graphs to determine which of multiple contract parties are responsible for a violating effect (action or absence of action), when one party's actions are a pre-requisite for those of other parties. Cardoso and Oliveira [2] model obligations with fixed deadlines in such a way that other parties are merely able to take penalties following the deadline, it is not an automatic or immediate act, therefore also allowing some flexibility for preserving business relationships. Also, longer-term issues are considered by Duran et al. [4], who examine how observation of fulfilment and violation of obligations can feed into a longer-term assessment of agents through *testimonials*.

There have been many recent approaches to the recording causal documentation so that the provenance of occurrences can be determined. It is applicable to a wide range of applications [11], and has particularly been considered in the context of workflow enactment, i.e. automatically recording documentation as each step of a workflow is executed [5, 19]. In our own work we have examined how provenance can be used to interpret and ask questions about the validity of experimental results [7]. Also, we have applied integrated provenance and electronic contract research in an alternative way in other work [6], in which we judge the likely trustworthiness of new contract proposals on the basis of past experience of similar contracts (as documented in provenance).

7 Conclusions

When a contract clause between parties is violated, a single fixed penalty is an inflexible way to manage the situation. In many real world cases, the party permitted to enact the

penalty may wish to take into account *mitigating circumstances*, for the sake of the long-term business relationship. Mitigating circumstances, and how to act when they occur, can be expressed in a policy document, but in order to judge circumstances against the policy we need a reliable record of what led to the violation occurring.

In this paper, we have provided an algorithm, and accompanying data structures, for evaluating whether violations were caused by mitigating circumstances, and acting accordingly. This makes use of a contract-based framework, by which we can define the contract clauses, and provenance technology, by which agents can document the *causes* of what occurs. In combination, this allows us to express and enact mitigating circumstances policies. We have shown how this applies in a concrete example in the aerospace domain.

This is preliminary work, which needs to be tested in practical applications. Future work will concern the re-usability of (parts of) mitigating cirumstances policies, and methodological guidelines for constructing them, both aimed at easing the process of implementing such policies in diverse applications.

Acknowledgements The research described in this paper is partly supported by the European Commission Framework 6 funded project CONTRACT (INFSO-IST-034418). The opinions expressed herein are those of the named authors only and should not be taken as necessarily representative of the opinion of the European Commission or CONTRACT project partners. We would also like to thank Lost Wax, and Camden Holt in particular, for discussions on the aerospace use case.

References

1. IST CONTRACT project. http://www.ist-contract.org, 2007.
2. H.L. Cardoso and E. Oliveira. Directed Deadline Obligations in Agent-based Business Contracts. In *Coordination, Organization, Institutions and Norms workshop at AAMAS (COIN@AAMAS) 2009*.
3. Chrysanthos Dellarocas. Contractual agent societies: Negotiated shared context and social control in open multi-agent systems. In *Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems (Agents-2000)*, Barcelona, Spain, June 2000.
4. Fernanda Duran, Viviane Torres da Silva, and Carlos J. P. de Lucena. Using testimonies to enforce the behaviour of agents. In Jaime Sichman and Sascha Ossowski, editors, *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN)*, pages 25–36, Honolulu, Hawai'i, May 2007.
5. Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW 2006)*, Lecture Notes in Computer Science. Springer, 2006. To appear.
6. Paul Groth, Simon Miles, Sanjay Modgil, Nir Oren, Michael Luck, and Yolanda Gil. Determining the trustworthiness of new electronic contracts. In *Proceedings of the 10th International Workshop on Engineering Societies in the Agents' World (ESAW 2009)*, 2009.
7. Paul T. Groth. *The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes*. PhD thesis, University of Southampton, September 2007.

8. Fabiola Lopez y Lopez, Michael Luck, and Mark d'Inverno. A normative framework for agent-based systems. *Computational and Mathematical Organization Theory*, 12(2–3):227–250, 2005.
9. I.R. Macneil. Relational contract theory: challenges and queries. *Northwestern University Law Review*, 94:877–907, 1999.
10. Felipe Meneguzzi, Simon Miles, Camden Holt, Michael Luck, Nir Oren, Nora Faci, and Martin Kollingbaum. Electronic contracting in aircraft aftercare: A case study. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, 2008.
11. Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *Journal of Grid Computing*, 5:1–25, 2007.
12. Simon Miles, Paul Groth, Steve Munroe, and Luc Moreau. Prime: A methodology for developing provenance-aware applications. *ACM Transactions on Software Engineering and Methodology*, June 2009.
13. Simon Miles, Steve Munroe, Michael Luck, and Luc Moreau. Modelling the provenance of data in autonomous systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems 2007*, pages 243–250, Honolulu, Hawai'i, May 2007.
14. Simon Miles, Nir Oren, Michael Luck, Sanjay Modgil, Felipe Meneguzzi, Nora Faci, Camden Holt, and Gary Vickers. *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*, chapter Electronic Business Contracts between Services. IGI Global, 2009.
15. Sanjay Modgil, Nora Faci, Felipe Meneguzzi, Nir Oren, Simon Miles, and Michael Luck. A framework for monitoring agent-based normative systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 153–160, Budapest, Hungary, May 2009. IFAAMAS.
16. Edward Muntaner-Perich, Josep Lluis de la Rosa, and Rosa Esteva. Towards a formalisation of dynamic electronic institutions. In Jaime Sichman and Sascha Ossowski, editors, *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN)*, pages 61–72, Honolulu, Hawai'i, May 2007.
17. Nir Oren, Sofia Panagiotidi, Javier Vazquez-Salceda, Sanjay Modgil, Michael Luck, and Simon Miles. Towards a formalisation of electronic contracting environments. In *Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in Agent Systems at AAAI 2008 (COIN 2008)*, 2008.
18. Lai Xu, Manfred A. Jeusfeld, and Paul W. P. J. Grefen. Detection tests for identifying violators of multi-party contracts. *ACM SIGecom Exchanges*, 5:19–28, April 2005.
19. Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi. Mining taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 2007.