Applying electronic contracting to the aerospace aftercare domain

Felipe Meneguzzi[a,*], Sanjay Modgil[b], Nir Oren[c], Simon Miles[b], Michael Luck[b], Noura Faci[d]

[a] *Robotics Institute*
Carnegie Mellon University
Pittsburgh – USA
[b] *Department of Informatics*
King's College London
London – UK
[c] *Department of Computing Science University of Aberdeen*
Aberdeen – UK
[d] *LIRIS*
Université Lyon 1
Lyon – France

Abstract

The CONTRACT project was a European Commission project whose aim was to develop frameworks, components and tools to model, build, verify and monitor distributed electronic business systems based on electronic contracts. In this context, an electronic contract provides a specification of the expected behaviours of individual services, with the assumption that these services are often enacted by autonomous agents. Using the theoretical tools created by the project, in this paper we describe the complete life cycle of instantiating an electronic contracting system using the CONTRACT framework within the aerospace aftercare domain. Thus, we use a natural language description of parts of the types of contracts used in this domain to generate individual norms amenable to a computational representation, and how these norms are used to generate a concrete contract monitor. Moreover, we describe a concrete implementation of CONTRACT agents in the AgentSpeak(L) language and how these agents interact within a concrete instantiation of CONTRACT.

Keywords:
CONTRACT, Norms, Monitoring, BDI, ATN

1. Introduction

As more and more business is conducted electronically over the internet, the need to guarantee transactions through formal electronic contracts is becoming increasingly important. In traditional business transactions, people need to meet and agree to terms of contracts before any exchange of goods and services takes place, but such a model is too slow, and inappropriate, in an interconnected world in which computer systems can replace human decision makers in closing deals and enacting decisions without constant supervision. In particular, systems of autonomous agents may be able to conduct business in an environment governed by formally documented norms, whereby different parties agree to the terms of business interactions specified in formal *electronic contracts*. These contracts must be represented in a way that allows autonomous agents to reason about them so that the agents can steer their behaviour towards compliance [1] and, more importantly, so that others can process such contracts and determine whether they have indeed been complied with.

*Corresponding Author
Email addresses: meneguzz@cs.cmu.edu (Felipe Meneguzzi), sanjay.modgil@kcl.ac.uk (Sanjay Modgil), n.oren@abdn.ac.uk (Nir Oren), simon.miles@kcl.ac.uk (Simon Miles), michael.luck@kcl.ac.uk (Michael Luck), noura.faci@univ-lyon1.fr (Noura Faci)

To address this need, we have sought to create a framework, and guidelines, for the development of electronic contracting systems for real world applications. Like their physical counterparts, electronic contracts contain a set of clauses describing the expected behaviour of the signatories. In this view, clauses in such electronic contracts are comprised of individual *norms* [2] that describe expected agent behaviour in terms of the deontic modalities of obligations, prohibitions and permissions. For example, in the aerospace aftercare domain, aircraft engine manufacturers are contracted not only to sell new engines, but also to provide for their long term maintenance, subject to various terms by the airlines, and depending on terms from the suppliers of the component parts of these engines.

While we have previously outlined our basic computational framework for electronic contracting [3], in this paper we review that framework and provide an instantiation of it for such an aerospace aftercare use case [4], demonstrating the end-to-end application of the techniques developed, and how the generic framework can be used in a concrete real-world application of electronic contracting. The resulting application provides a representation of the required electronic contracts used in the multi-agent system that simulates the aftercare domain, as well as a monitoring process that is capable of generating explanations for contract violations [3].

We start the paper by describing the domain in Section 2, including two specific scenarios in which electronic contracting is to be demonstrated. Next, we provide an overview of the contracting architecture in Section 3, explaining the components of an electronic contracting system following our framework. Once we have laid out the motivating aspects of our work, we proceed by describing the representation of the electronic contracts required for the aerospace domain in Section 4, showing how such contracts are represented originally in natural language, but eventually leading to the machine processable format needed in the system. After a contract has been created and signed by the relevant parties, we need to *enact* the contract, and monitor this enactment, as described in Section 5. The monitoring process detects when agents fail to comply with the norms they are committed to, but does not help to trace the *original cause* for a chain of violations, which may be critical in avoiding future failures. Hence, we describe a violation explanation mechanism in Section 6. We finish by showing the results of our practical implementation and empirical tests in Section 7.

2. An Aerospace Aftercare Domain

2.1. Domain Overview

The aerospace aftermarket is increasingly populated by customers buying a service rather than a product. For example, aircraft engine manufacturers provide long term commitments to make available operational engines for the aircraft of airlines (or *operators*) in order that their aircraft are not grounded while awaiting engines, and thus prevented from flying. Specifically, these commitments[1] consist of having minimum numbers of spare engines available at specific locations (a given engine manufacturer may service an aircraft operator at multiple sites) and not allowing any aircraft to be idle for greater than an agreed duration.

These minimum service level commitments are stipulated in aftercare contracts. If the commitments are violated (for example, when an operator's aircraft is grounded, awaiting functioning engines for a period of time greater than that agreed with the engine manufacturer), then engine manufacturers receive predetermined financial penalties. Based on these contracts, engine manufacturers establish repair contracts with service sites (usually located at airports) that are responsible for the actual repair of aircraft engines. Commitments on an engine manufacturer to have engines available for a given aircraft operator imply commitments in a repair contract, requiring that a service site repairs engines within a given time period.

Aftercare and repair contracts also contain other interdependent commitments that are secondary to the core service level and repair commitments. For example, a given aircraft operator

[1]The semantics and normative language we describe later make the notion of a commitment more concrete, by distinguishing between, and giving structure to, obligations, prohibitions and permissions.

Cycle	Aircraft engine usage — and hence wear — is measured in cycles. A return short-haul flight is one cycle. A long-haul flight may count as up to 4 cycles.
Engine	An engine model typically comes in 2–3 variants, such as for short or long-haul aircraft. Each aircraft has 2–4 engines.
Module	Modules are the main components of an engine and the basis of engine maintenance scheduling. An engine typically comprises 10–20 modules. Modules are manufactured and maintained by the engine manufacturer. Some modules are common between engine variants.
Part	Some modules contain parts (provided by a part supplier). Parts have a fixed maximum life but can fail before then.
Hard Life	A module is built to last until it reaches the end of its hard life. Then the module must be refurbished. The refurbished engine has a new hard life.
Scheduled Maintenance	Modules reach their hard life after a specified known usage so their refurbishment can be scheduled.
Unscheduled Events	Modules break down unpredictably and need to be repaired. Depending on the fault they may be able to make some more flights first.

Table 1: Terms used in the domain description

may place restrictions on the provenance of engines. Thus, an engine manufacturer may be committed to not using engines previously mounted on the aircraft of one of the operator's competitors. Similarly, an engine manufacturer may be committed to not using parts for engine repair supplied by specific part suppliers. These restrictions may in turn need to be stipulated in the repair contracts, so that if an engine manufacturer is committed to not using parts from a given part supplier, then the repair contract between the engine manufacturer and the service site responsible for the actual repair must also prohibit the service site from ordering parts from that part supplier.

In this business model, servicing and maintenance becomes a key driver of long term profitability for the engine manufacturer. Aftercare contracts are worth millions of Euros and can last several years. They are complex, with stipulated service levels and penalties for failure to meet them. To summarise key aspects of the domain, and to set up the example used throughout the paper, we list relevant terms from the aerospace aftermarket domain, with an explanation, in Table 1.

The aerospace aftermarket use case considered in this paper is inspired by Lost Wax's Aerogility [5] product, an agent-based decision support tool to simulate the aerospace aftermarket. Outlined by Jakob *et al.*[4], and further developed by Meneguzzi *et al.*[6], the use case is an ideal application for evaluating and validating the developed concepts and technologies.

2.2. Domain Agents

Engine usage is measured in terms of usage cycles, where the number of cycles clocked up by an engine depends on the length of the flight. Engines have a *hard life* represented by a predetermined number of cycles, after which they must be refurbished before being used again. Within our model, operators fly aircraft to fulfil a predetermined schedule of flights, which results in cycles being logged for the engines of the aircraft involved. When engines have clocked up enough cycles to end their hard life, an operator sends a request for maintenance to the engine manufacturer, which swaps used engines for new or already serviced ones. Manufacturers need to respond to maintenance requests within a certain time limit, otherwise they are violating the terms of their aftercare contract.

As we shall see in Section 3.3, agents subscribing to a contract can fulfil roles associated with the business goals of an application and administrative roles associated with the maintenance of the contracting environment itself. Business roles are application-specific and, in the case of this scenario, consist of the *aircraft operator*, *engine manufacturer*, *service site*, *part supplier* and *logistics*.

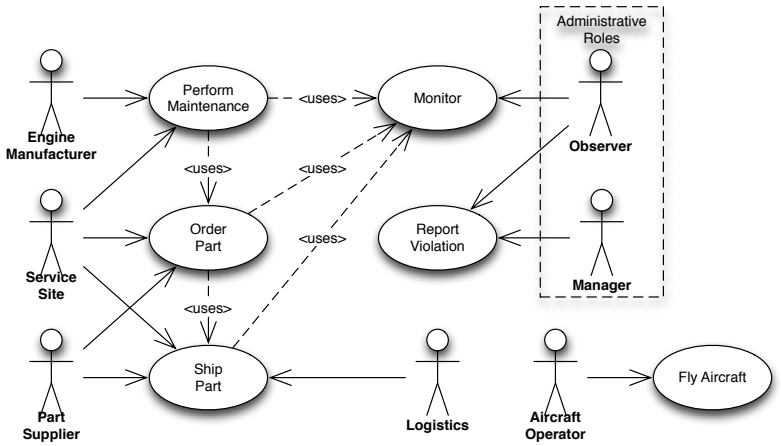


Figure 1: Use cases and interactions (in UML).

We also use both *observers* and *managers* as administrative contract parties in our system. In particular, an *observer* is responsible for monitoring maintenance requests from the operator and subsequent maintenance operations by the engine manufacturer, notifying the manager when a violation of contract terms is detected. A *manager* is responsible for receiving notifications of violations from the observer, and taking action towards remedying them which, in our current implementation consists of notifying a human operator. These interactions and their associated use cases are illustrated in the use case diagram of Figure 1, in which we emphasise administrative roles within the dashed box.

Aircraft operators are responsible for performing flights according to their designated schedule, notifying engine manufacturers of unscheduled events, and scheduling maintenance before an engine's hard life is reached. Engine manufacturers are responsible for performing maintenance operations by the contracted deadlines when these are scheduled, or as soon as possible when these are unscheduled, as well as maintaining a pool of operational engines according to contractual obligations.

Our application thus comprises, at the top level, *aircraft operators* signing aftercare contracts with *engine manufacturers*, including conditions of delivery deadlines and restrictions on the *part suppliers* allowed to supply replacement parts. Engine manufacturers, in turn, subcontract *service sites* at various airport hubs to service engines as and when they require maintenance. Service sites are obliged to deliver repaired engines within a maximum of five days[2], and thus need to make sure that all pre-requisites of the repair are performed well before this deadline, including the delivery of any required parts. During maintenance at service sites, parts may need to be ordered from *part suppliers* and shipped through *logistics* agents. When a service site needs to order parts, it sends requests for bids to all its known part suppliers, each of which then sends their bids to supply parts. In general, a service site will accept the bid with the lowest price and earliest delivery date estimate, and will turn down bids from part suppliers whose delivery estimate would render its own deadline commitments impractical, as well as bids from part suppliers forbidden in the contract terms. After a bid is selected and the parts order is placed, part suppliers ship the new parts to the service site. Once the parts arrive, the service site resumes repairing the engine, and readies it in the designated aircraft, notifying the engine manufacturer that the repair is complete. A summary of the use cases involved in our simulation is shown in the use case diagram of Figure 1, including the monitoring that is discussed in Section 5. For simplicity, we have introduced a number of restrictions on our current model, but these can be relaxed with no

[2]The number of days for delivery can change between contracts, but in the running example used throughout the paper, we have chosen to set this number at five to facilitate understanding.

impact on the underlying implementation of our model. These restrictions are as follows:

1. Only one type of engine is considered.
2. There is one engine manufacturer per operator.
3. The base system is up and running, with working aircraft, engines and parts, with which the electronic contracting is to be integrated. This contrasts with other applications in which the electronic contracting is part of an envisioned new system.

2.3. Supply chains

Given the complexity of modern aircraft engines, their production and maintenance involves complex supply chains, with parts sometimes coming from a very limited range of suppliers. As a consequence, problems at the bottom of the supply chain (part suppliers) may easily cascade to the top (aircraft operators). In particular, part suppliers may experience delays in delivering parts to the service site, which in turn may prevent the service site from repairing an engine on time, and result in the engine manufacturer violating its contract with the aircraft operator. These delays may occur for a number of reasons: a part supplier may take longer than expected to fabricate a new part; the logistics agent may delay shipping; or the service site may not find a permitted part supplier to supply parts.

In this paper, we focus on the supply chain from the engine manufacturer down to the part suppliers, so the simulations of Section 7 do not deal with the aircraft operators. Instead we address the interactions (and normative constraints) between engine manufacturer, service site, part suppliers and logistics agents. In our simulations, we use the monitor of Section 5 to detect contract violations and generate explanations for the origin of these violations. The explanations thus generated guide the improvement of the contracts and the simulation of the application domain.

3. The CONTRACT architecture

The CONTRACT[3] framework and architecture [6] allow electronic contracting technologies to be integrated into applications. This provides several benefits, as follows.

- Explicit formulations of permissions, obligations and prohibitions on contract parties can be reasoned over and acted on by software agents to best meet business objectives.
- Verification of the system, with regard to contracts, enables parties to determine whether it is possible to meet their future obligations.
- Well-specified mechanisms are available to store, maintain the integrity of, and access, contract documents.

Since the work presented in this paper aims, in part, to understand and assess the applicability of the architecture, we ensure that each architectural component is at least minimally implemented as part of our overall system. In this section, we briefly describe the framework and architecture, in the context of the scenario described above.

3.1. Overall Structure

The CONTRACT *framework* is a conceptual model for specifying applications using electronic contracting. The *architecture* is an instantiation of the contract administration aspects of the framework: a set of service-oriented middleware and multi-agent design patterns to support administration of electronic contracts.

[3] We name the framework and architecture after the IST-CONTRACT project, funded by the European Commission, in which this work was undertaken.

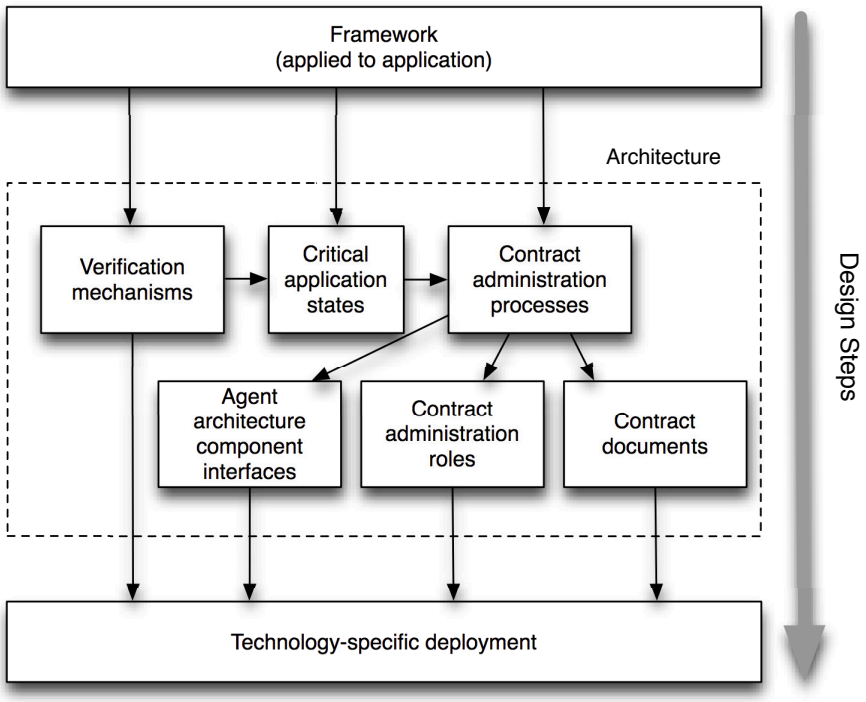


Figure 2: Overall structure of the CONTRACT architecture.

In Figure 2 we show the overall structure of the framework and architecture. As a whole, this can be seen as series of models and specifications, comprising a series of design steps for adapting application designs to utilise electronic contracts. The primary component is the framework, depicted at the top, which is the conceptual structure used to describe a contract-based system, including the contracts themselves and the agents to which they apply [3]. Each level in the figure provides support for the components below it. Arrows indicate where one model influences or provides input to another. In summary, given the CONTRACT framework as applied to an application (including a definition of the required business business roles), verification mechanisms validate the application model, generating the critical application states needed for the contract administration processes (such as monitoring). With these processes in place, the concrete individual agents (both business and administrative ones) are modelled, together with the contract documents, before finally arriving at a technology-specific deployment. In this sense, Figure 2 shows not only how each part of the CONTRACT architecture is related to each other, but the vertical axis in the figure also illustrates the sequence of activities (a methodology sketch) used to create one such application.

First, off-line verification mechanisms —built around BDD-based approaches to system verification— can check whether the contracts to be established obey particular properties, such as being achievable, given the possible states the world can reach [7]. Given this set of reachable states, together with the contracts themselves, we can determine which states are *critical* to observers during execution in order to detect whether inappropriate behaviour occurs. For example, a critical state of a contract-based system with regard to an obligation could indicate whether the obligation is fulfilled or fulfillable. When alerted to being in such a critical state, a contract party would know that their actions could have important effects on the contract outcome. More complex critical states (e.g. identifying whether an obligation is achieved, failed, or in danger of not being fulfilled), could be used to both monitor and explain the status of the contract, and to provide warnings to those monitoring the system. These *critical application states* provide part of

the specification for the events for which agent design artefacts (e.g. agent plans, environmental actions, etc.) need to be created. This specification is used in the creation of the different agents that comprise a CONTRACT instantiation, the roles that these agents play in an implemented system and the formal documents exchanged between the agents. At present, such critical states must be manually defined in order to be recognised, but we intend to investigate the automated identification of such critical states from a contract specification as future work. Preliminary work on identifying states that occur immediately before violation has been published as [8].

In the particular instantiation of CONTRACT for the aerospace domain of Section 2, we consider the deadlines for each order in the supply chain to be in critical states, so that if any deadline is missed, the agents in the system can start taking corrective action. For example, if a part supplier misses its deadline for delivering parts, the monitor can detect this and have the manager start corrective action before this cascades all the way to the airline, prompting a service site to procure parts from another supplier.

3.2. Contracts

The framework specification is used to determine suitable processes for administration of the electronic contracts through their lifetimes, including establishment, updating, termination, renewal, and so on. Such processes may also include observation of the system, so that contractual obligations can be enforced or otherwise effectively managed, and these processes depend on the critical states identified above. Once suitable application processes are identified, we can also specify the roles that agents play within them, the components that should be part of agents to allow them to manage their contracts, and the contract documents themselves.

More specifically, agreements between agents are formally described in electronic *contracts*, which document *obligations*, *permissions* and *prohibitions* (collectively *clauses*) on agents. Agents bound by contract clauses are said to be *contract parties*, and a contract specifies *contract roles*, which are fulfilled by contract parties, so that clauses apply to specific contract roles. The lifecycle of a contract may be broken down into five stages, as illustrated in Figure 3:

- *creation*, including the process of finding potential interaction partners and negotiating terms for a contract;
- *maintenance* and *update* of the formal representation of a contract document in a controlled repository;
- *fulfilment* of the contract clauses by the participants;
- *management*, consisting of overseeing the fulfilment of obligations by designated agents, and taking action when violations are detected; and
- *termination* or *renewal* of contracts when contracts are about to expire, or when their validity is violated.

Given that our aerospace aftermarket domain is focused on monitoring the supply chain rather than negotiation over contract terms, our instantiation neither considers nor elaborates processes for contract establishment, update and renewal.

3.3. Contract Parties

Contracts in our system are agreed upon by agents, which are assumed to be autonomous, pro-active, flexible (decision-making) and social. Agents engage in contract-directed interactions to fulfil the clauses specified in a contract. Contract interactions require a minimum of two agents fulfilling the role of participants. Some applications may require contract-related processes to have certain properties, for example that violations are acted on, or that the integrity of the contract documents is maintained. These requirements lead to obligations on (and the creation and use of) administrative parties, and contracts may document their required behaviour. We can roughly classify contract parties into two kinds.

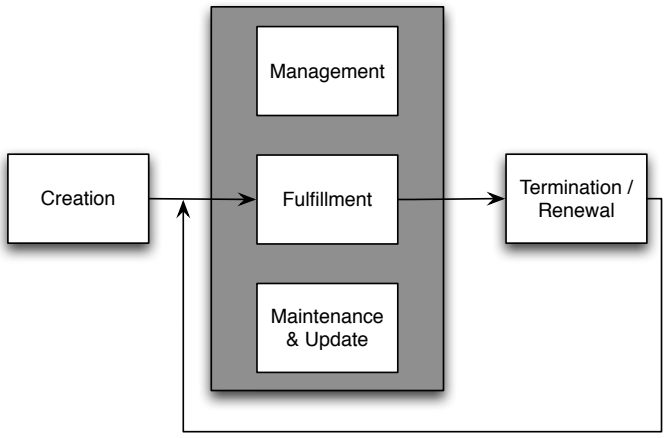


Figure 3: Life cycle of a contract.

Business Contract Parties Contracts are created for governing the actions and interactions of agents: the obligations on the business contract parties are largely concerned with the *business* of the application. In our use case example, the aircraft operator, engine manufacturer and part suppliers are business contract parties.

Administrative Contract Parties Agents are required to ensure that the contract is accessible, retains integrity and legitimacy, is monitored and enforced, and that it complies with other, similar administrative requirements in order to ensure the contract itself has force. The obligations on these agents relate to their administrative roles.

There may be any number agents playing any number of business roles. Moreover, the two administrative roles are always mandatory regardless of the domain, since there must be an agent responsible for providing trusted observations, and one responsible for signalling violations if a contract is to be enforced. The monitoring mechanism is a function that could be placed in either one of the administrative roles.

3.4. Enforcement

The CONTRACT framework includes two administrative contract party roles: *observer* and *manager*. An observer detects whether the system enters a critical state (success, violation, in danger of violation) with regard to a particular clause. In contrast, a manager reacts on the basis of observation, for example to inform a user of the problem, penalise a contract party in some way, and so on. Conceptually, these roles can be fulfilled by a single agent, but in practice, we have multiple observers because there are multiple things to observe and multiple points to observe from in a distributed system. We can also have multiple instances of the monitoring mechanism (detailed in Section 5), potentially including user-specific ones, because the results of monitoring can be used for different purposes. That is, a contract-wide monitor can be used to manage violations in accordance with the contract, while an agent-specific monitor just helps an agent know what to do next according to the contract. Moreover, the observer role can be implemented by observers that the electronic contract framework itself provides, and also by external observers specific to business parties.

Thus, there may be many agents performing the role of observer (or manager) primarily in order to allow the system to scale. If a system is small enough, and the electronic contract allows for it, there might be only one agent in the system fulfilling both roles of manager and observer. In our aerospace aftermarket system, the observers communicate with a monitor component that tracks the evolution of all norms in the system.

4. Representing Contracts

Electronic contracts are the central concept in the CONTRACT framework, thus making the representation of such contracts the vital element. In this section, therefore, we review our previous work on the specification of electronic contracts [2]. At its most basic, an electronic contract is composed of a set of *clauses*, each of which contains a set of *norms*. Norms are essentially rules that can be categorised as *permissions*, *obligations*, and *prohibitions*, which respectively specify what may be done, should be done, and should not be done. While agents typically comply with norms, they may decide not to do so if it serves their self-interest, or simply if they cannot do so. For example, an optimistic agent may over-commit on a manufacturing contract, agreeing to provide more goods than it is capable of manufacturing, after estimating that typically, large orders will not arrive simultaneously. If the latter does occur, it may be unable to meet one of its orders, thus violating a norm. In such scenarios, culpable agents are subject to sanctions (that may themselves be specified as *contrary-to-duty* obligations that come into force when other obligations are violated); in the above example, the manufacturer may be obliged to pay a fine if an order is not delivered on time.

We start by describing the abstract model of norms that underlies the monitoring mechanism used as part of the enforcement of the norms in Section 4.1. In this abstract model, norms are represented in first-order logic and provide the input format for the monitoring mechanism. With this model in mind, we review the natural language clauses from the aerospace aftercare scenario in Section 4.2. These clauses can be used as the starting point in the creation of the formal representation of electronic contracts. The formal representation of contract clauses can then be parsed by the monitoring component in order to extract the individual norms that are to be monitored. Notice that the individual first-order logic norms extracted from the formal contract are also the elements processed by individual agents to change their behaviour and comply with a contract [1].

4.1. Individual Norms

Philosophers and computer scientists have undertaken a vast amount of research into normative systems (e.g. [9, 10, 11]), most of which focus on deontic logic formalisations. While deontic logics provide formal models for reasoning about norms, their representations are somewhat abstract and do not readily lend themselves to implementation. To address this drawback, we have proposed a more readily implementable framework for representation and reasoning about norms [2]. While the framework makes no assumptions about the context in which the norms are specified, it aims to easily handle norms appearing within contracts. In [2], a norm $\mathcal{N}$ is a tuple:

$$\begin{aligned}\langle Type, \quad & ActivationCondition, \\ & NormCondition, \\ & ExpirationCondition, \quad Target\rangle\end{aligned}$$

where $Type$ identifies whether the norm is an obligation or permission, and the next three parameters are all predicates expressed in some logical language (which we leave unspecified). We assume prohibitions to be equivalent to obligations to ensure some state of affairs (the norm condition) does not hold, so for example $\langle PROHIBITION, AC, NC, EC, T\rangle$ is equivalent to $\langle OBLIGATION, AC, \neg NC, EC, T\rangle$. So, when the norm condition is true in an *obligation* the norm is interpreted as *not being* violated, whereas in a *prohibition* when the norm condition is true, the norm is interpreted as being violated. Here, the *ActivationCondition* specifies some state of the world in which $\mathcal{N}$ comes into force. Then, the *NormCondition* specifies some action or world state that the target of the norm must do/bring about (in the case of obligations), refrain from generating (in the case of prohibitions), or is permitted to do/bring about regardless of other norms (in the case of permissions), while the norm is in force. In turn, the *ExpirationCondition* specifies some world state in which $\mathcal{N}$ no longer has normative force. Finally, $\mathcal{N}$'s $Target$ set specifies which agents are responsible for, or allowed to make use of, the norm. For example, consider an obligation on

a service site SS to repair an aircraft engine E at some time T' that is within 7 days of receiving a request from an engine manufacturer EM at time T:

$$\begin{aligned}\langle \mathit{Obligation}, \quad & \mathit{received}(\mathit{engine\_repair\_request}, EM, E, T), \\ & \mathit{engine\_repaired}(E, T') \vee (T' \leq T + 7), \\ & \mathit{engine\_repaired}(E, T') \vee (T' > T + 7), \qquad SS\rangle\end{aligned}$$

It is important to note that the above norm can be considered to exist as an *abstract* norm that comes into force, and so only exists in its *instantiated* form when a specific repair request for a specific engine is received by a specific service site from a specific engine manufacturer. Thus, an abstract norm serves as a template for instantiated norms; when an abstract norm's activation condition evaluates to true, an instantiated norm is created and specialised to the situation (that is, the variables EM, E, T and SS are instantiated). Multiple instantiated norms may be created from a single abstract norm, and may exist at the same time. Once instantiated, a norm persists until its expiration condition is met, regardless of the state of its activation condition. In an abuse of notation, we assume that T' represents the current time, *i.e.* that the clauses with T' contain a conjunction with a function $\mathit{current\_time}(T')$, *e.g.* when we state $\mathit{engine\_repaired}(E, T') \vee (T' \leq T + 7)$, we assume the expression is $(\mathit{engine\_repaired}(E, T') \vee (T' \leq T + 7)) \wedge \mathit{current\_time}(T')$.

4.2. Natural Language Clauses

In this section we introduce the norms in natural language for the aerospace aftercare domain we introduced in Section 2. As we further describe the domain, we emphasise the clauses using tables summarising the structural components given in Section 4.1.

The scenarios we describe in this section assume a single engine manufacturer — *boing* — that has individual contracts with multiple airline operators, and maintains pools of available engines at a service site, *heathhedge*. Engine repair and servicing by *heathhedge* can require new parts that are available at the sites or that need to be ordered from part manufacturers *part manufacturer 1*, *part manufacturer 2* and *part manufacturer 3*. In what follows, we summarise each clause in the contract in tabular form to help the reader relate the natural language clause to its computational representation throughout the rest of the paper. Each contract clause contains five fields (as illustrated in Table 2). The first two fields identify *who* is bound to comply with the clause and what *type* of deontic *modality* is expressed by the clause. Normative stipulations are expressed through the *norm condition*, which specifies any possible actions or world states to which the clause modality refers. So if the clause is an *obligation* with a norm condition n, an agent subject to this clause must ensure that n. Moreover, *activation* and *expiration* conditions describe, respectively, the state in which a clause comes into force and the state in which a clause ceases to be in force.

The contract C we are interested in modelling is a short term contract between *boing* and *heathhedge*. C has dependencies on a long term aftercare contract between an airline operator, *hardjet*, and *boing*. In the long term aftercare contract:

- there is an obligation *ObGrounded* on *boing* to ensure that no *hardjet* plane is grounded for greater than a certain period of time awaiting a repaired or serviced engine; and
- *hardjet* states provenance restrictions on parts or modules for use in the engines made available by *boing*, encoded as permissions and prohibitions on *boing*.

The obligation *ObGrounded* in the long term contract implies an obligation, which we name *Ob*1*Repair*, on *heathhedge* in the short term contract C: *Ob*1*Repair* obliges *heathhedge* to repair a *hardjet* engine E for *boing* within seven days. Obligation *Ob*1*Repair* is implied by *ObGrounded* (although quite how this dependency is formalised is beyond the scope of this paper) in the sense that if no engine is available to *boing* for swapping into a *hardjet* plane P (for example, because all available engines do not conform to *hardjet*'s provenance restrictions on parts and modules for use in its planes), then the engine E on plane P must itself be removed by *boing* and

Ob1Repair	**Description**
Target	heathhedge
Type	OBLIGATION
Activation Condition	engine repair ordered
Norm Condition	repair engine within 7 days
Expiration Condition	engine repaired within 7 days

Table 2: Obligation on Service Site

Per1Order	**Description**
Target	heathhedge
Type	PERMISSION
Activation Condition	engine repair ordered
Norm Condition	order part from part manufacturer 1
Expiration Condition	part ordered from part manufacturer 1

Table 3: Permission on Service Site

given for service to *heathhedge* before being placed back on P. Thus, one would want that the service site *heathhedge*'s turn-around time period for repair, together with the time to remove and replace the engine by *boing*, is less than the maximum time that a plane can be grounded without *boing* incurring penalties for violation of C in the long term aftercare contract. Furthermore, the provenance restrictions on *boing* in the long term contract are inherited by *heathhedge*, and apply when *heathhedge* needs to order parts for repair of engines for *boing*.

More specifically, we have the following norms in our contract (shown as Tables 2 through 5).

- *Ob*1*Repair*: Contract C obliges *heathhedge* to repair engines for *boing* within seven days.
- *Per*1*Order*: Contract C permits *heathhedge* to source parts for engines for *boing*, from *part manufacturer 1*.
- *Per*2*Order*: Contract C permits *heathhedge* to source parts for engines for *boing*, from *part manufacturer 2*.
- *Pro*1*Order*: Contract C prohibits *heathhedge* to source parts for engines for *boing*, from *part manufacturer 3*.

We note that, following the semantics of prohibitions being negated obligations in our framework, the prohibition on Table 5 could equivalently be defined as an obligation not to order a part from manufacturer 3. The permissions in Tables 3 and 4 are used to encode the restrictions on the provenance of the parts (and so they cease to be active once parts have been ordered from an allowed supplier). Notice that although the instantiations of these permissions expire (once a particular part is delivered), the abstract permissions remain in the system so when new engines need to be repaired (and new parts ordered), these permissions are again instantiated.[4] Finally, in any implementation of a norm monitoring system, there must be an underlying assumption of the default normative modality for each action (*i.e.* , an action that is not mentioned by any norm must be assumed to be either permitted or prohibited); in our implementation, we consider the default deontic modality to be a permission. That is, unless explicitly prohibited, an action is permitted. Similarly, any obliged action is (implicitly) permitted. This type of permission is referred to as a *weak* permission in the literature [12]. Such weak permissions can be contrasted with the *strong* permissions of Tables 3 and 4. Here, the permissions identify states of affairs that cannot be prohibited. Nevertheless, in order to enable explanation, we occasionally explicitly specify a weak permission in order to instrument the monitoring explanations, as detailed in Section 6.

Three scenarios can then be modelled, each beginning with the following actions.

[4]We further detail the notion of norm instantiation in the explanation of contract monitoring in Section 5.

Per2Order	**Description**
Target	heathhedge
Type	PERMISSION
Activation Condition	engine repair ordered
Norm Condition	order part from part manufacturer 2
Expiration Condition	part ordered from part manufacturer 2

Table 4: Permission on Service Site

Pro1Order	**Description**
Target	heathhedge
Type	PROHIBITION
Activation Condition	engine repair ordered
Norm Condition	order part from part manufacturer 3
Expiration Condition	part ordered from part manufacturer 3

Table 5: Prohibition on Service Site

1. *hardjet* sends a request to *boing* for an engine for one of its planes. The engine itself is then removed for repair since no alternative engine is available.
2. *boing* orders a repair of the engine from *heathhedge*.

These two steps are then followed by one of three alternate sequences of events, and listed in the scenarios below. The different outcomes of each sequence of events is later used to illustrate the generation of explanations for violations in Section 6.

4.2.1. Scenario 1

Scenario 1 illustrates the successful fulfilment of the contract clauses.

3 *heathhedge* orders a part for the engine from *part manufacturer 1*.
4 *part manufacturer 1* informs *heathhedge* that the delivery time for the part is 3 days.
5 The delivery time is acceptable for *heathhedge* since it gives the service site enough time to complete repair of the engine within the obliged seven day period.
6 *part manufacturer 1* delivers the part to *heathhedge* in 3 days.
7 The engine is repaired and ready for *boing* 6 days after receipt of the order for repair from *boing*, and so *heathhedge* has fulfilled its obligation to repair the engine within 7 days.

4.2.2. Scenario 2

Scenario 2 illustrates violation of the obligation $Ob1Repair$.

3 *heathhedge* orders a part for the engine from *part manufacturer 1*.
4 *part manufacturer 1* informs *heathhedge* that the delivery time for the part is 5 days.
5 The delivery time is unacceptable for *heathhedge* since it does not give the service site enough time to complete repair of the engine within the obliged seven day period.
6 *heathhedge* then orders the part from *part manufacturer 2*.
7 *part manufacturer 2* informs *heathhedge* that the delivery time for the part is 3 days.
8 The delivery time is acceptable for *heathhedge*.
9 However because of delays in transport, the part is received from *part manufacturer 2* after 5 days.
10 Because of the delay in receipt of the part, the engine is repaired and ready for *boing* 8 days after receipt of the order for repair from *boing*, and so *heathhedge* has violated its obligation to repair the engine within 7 days.

4.2.3. Scenario 3

Scenario 3 illustrates violation of the prohibition $Pro1Order$.

3 *heathhedge* orders a part for the engine from *part manufacturer 1*.
4 *part manufacturer 1* informs *heathhedge* that the delivery time for the part is 5 days.
5 The delivery time is unacceptable for *heathhedge*.
6 *heathhedge* then orders the part from *part manufacturer 2*.
7 *part manufacturer 2* informs *heathhedge* that the requested part is out of stock.
8 *heathhedge* orders the part from *part manufacturer 3*.
9 *part manufacturer 3* informs *heathhedge* that the delivery time for the part is 3 days.
10 The delivery time is acceptable for *heathhedge*.
11 *part manufacturer 3* delivers the part to *heathhedge* in 3 days.
12 The engine is repaired and ready for *boing* 6 days after receipt of the order for repair from *boing*, and so *heathhedge* has fulfilled its obligation to repair the engine within 7 days. However, it has only been able to do so by violating a prohibition to order parts from *part manufacturer 3*.

5. Monitoring contracts

In this section, we build on the representation of Section 4, and review the CONTRACT monitoring framework [13, 3]. Now, the goal of the monitoring framework is to be able to identify the status of a norm at any point in time. Typically, this status indicates whether a norm is abstract or instantiated, as well as whether it has been violated or expired. The ability to determine and reason about the status of a norm in this way is useful not only to agents interacting with each other within the system, but also for activation of other norms. For example, a sanction may take the form of a contrary-to-duty norm obliging an agent to pay a penalty, which may come into effect if the status of some other norm is *violated*.

5.1. Monitoring for Compliance

In the CONTRACT monitoring framework [3], *monitors* receive observations from observers (Section 3.4) that are explicitly entrusted by all contract parties to accurately report on the state of the world. These observations are then processed, together with *Augmented Transition Network* (ATN) [14] representations of norms, to determine their status. Once the status of a norm is ascertained through the monitoring process, the decision of what actions are to be taken is delegated to manager agents, which might apply sanctions for violations, and rewards for fulfilment, as appropriate. This flow of information from the interacting agents through the monitor and to managers is illustrated in Figure 4. Here, the use of trusted observers ensures some degree of certainty that a norm will be reported as violated if and only if it has *actually* been violated, and so some assurance that sanctions will only be applied as and when appropriate. Such assurances are important in order to encourage deployment of agents in electronic contracting environments.

For example, consider an obligation on a service site to pay for engine parts within a certain time after receipt of these parts. Recognition that the payment has been made may be based on an observer reporting that the monies for payment have been deposited in the bank account of the part supplier, where the observer may be the bank itself, and where the bank has been explicitly entrusted by the service site and part supplier (in the relevant supply contract) to report that the monies have been paid if and only if they have in actuality been paid. Thus, if no such report of payment is received by a monitor, then there is some degree of certainty that the monitor's reporting that the obligation has been violated is accurate.[5]

[5]Contrast this with either the service site or part supplier reporting that the monies have been paid, where it may be in the interest of these parties to mis-report the truth of the matter.

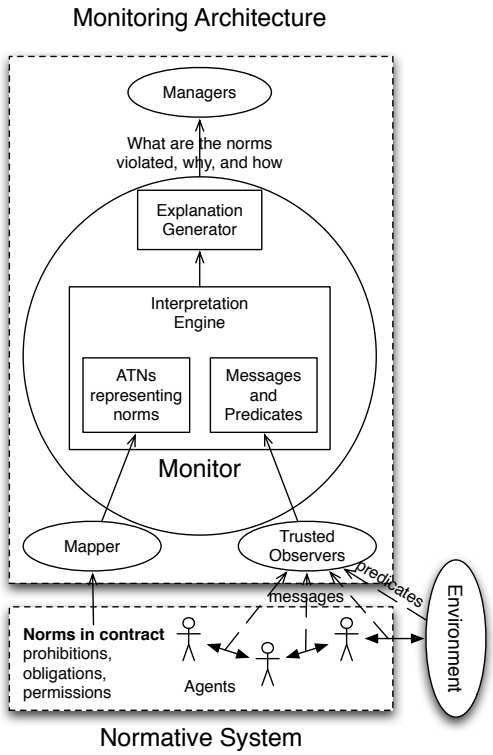


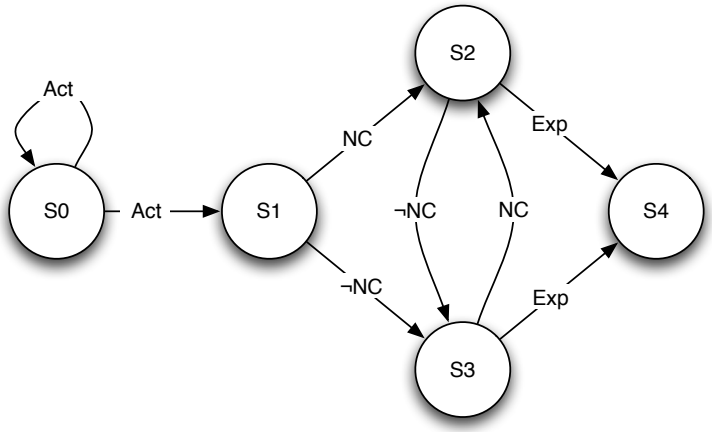
Figure 4: Monitoring architecture.

Observations relayed by trusted observers to monitors may either be messages observed as having been sent to and from contract parties (for example, a message received by a service site requesting repair of an engine), or predicate logic descriptions of properties holding in the world (for example, that an engine has been repaired, or that an action has occurred). In either case, these observations are processed, together with ATNs, to determine the status of the represented norms.

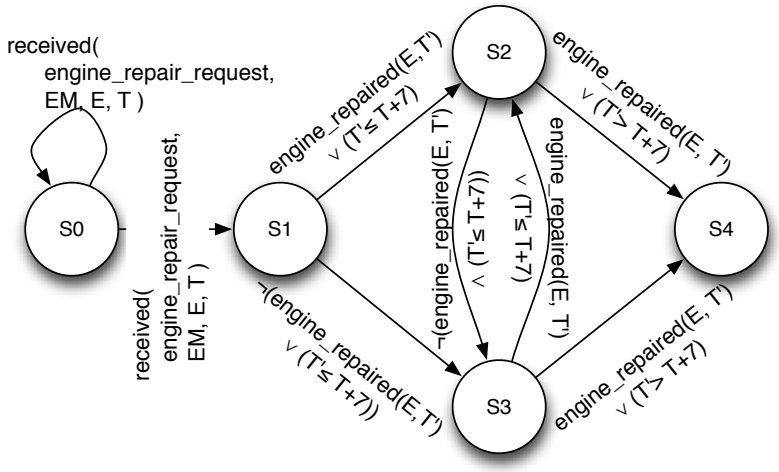
These ATNs are essentially 5-node directed graphs in which each node represents a distinct state of the same norm. Based on messages received from observers describing the states of interest specified by the norm's components, *ActivationCondition*, *NormCondition* and *ExpirationCondition*, the monitor matches the messages with the labels of the ATN's arcs that describe the corresponding states of interest, so as to transition the ATN from one node to the next. Norm ATNs contain five states, together with transitions modelling the evolution of a norm as each of its conditions are detected as true by the monitor. This is described below, and illustrated in Figure 5a, which shows a generic ATN representation of a norm.

When the monitor is initialised, all monitored norms are converted into ATNs; abstract norm ATNs are always in state $S0$, with node $S0$ indicating that a norm ATN is in its abstract form. If the monitor receives observations indicating that the activation condition holds, then the abstract ATN is instantiated. Note that state $S0$ of the template ATN of Figure 5a contains one transition to $S1$ and an additional transition back to $S0$. The former transition to $S1$ represents the generation of an instantiated copy of the abstract ATN; instantiated in the sense that the variables in the arc labels are instantiated based on the unification of the observation with the activation condition. This instantiated ATN is transitioned across the arc labelled by the activation condition, to $S1$, in which case the norm is said to be activated. The transition to $S0$, represents the preservation of the abstract ATN, for possible further instantiations. This lifecycle is a linear sequence of different states contrarily to abstract norm ATN representation that includes alternative transitions.

Thus, in our example obligation, if an observer relays a message received by a specific service



(a) ATN template used to represent norms. *Act* corresponds to *ActivationCondition*, *NC* corresponds to *NormCondition*, and *Exp* corresponds to *ExpirationCondition*



(b) Abstract ATN for norm example.

Figure 5: Norm ATNs

site from an engine manufacturer requesting repair of a specific engine, then the ATN for the obligation is transitioned to $S1$. If, subsequently, the monitor receives observations indicating that the normative condition holds, then the ATN is transitioned to $S2$, otherwise it transitions to $S3$. $S2$ thus represents the non violated state of an obligation, or the *made use of* state of a permission, and $S3$ the violation state of an obligation, or the *not made use of* state of a permission.

In the example obligation, the normative condition can be expressed as $engine\_repaired(E, T') \vee (T' \leq T + 7)$ so, if at the time of instantiation (T), either of these disjuncts is observed as holding (if the engine is repaired at time T, or if T' is within the seven day time window), then the ATN transitions from the instantiated state $S1$, to the non-violated state $S2$. If the normative condition does not hold; that is, if neither disjunct is observed as holding (the engine is not repaired and the current time is greater than seven days after activation), then the ATN transitions to the violated state $S3$. Of course, practically speaking, the latter transition will never take place for our example norm, as the time of instantiation T will always be less than $T+7$ days. In general, however, norms may transition immediately from the instantiated $S1$ to the violated $S3$. Consider for example, a norm obliging that some file be made available for download after a specific date. If, when

instantiated on the date in question, the file is not available for download, then the corresponding ATN will immediately transition to $S3$. Notice also that, in general, norms may toggle between $S2$ and $S3$ (as in the case of the download obligation when the file may alternately be available / not available for download). Finally, if at any time point T', the monitor receives observations indicating that norm expiration holds (either $engine\_repaired(E, T')$ or $T' > T + 7$ is observed as holding) then the ATN transitions to the expiration state $S4$. Since this is effectively a terminal state, the ATN is no longer processed. However, notice, that before transitioning to the expired state, it must be first be checked as to whether the ATN should be first be transitioned from the non-violated to the violated state ($S2$ to $S3$). Consider that the ATN is in the non-violated state $S2$, and the current time T' is greater than $T + 7$, *and* the engine is not repaired. Then, the ATN transitions to $S3$, the violated state, prior to then subsequently transitioning to the expired $S4$ state. This ensures that the norm is detected as having been violated prior to expiration. The abstract ATN corresponding to the norm from Section 4.1 is illustrated in Figure 5b.

More concretely, recalling the sequence of events described in Section 4.2 when an order is received by the engine manufacturer, a message $received(engine\_repair\_request, boing, engine1, 10)$ arrives at the service site (and is perceived by the observer). This message will unify the activation condition of the norm in Table 2 (as well as the activation conditions of the norms in Tables 3 through 5), which will cause an instantiation of each one of these norms to be created. Similarly, since these conditions are also the labels on the arc $S0 \rightarrow S1$ in their corresponding ATNs, these new instantiations will transition to $S1$. For example, the ATN of Figure 5b will generate an instantiated ATN with its variables EM, E and T bound, respectively, to *boing*, *engine*1 and 10. At this point, the monitor will be keeping track of four abstract ATNs (the ones originally in the system), plus four instantiated ATNs with bound variables. Notice that if more engine repair orders come in, new instantiations of each ATN will be created, allowing the monitoring mechanism to track each new instantiation separately. As further messages indicating the passage of time or the status of the engine repair arrive, further transitions will occur in the instantiated ATNs. In our example, in subsequent points in time messages indicating the current time will be received, transitioning the ATN either to $S2$ (if the engine is repaired), or to $S3$ (if the engine is not yet repaired). One possible development is for a message indicating $engine\_repaired(engine1, 11)$ to be received, the instantiated ATN will transition from $S2$ to $S4$, causing the norm to be fulfilled. Alternatively, if a message $current\_time(18)$ is received before the engine is repaired, this ATN will transition to $S4$ from $S3$ and indicate a violation.

This ATN-based mechanism allows the monitor to avoid the state-space explosion that would occur if a single giant ATN was to be created for an entire contract. Moreover, by modelling ATN instantiations as separate entities from the generic norms, it avoids the need to know at deployment time all possible instantiations of each norm.

5.2. Monitoring algorithm

Now that we have described the ATN mechanism that forms the core of the CONTRACT monitor component, we proceed to describing the monitoring mechanism. The monitor itself fulfils a number of functions. First, it acts as a store for all ATNs within the system. Second, it is responsible for routing messages to the appropriate ATNs. Third, it must send out notifications to the appropriate agents when a norm is instantiated, violated, fulfilled or expires (that is, when a norm's status changes in a specific way).

From a high-level point of view, the monitor maintains a repository of all norm ATNs (i.e. both abstract and instantiated ones). Whenever new messages arrive, the monitor tries to match them with the transition arcs leaving the current state of each ATN. As ATNs are further instantiated, new ATNs may be added to the monitor's repository, and as ATNs transition to the expired state, the monitor removes them from its repository. Thus, the monitor is able to add, remove, and query ATNs.

As we have seen, message routing requires sending any received messages to the appropriate ATNs, causing them to transition between states as described above. Checking whether an ATN can transition is performed using unification, and represented in the monitoring procedure of Algorithm 1, as the *satisfied* function of Lines 7 and 13. The monitor must subscribe to some observers

based on the events occurring in the labels of instantiated norm ATNs, which can be implemented as an ECA (Event-Condition-Action) rule mechanism. The monitor is also responsible for passing one other type of message to ATNs, namely time messages, which are sent to all ATNs at regular intervals, indicating the passing of time. ATNs process these messages differently, transitioning only when a timeout condition on the edge successfully processes the time message.

For example, in the ATN of Figure 5b, these messages would be used to process the temporal conditions ($T' \leq T + 7$) and ($T' > T + 7$). These interactions are illustrated in Figure 6, but since the monitoring algorithm handles each norm ATN independently (both abstract and instantiated), it is easy to distribute it across multiple computers, each handling a specified number of ATNs.

We can conceptually view the monitoring process as following the algorithm described in Algorithm 1, in which the monitor repeatedly dispatches messages to ATNs. If an abstract ATN's activation condition is satisfied (line 7) then the ATN is instantiated (lines 8 to 10) and added to the set of instantiated ATNs. Messages are then dispatched to all instantiated ATNs in the system. If the message satisfies the requirements on an arc leaving the current node (line 12), a transition takes place (line 14), and agents are notified of the transition as appropriate (lines 15 to 18). Finally, if the transition leads to an expired state, the instantiated norm is removed from the instantiated norm set (line 19). We assume that timestep messages are treated as any other message.

It is important to note that the monitoring algorithm is not overly complex, since the processing of a single message has polynomial complexity on the number of ATNs in the monitor's repository, with the most complex part of the algorithm being the unification performed for each message to each arc departing from the current node of each ATN. Nevertheless, it is possible for a system to have a huge number of norms, in which case our algorithm is easily distributable, since one can imagine multiple monitors each containing a subset of the original abstract ATNs.

6. Explaining contract violations

Now that we have described the way in which the CONTRACT monitoring mechanism detects the current state of individual norms, we can proceed to describe how this monitoring mechanism can be extended to provide richer explanations as to the causes of particular norm states. We start by defining a basic explanation mechanism in Section 6.1, and then proceed to describe how more detailed explanations can be generated based on overlapping conditions of multiple norms in Section 6.2. Finally, we show how explanations are generated in the context of the aerospace domain, illustrating it with two monitoring scenarios and how the explanations help refine the system in Section 6.3.

6.1. Monitoring explanations

We consider first how a simple explanation could be formulated by the monitor. As we have seen, the monitor detects that a norm is violated when the ATN associated with it transitions to a

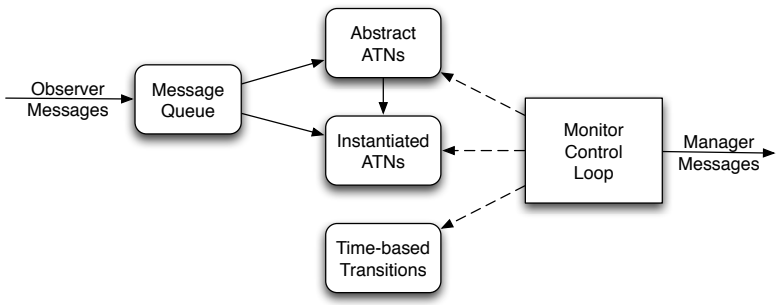


Figure 6: Overview of the monitor control loop

Algorithm 1 Monitor control loop

Require: Message queue Q_{msg}
Require: Message store M_{St}
Require: Set of abstract norm ATNs $\mathcal{X}_{Abs}$
Require: Set of instantiated norm ATNs $\mathcal{X}_{Inst}$

```
1: while Monitor is active do
2:   while Q_msg is not empty do
3:     Retrieve Msg from head of Q_msg
4:     Add Msg to M_St{First, deal with messages}
5:     for all Abstract norm ATN A in X_Abs do
6:       for all Arcs Act in A do
7:         if satisfied(M_St,Act) then
8:           create I, an instantiated version of the ATN I of A
9:           add I to X_Inst
10:          move I to state S1
11:    for all Instantiated norm ATN I in X_Inst, with current state C, do
12:      for all Arcs a leaving C, going to T, do
13:        if satisfied(M_St,a) then
14:          move I to state T
15:          if T is S3 then
16:            Notify manager of violation
17:          else if T is S4 then
18:            Notify manager of expiration
19:            remove I from X_Inst
```

particular state, S3. Since every ATN is associated with a norm, an initial approach to explaining a contract violation is to state which norm in the contract has been violated.

However, an explanation consisting solely of the norm instance that was violated is often inadequate for diagnosing any deeper causes for a violation, as it expresses only the *fact* and not the *causes* of the violation. A refinement of the explanation would be to include the ATN transition that led to the violating state; that is, the observation of a message or state by which the monitor detected that the obligation or prohibition had not been fulfilled. This refined explanation, which we refer to as a *single-ATN explanation* below, allows a designer to pinpoint the *immediate cause* of a norm being violated. However, it does not help to determine the overall circumstances relevant to violation, for which we would also need to explain the *indirect causes* leading up to the violation.

We therefore need to consider ways to recognise other observations of relevance to a violation, thus building up a richer picture of the surrounding circumstances. Notice that, as shown in the monitoring algorithm of Section 5.2, each observation arriving at the monitor is supplied to *every* ATN being monitored, allowing the monitor to detect if a single observation causes more than one ATN to activate, expire, or transition between violation and non-violation. If multiple norms are affected (that is, if multiple ATNs are activated, violated, unviolated or expired) by the *same* observation, then violations or fulfilments of those norms have a *common* cause.

Consequently, we can enhance our explanations of one violation by reporting the observations that caused transitions in other ATNs with the same arc labels. For example, an engine requiring repair is an activation condition for both returning the engine to working order, and for ordering parts necessary to make repairs, with each norm being monitored separately. If the engine is not repaired in time, violating the first obligation, then observations relating to the ordering of parts may help explain why this was the case. An explanation using this enhancement can thus take the form of a set of single-ATN explanations *chained* together by the observations common to each ATN. It is important to note here that we assume a completely observable environment, and thus the explanations will be deterministic, and based on the ATNs available to the monitor.

6.2. Refining Explanations

The primary idea of our monitor is to detect violations of norms stated in a contract agreed between the contract parties, such as engine manufacturers and site services. Now, the obser-

vations that the monitor receives (through subscribing to trusted observers) are just those that indicate these norms are activated, violated or not, or expired. However, as described above, our explanations of violations of the norms comprise the relevant observations received by the monitor. Therefore, the explanations that the monitor can give are limited to observations directly relevant to determining the fulfilment of the contract clauses. This is often inadequate for good explanations. For example, a contract may place no obligations, prohibitions or permissions on how engine parts are supplied, and so the monitor will receive no observations regarding this supply, but it may be exactly this factor that has led to the violation of the obligation to repair an engine (that there is something wrong with the part supply chain).

In our approach, we aim to improve explanations generated by the same monitoring machinery described above, by adding ATNs representing norms not present in the contract but helpful to monitor, purely to build better explanations: *explanation ATNs*. So, for the example above, we add explanation ATNs for monitoring the part supply chain to ensure that the monitor had records of observations relating to part supply and determine where part supply problems and repair violations had a common cause.

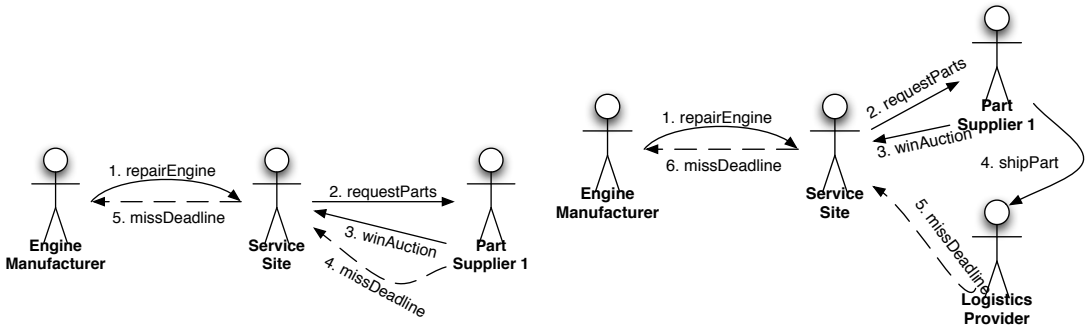
Such an explanation ATN has no qualitative difference from an ATN representing a contract clause. Every explanation ATN works similarly to a *permission* (rather than obligation or prohibition) in terms of detecting a violation, because it does not state what should happen, but what could happen; for example, parts *could* be delivered by this supplier. There is, therefore, no notion of an explanation ATN itself being violated, since permissions cannot be violated. Unlike permissions though, explanation ATNs have no normative power, as they are only used in case an actual norm-derived ATN detects a violation. Below, we consider how this explanation mechanism is used in our aerospace case study.

6.3. Monitoring Examples

In order to show how the explanations generated by our monitor are used to detect both the origin of violations and potential problems in the aerospace aftercare simulation, we have designed two typical scenarios that demonstrate the use of the monitor explanations in improving the norm-regulated multiagent system [15]. As we have seen in Section 2, violations at the bottom of a complex supply chain can easily cascade to the top. Thus, in both scenarios that follow an obligation on an engine manufacturer to make available engines to an airline operator is fulfilled or violated based on whether the sub-contracted service site repairs the engine for the engine manufacturer within a given time period. Furthermore, provenance restrictions on parts, dictated by the airline operator to the engine manufacturer, translate to permissions and prohibitions on the service site's ordering of parts from part suppliers.

In the scenarios that follow, we describe the monitoring of a contract between the service site and engine manufacturer. These scenarios illustrate how the additional explanation ATNs allow the inference of a chain of events that lead to a violation. The norms encoded include the obligation on the service site to repair engines in a given time frame, as specified in Section 4.1 (its ATN representation is the example in Section 5). We identify this obligation as *Ob1* in the examples below. In addition, the contract includes the following clauses, all imposed on the *service site* agent.

- A permission to order parts from part supplier 1, *Per1*, represented by an ATN as in Figure 5a, where:
  - Activation condition (Act) = engine repair request received at time T (the same activation condition as *Ob1*)
  - Normative condition (NC) = order parts from part supplier 1 (so ¬NC = not order part from supplier 1)
  - Expiration condition (Exp) = part ordered from part supplier 1
- A permission to order parts from part supplier 2, *Per2*, represented in the same way as *Per1* except that part supplier 2 replaces part supplier 1 in its specification.



(a) Original scenario representation.

(b) Expanded representation.

Figure 7: Tracing Violations

- A prohibition on the service site from ordering parts from part supplier 3 (expressed as an obligation not to order parts from part supplier 3), *Pro1*, represented by an ATN with:
  - Activation condition (Act) = engine repair request received at time T (the same activation condition as the obligation)
  - Normative condition (NC) = not order parts from part supplier 3 ($\neg$NC = order part from supplier 3)
  - Expiration condition (Exp) = false (the norm never expires).

6.3.1. Scenario 1: Tracing Violations

In the first scenario considered (depicted in Figure 7a), the following events are simulated.

1. The service site receives a repair request. This event activates *Ob1*, so the corresponding ATN transitions to $S1$ and then immediately to the non-violated state $S2$, given that the 7 days have not elapsed. Similarly, *Per1* and *Per2* are activated and transitions to state $S3$ (indicating that the permission has not been taken advantage of).
2. In a subsequent bidding phase, the service site makes a request for a required part to *part supplier 1*, which in turn responds by indicating that it has the part in stock and the part can be delivered in some given time.
3. *Part supplier 1*'s expected delivery time is acceptable to *service site* (in the sense that it leaves enough time for service site to fulfil its time-constrained obligation to the engine manufacturer). The *service site* then makes use of *Per1* to order the part from *part supplier 1*, and the corresponding ATN (that has also been activated by receipt of the repair request and so is already in $S1$) transitions to $S2$ and then $S4$ (the permission has been made use of and then expires). Notice that in Figure 7a, the bidding phase and subsequent use of the permission is indicated by the *winAuction* arrow.
4. On the 8th day after activation, no message informing the engine manufacturer of delivery of the engine has been observed. *Ob1*'s ATN transitions to the violation state $S3$ and then immediately expires, transitioning to $S4$.

Following this scenario, the monitor will report that *Ob1* has been violated. However, if it only monitors the four contractual norms specified above (*Ob1*, *Per1*, *Per2*, and *Pro1*), the explanation that the monitor can provide in this scenario consists only of the observations that:

1. a repair request was received (activation condition of *Ob1* and *Per1*);
2. a part was ordered from *part supplier 1* (normative and expiration condition of *Per1*); and
3. the engine was not repaired within the time limit (negation of normative condition and expiration condition of *Ob1*).

As discussed in Section 6.2, this explanation is inadequate. To properly understand what has occurred, we would want to know more; for example, what availability information the part supplier provided and thus whether the service site was justified in ordering a part from that supplier on the basis of that information. To do this, we add explanation ATNs (ATNs for possible behaviour not explicitly permitted in the contract) regarding the bidding process. Figure 8 shows the explanation that results from chaining explanation ATNs once a violation occurs. Each box depicts a single-ATN explanation, with the majority being explanation ATNs causally connected to *Per1*. Bold arrows represent the explanation-chain connection between ATNs, and the common events that allow these connections to be established are highlighted in each ATN.

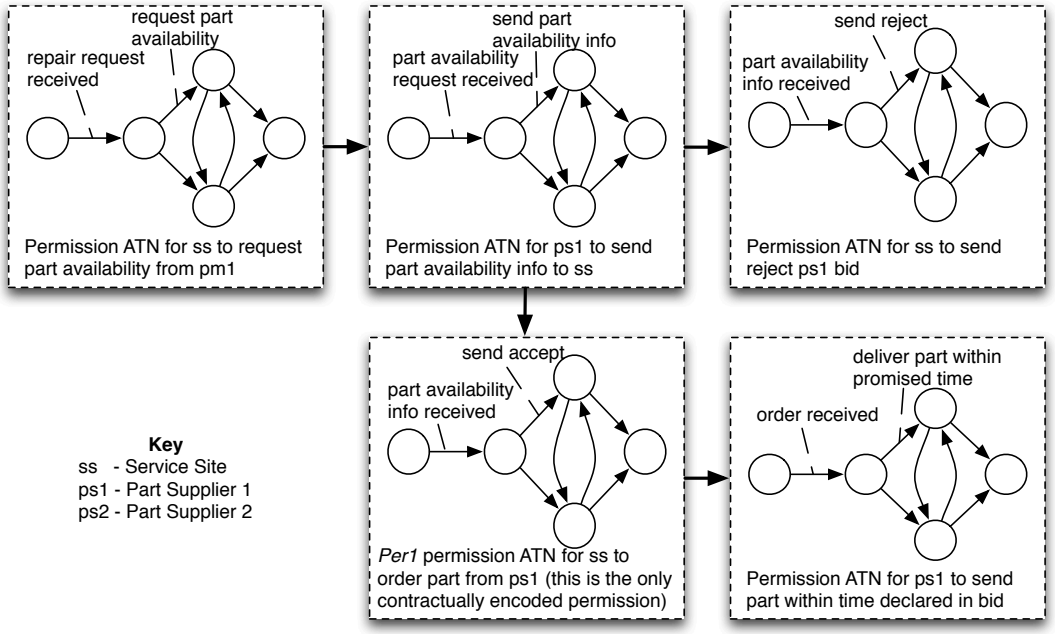
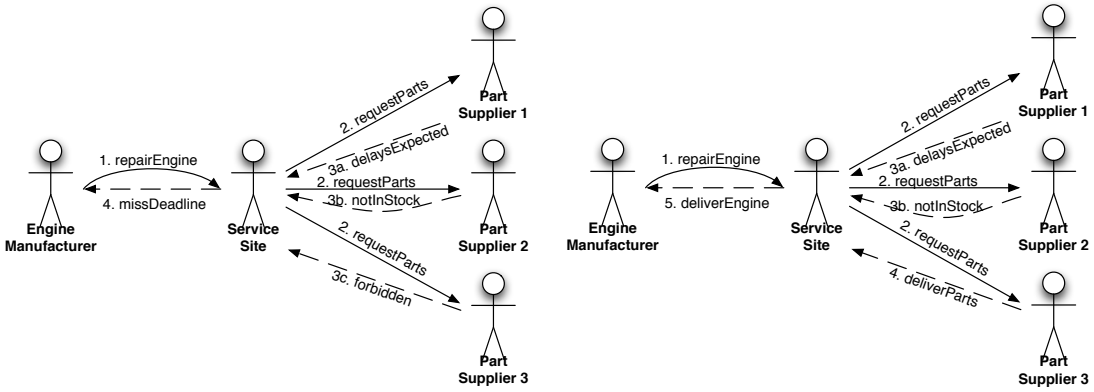


Figure 8: Chained explanation ATNs for Scenario 1.

To further illustrate the point, the user receiving such an explanation may still not be satisfied that enough information has been provided to pinpoint the events leading to violation. Therefore, they may add further explanation ATNs regarding the logistics provider used to deliver the parts from part supplier to the service site. Such a scenario is depicted in Figure 7b, which shows the sequence of events we are interested in capturing whenever a violation occurs. This *drilling down* to include inter-connected, relevant details by adding to that part of the system that is monitored can continue indefinitely, but the key aspect is that the full monitoring process only occurs for events deemed potentially relevant to the contractual norms by the user; that is, we are not trying to monitor and connect all events in the system to the violation of contracts.

6.3.2. Scenario 2: Relaxing Prohibitions

In the second scenario illustrated in Figure 9a, the *service site* again violates its obligation to repair the engine within 7 days, but the explanation of the violation above indicates that this is because both the permitted part suppliers *part supplier 1* and *part supplier 2* are not able to provide successful bids. *Part supplier 1* cannot supply the part in time, and *part supplier 2* does not have the part in stock. *Service site* is prohibited from sourcing parts from the only other available part supplier *part supplier 3*, and so violates its obligation to the engine manufacturer. After analysing the explanation, a designer may conclude that the prohibition *Pro1* should be relaxed in case no other part suppliers are available. Thus, *part supplier 3* can be sourced in such exceptional circumstances so that obligation deadlines can be met, as illustrated in Figure 9b.



(a) Original scenario leading to failure.

(b) Relaxed restrictions lead to success.

Figure 9: Relaxing prohibitions

The above illustrates how sophisticated monitoring and explanation of violations feed into a more general methodology for refining contractual specifications. The approach is particularly applicable in complex scenarios in which the reasons why high level goals (such as obligations to have minimum numbers of serviced engines readily available for airline operators) cannot be realised are not immediately apparent. Such scenarios occur in simulations that aim to find an optimal compromise between utility and norm compliance[6], which also allow parameters affecting norms to be changed (for example, adding or relaxing prohibitions) in order to test various contractual specifications, thus arriving at an optimal specification that maximises the chances of realising the high level goals.

7. Implementation and Experiments

To allow the evaluation of the contracting framework separately from the specific use case, we have divided the system into two main parts: the aftercare simulation and the contract-related functionality. The simulation part of our system implements a small subset of the aftercare scenario, including the scheduling of flights, update of engine usage information and engine maintenance operations. The contracting part of our system focuses on the communication and monitoring of requests by contract partners to fulfil their obligations, and taking action when violations of these obligations are detected. Our experiments consisted of running the system with a contract containing the clauses described in Section 4.2, in an environment containing both administrative and business roles. We note that, although the work of LostWax on aerospace aftercare simulation involves hundreds of agents aiming at identifying problems in a large supply chain, our objective here is different. The objective of our scenario and simulation was to demonstrate key characteristics of the contracting framework:

- a chain of events leading to fulfilment;
- a chain of events leading to violation;
- detection of violation and fulfilment by the monitoring mechanism;
- explanation of a detected violation;
- contract clauses with deadlines; and

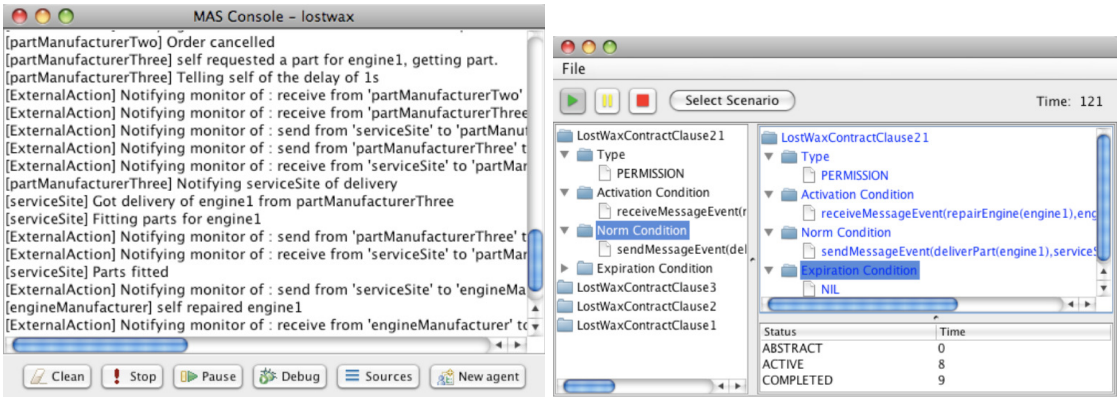
[6]For example, the simulations conducted in the Aerogility software, from which the scenario of this paper has been inspired.

- multiple-party contracts interacting in an environment.

The aerospace scenario we developed with LostWax exhibited all of these characteristics, and was thus used as a testbed for the contracting framework. This number of agents is representative of the number of parties expected to participate in one contract of the type described in the paper (namely, with one engine manufacturer, one service site, and three part manufacturers, besides the administrative roles). The experiments consisted of submitting dozens of requests for maintenance, in series, while subjecting the part suppliers with various levels of delay, which induced to various chains of events leading to fulfilment and violation. While some of these chains of events were reported in Section 6, in this section we focus on the description of the implementation of our prototype.

7.1. Agent development environment

As we have seen, the architecture developed for the CONTRACT framework is driven by events in the environment that are associated with certain conditions specified in contract clauses. These events drive complying agents to adopt plans to fulfil their obligations. Such a mechanism lends itself very well to implementation through reactive-planning BDI agents, such as PRS [16], and AgentSpeak(L) [17]. In consequence, our CONTRACT demonstrator was implemented using Jason [18], which is a Java-based AgentSpeak(L) [17] interpreter. More specifically, AgentSpeak(L) [17] is an agent language, as well as an abstract interpreter for the language, and follows the *beliefs, desires and intentions* (BDI) model of practical reasoning [19]. In simple terms, a BDI agent tries to realise the *desires* it *believes* are possible by committing to carrying out certain courses of action through *intentions*; in AgentSpeak(L), this is simplified in that an agent chooses plans of action that are considered possible by the agent's beliefs, making the notion of desires implicit in the plan representation. The language of AgentSpeak(L) allows the definition of *reactive procedural plans*, so that plans are defined in terms of events to which an agent should react by executing a sequence of steps (*i.e.* a procedure). Plan execution is further constrained by the context in which these plans are relevant.



(a) Simulation interface.

(b) Monitor interface.

Figure 10: Prototype graphical user interface.

A graphical user interface allows users to explore the norms and see the violation or fulfilment states reported by the monitor during run-time, and screenshots of this are shown in Figure 10. The screenshot of Figure 10a shows a log of the actions taken by the agents as well as the messages exchanged between them, while that of Figure 10b shows the formalised contract (top left), instantiated norms (top right) and the current monitored status of an instantiated norm (bottom right).

Since our implementation is based on an AgentSpeak interpreter, we briefly review its language

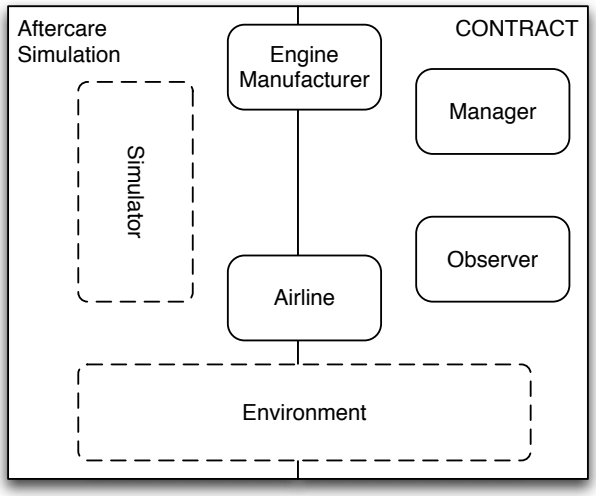


Figure 11: Main parts of the system.

and semantics in what follows.[7] The control cycle of an AgentSpeak(L) interpreter adopts plans in reaction to events in the environment and executes their steps. If the step is an action it is executed, while if the step is a goal, a new plan for the goal is added into the intention structure. Failures may take place either in the execution of actions, or during the processing of subplans. When such a failure takes place, the plan that is currently being processed also fails. Thus, if a plan selected for the achievement of a given goal fails, the default behaviour of an AgentSpeak(L) agent is to conclude that the goal that caused the plan to be adopted is not achievable. In order to better understand the relationship between the control cycle and the plan library, it is necessary to introduce the notation of AgentSpeak(L) plans. Events on an agent's data structures that can trigger the adoption of plans consist of additions and deletions of goals and beliefs, and are represented by the plus (+) and minus (−) sign respectively. Goals are distinguished into *test goals* and *achievement goals*, denoted by a preceding question mark (?), or an exclamation mark (!), respectively. For example, the addition of a goal to achieve g is represented by $+!g$. Belief additions and deletions arise as the agent perceives the environment, and are therefore outside its control, while goal additions and deletions only arise as part of the execution of an agent's plans. Plans in AgentSpeak(L) are represented by a header comprising a triggering condition and a context, as well as a body describing the steps the agent takes when a plan is selected for execution. If e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \& \ldots \& b_m \leftarrow h_1; \ldots; h_n.$ is a plan.

Below, we describe the implementation of the aerospace aftermarket scenario by detailing the descriptions of the various roles and their corresponding agents in terms of the AgentSpeak(L) plans used by these agents in fulfilling their goals.

7.2. Roles and Agents

We have developed agents, and their corresponding behaviours, to fulfil goals in both the simulation and the contract-related parts of the system, as illustrated in Figure 11. The administrative roles play no part in the simulation of the aftercare services; since these roles are in charge of maintaining the contract system (the infrastructure), they are not really part of the aftercare simulation, and simply support it. By contrast, the manufacturer and the operator have goals that span both areas, since their goals relate to the simulation as well as the fulfilment of contrac-

[7] For a full description of the language and its semantics, refer to d'Inverno *et al.* [20].

tual obligations. These agents and roles are summarised in Table 6, and further detailed in the following sections.

Agent/Role	Goals
Operator	Perform flights according to schedule Notify engine manufacturer of unscheduled events Schedule maintenance ahead of time
Manufacturer	Perform scheduled maintenance according to deadlines Perform unscheduled maintenance as soon as possible
Manufacturer/ Operator	Maintain the observer informed of the communication between themselves
Service Site	Perform maintenance operations whenever the manufacturer requests
Part Supplier	Respond to bids for parts with an accurate estimation of cost and delivery dates Deliver parts on time, once awarded a contract
Observer	Notify the manager when violations occur Monitor maintenance requests from the airline operator Monitor maintenance actions from the engine manufacturer
Manager	Receive notifications from the Observer and detect contractual violations Notify relevant parties of violations

Table 6: Summary of roles.

Regarding the actual implementation of these agents, we do not provide excessive detail for the business roles from the domain, as their implementation is not central for the understanding of an instantiation of the CONTRACT system, but focus instead on the administrative roles. Thus, Sections 7.2.1 through 7.2.4 do not refer to AgentSpeak(L) code snippets, while Sections 7.2.5 and 7.2.6 contain a more detailed analysis of their respective implementations.

7.2.1. Airline Operator

Airline operators manage a fleet of aircraft and have a flight schedule that must be fulfilled throughout the system execution. When the system starts up, operators receive information regarding their aircraft and engines as perceptions specifying the identity, ownership and location of each aircraft, as well as the identity, location, number of usage cycles and provenance of each engine. Here, an engine may be either mounted on an aircraft, or stored in an engine pool, and its provenance comprises a list of each aircraft the engine has ever been mounted on. Moreover, operators receive the schedule of flights that must be flown by means of perceptions identifying the scheduled time, the airline responsible for executing the flight, the designated aircraft, and origin and destination of each flight. After having the simulation data internalised in the belief base, airline operators seek to sign maintenance contracts with engine manufacturers to provide for the aftercare of the engines used by their fleet. Initially, an operator broadcasts its intention to sign maintenance contracts, and engine manufacturers that are willing to provide this service reply with a message to accept the contract. Upon receipt of an acceptance by an engine manufacturer, an operator signs the contract.

7.2.2. Engine Manufacturer

Engine manufacturers build and own a pool of aircraft engines that are sold or leased to airline operators that may also contract them to carry out regular maintenance on such engines. Like the airline operator, the engine manufacturer has its belief base initialised at the beginning of the simulation with information on the engines it can maintain. As operators send requests for maintenance contracts, manufacturers receive these requests and decide whether or not to accept them, either accepting or turning down the requests.

Once a contract is established, an engine manufacturer is obliged to respond to scheduled maintenance requests within a predetermined time frame, and to unscheduled requests as soon

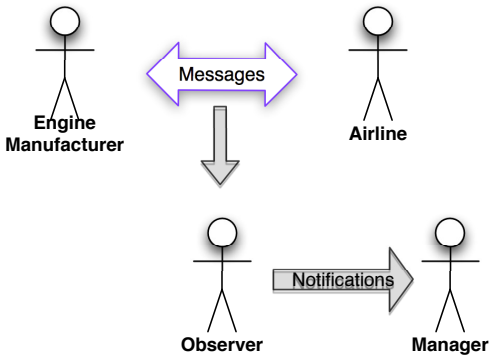


Figure 12: Flow of messages among the agents.

as possible. Requests for maintenance are received by the manufacturer and, if they are valid, a service site is designated and the request for maintenance forwarded to it. Once a service site notifies a manufacturer that maintenance is completed, it informs the operator that the aircraft is ready to fly by sending it a confirmation message.

7.2.3. Service Sites

Service sites are the the locations where maintenance operations actually take place, and are used by the engine manufacturers to carry out requested maintenance. Whenever an engine manufacturer sends a request to repair an engine at a service site, the service site broadcasts a request for bids to supply parts, and awards a supply contract to the best bid from a permitted supplier, waiting for the parts to be delivered. When a service site receives a confirmation from the part supplier that the ordered parts were delivered, it carries out the requested maintenance, and then notifies the requesting engine manufacturer.

7.2.4. Part Suppliers

Part suppliers are responsible for supplying parts to service sites, responding to requests for parts with bids for supply contracts. Once a bid is accepted by a service site, a part supplier is obliged to deliver parts within the accepted deadline for the price specified in the bid. After a part is supplied, the service site is notified of the delivery.

7.2.5. Observer

An observer is responsible for monitoring the activities of contract parties and detecting whether or not any contract violations take place.[8] In the aftercare scenario, an observer only monitors the requests and responses to maintenance operations. In order to monitor these requests, our observer implementation leverages the fact that agents representing contract parties include plans that comprise a communication layer for CONTRACT-related communication. For example, if an operator wishes to send a maintenance request to an engine manufacturer, instead of directly invoking a communication action, it uses plans that include communication actions, as well as replicating sent messages to the observer. This functionality is implemented by a plan that ensures that all messages exchanged between contract parties are also forwarded to the observer, keeping it up to date regarding the status of contractual commitments.

The plans to send messages and to notify the observer are shown in Listing 1, and the flow of messages from each role is illustrated in Figure 12. Such an approach is similar in spirit to that of Garcia-Camino *et al.*[21], since agents are not directly aware that their contractual obligations are monitored by an external agent through each agent's communication layer.

[8] As a consequence, the observer runs the monitoring algorithms of Section 5

```
+!contractSend(Target,SpeechAct,Message) : true
   <- .send(Target, SpeechAct, Message);
      !notifyObserver(Target,SpeechAct,Message).

+!notifyObserver(Target,SpeechAct,Message) : observer(Observer)
   <- .my_name(From);
      .send(Observer, tell, message(From,Target,Message)).
```

Listing 1: Communication plans.

```
+message(From,To,Message) [source(From)] : true
   <- !handleMessage(Message, From, To).
```

Listing 2: Plan to handle messages between contract parties.

Since the observer is not a compulsory part of the system, its existence and identity is not known in advance by airline operators or engine manufacturers. Therefore, at the beginning of the simulation, all contract parties broadcast a message requesting observers to identify themselves. If an observer is present in the system, it replies with its identity, which allows contract parties to forward their communication to it.

Observed messages are handled by the observer in a generic way, as illustrated by the plan in Listing 2. In order to detect violations of deadlines for maintenance, observers generate triggers associated with the deadline for maintenance operations. Whenever an observer detects a request for maintenance from an operator, it awaits confirmation from the manufacturer that maintenance has been performed (*i.e. maintenanceDone*), as shown in the plans of Listing 3.

If the deadline for this maintenance request is reached without the observer having perceived a *maintenanceDone* message, it notifies the manager responsible for the actual handling of the violation within the system. This is illustrated by the plans in Listing 4.

7.2.6. Manager

Whenever a violation is detected by the observer, it notifies the manager, which is responsible for taking some sort of remedial action. In our system, managers have a generic plan to handle violation notifications in some pre-defined way, as illustrated in Listing 5. In this example, a plan reacts to the perception of a violation (*violation(Violation, From, To)*), which is forwarded to the *!handleViolation* plan where the violation is actually handled by the manager. In the aftercare scenario, the only possible violation relates to an engine manufacturer not honouring the deadline for scheduled maintenance (represented as *maintenance(Time)*) and the handling of this violation consists simply of informing a human user of the violation through a console message. Finally, we note that in the current implementation of the manager's monitoring, the explanation generation mechanism has not been implemented and so is left for future work.

```
+!handleMessage(
     requestMaintenance(Time,Plane,Location,Engine), From, To)
   : true
   <- !print("Handling request for maintenance from ",From," to ", To);
      //When I hear a maintenance request,
      //store the request
      +maintenanceRequested(Time,Plane,Location,Engine,From,To);
      ?maintenanceDeadline(Deadline);
      TriggerDeadline = Time+Deadline;
      //And create a trigger
      +trigger(maintenance,TriggerDeadline).

+!handleMessage(
      maintenanceDone(Time,Plane,Engine), From, To)
    : maintenanceRequested(TimeReq,Plane,Location,Engine,To,From)
   <- +maintenancePerformed(Time,Plane,Location,Engine,From,To).
```

Listing 3: Observer plans to handle maintenance requests.

```
+!checkMaintenanceDone(maintenance(Time,Plane,Engine))
   : maintenanceRequested(Time,Plane,Location,Engine,From,To) &
     maintenancePerformed(TimeDone,Plane,Location,Engine2,To,From) &
     time(Now) &
     (TimeDone < Now)
   <- true. //No violation detected

+!checkMaintenanceDone(maintenance(Time,Plane,Engine))
   : maintenanceRequested(Time,Plane,Location,Engine,From,To) &
     time(Now)
   <- .send(manager, tell, violation(maintenance(Now),To,From)).
```

Listing 4: Observer plan to detect violations.

```
+violation(Violation,From,To) [source(S)]
   : true
   <- !handleViolation(Violation,From,To).

+!handleViolation(maintenance(Time),From,To)
   : true
   <- .print(From, " should have done maintenance for ",To," by time ",Time,", but it did not.").
```

Listing 5: Manager plan to handle violations.

8. Conclusions and Related Work

In this paper we have shown the practical applicability of the CONTRACT project framework by instantiating its architecture within an aerospace domain.[9] Importantly, the domain was used throughout, to inform the development of the framework, being particularly suited to the application of CONTRACT technologies. We have described the components of the framework with a view towards practical implementation, showing how real agents subjected to electronic contracts can operate, and showing how the administrative roles can be implemented in a popular agent programming language.

Furthermore, we have developed a simulator for the aircraft aftercare case study [6]. This simulator was used in the development of the CONTRACT framework and in the validation of the monitoring mechanism. Data obtained from the prototype can impact on the development of Lost Wax's *Aerogility* tool, as the results indicate the utility of contractual encoding of certain agent behaviours, and the monitoring of these behaviours to enable diagnosis, and thus instigate remedial measures, and contract refinement.

In consideration of existing related work, there are two particular efforts worth mentioning here. First, the AMELI middleware [22] is strongly related to our work in CONTRACT in that it also provide a framework in which norms are managed and monitored, and some of the key components in AMELI have analogues in CONTRACT . AMELI, however, focuses on electronic institutions, and views norms as hard constraints, which cannot be violated, as opposed to the norms described in CONTRACT , which can be violated and entail punishments. Moreover, since it assumes inviolable norms, AMELI does not include mechanisms for *explaining* violations, on which the work presented in this paper focuses significant attention. Second, the ATN model provides us with the ability to detect *when* a norm enters some state and also potentially allows us to understand *why* this state transition occurred. This latter understanding is obtained by examining the logical formulae that caused the transition to occur, and understanding what caused these to evaluate to true. Now, in complex domains, this latter task may not be trivial, and [23] have examined how graphical techniques can be used to enhance this understanding. They propose a system by which graphical models can be manipulated and expanded in order to allow a user to drill down to the root cause of a system transition, but they do not provide the extensive implemented system as we have done here.

In conclusion, our contribution in this paper is twofold: we have demonstrated the practicality of the technologies developed for contract-enabled e-business systems, in particular norm moni-

[9] The CONTRACT platform is available for download at http://ist-contract.sourceforge.net/

toring and violation explanation; and we have shown how these technologies can be deployed in a real multi-agent simulation.

Acknowledgments: The research described in this paper was 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. The first author was partly supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) of the Brazilian Ministry of Education.

References

[1] F. Meneguzzi, M. Luck, Norm-based behaviour modification in BDI agents, in: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 2009, pp. 177–184.

[2] N. Oren, S. Panagiotidi, J. Vazquez-Salceda, S. Modgil, M. Luck, S. Miles, Towards a formalisation of electronic contracting environments, in: Proceedings of Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI 2008, Chicago, Illinois, USA, 2008, pp. 61–68.

[3] S. Modgil, N. Faci, F. Meneguzzi, N. Oren, S. Miles, M. Luck, A framework for monitoring agent-based normative systems, in: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 2009, pp. 153–160.

[4] M. Jakob, M. Pěchouček, J. Chábera, S. Miles, M. Luck, N. Oren, M. Kollingbaum, C. Holt, J. Vázquez, P. Storms, M. Dehn, Case studies for contract-based systems, in: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 2008, pp. 55–62.

[5] LostWax, Aerogility, http://www.aerogility.com/ (2007).

[6] F. Meneguzzi, S. Miles, C. Holt, M. Luck, N. Oren, S. Modgil, N. Faci, M. Kollingbaum, Electronic contracting in aircraft aftercare: A case study, in: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 2008, pp. 63–70.

[7] A. Lomuscio, H. Qu, M. Solanki, Towards verifying contract regulated service composition, in: Proceedings of the 2008 IEEE International Conference on Web Services, Washington, DC, USA, 2008, pp. 254–261.

[8] S. Miles, N. Oren, M. Luck, S. Modgil, N. Faci, C. Holt, G. Vickers, Modelling and administration of contract-based systems, in: Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems (BRMAS 2008) at AISB 2008, Aberdeen, Scotland, 2008, pp. 19–24.

[9] J. F. Horty, Agency and Deontic Logic, Oxford University Press, 2001.

[10] C. Krogh, The rights of agents, in: M. Wooldridge, J. P. Müller, M. Tambe (Eds.), Agent Theories, Architectures, and Languages II, Vol. 1037, Springer, 1996, pp. 1–16.

[11] F. Lopez y Lopez, Social power and norms: Impact on agent behaviour, Ph.D. thesis, University of Southampton (2003).

[12] D. Makinson, L. van der Torre, Permission from an input/output perspective, J. Philosophical Logic 32 (4) (2003) 391–416.

[13] N. Faci, S. Modgil, N. Oren, F. Meneguzzi, S. Miles, M. Luck, Towards a monitoring framework for agent-based contract systems, in: M. Klusch, M. Pechoucek, A. Polleres (Eds.), Proceedings of the Twelfth International Workshop on Cooperative Information Agents, Prague, Czech Republic, 2008.

[14] W. A. Woods, Transition network grammars for natural language analysis, Communications of the ACM 13 (10) (1970) 591–606.

[15] F. Meneguzzi, Extending agent languages for multiagent domains, Ph.D. thesis, King's College London (2009).

[16] F. F. Ingrand, M. P. Georgeff, A. S. Rao, An architecture for real-time reasoning and system control, IEEE Expert, Knowledge-Based Diagnosis in Process Engineering 7 (6) (1992) 33–44.

[17] A. S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in: W. V. de Velde, J. W. Perram (Eds.), Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Vol. 1038 of LNCS, Springer-Verlag, Budapest, Hungary, 1996, pp. 42–55.

[18] R. H. Bordini, J. F. Hübner, M. Wooldridge, Programming multi-agent systems in AgentSpeak using Jason, Wiley, 2007.

[19] M. E. Bratman, Intention, Plans and Practical Reason, Harvard University Press, Cambridge, MA, 1987.

[20] M. d'Inverno, D. Kinny, M. Luck, M. Wooldridge, A formal specification of dMARS, in: M. P. Singh, A. S. Rao, M. Wooldridge (Eds.), Agent Theories, Architectures, and Languages, Vol. 1365 of Lecture Notes in Computer Science, Springer-Verlag, London, UK, 1998, pp. 155–176.

[21] A. García-Camino, J. A. Rodríguez-Aguilar, W. Vasconcelos, A distributed architecture for norm management in multi-agent systems, in: Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN), Honolulu, HI, USA, 2007, pp. 275–286.

[22] M. Esteva, B. Rosell, J. A. Rodriguez-Aguilar, J. L. Arcos, Ameli: An agent-based middleware for electronic institutions, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 236–243.

[23] M. Croitoru, N. Oren, S. Miles, M. Luck, Graphical norms via conceptual graphs, Knowledge-Based Systems 29 (0) (2012) 31 – 43.