Citation for published version (APA):

Citing this paper
Please note that where the full-text provided on King's Research Portal is the Author Accepted Manuscript or Post-Print version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version for pagination, volume/issue, and date of publication details. And where the final published version is provided on the Research Portal, if citing you are again advised to check the publisher's website for any subsequent corrections.

General rights
Copyright and moral rights for the publications made accessible in the Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the Research Portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the Research Portal

Take down policy
If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Abstract. Mirroring the paper versions exchanged between businesses today, electronic contracts offer the possibility of dynamic, automatic creation and enforcement of restrictions and compulsions on agent behaviour that are designed to ensure business objectives are met. However, where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfill them all; computer-parsable electronic contracts may allow such verification to be automated. In this paper, we describe a conceptual framework and architecture specification in which normative business contracts can be electronically represented, verified, established, renewed, etc. In particular, we aim to allow systems containing multiple contracts to be checked for conflicts and violations of business objectives. We illustrate the framework and architecture with an aerospace example.

1 Introduction

It has often been argued that agents interacting in a common system, society or environment need to be suitably constrained in order to avoid and solve conflicts, make agreements, reduce complexity, and in general to achieve a desirable social order (e.g. [4, 5]). For many, this role is fulfilled by norms, which represent what ought to be done by a set of agents. Views of norms differ, and include fixed laws that must never be violated as well as more flexible social guides that merely seek to bias behaviour in different ways. Yet the obligations, prohibitions and permissions that may affect agent behaviour in a normative system can also be documented and communicated between agents in the form of contracts. Electronic contracts, mirroring the paper versions exchanged between businesses today, offer the possibility of dynamic, automatic creation and enforcement of such restrictions and compulsions on agent behaviour. However, where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfill them all; computer-parsable electronic contracts may allow such verification to be automated.

There are two pre-requisites to realistically applying an electronic contracting approach in real-world domains. First, to exploit electronic contracts, a well-defined conceptual framework for contract-based systems, to which the application entities can be mapped, is needed. Second, to support the management of contracts through all stages of the contract life-cycle, we need to specify the functionality required of a contract management architecture that would underly any such system, leading to ready-made implementations for particular deployments of that architecture.

The CONTRACT project\(^1\) aims to do just this. Funded by the European Commission as part of its 6th Framework Program, the project seeks to develop frameworks, components and tools that “make it possible to model, build, verify and monitor distributed electronic business systems on the basis of dynamically generated, cross-organisational contracts which underpin formal descriptions of the expected behaviours of individual services and the system as a whole.” In this context, this paper documents the CONTRACT project’s work on both of the pre-requisites identified above. The aims and vision of the project are described in full elsewhere [17]. More specifically, the technical contributions described in this paper are: the specification of a model for describing contract-based systems; the specification of an architecture for managing such systems; and the mapping of an aerospace application to those models.

Our approach takes a distinct approach in several respects. First, its development is explicitly driven by a range of use cases provided by a diverse set of small and large businesses. One consequence of this diversity is that our approach must account for different practices and possibilities in each stage of the lifecycle of a contract-based system. It is therefore defined in terms of abstract process types, to be instantiated in different ways for different circumstances. We provide a non-exhaustive set of options for instantiating these process types, and technologies to support these processes. A more specific requirement addressed by our approach is in managing not just fulfillment or violation of contractual obligations, but also other states of the system with regard to those obligations, such as being in danger of violation, being expected to easily fulfil an obligation, and application-specific states.

In the next section, we provide an overview of the overall structure, introducing the conceptual framework and applying it to a running example. Then, in Section 3, we discuss how the contractual obligations imply critical states of the system that we may wish to detect and react to in order to effectively manage the system. In Section 4, we describe the architecture: the process types that are required to manage contract-based systems and components that can support such processes. We discuss related work in Section 5 and conclude with future work in Section 6.

2 CONTRACT Framework and Architecture

The models and procedures comprising the CONTRACT framework and architecture are shown in Figure 1. The primary component of this is the framework itself, depicted at the top of the figure, which is the conceptual structure used to describe a contract-based system, including the contracts and the agents to which they apply.

\(^1\) King’s College London, UK, email: simon.miles@kcl.ac.uk
\(^2\) Lost Wax, UK
\(^3\) www.ist-contract.org
From the framework specification of a given application, other important information is derived. First, understanding the contractual obligations of agents allows us to specify the critical states that an application may reach. A critical state of a contract-based system with regard to an obligation essentially indicates whether the obligation is fulfilled or fulfillable, e.g. achieved, failed, in danger, etc. This is discussed in detail in Section 3. A state-based description, along with the deontic and epistemic implications of the specified contracts, can then be used to verify a system either off-line or at run-time [11] (though we do not discuss this further here).

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 administration 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. Such process types and roles are described in Section 4.

2.1 A Contract-Based System Framework

We first specify a conceptual framework by which contract-based systems can be described. This framework provides a clear indication of how particular applications can exploit contracts and how they must be supported in managing them. By being abstract and generic, such a framework may also be used to translate contract data from one concrete format to another.

Contracts document obligations, permissions and prohibitions (collectively clauses) on agents and are agreed by those agents. Simply, obligations are statements that agents should do something and prohibitions are statements that they should not. Since agents are autonomous, external control can only be specified through normative structures such as contracts. Thus, permissions are defined as exceptions to prohibitions: if something was not prohibited, it is not meaningful for a permission to be granted.

The agents obliged, permitted and prohibited in a contract are parties to that contract, which specifies contract roles, played by agents within it. Each clause in a contract applies to roles, to which agents are assigned, and each agent can hold multiple contracts with the same or different parties. Finally, a contract proposal is one that has not yet been agreed. These concepts are summarised in Table 1.

Table 1. The primary concepts in the CONTRACT framework

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>A named part that can be played by an agent in a system.</td>
</tr>
<tr>
<td>Obligation</td>
<td>A statement that an agent playing a given role should do something.</td>
</tr>
<tr>
<td>Prohibition</td>
<td>A statement that an agent playing a given role should not do something.</td>
</tr>
<tr>
<td>Permission</td>
<td>An exception to a prohibition for an agent playing a given role</td>
</tr>
<tr>
<td>Clause</td>
<td>An obligation, prohibition or permission.</td>
</tr>
<tr>
<td>Assignment</td>
<td>A statement that an agent should play a given role.</td>
</tr>
<tr>
<td>Proposal</td>
<td>A document containing a set of clauses and assignments,</td>
</tr>
<tr>
<td>Contract</td>
<td>A proposal to which all assigned agents have agreed.</td>
</tr>
</tbody>
</table>

2.2 Aerospace Use Case

To test and illustrate the efficacy of our approach, we adopt an engineering application, based on the aerospace aftercare market, targeted by Lost Wax’s agent-based Agility platform [1], and used as a running example through the paper.

The application concerns the continued maintenance of aircraft engines over their lifetime. In this domain, an engine manufacturer is contractually obliged to ensure operators’ aircraft have working engines. For an engine to be working, it should not be overdue for regular servicing or left waiting to be fixed after a fault is discovered. An aircraft’s engine can be replaced when it lands at some location if there is a suitable spare engine present at that location. As well as replacing engines to ensure continued operation of the aircraft, an engine manufacturer will service the engines it has removed, so that the serviced engine can be added back into circulation (the “engine pool”) and used to replace other engines.

In addition to long-term contracts between engine manufacturers and operators, we consider short-term contracts regarding particular instances of servicing engines. These sit in the context of long-term contracts but, by being specified explicitly, allow the parties to use and commit to resources more flexibly. In a long-term contract between an aircraft operator and an engine manufacturer, the manufacturer agrees to service the operator’s aircraft to some overall specified standard over the duration of the contract. Such a contract is provided in Table 2, using the framework concepts. Here, the operator specifies a preferred time within which the manufacturer must service an aircraft, and the manufacturer is obliged to meet this in 90% of cases. If the manufacturer does not meet short-term contract requirements (see below), penalties are deducted from the long-term payment the operator is obliged to make. The operator is obliged to provide adequate engine data so that the manufacturer can fulfil their servicing obligations. Finally, the operator may have demands on the provenance of an engine: operator A may be happy to re-use engines previously used by operators B or C but not those used by D.

In this context, a short-term contract concerns the servicing of a particular aircraft at a particular time (see Table 3). It is again between two parties: the aircraft operator and the engine manufacturer.
3 Critical States of Contract-Based Systems

In this case, the manufacturer has more specific obligations: that they must either service an aircraft in the preferred timescale or pay a penalty, and that they must service it within a maximum period. The limitations on provenance apply to the particular short-term servicing as they do in the long-term aftercare.

<table>
<thead>
<tr>
<th>Roles</th>
<th>Manufacturer, Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>Manufacturer agrees to servicing contracts requested by operator during aftercare contract period.</td>
</tr>
<tr>
<td>O2</td>
<td>Manufacturer services engines within the preferred time specified by the servicing contracts in at least 90% of cases.</td>
</tr>
<tr>
<td>O3</td>
<td>Operator pays for servicing or engines, minus any penalties.</td>
</tr>
<tr>
<td>O4</td>
<td>Operator must supply engine health data to the manufacturer in an adequate time to allow problems requiring unscheduled maintenance to be detected.</td>
</tr>
<tr>
<td>P1</td>
<td>Manufacturer is prohibited from supplying engine parts previously used by other operators not on an approved list (if one is given) or on a disapproved list (if one is given).</td>
</tr>
<tr>
<td>P2</td>
<td>Manufacturer services engine in maximum time.</td>
</tr>
<tr>
<td>P3</td>
<td>Manufacturer services aircraft in preferred time, or pays penalty (taken out of aftercare contract payment from operator).</td>
</tr>
<tr>
<td>P4</td>
<td>Manufacturer services engine in maximum time.</td>
</tr>
</tbody>
</table>

Table 2. Long-term aftercare contract

Such formal documents of agreements are important, especially when there are multiple agreements and when these agreements can interact, because they can reveal critical points of potential or actual conflict. If it is possible to examine such contracts, and determine where these critical points lie, so that they may be monitored for violations, or even modified to avoid the potential for violation. In what follows, these aims inform the elaboration of our architecture. For example, we classify obligations into three types:

- An obligation to achieve some state $G$, e.g. to pay some amount to another agent.
- An obligation to maintain some state $H$, e.g. to keep an aircraft in working order.
- An obligation to behave in some way, where that behaviour is to fulfil obligation $O(X)$ whenever event $E(X)$ occurs, e.g. when aircraft $X$ requires servicing, to service $X$ in an acceptable time.

For an achievement obligation, there are three significant states:

<table>
<thead>
<tr>
<th>State</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-achievement</td>
<td>Not $G$ Agent obliged to achieve $G$</td>
</tr>
<tr>
<td>Succeeded</td>
<td>$G$</td>
</tr>
<tr>
<td>Failed</td>
<td>$G$ unachievable Agent obliged to achieve $G$</td>
</tr>
<tr>
<td>Cancelled</td>
<td>No agent obliged to achieve $G$</td>
</tr>
</tbody>
</table>

Table 4. Basic states (top) and sample pre-achievement sub-states (bottom) of an achievement obligation

3.1 Obligation States

Each obligation implies a set of states for the system with regard to that obligation. In part, these can be specified independently of the application. For example, we classify obligations into three types:

- An obligation to achieve some state $G$, e.g. to pay some amount to another agent.
- An obligation to maintain some state $H$, e.g. to keep an aircraft in working order.
- An obligation to behave in some way, where that behaviour is to fulfil obligation $O(X)$ whenever event $E(X)$ occurs, e.g. when aircraft $X$ requires servicing, to service $X$ in an acceptable time.

For an achievement obligation, there are three significant states:

3.2 Significant Sub-States

In addition to the application-independent system states above, applications often refer to significant sub-states part-way between an obligation coming into force and its success or failure. Examples of these are shown in the bottom portion of Tables 4, 5 and 6. For example, an application may need to detect whether an obligation is in danger of violation and so allocate more resources to ensure that it is fulfilled instead, implying a Danger critical state of the system with regard to that obligation as shown in Table 4 (bottom). On the other hand, if an obligation is being fulfilled unexpectedly easily, an
4.2 Maintenance and Update

The continued existence and integrity of a contract after establishment is important in reliable systems. As with establishment, there

4.1 Establishment

There are many potential ways to establish a contract, varying in complexity. To give an illustration, we present two below.

Full Proposal Establishment Process Here, one party, the proposer, creates a full proposal, excluding some assignments of roles to agents, and signs it. It then uses a registry to discover agents that may fulfill the unassigned contract roles. For each unassigned role in turn, it offers the proposal to an agent, a potential party it is satisfied can assume that role. If the party is willing, it signs the proposal and returns it. When the last role is filled, a contract is established.

Template Discovery Establishment Process Alternatively, a process may be used in which an agent discovers a contract template that may be instantiated in a way that fulfills its goals. This implies the use of a template repository, where templates can be stored. Such templates may have some assigned roles; that is, they may describe services for which a provider is willing to negotiate terms.

Table 5. Basic states (top) and sample maintained sub-states (bottom) of a maintenance obligation

<table>
<thead>
<tr>
<th>State</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>Not H in danger or becoming true</td>
</tr>
<tr>
<td>succeeded</td>
<td>H achievable</td>
</tr>
<tr>
<td>failed</td>
<td>H unachievable</td>
</tr>
<tr>
<td>cancelled</td>
<td>No agent obliged to maintain H</td>
</tr>
<tr>
<td>Sub-state</td>
<td>Additional Properties</td>
</tr>
<tr>
<td>initial</td>
<td>Not H in danger or becoming true</td>
</tr>
<tr>
<td>succeeded</td>
<td>H achievable</td>
</tr>
<tr>
<td>failed</td>
<td>H unachievable</td>
</tr>
<tr>
<td>cancelled</td>
<td>No agent obliged to maintain H</td>
</tr>
</tbody>
</table>

Table 6. Basic states (top) and sample pre-trigger sub-states (bottom) of a behaviour obligation

3.3 Example

As an example, in Table 7, we enumerate critical states for an achievement obligation, indexed O2, in the long-term aftercare contract of Table 2. It is an achievement obligation as it describes an eventual state of the system in which a state has been achieved, i.e. 90% of servicing cases were performed in the preferred time period. When the contract first comes into force, i.e. the system time is within the contract period, the state Pre-achievement: Initial holds. In this state, insufficient cases have been performed to determine whether success is likely. After 5% of cases are performed, the system will be in either Pre-achievement: Satisfactory or Pre-achievement: Danger states, and may vary between them over the contract period. Pre-achievement: Satisfactory holds where over 5% of cases are performed within the preferred time, while Pre-achievement: Danger holds where between 5% and 10% of cases exceeded that time. The value of taking account of these two states is that transfer of resources between fulfillment of different obligations can be triggered by changes of state. Eventually, the system will reach either succeeded state, where the contract period is exceeded and over 90% of cases were performed on time, or Failure state, where over 10% have exceeded the preferred time. The choice of the appropriate sub-states (Pre-achievement: Satisfactory and Pre-achievement: Danger in this case) is entirely application dependent: considering more states allows finer control as appropriate, but may also add overheads.

Table 7. States of aftercare contract obligation O2

<table>
<thead>
<tr>
<th>State</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>Less than (estimated) 5% of servicing cases performed and within contract period</td>
</tr>
<tr>
<td>succeeded</td>
<td>Between 5% and 10% of cases exceeded preferred time and within contract period</td>
</tr>
<tr>
<td>failed</td>
<td>More than 10% of cases exceeded preferred time beyond contract period</td>
</tr>
</tbody>
</table>

4 Architecture of Contract-Based Systems

Aside from modelling contract-based systems using the CONTRACT framework, we also address the issue of administration: how to manage the processes involved in creating, maintaining, acting on and otherwise processing contracts and contract proposals. We identify four key process types in the contract life-cycle.

Establishment brings about the existence of the contract. Maintenance and Update ensures a contract’s integrity over time. Fulfilment brings about the fulfillment of obligations while observing its prohibitions. Termination or Renewal end the normative force of the contract, or renew it to apply for a longer period.

Each of these process types can be instantiated in different ways, depending on the application and its deployment. The choice dictates the roles agents must play to fulfill the administration duties implied. Below, we examine each process type in turn.
are multiple ways in which this can be achieved, and the functionality that needs to be provided depends on the particular contract and application.

**Contract Store Maintenance Process** Here, contract parties use a contract store to maintain and control access to contracts. The store is obliged only to allow agents to change the contract when all parties send a signed agreement of the change to be made.

**All Party Signature Maintenance Process** In this process, integrity is preserved by the contract being signed by all parties in a way that prevents editing without detection; for example, digital signatures based on reliable certificates. The signed document includes the contract itself, and an indication of whether it is a revision of a previous version. Each party should check the signatures of the contract before accepting it as binding.

### 4.3 Fulfilment

For every contractual obligation and prohibition, there are certain processes that can be performed to help ensure they are fulfilled. As with the processes above, these imply particular administrative roles that must be played by agents. The administrative roles carry with them obligations, prohibitions and permissions, which may be documented in the same contract as the one that is the target of administration, or another contract. The processes below often refer to particular system states with regard to obligations: these are the states specified in Section 3.

**Observation of Fulfilment Process** An observer observes state changes to determine whether contractual obligations are being fulfilled. It can notify other agents when an obligation is being violated or in danger of violation\(^4\). An observer \(X\) is in an obligation pattern of the following form:

\[ X \text{ is obliged to observe for critical state } S \text{ of contract clause } C, \text{ and notify registered listeners when it occurs.} \]

**Management of Fulfilment Process** A manager is an agent that acts when an obligation is not being fulfilled, is in danger of not being fulfilled or a prohibition is breached. It knows about the problem by (conceptually at least) registering to listen to the notifications from an observer. Manager is a role, and one agent may play the role of both manager and observer. The nature of the action taken by a manager may vary considerably. In highly automated and strict applications, an automatic penalty may be applied to a party. In other cases, a management agent may be a human who decides how to resolve the problem. Alternatively, a manager may merely provide analysis of problems over the long term, so that a report can be presented detailing which obligations were violated. A manager \(X\) is in an obligation pattern of the following form:

\[ X \text{ is obliged, whenever the system reaches a critical state } S \text{ of contract clause } C, \text{ to perform action } A. \]

Note that we avoid the sometimes used term, *enforcer*, because enforcement implies action either in the case of failure or in the case of likely failure, and often sanctions or financial penalties. In the applications we are considering, likely success is also a good state to act upon, e.g. to reallocate resources used for the successful obligation, and actions may take softer forms, e.g. notified humans renegotiate the contract, so the more generic term, manager, is used.

An example of an observer’s obligation in the aerospace application is shown in Table 8 (top), and of a manager’s obligation in Table 8 (bottom). The observer, Checker, is obliged to check that a Danger state has not been reached for the number of suitable engines available at a given location, and the manager, Enforcer, listens to observations on this state and rectifies the situation when it occurs.

### 4.4 Termination and Renewal

Termination of a contract means that the obligations and other clauses contained within it no longer have any force. A contract may be terminated in several ways: (i) it may terminate naturally if the system reaches a state in which none of its clauses apply, e.g. a contract’s period of life expires or all obligations have been met; (ii) it may terminate by design if the contract has an explicit statement that the contract is terminated when a particular event occurs (e.g. if one party fails to meet an obligation, the contract is terminated and all others are released from their obligations); or (iii) it may terminate by agreement, if parties agree that the contract should no longer hold, and update it accordingly (in line with the process chosen for the Maintenance and Update type above). Renewal of a contract means that a contract that would have imminently terminated naturally is updated so that termination is no longer imminent (again depending on the Maintenance and Update process type above).

### 4.5 Administrative Roles and Components

The processes above all require the fulfilment of particular administrative roles, e.g. contract store, registry, observer, manager. For some of these components, we can provide generic implementations. For example, a contract store, based solely on contract documents and having nothing to do with the application itself, is easy to implement generically. Others, such as managers, need to have application-specific instantiations, as dealing with a contractual obligation not being fulfilled varies greatly between applications. Specifying the components further is largely a technology-dependent issue out of scope of this paper.

### 5 Related Work

There has been much previous work on various aspects of contract-based system modelling, enactment and administration, and our approach is intended to build on and be compatible with other ideas presented elsewhere. For example, there are many approaches to negotiation which may be used in the establishment of contracts [12], and the administration of contracts can integrate with other useful behaviour, such as observation of fulfilment and violation of obligations.

\[^4\] Note that we do not use another term sometimes appearing in the literature, *monitor*, because it is also often used to refer to quantitative infrastructure-level measurements of system metrics; in the case of many business applications, observation is of discrete high-level states.

<table>
<thead>
<tr>
<th>Roles</th>
<th>Obligations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checker, Manufacturer, Operator</td>
<td>Checker monitors the number of engines available to the manufacturer at a given location that are suitable for a given operator, and notifies registered agents if it falls below a minimum quantity.</td>
</tr>
<tr>
<td>Enforcer, Checker</td>
<td>Enforcer, on hearing from checker that the number of suitable engines at a location has fallen below a minimum level, transports a suitable engine from another location.</td>
</tr>
</tbody>
</table>

Table 8. Engine supply checking contract (top) and Engine supply enforcement contract (bottom)
6 Conclusions and Future Work

In this paper, we have presented the CONTRACT conceptual framework and architecture, and shown how they apply to aircraft aftercare. By creating a technology-dependent implementation along these lines, an application can take advantage of the reliable coordination provided by electronic contracts. The CONTRACT project aims to allow multi-agent systems to be verified on the basis of their contracts, building on work by Lomuscio et al. on deontic interpreted systems [11]. While this verification is beyond the scope of this paper, it places a requirement on our framework that the properties of the target system are identified and isolatable, and a requirement on the architecture that such information can be captured in order to pass to a verification mechanism. Perhaps equally importantly, we also aim for an open source implementation built on Web Services technologies, requiring the architecture to be compatible with such an objective. Finally, taking a very practical standpoint, we intend to construct a methodology to guide development of applications that use electronic contracts through the process from conceptual framework to deployment.

Acknowledgement: The CONTRACT project is co-funded by the European Commission under the 6th Framework Programme for RTD with project number FP6-034418. Notwithstanding this fact, this paper and its content reflects only the authors’ views. The European Commission is not responsible for its contents, nor liable for the possible effects of any use of the information contained therein.

REFERENCES