Electronic Business Contracts between Services

Simon Miles, Nir Oren, Michael Luck, Sanjay Modgil and Felipe Meneguzzi
King's College London, UK
Nora Faci
University of Lyon, France
Camden Holt and Gary Vickers
Lost Wax, UK

ABSTRACT

Electronic contracts mirror the paper versions exchanged between businesses today, and offer the possibility of dynamic, automatic creation and enforcement of restrictions and compulsions on service behaviour that are designed to ensure business objectives are met. Where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfil them all, yet computer-parsable electronic contracts may allow such verification to be automated. In this chapter, we describe a conceptual framework and architecture specification in which normative business contracts can be electronically represented, verified, established, renewed, and so on. 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 aftermarket example.

INTRODUCTION

It has often been argued that independent entities, such as business services, 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 (Conte & Castelfranchi, 1993; Conte, Falcone & Sartor, 1999). For many, this role is fulfilled by norms, which represent what *ought* to be done by a set of services (when performing functions on behalf of their owning business). 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 service behaviour in a normative system can also be *documented* and communicated between services 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 service behaviour. However, where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfil them all; computer-parsable electronic contracts may allow such verification to be automated.

In a peer-to-peer system, organisations, and the services performing functions on their behalf, act as independent peers, with no overall authority, and contracts are necessary to add predictability to behaviour between them. Where there is multiple, independently owned alternatives for a resource or service, contract technology is of particular use. By providing and monitoring contract compliance, applications can make better decisions on which resources or services to take advantage of in the future, a particular problem on Grid systems with a range of reliability issues.

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 underlie any such system, leading to ready-made implementations for particular deployments of that architecture. The CONTRACT project (CONTRACT, 2008) 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 chapter documents the CONTRACT project's work on both of the pre-requisites identified above. More specifically, the technical contributions described in this chapter 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 is distinct in several respects. First, its development is explicitly driven by a range of use cases (Jakob et al., 2008) 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 fulfilment 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. We then 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. After that, we describe the architecture: the process types that are required to manage contract-based systems and components that can support such processes. Finally, we discuss related work and conclude with future work.

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 services to which they apply.

From the framework specification of an application, other information is derived. First, understanding the contractual obligations of services 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: achieved, failed, in danger, and so on. This is discussed in the following section. A state-based description, along with the deontic (concerning duties) and epistemic (concerning knowledge) implications of the specified contracts, can then be used to verify a system either off-line or at run-time (Lomuscio & Sergot, 2003) (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, update, termination, and renewal. Such processes also include observation of the system, so that contractual

obligations can be enforced or otherwise managed, and these processes depend on the critical states identified above. Once suitable administration processes are identified, we can also specify the roles that services play within them, the components that should be part of services to allow them to manage their contracts, and the contract documents themselves. Such process types and roles are described further below.

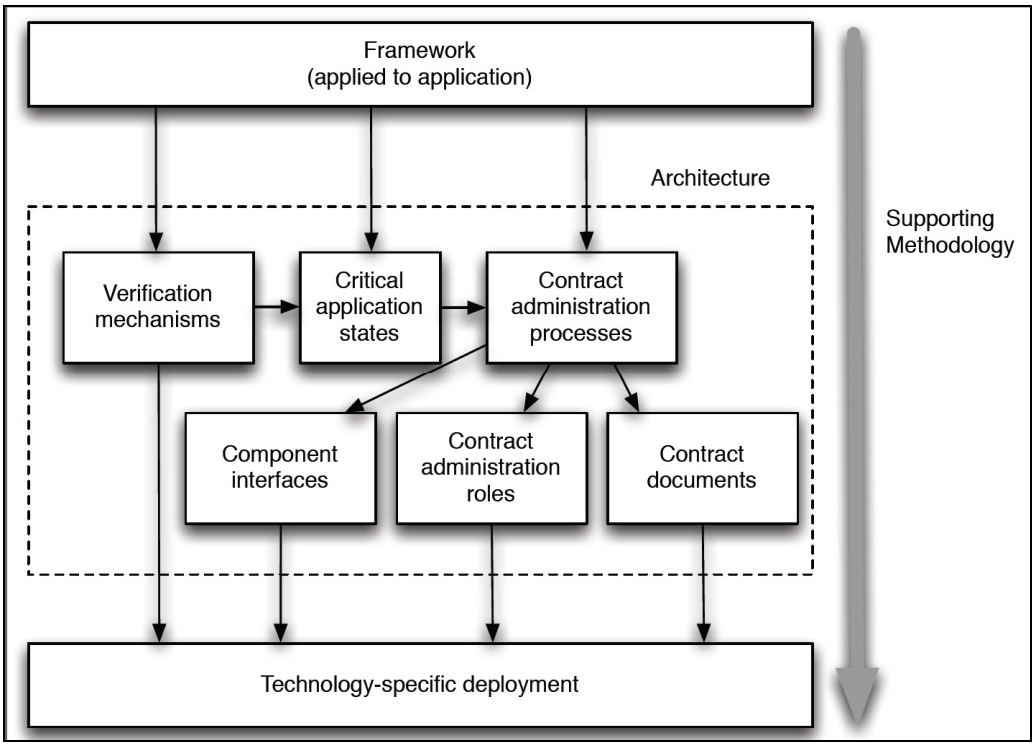

Figure 1. The overall structure of the CONTRACT architecture and framework

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 be used to translate contract data from one concrete format to another.

Contracts document *obligations*, *permissions* and *prohibitions* (collectively *clauses*) on services and are *agreed* by those services (strictly, it is the *agent* enacting the service's logic (W3C, 2008) to which impositions on behaviour apply). Put simply, obligations are statements that services should do something and prohibitions are statement that they should not. Permissions are defined as exceptions to prohibitions: if something was not prohibited, it is not meaningful for a permission to be granted.

The services obliged, permitted and prohibited in a contract are parties to that contract, which specifies *roles* played by services within it. Each clause in a contract applies to roles, to which services are *assigned*, and each service can hold multiple contracts with the same or different parties. The obligation, permission or prohibition defined in a clause is on the service(s) assigned to the role to which the clause applies. A *contract proposal* is a contract that has not yet been agreed by its parties. The concepts are summarised in Table 1.

Role	A named part that can be played by a service in a system.
Obligation	A statement that a service playing a given role should do something.
Prohibition	A statement that a service playing a given role should not do something.
Permission	An exception to a prohibition for a service playing a given role under given circumstances.
Clause	An obligation, prohibition or permission.
Assignment	A statement that a service should play a given role.
Proposal	A document containing a set of clauses and assignments, where every role referred to which each clause applies has been assigned to a service.
Contract	A proposal to which all assigned services have agreed.

Table 1. The primary concepts in the CONTRACT framework

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 Aerogility platform (Lost Wax, 2008), and used as a running example through the chapter.

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.

Roles		Manufacturer, Operator
Obligations	**O1**	Manufacturer agrees to servicing contracts (defined in Table 3) requested by operator during aftercare contract period.
	O2	Manufacturer services engines within the preferred time specified by the servicing contracts in at least 90% of cases.
	O3	Operator pays for servicing of engines, minus any penalties.
	O4	Operator must supply engine health data to the manufacturer in an adequate time to allow problems requiring unscheduled maintenance to be detected.
Prohibitions	**P1**	Manufacturer is prohibited from supplying engines with parts previously used by other operators not on an approved list (if one is given) or on a disapproved list (if one is given).
Permissions	**R1**	Manufacturer is allowed to supply engines with parts previously used by other operators on an approved list (if one is given).

Table 2. Long-term aftercare contract

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. 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 in the particular short-term servicing as they do in the long-term aftercare.

Roles		Manufacturer, Operator
Obligations	**O5**	Manufacturer services aircraft in preferred time, or pays penalty (taken out of aftercare contract payment from operator).
	O6	Manufacturer services engine in maximum time.
Prohibitions	**P2**	Manufacturer is prohibited from supplying engines with parts previously used by other operators not on an approved list (if one is given) or on a disapproved list (if one is given).
Permissions	**R2**	Manufacturer is allowed to supply engines with parts previously used by other operators on an approved list (if one is given).
	R3	Operator is allowed to take a penalty from the manufacturer if an aircraft is left on the ground for longer than the preferred time agreed.

Table 3. Short-term servicing contract

Such formal documentation of agreements is important, especially when there are multiple agreements and when these agreements can interact, because they can reveal points of potential or actual conflict. If it is possible to examine such contracts, and determine where these points lie, then one can monitor for violations, or even instigate measures to pre-empt violation. In what follows, these aims inform the elaboration of our architecture. For example, a short-term conflict between two servicing contracts in the aerospace domain occurs when a manufacturer is obliged to service two operators' aircraft at the same time, but can only service one due to a lack of resources. Long-term conflicts are also present, as in a conflict between a servicing contract and an aftercare contract arising when a manufacturer must choose between servicing one operator's aircraft within the maximum time limit and servicing another operator's aircraft within the preferred time, where the manufacturer is in danger of not having serviced the latter operator's aircraft within the preferred time limit for 90% of cases.

CRITICAL STATES OF CONTRACT-BASED SYSTEMS

As mentioned above, a critical state of a contract-based system with regard to an obligation essentially indicates whether the obligation is fulfilled or fulfillable: achieved, failed, in danger, and so on. By identifying the *critical states* of the system with respect to given contractual obligations, it is then easier to determine which of these needs to be checked for and acted upon to ensure that the system performs well. A state-based description can also be used as a basis for verifying whether the system will always result in a desirable state.

Obligation States

Each obligation implies a set of states for the system with regard to that obligation. We classify obligations into three types:

- An obligation to *achieve* some state G, for example to pay an amount
- An obligation to *maintain* some state H, for example to keep 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, for example when aircraft X requires servicing, to service X in an acceptable time.

In part, the critical states of an obligation can be specified independently of the application in which the obligation has force, as we do below for each of the three classes of obligation named.

For an achievement obligation, there are three critical states: *Pre-achievement, Succeeded* and *Failed*. Each has particular properties with regard to the goal state G, as shown in Table 4 (top). In *Pre-achievement*, the goal state is achievable but not yet achieved; in *Succeeded*, the system is in the goal state; and in *Failed*, the goal state is no longer achievable.

State	**Properties**
Pre-achievement	Not G G achievable Service obliged to achieve G
Succeeded	G
Failed	G unachievable Service obliged to achieve G
Cancelled	No service obliged to achieve G

Sub-State	**Additional Properties**
Initial	
Danger	G in danger of becoming unachievable
Likely Success	Success G' achieved, where G' is a significant subset of G

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

Similarly, a maintenance obligation implies three significant states, as shown in Table 5 (top). In the *Maintained* state, the system is in the goal state; in *Succeeded*, the system can no longer leave the goal state; in *Failed*, the system has left the goal state.

State	**Properties**
Maintained	H Not H achievable Service obliged to maintain H
Succeeded	Not H unachievable
Failed	Not H Service obliged to maintain H
Cancelled	No service obliged to maintain H

Sub-State	**Additional Properties**
Initial	
Danger	Not H in danger of becoming true

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

As described above, a behaviour obligation triggers the imposition of a further obligation, which we will call the *triggered obligation*, on particular events occurring. The significant states of a behaviour obligation depend on the triggered obligation, but the behaviour obligation has some states of its own, as shown in the top of Table 6. In the *Pre-trigger* state, the triggering event has not yet occurred; in the *Reaction Active* state, an event has occurred

but the obligation it has triggered into taking force has not yet reached a *Succeeded* or *Failed* state; in Reaction Failed, that reaction obligation has reached a *Failed* state, and so the behaviour obligation as a whole has failed; in *Reaction Succeeded* state, the particular reaction obligation has succeeded; and in *Succeeded*, no more applicable events can ever occur and so the behaviour obligation as a whole has succeeded. All obligations also imply a state, *Cancelled*, when the obligation no longer holds.

State	**Properties**
Pre-trigger	No new E(X) has occurred Service obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Active	E(a) occurred As Pre-achievement, Maintenance or Pre-trigger state for G(a)/B(a) Service obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Failed	E(a) occurred As respective Failure state for reaction G(a) or B(a) Service obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Succeeded	E(a) occurred As respective Succeeded state for reaction G(a) or B(a) Service obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Succeeded	E(X) can never occur again
Cancelled	No service obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)

Sub-State	**Additional Properties**
Initial	
Imminent	E(X) is likely to occur imminently
Likely Complete	E(X) is unlikely to occur again

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

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 are shown in the bottom portions of Tables 4, 5 and 6. 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). Or, if an obligation is being fulfilled unexpectedly easily, an application may take advantage by transferring resources being used in support of this obligation to other tasks, for example the *Likely Complete* critical state in Table 6. Interpretation of concepts such as danger or likelihood are application-specific.

Example

As an example, in Table 7 we enumerate critical states for an achievement obligation, O2 in the long-term aftercare contract. It is an achievement obligation as it describes an eventual state of the system, i.e. 90% of servicing cases were performed in the preferred time period. When the contract first comes into force, i.e. 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, 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 5% of cases were performed within the preferred time, while *Pre-achievement: Danger* holds where between

5% and 10% exceeded that time. The value of taking account of these two states is that transfer of resources between fulfilment 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.

Pre-achievement: Initial	Less than (estimated) 5% of servicing cases performed and within contract period
Pre-achievement: Satisfactory	Over 5% of cases performed, less than 5% exceeded preferred time and within contract period
Pre-achievement: Danger	Between 5% and 10% of cases exceeded preferred time and within contract period
Succeeded	Less than 10% of cases exceeded preferred time and beyond contract period
Failed	More than 10% of cases exceeded preferred time

Table 7. States of aftercare contract obligation O2

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.

The life-cycle of a contract is viewed as follows. First, a potential contract party discovers services which may provide the functionality they require, and specifies a proposal and negotiates over it with the potential service providers. As part of this process, the parties will agree to how compliance to the obligations will be monitored (see Third-Part Monitoring below). The contract will be agreed to and preserved in independent storage. The contract parties can then perform actions to their obligations in accordance with prohibitions and permissions. This behaviour will be observed and checked by the agreed independent parties. The contract will eventually terminate, possibly leading to renewal if the service required is ongoing.

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 fulfilment 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* services must play to fulfil the administration duties implied. Below, we examine each process type in turn.

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 services, and signs it. It then uses a *registry* to discover services that may fulfil the unassigned contract roles. For each unassigned role in turn, it offers the proposal to a service, 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 a service discovers a contract *template* that may be instantiated in a way that fulfils 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.

Maintenance and Update

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

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 services. 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 the previous section.

Observation of Fulfilment Process: An *observer* observes state changes to determine whether contractual obligations are being fulfilled. It can notify other services when an obligation is being violated or in danger of violation. An observer X is in an obligation pattern of the following form: “X is obliged to observe for critical state S of contract clause C, and notify registered listeners when it occurs.”

Management of Fulfilment Process: A *manager* is a service 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 service 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 service 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 is obliged, whenever the system reaches a critical state S of contract clause C, to perform action A.”

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.

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, for example when 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 an event occurs (for example, 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).

Roles	Checker, Manufacturer, Operator
Obligations	Checker monitors the number of engines available to the manufacturer at a given location that are suitable for a given operator, and notifies registered services if it falls below a minimum quantity.
Roles	Enforcer, Checker
Obligations	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.

Table 8. Engine supply checking contract (top) and Engine supply enforcement contract (bottom)

Administrative Roles and Components

The processes above all require the fulfilment of particular administrative roles, for example a contract store, registry, observer, or 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. Further details on the specification of these components are available from the CONTRACT website (Contract, 2008).

THIRD-PARTY MONITORING

Detecting and handling obligation violations is essential, in particular as contracts often specify how to react in such circumstances, e.g. the operator taking a penalty for late servicing in clause R3 in Table 3. However, this requires independent contract parties to hold a consistent view of whether a violation has occurred – which is not always trivial to achieve in any distributed system.

As part of our architecture, we attempt to meet this need by two complementary measures. Full details would exceed the scope of this chapter, but we summarise the ideas below and point interested readers to existing publications (Modgil et al. 2009; Meneguzzi et al. 2009).

First, we allow contract parties to name and agree in the contract which observers are jointly *trusted* by all signing parties. The reason for this joint trustworthiness in an observer cannot be application-independent. For example, in a financial situation, a bank may be a

trusted third-party observer, whereas in a remote procedure call we may have to rely on the combined reports of the caller and the callee to obtain a trustworthy observation. We, therefore, simply provide the mechanism to declare trusted observers and leave establishing trust to other mechanisms.

Second, we provide a generic, independent monitoring component. This takes as input, a translation of the contract into *augmented transition networks (ATNs)*. Each ATN corresponds to one clause, and consists of a series of nodes connected arcs labelled with observable messages. As messages pass between contract parties, trusted observers report this to the independent monitor, which follows the arcs in the ATNs. This tracking ultimately allows the monitor to declare (to manager services), that a clause is fulfilled or violated. The trace of messages observed also acts as a means to *explain* violations, so providing some supporting evidence for redress.

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 (Lopes Cardoso & Oliveria, 2000), and the administration of contracts can integrate with other useful behaviour, such as observation of fulfilment and violation of obligations potentially feeding into a longer-term assessment of systems (Duran, Torres da Silva, & de Lucena, 2007). Work on multi-party contracts (Xu, 2004) adopt modelling techniques specifically designed to enable detection of parties responsible for contract violation, but do not use normative concepts to regulate behaviour, or model other contract administration processes.

In addition, the wider domains of normative systems and agreement in service-oriented architectures informs our work. Concepts such as norms specifying patterns of behaviour, contract clauses as concrete representations of dynamic norms, management or enforcement of norms itself being a norm, are all already established in the literature (Dellarocas, 2000; Duran, Torres da Silva, & de Lucena, 2007; García-Camino, 2007; Lopez y Lopez, Luck, & d'Inverno, 2005).

However, the approach in this chapter is distinct in that it is concerned with the development of practical system deployments for business scenarios. In particular, business systems operate in the context of wider organisational and inter-organisational processes, so that commitments, providing assurance over the actions of others assumes great importance. While potentially less flexible over the short term, explicit contracts provide just such commitments and are therefore more appropriate for business systems than more flexible, less predictable *ad hoc* approaches (Ghijsen, Jansweijer, & Wielinga, 2007; Muntaner-Perich, de la Rosa, & Esteva, 2007).

We also believe our system to be more widely applicable than some other approaches. By classifying processes into types with different instantiations, the architecture can be incorporated into a wider range of application domains and deployments than fixed protocols would allow. In addition, we describe how administrative functions, such as storing or updating a contract, can be achieved. This contrasts with specifications such as WS-Agreement and Web Services Service Level Agreement, where the specifications cover only part of the necessary administration (Andrieux & Czajkowski, 2004; Ludwig, Keller, Dan, King, & Franck, 2003). Abstract architectures for electronic contracting, and associated case studies, have been described elsewhere; most notably in the work of Grefen and Angelov (2002; Angelov & Grefen, 2006). However, accommodation of deontic specifications in order to regulate service behaviour is not modelled in this work. Our approach aims for broad

observation and management of obligations and prohibitions, so as to verify whether they are being achieved, prevent failure when in danger of violation and take advantage of success when obligations are being easily met. Some existing work does consider system states with regard to contract clauses (Lopes Cardoso & Oliveira, 2000), but none, to our knowledge, classifies obligations and the critical states they imply as we have done in this chapter, a necessary pre-requisite to observing and managing obligation fulfilment in accordance with a particular application.

Others have raised the issue that observers and managers have, themselves, to be observed and managed (Jones & Sergot, 1993). Here, by modelling observers and managers as services, we allow for the same contract framework to apply to them. However, this clearly has its limits and at some point *trust* between businesses must be explicitly modelled in the system, a topic to be addressed in future work.

CONCLUSIONS AND FUTURE WORK

In this chapter, 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 service-oriented systems to be verified on the basis of their contracts, building on work by Lomuscio et al. on deontic interpreted systems (Lomuscio & Sergot, 2003). While this verification is beyond the scope of this chapter, 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 have begun to construct a methodology to guide development of applications that use electronic contracts through the process from conceptual framework to deployment. To ensure wide applicability, this will be applied to CONTRACT's other test applications in insurance settlement, software provisioning and certification testing.

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 chapter 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

Andrieux, A., Czajkowski K., Dan A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., & Xu, M. (2004). Web services agreement specification (WS-Agreement) (Tech. Rep). Global Grid Forum.

Angelov, S., & Grefen, P. (2006). A case study on electronic contracting in on-line advertising - status and prospects. In L. M. Camarinha-Matos, H. Afsarmanesh, and M. OIlus (Ed.), *Network-Centric Collaboration and Supporting Frameworks - Proceedings 7th IFIP Working Conference on Virtual Enterprises*, (pp. 419–428). Springer.

Conte, R., & Castelfranchi, C.. (1993). Norms as mental objects. From normative beliefs to normative goals. In C. Castelfranchi and J.-P. Mueller (Ed.), 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '93 (pp. 186–196). Springer.

Conte, R., & Falcone, R., & Sartor, G. (1999). Agents and norms: How to fill the gap? *Artificial Intelligence and Law 7*, 1–15.

CONTRACT. Project website. Retrieved November 12, 2008, from http://www.ist-contract.org/

Dellarocas, C. (2000). Contractual agent societies: Negotiated shared context and social control in open multi-agent systems. In R. Conte and C. Dellarocas (Ed.), *Social Order in Multiagent Systems* (pp. 113-133). Kluwer Academic Publishers.

Duran, F., Torres da Silva, V., & de Lucena, C. J. P. (2007). Using testimonies to enforce the behaviour of agents. In J. Sichman and Sascha Ossowski (Ed.), *Coordination, Organizations, Institutions, and Norms in Agent Systems III* (pp. 25–36). Springer.

García-Camino, A. Ignoring, forcing and expecting concurrent events in electronic institutions. (2007). In J. Sichman and Sascha Ossowski (Ed.), *Coordination, Organizations, Institutions, and Norms in Agent Systems III* (pp. 15-26). Springer.

Ghijsen, M., Jansweijer, W., & Wielinga, R. (2007). Towards a framework for agent coordination and reorganization, AgentCore. In J. Sichman and Sascha Ossowski (Ed.), *Coordination, Organizations, Institutions, and Norms in Agent Systems III* (pp. 13–24). Springer.

Grefen, P., & Angelov, S. (2002). 'On τ, μ, π and ε-contracting. In C. Bussler, R. Hull, S. McIlraith, M. E. Orlowska, B. Pernici, and J. Yang (Ed.), *Proceedings of the CAiSE Workshop on Web Services, e-Business, and the Semantic Web* (pp. 68–77). Springer.

Jakob, M., Pechoucek, M., Chábera, J., Miles, S., Luck, M., Oren, N., Kollingbaum, M., Holt, C., Vázquez, J., Storms, P. & Dehn, M. (2008). Case studies for contract-based systems. In M. Berger, B. Burg, and S. Nishiyama (Eds.), *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)- Industrial and Applications Track* (pp. 55-62). INESC.

Jones, A. J, I., & Sergot, M. J. (1993). On the Characterisation of Law and Computer Systems: The Normative Systems Perspective. In J.-J.Ch. Meyer and R.J. Wieringa (Ed.), Deontic Logic in Computer Science: Normative System Specification (pp. 275–307). John Wiley & Sons.

Lomuscio, A., & Sergot, M. (2003). Deontic interpreted systems. *Studia Logica, 75*(1), 63-92.

Lopes Cardoso, H., & Oliveira, E. (2000). Using and evaluating adaptive agents for electronic commerce negotiation. In M. C. Monard and J. Simão Sichman (Ed.), *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI: Advances in Artificial Intelligence* (pp. 96–105). Springer.

Lopes Cardoso, H. & Oliveira, E. A contract model for electronic institutions. In J. Sichman and Sascha Ossowski (Ed.), *Coordination, Organizations, Institutions, and Norms in Agent Systems III* (pp. 73–84). Springer.

Lopez y Lopez, F., Luck, M., & d'Inverno, M. (2005). A normative framework for agent-based systems. *Computational and Mathematical Organization Theory, 12*(2–3), 227–250.

Lost Wax. Aerogility. Retrieved November 6, 2008, from http://www.aerogility.com/

Ludwig, H., Keller, A., Dan, A., King, R. P., & Franck, R. (2003). Web service level agreement (WSLA), language specification (Tech. Rep.). IBM Corporation.

Meneguzzi, F., Modgil, S., Oren, N., Miles, S., Luck, M., Faci, N., Holt, C. & Smith, M. (2009). Monitoring and Explanation of Contract Execution: A Case Study in the Aerospace Domain. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009) – Industrial and Applications Track*, to appear.

Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., & Luck, M. (2009). A Framework for Monitoring Agent-Based Normative Systems. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, to appear.

Muntaner-Perich, E., de la Rosa, J. L., & Esteva, R. (2007). Towards a formalisation of dynamic electronic institutions. In J. Sichman and Sascha Ossowski (Ed.), *Coordination, Organizations, Institutions, and Norms in Agent Systems III* (pp. 61–72). Springer.

W3C. Web Services Architecture. Retrieved November 11, 2008, from http://www.w3.org/TR/ws-arch/

Xu, L. (2004). A multi-party contract model. *ACM SIGecom Exchanges, 5*(1), 13–23.

KEY TERMS AND DEFINITIONS

Role	A named part that can be played by a service's agent in a system.
Obligation	A statement that an agent playing a given role should do something.
Prohibition	A statement that an agent playing a given role should not do something.
Permission	An exception to a prohibition for an agent playing a given role under given circumstances.
Clause	An obligation, prohibition or permission.
Assignment	A statement that an agent should play a given role.
Proposal	A document containing a set of clauses and assignments, where every role referred to which each clause applies has been assigned to an agent.
Contract	A proposal to which all assigned agents have agreed.