Case Studies for Contract-based Systems

Michal Jakob
Michal Pěchouček
Czech Technical University in Prague
Technická 27
166 27 Praha 6, Czech Republic
jakob@labe.felk.cvut.cz

Jiří Chábera
Certicon a.s.
Václavská 12/316
120 00 Praha 2, Czech Republic

Simon Miles,
Michael Luck
Nir Oren,
Martin Kollingbaum
King's College London
London WC2R 2LS, UK
simon.miles@kcl.ac.uk

Camden Holt
Lost Wax
72 Lower Mortlake Road
Richmond TW9 2JY, UK

Javier Vazquez
Universitat Politècnica de Catalunya
C/ Jordi-Girona 1-3
E-08034 Barcelona, Spain

Patrick Storms
Y'All BV
Grotestraat 182
NL-5141HD Waalwijk, Netherlands

Martin Dehn
Fujitsu EST
Frankfurter Ring 211
80807 München, Germany

ABSTRACT

Of the ways in which agent behaviour can be regulated in a multi-agent system, electronic contracting – based on explicit representation of different parties' responsibilities, and the agreement of all parties to them – has significant potential for modern industrial applications. Based on this assumption, the CONTRACT project aims to develop and apply electronic contracting and contract-based monitoring and verification techniques in real world applications. This paper presents results from the initial phase of the project, which focused on requirement solicitation and analysis. Specifically, we survey four use cases from diverse industrial applications, examine how they can benefit from an agent-based electronic contracting infrastructure and outline the technical requirements that would be placed on such an infrastructure. The designed CONTRACT architecture is presented and we describe how it may fulfil these requirements. In addition to motivating our work on the contract-based infrastructure, the paper aims to provide a much needed community resource in terms of use case themselves and to provide a clear commercial context for the development of work on contract-based system.

Keywords

contract-based systems, contracts, multi-agent systems, services, monitoring, architecture

1 INTRODUCTION

Of the ways in which agent behaviour can be regulated in a multi-agent system, *electronic contracting* may be the most suitable for industrial applications. In part, this is because it explicates different parties' responsibilities, and the agreement of all parties to them, allowing businesses to operate with expectations of the behaviour of others, but providing flexibility in how they fulfil their own obligations. Additionally, it mirrors existing (non-electronic) practice, aiding adoption.

In technical terms, an electronic contracting approach, where publicly declared commitments in the form of contract documents between application-specific agents are created and reasoned about, provides the following benefits:

- Contracts abstract away detailed implementation information of individual actors and model dependencies between them explicitly.
- Contracts are publicly observable, improving run-time monitoring of the system as a whole.
- Contracts can model functional as well as non-functional properties of an interaction.
- Contracts correspond to the types of relationships occurring in organisations in which business software systems are deployed and, therefore, allow software engineers to perform a more intuitive analysis.
- Contracts can be linked to more general social structures such as social laws, rules, norms and institutions, which provide useful metaphors for system design.

While contract-based infrastructures have previously been proposed, e.g. [4, 6], with each applied to particular scenarios, without a principled study of requirements on electronic contracting across a range of industrial use cases, there is a danger that they may be designed in a non-reusable way, fulfilling only individual application needs. An analysis of electronic contracting requirements may also highlight open questions for research into open agent systems.

The CONTRACT project [11] aims to develop and apply electronic contracting and contract-based monitoring and verification techniques in real world applications. However, rather than focus on simply building particular instantiations of an architecture, the project also seeks to address one of the fundamental barriers to the adoption of agent technologies, the lack of a sufficient range of case studies [3].

While several efforts have been made to report on particular applications in support of this aim (for example, [1] and previous AAMAS Industry Tracks), these have tended to be one-off applications that are inadequate to show broad applicability and relevance of generic techniques across a range of domains or indeed applications themselves. Some efforts have sought to use similar examples to illustrate the use of particular techniques, such as the conference management system in the case of methodologies for agent oriented software engineering (e.g., [2]), but these are to some extent toy examples rather than real use cases.

In contrast, our effort to develop a contract architecture is unashamedly tied directly to our efforts in eliciting requirements from real business cases, and developing prototype systems for real applications. In this paper, therefore, we survey four use cases

from diverse industrial applications, examine how they can benefit from an agent-based electronic contracting infrastructure, draw out the technical requirements this would place on such an infrastructure, and describe how our architecture may fulfil them. The aim is threefold: first to motivate and inform our work on the contract-based infrastructure; second to provide a community resource in terms of the use cases themselves, to facilitate understanding, comparison and evaluation of competing techniques and architectures; and third to provide a clear commercial context for the development of work in this area, and demonstrating the business benefit to be gained.

2 USE CASES

To this end, the CONTRACT project has sought to capture requirements in four application areas, each represented by one industrial partner: Modular certification testing (provided by Certicon), Service procurement in the insurance industry (Y'All), Aircraft engine aftercare (Lost Wax), and Service level agreement management in software engineering (Fujitsu).

The use cases were captured in a multi-round dialogue, the basis of which comprised detailed use case templates completed by each industrial partner. The emphasis was on capturing business relationships and contracts in each domain as these provide a solid foundation for the description of technical requirements and specific use case scenarios. In this paper, we provide only a brief overview of each of the four use cases – for detailed descriptions, see [10]. Each overview below consists of a description of the application area, business entities and services involved in the business model, and selected main contracts formally underpinning business interactions in the domain.

2.1 Use Case 1: Modular Certification Testing

Modular certification testing allows a large number of heterogeneous and independent businesses to flexibly collaborate on the provision of certification services. The model has been applied to computer literacy testing using the European Computer Driving Licence (ECDL[1]) concept, first in the Czech Republic and later in Slovakia, and can be equally well applied to other certification programmes of similar structure.

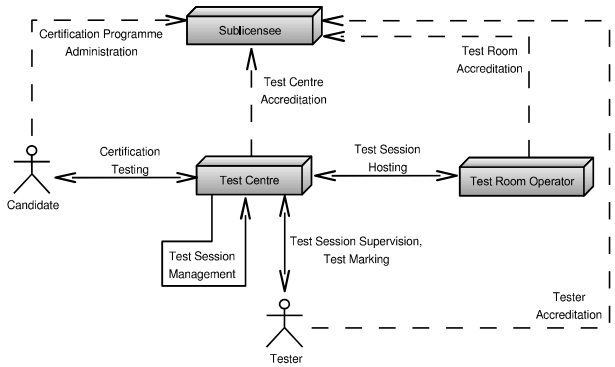

Figure 1: Overview of the certification business domain showing business entities and services they provide

[1]The European Computer Driving Licence® (or ECDL) is the registered trade mark of The European Computer Driving Licence Foundation Limited in Ireland and other countries.

2.1.1 Actors

There are three categories of business entities cooperating in the certification business domain (see Figure 1 for an overview):

- **accreditation institutions** (the national licensee and sub-licensees) are the subjects responsible for the proper operation of the certification programme;
- **suppliers** (accredited test centres, test room operators and testers) provide services required for certification testing (both to customers and between themselves);
- and **consumers** are the recipients of certification testing services.

From the perspective of the use case, the most important business entities in the domain are the accredited test centres. Test centres offer certification testing to candidates who wants to get certified in the respective certification programme. Candidate testing is a complex process and requires a combination of multiple services, including the provision of testing facilities (test rooms), the supervision of test sessions and marking of tests. Although a test centre can provide all these service internally, in many cases it is advantageous to procure these services on the supplier market. Business cooperation between test centres and their subcontractors on the supplier market is the primary focus of the use case.

2.1.2 Contracts

Out of a number of different contracts governing the provision of services in the domain, the use case primarily focuses on the following:

- **Certification Test Contract** is a business-to-consumer contract specifying the terms and conditions of certification testing between a certification candidate and an accredited test centre.
- **Test Room Rental** and **Tester Hire** are business-to-business contracts governing the provision of testing facilities and test supervision/test marking services, respectively.

2.1.3 Benefits of Contract-based Technology

Contract-based technology should primarily be applied to enable test centres to establish and manage supplier relationships with accredited test room operators and accredited testers, as well as to their relationships to customers. The application of electronic contracting should lead to increased flexibility by which test centres can respond to consumer demand for certification testing by forming on-demand business partnerships. Automated contract management and monitoring should result in significant reduction in labour cost, better utilization of resources, higher reliability and consequently improved quality of service to the consumer.

2.2 Use Case 2: Dynamic Insurance Settlement

The Insurance domain relies heavily on traditional ways of claims handling. Every aspect of a claim is dealt with by different specialists working in different departments of a company, or in different companies involved in the total claims handling process. Therefore, the whole process is very costly. Nowadays, the insurance market is increasingly seeking ways to economize on claims handling by increasing the level of process automation and improving the integration of the different parties (e.g., victims, witnesses, surveyors/experts, lawyers, insurance companies, middlemen and doctors) and systems involved.

The Dynamic insurance settlement use case describes a *CarRepairGrid* system to be built for an envisioned new company, *DamageSecure (Figure 2)*, which manages and controls all businesses involved in dealing with car damage claims for several insurance companies. The goal of DamageSecure is to enhance the quality and efficiency of the total damage claims handling process between consumers, damage repair companies and insurance companies. CarRepairGrid reasons about repairs of damaged cars that are insured at insurance companies in order to settle the claim under the best circumstances (lowest prices, highest quality, as soon as possible, as close as possible, etc).

2.2.1 Actors

In all, five types of entities are involved in business transactions in the insurance domain:

- **Customers** who are the holders of insurance policies
- **Repair companies** which repair and replace damages.
- **Insurance companies** which inspect, approve and pay approved claims.
- **DamageSecure** which operates the CarRepairGrid and acts as a broker between the other parties in the domain. It offers services to both Insurance Companies (centralized procurement of services) and Repair Companies (centralized selling of services to Insurance Companies).
- **Experts** who perform counter-expertise for Damage Secure

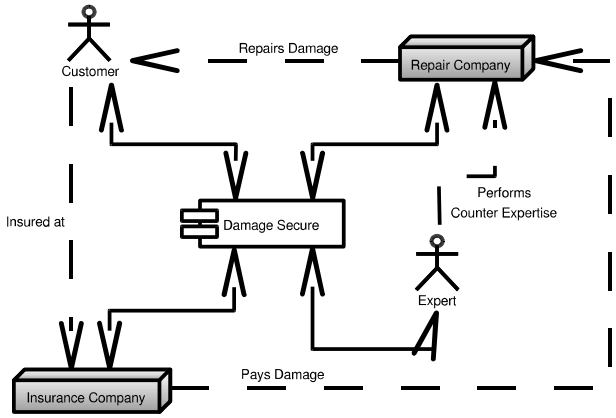

Figure 2: Overview diagram of the insurance domain

2.2.2 Contracts

In terms of contracts regulating business transactions in the domain, the use case is focused on:

- **Overall Contract** which specifies the relation between an Insurance Company and DamageSecure.
- **Repair Contract** which specifies the relation between DamageSecure and a Repair Company.

The Overall Contract is a good example of a long-term contract between a supplier and a broker defining a contractual framework within which targeted, short-term repair contracts are established. Repair contracts are the primary interest from the project's perspective because they have significantly higher volume and dynamics than the rather static Overall Contracts.

2.2.3 Benefits of Contract-based Technology

The biggest opportunity for the application of automated contract technology lies with DamageSecure. Contract technology can significantly improve insurance claims handling by enabling automated, contract-based matching of repair requests to repair companies as well as automated monitoring of the claims handling process. This is expected to lead to a decreased cost due to reduced manual labour, increased competition and improved efficiency of the claims handling market. With estimated 100 000 damages per year, the automation of CarRepairGrid could potentially save 172M Euros. Greater variety of customized insurance policies and a wider range of repair options together with accelerated claim handling are the other benefits expected from automated contract-based claim handling.

2.3 Use Case 3: Aerospace Aftermarket

The aerospace aftermarket is increasingly populated by customers buying a service rather than a product. Here, the aircraft engine manufacturer is responsible for providing a specified number of serviceable engines so that the airline operator's aircraft can be kept flying. The engine manufacturer is paid by the hour when the engines are available and may face a penalty if planes are on the ground waiting for a serviceable engine. 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.

A unique feature of this use case is the Aerogility system, an agent-based decision support tool developed by LostWax to simulate aerospace aftercare. Aerogility will be able to show the effects of variations in contracts – not only in profitability but also the different aftercare strategies needed to meet the revised contract.

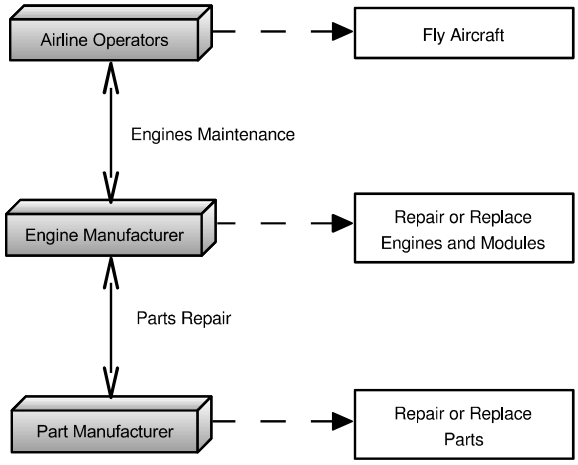

Figure 3: Actors and services in the Aerospace aftermarket domain

2.3.1 Actors

From the perspective of the use case, there are three relevant types of businesses in the domain (see Figure 3):

- **Airline Operators** are customers for aftercare contracts. Each operator has its own fleet of aircraft which need to be kept in service.
- **Engine Manufacturers** are suppliers of aftercare contracts. They attempt to fulfil the service levels specified in the contracts or else incur penalties.
- **Part Manufacturers** manufacture and deliver engine parts. They have contracts to supply new or refurbished parts of given types to engine manufacturers**.**

2.3.2 Contracts

The provision of services is regulated by contractual agreements between respective parties. The use case explicitly models the following two contracts:

- **Aftercare Contract** which specifies the terms and conditions under which the aircraft manufacturer undertakes to supply and maintain engines for the operator's aircraft. The Aftercare contract specifies e.g. serviceable engine rate, airports and routes on which the operator operates, penalties applicable if agreed service levels are not met, etc.
- **Parts Supply Contract** which regulates how the engine manufacturer asks a part manufacturer to make and deliver new parts or refurbished old parts of a given type over a given period. Part Supply Contract specifies e.g. locations where part supplies should be delivered, cost of parts, delivery times, etc.

Note that although Aftercare and Parts Supply contracts can be relatively long-term, they provide a framework under which a large number of specific service requests are handled. The Parts Supply Contract is more speculative and simpler than the Aftercare Contract, but is potentially more dynamically and frequently created and may provide a useful extra test of the developed contract-based technology.

2.3.3 Benefits of Contract-based Technology

In contrast to the other use cases, which envision the deployment of the developed technology directly into the application domain, here the contracting technology will be used in the simulated environment provided by the Aerogility simulation system. Rather than automating operation in the application domain, the emphasis of the use case is on investigating collaboration patterns emerging from contract-driven interaction between parties in the domain. By using contract monitoring and verification techniques, operators and manufacturers can investigate the properties of contracts they are involved in, and analyse the impact of their potential modification. Potentially, this could lead to recommendations of change on the basis of current contracts to be fulfilled.

2.4 Use Case 4: SLAs in Software Engineering

Service level agreements (SLA) play an increasing role in IT service management. Customers expect from their IT service providers high quality and flexible services at reasonable cost, meeting customer requirements. One of the major tasks of service level management (SLM) consists in grasping requirements of end users and offering respective services. In this process, the quality and quantity of services at acceptable costs are negotiated, defined, measured, and continuously improved. To ensure stable and reliable operation of the IT infrastructure of an organization with a high degree of performance, the responsible managers within the organization establish, verify and monitor contracts with service providers.

2.4.1 Actors

Business collaboration in the domain takes place between a customer who has an identified need for a specific IT service to contribute to its business, and an IT service provider who can deliver the requested services to the customer. IT services provided range from processing a single help desk call to the development of a new software system. From this broad range, the use case focuses primarily on software engineering services for developing new systems or enhancing existing software solutions based on change requests issued by customers

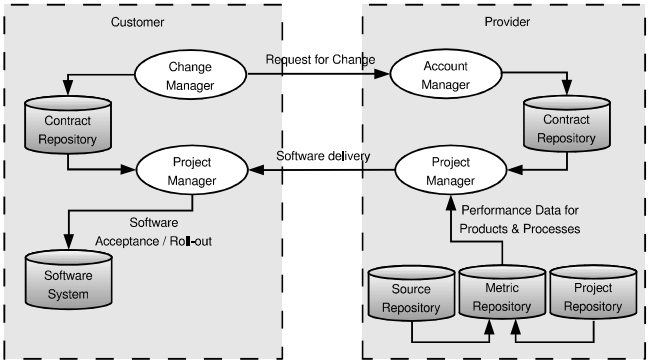

Figure 4: Actors and resources involved in the provision of software engineering services

Figure 4 depicts business collaboration patterns corresponding to the provision of software engineering services, including a high-level view of internal services and resources involved at each party.

2.4.2 Contracts

The provision of services in the use case domain is governed by two key contracts:

- **Change Request Agreement** contract which specifies the characteristics of the delivery process of the software modification and the quality properties of the software to be delivered.
- **Service Level Agreement** contract which specifies the service level for technical support of a software system in terms of characteristics of the delivery process, e.g., response times for reported incidents.

In addition to a simple scenario consisting of just one customer and one provider, more complicated collaboration patterns are also envisioned. A more complicated scenario can involve a customer and a provider with one main agreement defining default conditions and obligations, and any number of sub-contracts, e.g., for single change requests, which have to meet the rules of the main agreement, while they can contain extensions as long as they remain consistent with the conditions and obligations of the main agreement.

2.4.3 Benefits of Contract-based Technology

Due to compliance issues and legal regulations, IT service providers and customers increasingly tend to measure the quality

of software products and the performance of software engineering and technical support processes. Automated contract technology can greatly help this task by supporting continuous monitoring of performance indicators before and during service delivery, issuing early warnings in case of the risk of not meeting the conditions and obligations set in the agreement, and thus preventing contract violations. This decreases the risk of delivering IT services too late or with low quality, reduces costs for penalties and increases customer satisfaction.

Representation of IT service agreements in a machine interpretable way can also greatly improve contract management, providing an accurate and up-to-date view of contracts and the state of corresponding commitments, and opening a way for the optimization of the provisioning process. In scenarios with multiple dependent contracts, requesting changes to already established contracts can be better controlled, avoiding inconsistencies and unnecessary business disputes. In addition, well-defined syntax and structure of electronic contracts allows a clear definition of the obligations for the IT service provider and the respective expectations of the customer.

3 REQUIREMENTS

Following solicitation of the use cases, we performed an analysis to determine technical requirements. Analysis was performed on the complete descriptions of the use cases [10], which provide significantly more information. Detailed analysis has been omitted due to space constraints, but we hope the connection between use cases and extracted requirements will be evident to the reader.

Categories of requirements were identified: general, storage and representation, life-cycle management, monitoring and deployment[2], described below.

3.1 General Requirements

The first group contains generic requirements which concern multiple areas of functionality of a contract-based system.

R1	The system supports all stages of the contract lifecycle, namely negotiation, creation, execution and termination.
R2	The system allows for short-term contracts created only for the purpose of fulfilling one-off service requests (as opposed to long-running contracts)
R3	The system allows for contracts that operate over long, defined periods (as opposed to short-lived one-off requests)
R4	The system allows for obligations that come into force on being triggered by (possibly unpredictable) domain events.
R5	Exposure of sensitive information about internal processes of each business partner is minimized.
R6	Contracts are secured from unauthorized access, in particular they are not revealed to competitors.
R7	When required, important decisions can be escalated to a human decision maker for ratification.
R8	A party can have a number of different contracts with different parties active at the same time

Table 1. General Requirements

[2] In the CONTRACT project, requirements on design-time verification of contract-based system have also been analysed. Although a very important component of the project, verification have been omitted in this paper due to lack of space required for the proper exposition of the verification functionality. Please refer to [10] and [11] for more information on this aspect of contract-based systems.

3.2 Storage

The ability to store and manipulate contracts is crucial for any contract-based system. It represents the basis on which the more advanced functionality of contract management and monitoring can be built. An important requirement from all four use cases is support for contract templates as a way to represent and manipulate classes of contracts that can be instantiated by inserting details.

R9	The system stores contracts throughout their life-cycle. Basic persistent storage operations (create, retrieve, update and delete) are supported.
R10	The system can handle multiple versions of the same contract.
R11	The system supports contract templates and operations with them (storage, creation, modification etc.)
R12	Contract hierarchies (such as a master contract and all its subcontracts) are supported for contracts and contract templates.
R13	Contracts can be annotated with metadata which can be used for the organisation of contracts (e.g. searching, sorting and grouping)
R14	Contracts and contract templates can be searched/browsed using a rich set of criteria (contract status, contract parties, time interval etc.)
R15	Dependencies between contracts are represented and can be analysed.

Table 2. Contract Storage and Representation Requirements

3.3 Life-Cycle Management

During their lifetime, contracts can go through a number of different stages. At each stage, each contract is ascribed a contract status which denotes how the contract should be currently interpreted. The ability to track and manipulate contract status throughout the contract life-cycle has been identified as a third important group of requirements.

R16	Authorized users can manually change, stop or re-negotiate (modify obligations) contracts when needed.
R17	CONTRACT should allow for one party to break a contract where the other party/parties do not fulfil their obligations.
R18	There is a mechanism allowing contract parties to terminate their contracts.
R19	Existing contracts can be extended / renewed, potentially with modified parameters.
R20	Contract status can be either derived automatically from other back-office systems or entered manually by authorized staff.
R21	Contract instances can be created by filling in details (such as price, or delivery date) into pre-specified contract templates.

Table 3. Contract Life-Cycle Management Requirements

3.4 Monitoring

The ability to evaluate at runtime to what extent the behaviours of agents in a system conform to agreements made presents a strong motivator for the introduction of explicit contracts into information systems. Requirements in this regard concern the ability to determine fulfilment states from the individual clause up to the whole-contract level, both for the internal use by an agent (for decision making, resource allocation etc.) and for use by independent third-parties.

R22	The system provides information on the fulfilment state of contracts.
R23	The system can detect whether a particular clause in being fulfilled
R24	The system can monitor the levels of fulfilment defined in terms of process and service-related metrics. Fulfilment metrics can either be evaluated automatically or entered manually by a human expert. Aggregated degree of fulfilment can be calculated from partial metrics.
R25	The system detects and reports violations of active contracts.
R26	The system can issue warnings when there is a risk of contract violation.
R27	The system can evaluate the degree to which a party adheres to a contract. The assessment can be also carried out over a group of contracts or contract parties. Both run-time analysis and historic analysis based on recorded data should be supported.
R28	There is a mechanism for resolving disputes between contract parties over the fulfilment of contract provisions. The mechanism can involve an independent third party.

Table 4. Monitoring Requirements

3.5 Deployment

The use cases described anticipate different ways in which the components implementing the contracting functionality would best be deployed in a real-world system.

R29	Centralized deployment is supported whereby contracts are stored with a single system and are accessed remotely by all partners.
R30	For security and sensitive information protection, distributed deployment should be considered whereby contracts are stored and managed in a distributed way by systems deployed at partner sites.
R31	For security reasons, hosting the CONTRACT engine by an independent 3rd party should be considered.

Table 5. Deployment Requirements

4 ARCHITECTURE

The requirements of the previous section can be implemented in different ways. The CONTRACT project proposed a *framework* and *architecture* to, respectively, provide a conceptual mapping of applications to contract technology and provide an infrastructure for the administration of contract-related processes. In the following, we briefly describe the key design decisions of the framework and architecture, starting from basic concepts. The full technical details, including design patterns, service interfaces and agent reasoning behaviour, are outside the scope of this paper. Where a requirement is addressed by the description we add annotation ***[Rn]*** where Rn is the requirement's reference above.

4.1 Overall Structure

The models and procedures comprising the CONTRACT framework and architecture are shown in Figure 5. The primary component of this is the framework itself, depicted at the top of the figure.

From the framework specification of a given application, other important information is derived. First, off-line verification mechanisms can check whether the contracts to be established obey particular properties, such as being achievable, given the possible states the world can reach. From this, and the contracts themselves, we can determine which states are critical to observe during execution to ensure appropriate behaviour. 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 of not being fulfilled, etc.

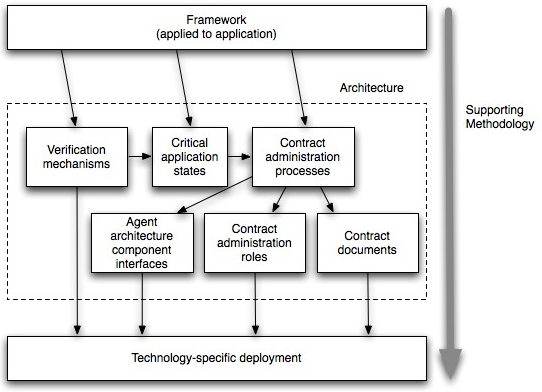

Figure 5: The overall structure of the CONTRACT architecture and framework

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.

4.2 Framework

A *contract* documents *obligations, permissions* and *prohibitions* (collectively *clauses*) on agents and is *agreed to* by those agents. The agents obliged, permitted and prohibited in a contract are the *parties* to that contract. One agent can hold multiple contracts with the same or different parties ***[R8]***. A contract specifies *contract roles*, which are named roles played by agents with regard to the contract. Each clause is documented as applying to contract roles, and agents are assigned to the contract roles in the contract, the combination of which means that the clauses apply to those agents. A *contract proposal* is a contract that has not yet been agreed, but may be agreed later (often, a contract proposal is a contract under negotiation).

One contract, the *child contract*, may reference *parent contracts*, which are other contracts whose contents are implicitly included in the child contract. The roles in the child contract are mapped, within the child contract, to those in the parent contracts, so that any agent assigned to a role in the child contract may also be assigned to roles in the parent contracts. Using multiple connected child and parent contracts, hierarchies of contracts can be built up ***[R12, R15]***.

A *contract template* documents a set of generalised clauses, where the clauses may contain *parameters* with no assigned values, and contract roles may not be assigned to agents. These parameters may be viewed as unbound variables. By assigning values to the parameters and agents to roles, the contract template is *instantiated* into a contract proposal.

A contract can have a *contract status*, which denotes something of how the contract should currently be interpreted ***[R14]***. For instance, in some applications where it makes sense, a contract

may have the status 'agreed but not ratified'. Contract status can be referred to by clauses of the contract (and in other contracts and agent communications). One contract status is so commonly required, we name it explicitly: *terminated*, which denotes that the contract no longer holds. By defining a generic terminated status, storage for contract documents can make use of it (e.g. for garbage collection). Contract status is updated using the same mechanisms as any other part of a contract ***[R20]***.

4.3 Contract Parties

Administrative contract parties, provide administrative support for a contract during its life cycle. They can be classified by the general role they play for the contract, and the responsibilities of that role. This model places no constraints on the number of agents playing each role, or their position in the system, allowing administration of contracts to be deployed as best fits the application ***[R29, R30, R31]***. For example, an agent fulfilling the *observer* role listens to environment events and observes state changes to determine whether contractual obligations are being fulfilled or not ***[R23]***. The observer can notify listeners, when an obligation is not being fulfilled ***[R25]*** or in danger of not being fulfilled ***[R26]***.

Another administrative role is that of a *manager:* agents playing this role know about a breach of contract by (conceptually at least) registering to listen to the notifications from an observer. 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 fine may be taken from a party. In other cases, a management agent may be a person that decides how to resolve the problem. Alternatively, a manager may merely provide analysis of problems over a long term, so that a report can be presented detailing which obligations were missed ***[R24, R27]***. A manager can be, or act with, a third-party arbiter, independent from the managed contract's other parties ***[R28].*** Note that administrative contract party roles may be played by humans as well as software agents ***[R7]***.

A *contract store* provides persistent and access controlled storage of contracts ***[R9]*** and contract proposals. Aside from being a general document store with version control ***[R10]***, it provides some contract-specific functionality.

- Allows storage and retrieval of contracts and proposals.
- Enforces access control on contracts so that they can only be retrieved by parties to the contract ***[R5, R6]***.

Retrieval of contracts can be made either by a specific contract accession ID, for that version of the contract document, or by searching for contracts meeting given criteria ***[R14]***, including a store-specific category to which they may belong ***[R13]***. Updating a contract leads to the creation of a new version in the store, and a notification sent to registered listeners, such as the contract parties. A contract store can also provide analysis functions over the contents of contracts it contains.

A *contract template store* provides storage of contract templates and querying facilities for finding templates that match particular criteria, e.g. contain obligations to fulfil particular generalised goals ***[R11]***.

4.4 Contract Lifecycle

The life cycle of a contract consists of up to five stages: *creation*, *fulfilment*, *maintenance and update* (ensuring access, security and integrity, controlled change etc.), *management* (observing handling violations etc.), and *termination and renewal*. Each stage defines one *process type,* which will be instantiated by different processes depending on the application. The contract is first created, then fulfilled and possibly enforced and/or updated, before at some point being terminated or renewed, the latter leading to further life of the contract. The architecture supports all stages of the life cycle ***[R1],*** which may be long-lasting or brief depending on the contract's period of use (which may or may not be fixed beforehand) ***[R2, R3]***.

As an example of one process fulfilling the *creation* process type, contracts can be created from templates that have been found to potentially meet the initiating agent's goals ***[R11, R21]***.

The choices of *management and update* process dictate which administrative contract parties are used. The continued existence and integrity of a contract after creation is an important factor in a reliable system. Also, contract updates should be allowed only in a well-managed way ***[R16].***

Termination of a contract means that the obligations and other clauses contained within it no longer have any force. There are a number of ways that a contract may be terminated.

- A contract may terminate *naturally*, if the system reaches a state in which none of its clauses apply, e.g. the contract's period of life expires, all obligations have been met etc.
- A contract 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 its obligation, the contract is terminated and all others are released from their obligations ***[R17]***.
- A contract may terminate *by agreement*, if all parties agree that the contract should no longer hold and modify it to a *terminated* contract status ***[R18]***. The contract maintenance and update processes dictate how to update contracts.

Renewal of a contract means that a contract that would have imminently terminated naturally is modified (updated) so that termination is no longer imminent ***[R19]***.

4.5 Critical Application States

Obligations may be classified into three types: *achievement, maintenance* and *triggered*. An achievement obligation obliges the assigned party to bring the system to some state. A maintenance obligation obliges the assigned party to keep the system in some state. Finally, a triggered obligation obliges the assigned party to react to events of a given type by taking on an additional obligation related to that event, i.e. whenever event E(X) occurs, obligation O(X) is in force ***[R4]***.

For each obligation, there are *critical application states* which can be identified. These critical states are the states of the system that there is value in observing and managing using administrative contract parties. For example, a general achievement obligation may be divided into the states:

- Pre-Achievement State, where the goal has not yet been achieved but is still achievable.
- Achievement State, where the goal has been achieved.
- Failure State, where the goal has not been achieved, and is no longer achievable.

As a variation in some applications, multiple Pre-Achievement States may be specified to denote more detailed relations between the system state and obligation. For example, a danger state can be added to denote that the goal is in danger of not being achieved ***[R26]***.

5 RELATED WORK

Related work falls into two broad categories: service-oriented and agent-oriented. There is a significant gap between the type of e-Business systems that technology such as ebXML and Web services allow to be built and the technologies that exist so as allow for the modelling and verification of such systems. One of the primary promising approaches for filling this gap is the use of electronic contracts as an explicit specification of obligations, permissions and prohibitions that regulate the activities and interactions within a distributed software system. In existing work (including ebXML, WSLA [8], WS-Agreement [9] as well as that planned in OASIS), efforts and results to date either do not focus specifically on system specification or do not provide any formal verification tools. Software by Contract and the newly released Microsoft INDIGO platform are a case in point since they focus on low level specification of method execution properties rather than more general types of obligations such as performance actions if particular conditions arise or commitment to availability at certain dates and times.

In multi-agent systems, there has been much previous work on contract-based systems, 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 [4], 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 agents [5]. In addition, the wider domains of normative systems and agreement in service-oriented architectures inform our work. Concepts such as norms specifying patterns of behaviour for agents, contract clauses as concrete representations of dynamic norms, management or enforcement of norms itself being a norm, are already established in the literature [5, 6, 7].

A detailed comparison of approaches is outside the scope of this paper. However, there has been no analysis across a range of business cases to determine the requirements of electronic contracting, as is presented in this paper.

6 CONCLUSIONS

The CONTRACT 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 that underpin formal descriptions of the expected behaviours of individual services and the system as a whole. Unlike traditional academic projects, CONTRACT is predicated on the industrial context, and the business case from four distinct business domains, in which the contract-based infrastructure is likely to bring real benefits.

In this paper, we have described the four main use cases in which our work is situated, detailing the requirements aggregated from them, and showing how our contract architecture satisfies them. This is the first step in our efforts to develop prototype systems for real applications. Crucially, in addition to adopting the use cases to motivate and inform our work on the contract-based infrastructure, we seek to address the noted lack of community resources in a library of real case studies, to facilitate understanding, comparison and evaluation of competing techniques and architectures, as well as providing a clear commercial context for the development of work.

ACKNOWLEDGEMENTS

The research described in this paper is partly supported by the European Commission Framework 6 funded project CONTRACT (INFSO-IST-034418). The opinions expressed herein are those of the named authors only and should not be taken as necessarily representative of the opinion of the European Commission or CONTRACT project partners.

REFERENCES

[1] R. A. Belecheanu, S. Munroe, M. Luck, T. Payne, T. Miller, P. McBurney, M. Pechoucek, Commercial Applications of Agents: Lessons, Experiences and Challenges, in *Proc. 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (Industry Track)*, Hakodate, Japan, 2006.

[2] D. Santos, M. Blois, R. Bastos, Developing a Conference Management System with the Multi-agent Systems Unified Process: A Case Study, in *Proc. 8th Int. Workshop on Agent Oriented Software Engineering*, 2007.

[3] T. Wagner, L. Gasser, M. Luck, J. Odell and T. Carrico, Impact for Agents, in *Proc. 4th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (Industry Track)*, 93-99, Utrecht, Netherlands, 2005.

[4] H. Lopes Cardoso and E. Oliveira, Using and evaluating adaptive agents for electronic commerce negotiation, in *Proc. 7th Ibero-American Conference on AI: Advances in Artificial Intelligence*, volume 1952 of LNCS, 96–105, 2000.

[5] F. Duran, V. Torres da Silva, and C. J. P. de Lucena, Using testimonies to enforce the behaviour of agents, in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems*, pages 25–36, 2007.

[6] C. Dellarocas, Contractual agent societies: Negotiated shared context and social control in open multi-agent systems in *Workshop on Norms and Institutions in Multi-Agent Systems*, Agents 2000, Barcelona, Spain, June 2000.

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

[8] Ludwig, Heiko. "Web Service Level Agreement (WSLA) Language Specification". IBM, 2003.

[9] A. Andrieux, et al. "Web Services Agreement Specification (WS-Agreement)". http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graapagreement.pdf, Version 1.1. Draft 18. 2004.

[10] M. Jakob, et al. "Use Case Outlines and Requirements", CONTRACT project deliverable D6.1, available from http://www.ist-contract.org/, September 2007

[11] IST CONTRACT project, http://www.ist-contract.org/