Abstract

Converting a conventional contract into an electronic equivalent that can be executed and enforced by computers is a challenging task. The difficulties are caused by the ambiguities that the original human-oriented text is likely to contain. The conversion process involves the conversion of the original text into mathematical notation. This paper discusses how standard conventional contracts can be described by means of Finite State Machines (FSMs). This mathematical description helps eliminate ambiguities from the original text before the contract is coded into a computer program. The paper describes how to map the rights and obligations extracted from the clauses of the contract into the states, transition and output functions, and input and output symbols of a FSM. The FSM representation can be used to guarantee that the clauses stipulated in the contract are observed when the contract is executed. The paper describes the middleware required for the enactment of the contract represented as a FSM.

Keywords: Contract, electronic contract, finite state machine, contract representation, contract enforcement, model-checking, validation, correctness requirements.
Run-time Monitoring and Enforcement of Electronic Contracts
Carlos Molina-Jimenez, Santosh Shrivastava, Ellis Solaiman, and John Warne
School of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, England
{Carlos.Molina, Santosh.Shrivastava, Ellis.Solaiman, J.P.Warne}@ncl.ac.uk

Abstract

Converting a conventional contract into an electronic equivalent that can be executed and enforced by computers is a challenging task. The difficulties are caused by the ambiguities that the original human-oriented text is likely to contain. The conversion process involves the conversion of the original text into mathematical notation. This paper discusses how standard conventional contracts can be described by means of Finite State Machines (FSMs). This mathematical description helps eliminate ambiguities from the original text before the contract is coded into a computer program. The paper describes how to map the rights and obligations extracted from the clauses of the contract into the states, transition and output functions, and input and output symbols of a FSM. The FSM representation can be used to guarantee that the clauses stipulated in the contract are observed when the contract is executed. The paper describes the middleware required for the enactment of the contract represented as a FSM.

1. Introduction

The concept and the use of contracts are not new to today’s society. Legal contracts can be traced back to ancient times [1]. There are records that indicate that legal contracts were used by the Mesopotamians circa 2300-428 BC for selling and purchasing slaves, land, crops and for establishing partnerships between two or more landowners.

Since hard-copy contracts have been used for a long time, we know how to write (for example in English), interpret and execute a conventional contract; unfortunately, contracts in the electronic world are not yet well understood. In particular, converting a conventional contract into an executable contract is not a trivial process.

Precisely we define a conventional contract as a document that stipulates the rights and obligations that two or more signatories agree to honour during their interactions.

An executable contract (x-contract) is the electronic version of a conventional contract and is a piece of software meant to be executed by computer (with little or no human intervention) to monitor and enforce the rights and obligations of the signatories at run-time.

The interest of this work is focussed on business contracts. We assume that business interactions between enterprises take place by means of the execution of inter-enterprise business processes. Thus in our work the contracts are conceptually located between the interacting parties and meant to drive the execution of the inter-enterprise business processes. Business processes vary in complexity from rather small such as the purchase of a book to rather complex such as the sharing of ATM cash machines between two or more banks. We assume that a complex business process can always be decomposed into two or more business processes of lower complexity that perform specific and individual activities. This decomposition can be conducted several times until the complexity of the resulting sub-processes is manageable.

This decomposition approach is of great relevance for the implementation of x-contracts. With x-contracts the interaction between the business partners can be thought as taking place through individual sub-processes that are regulated by individual sub-contracts. Naturally, each sub-contract contains only the rights and obligations to regulate the activities involved by the particular sub-process. For example, two business partners might have a contract that contains two sub-contracts: one for processing purchase orders for perishable goods and a different sub-contract for tinned food. To execute a complete business contract a parent contract is given the information and the power to create, coordinate and terminate one or more instances of the same or different sub-contracts as needed. When an instance of a sub-process is instantiated or terminated by the business partners, its corresponding electronic sub-contract is instantiated or terminated by the parent contract. In this paper we are concerned with sub-contracts only. Yet we can briefly mention that the parent contract can be realised as a workflow script that manages the set of finite
state machines that represent the set of sub-contracts that compose the whole contract. In the rest of this paper we refer to our sub-processes and sub-contracts simply as processes and contracts.

Before a conventional contract can be implemented, the contract clauses have to be described in a formal notation. There are several alternatives for describing contracts formally and several factors to be considered before choosing one of them. One of the crucial factors to be taken into account is the means that different alternatives offer for validating the correctness requirements of the contract. These correctness requirements are imposed by the contractual parties and meant to give a certain degree of assurance that the x-contract will not run into unexpected situations. We have found that Finite State Machines (FSMs) are quite adequate for describing contracts, and for validating their correctness properties.

The problem we are investigating can be described as follows: given a sub-contract that has been derived from a conventional contract how can it be described by FSMs? We are not investigating how to negotiate contracts over the Internet; we assume that the contract already exists. Our goal is to express it in FSMs, possibly, after editing the original text to correct ambiguities. Another aspect we investigate is what run-time infrastructure is required for monitoring and enforcement of x-contracts. We addressed these issues in a previous paper [2]; this paper is an extended and revised version of the previous one.

The rest of this paper is organised as follows. We introduce the concepts of rights and obligation in Section 2. In Section 3, we demonstrate, with the help of a rather simple yet illustrative contract, how rights and obligations can be represented with FSMs. Monitoring and enforcement of contracts is discussed in Section 4. In Section 5, we discuss contract ambiguities, contract validation and justify why we consider FSMs convenient for representing contracts formally. Section 6 provides the reader with a realistic business contract example represented as FSMs after being validated and corrected to eliminate its ambiguities. In Section 7, we discuss the middleware layer (B2Bobjects) we use for contract management. The similarities and dissimilarities between our research approach and other influential paradigms are discussed in Section 8. After drawing some conclusions, we close our discussion in Section 9.

2. Rights and obligations in a contract

Each entry in a contract is called a term or a clause. The clauses of a contract stipulate how the signing parties are expected to behave. In other words, they list the rights and obligations of each signing party.

A right is an action that a signing entity can do if it wishes to. For example, a contract might stipulate that Alice, as a manager of enterprise E1, has the right to send an offer to sell to Bob, the manager of enterprise E2.

Because this is a right, it is up to Alice to send or not to send the offer to Bob; Bob need not be disappointed if he does not receive the offer. Similarly, an obligation can be defined as a duty that an entity is expected to perform.

A failure to perform such a duty means a breach of the contract. For example, a contract might stipulate that upon receiving an offer to sell from Alice, Bob has the obligation to reply to her with an OfferAccepted or OfferRejected message.

The rights and obligations stipulated in a contract can be abstracted and grouped into a set of Rights (R) and a set of Obligations (O). Let us assume that ME1 and ME2 are the managers of enterprises E1 and E2 respectively. Likewise let us assume that enterprises E1 and E2 have a contract with each other signed by their managers. The sets of rights and obligations of this contract will be denoted by $R = \{ R_{ME1}^{ME1}, R_{ME1}^{ME2}, R_{ME2}^{ME1}, R_{ME2}^{ME2} \}$ and $O = \{ O_{ME1}^{ME1}, O_{ME1}^{ME2}, O_{ME2}^{ME1}, O_{ME2}^{ME2} \}$, respectively.

The sets $R$ and $O$ indicate that the manager of enterprise E1 has agreed to honour $m$ rights and $p$ obligations. Similarly, the manager of enterprise E2 has agreed to honour $n$ rights and $q$ obligations. We assume that $m, n, p, q$ are integers and equal or greater than zero. For example, $R_{ME1}^{ME1}$ is a right expected to be honoured by the manager of enterprise E1, whereas $R_{ME2}^{ME2}$ is a right expected to be honoured by the manager of enterprise E2. Obviously, for a contract to make sense it should have at least one right or one obligation. Note that for the sake of simplicity, in this paper we discuss examples of contracts with only two contracting parties. However, all our concepts, models, and examples can be generalised to $n$ parties as long as $n \geq 2$ is finite.

Note that the execution of a right or an obligation such as SendOfferAccepted will, at a lower level of abstraction, demand access to one or more objects such as files, databases and printers. A question that arises here is whether Alice and Bob have the right to access the objects affected by their operations. This is an issue of authentication and access control and falls out of the scope of this paper. We believe that at object level, rights to access resources can be implemented using Role-Based Access Control mechanisms [3].

3. Description of contracts using FSMs

A finite state machine $M$ is defined as the quintuple $[S, I, Z, \delta, \lambda]$, where $S = \{ s_1, s_2, \ldots, s_n \}$, $I = \{ i_1, i_2, \ldots, i_n \}$ and $Z = \{ z_1, z_2, \ldots, z_p \}$ are finite nonempty sets of states, input symbols and output symbols, respectively. $\delta : S \times I \rightarrow S$ is the transition function and $\lambda : S \times I \rightarrow Z$ is the output function.
function. Informally, M describes an abstract system that stays in a given state until it receives an external stimulus.

When such stimulus is received, the system reacts by doing something (for example, sending an output signal) and then moves to a different state. Note that do something might mean do nothing in some circumstances and that the new state is not necessarily different from the previous. The behaviour of this abstract system matches the behaviour of a business contract. At a given time a contract can be at any of n possible states (states1, state2,...,statesn). If the contract is in a given state (for example, WaitingForOffer), there is a finite and well defined set of events (event1, event2,...,eventn) that can affect the future behaviour of the contract. Examples of events are OffertRejected and OfferAccepted. The occurrence of event determines what objects (variables, files, database, etc.) within the system change their values, that is, the event determines to which new state the contract switches. Similarly, there is a finite and well defined set of operations (operation1, operation2,...,operationn) that can be executed when the contract is in statei. The event determines the operation to be executed.

Fig.1 shows the graphical representation we will use in the paper, where e and o stand for event and operation, respectively (a null operation will be represented by e).

Fig. 1. FSM notations.

It follows that a contract can be represented as a set of FSMs, one for each contracting party, that interact with each other. The physical location of each FSM is irrelevant to the functionality of the contract and is decided at the time of implementation. Conceptually, we can assume that a FSM is located within each contracting party and that these FSMs communicate with each other through communication channels.

3.1. Example of a simple contract

To show how the rights and obligations of a contract can be represented by means of FSMs we will now discuss a simplified version of a contract meant to be used by a purchaser and a supplier. The text of the contract is shown in Fig.2.

Fig.2. Simplified version of a contract.

To help follow the discussion, the rights and obligations of the purchaser and the rights and obligations of the supplier have been extracted from the contract, enumerated as R1p, O1p, R1s and O1s respectively and grouped into the set of rights R = {R1p, R1s} and set of obligations O = {O1p, O1s, O2p, O2s}. The meaning of each right and obligation is described in Fig.3.

Fig.3. Rights and Obligations of the contract signatories.

The representation of this contract by means of FSMs is shown in Fig.4. In the figure, the number of each right and obligation together with its corresponding operation is shown on the arcs of the FSMs. We have to mention that the contract we are discussing here is rather simple and looks correct at a first glance, however, as an attentive reader would have noticed; it suffers from inconsistencies (this will be discussed at large in Section 6).

3.2. Contract templates

In the business world, there is a family of applications where the contracting parties resort to fairly standardized contract templates which are offered ready to be filled in and signed. Examples of these templates are tenant agreements that are offered on the take-it-or-leave-it basis which means that the clauses of the contract are not negotiable.
means complying with the purchaser’s obligations. Since each operation $o_i$ is paired to an event $e_j$, the operation $o_i$ can be executed only after the occurrence of $e_j$. So, in state $e_i$, $e_j$ can be triggered by an operation performed internally within the purchaser’s enterprise, or by an operation performed externally within the supplier’s enterprise. For example, $e_i$ might be deliberately triggered by the purchaser (for example, when the purchaser wishes to send a purchase order) or it can be triggered by a message received from the supplier (for example, when the supplier wishes to offer a new item to the purchaser). Exercising a right (obligation) operation at one side of the contract might or might not have an effect at the other side. This depends on what the x-contract stipulates.

4. Monitoring and enforcement

To reason about how the contractual rights and obligations can be monitored and enforced by FSMs, it is useful to look at the rights and obligations a contracting party has in a given state of the execution of the x-contract. In terms of FSMs, this is equivalent to looking at the set of operations that can be executed when the FSM of the contractual party is in state $i$. It is useful to classify this set into two subsets: the subset of operations the owner of the FSM has the right to perform and the subset of operations that person has the obligation to perform, $\{o_1, o_2, ..., o_m\}$ and $\{o_{m+1}, o_{m+2}, ..., o_p\}$, respectively.

We consider an example involving a purchaser and a supplier. Let us say, the execution of the x-contract at the purchaser’s side is in state $state_1$ (see Fig.5). Executing an operation from the subset $\{o_1, o_2, ..., o_m\}$ means exercising a right given by the x-contract. Similarly, executing an operation from subset $\{o_{m+1}, o_{m+2}, ..., o_p\}$ means complying with the purchaser’s obligations. Since each operation $o_i$ is paired to an event $e_j$, the operation $o_i$ can be executed only after the occurrence of $e_j$. So, in state $e_i$, $e_j$ can be triggered by an operation performed internally within the purchaser’s enterprise, or by an operation performed externally within the supplier’s enterprise. For example, $e_i$ might be deliberately triggered by the purchaser (for example, when the purchaser wishes to send a purchase order) or it can be triggered by a message received from the supplier (for example, when the supplier wishes to offer a new item to the purchaser). Exercising a right (obligation) operation at one side of the contract might or might not have an effect at the other side. This depends on what the x-contract stipulates.

![Fig. 4. Representation of an ambiguous contract by means of FSMs.](image)

![Fig. 5. Interactions through rights and obligations.](image)
execute the operations he has the obligation to execute. The purchaser’s FSM works in a similar way.

5. Ambiguities in contracts

As we briefly mentioned in Section 3.1, the contract shown in Fig.2, suffers from ambiguities; namely, it does not specify the duration within which the supplier must send an offer to the purchaser. Neither does it specify the duration within which the notification about rejecting or accepting the offer should be sent. The x-contract can still be implemented and enacted but the purchaser’s FSM will hang silently until the supplier decides to send an offer. If for some reason the supplier forgets to send his offer, the two FSMs will hang silently forever or until the purchaser or the supplier use another channel (for example a telephone) to investigate the problem. Omissions like these are common rather than exceptional in conventional contracts where humans rely on common sense to interpret and deal with them; unfortunately, in x-contracts this kind of omissions are not acceptable because they would drive an x-contract into undesirable situations; thus the implementer of the x-contract must detect and eliminate them before deploying the x-contract. This is where the validation tools supported by or offered by the formal notation chosen for representing the contract become extremely important. One of the motivations we had for choosing FSMs for representing contracts was that the contract represented as FSMs can be validated using the model checking tools that were originally developed for validating communication protocols. Once the contract is converted into FSMs the implementer can use a model checker, such as Spin [4] to verify that the contract is free of undesirable ambiguities; technically speaking, the implementer can use the model checker to verify that the contract satisfies certain safety and liveness properties. The model checker used should have the means for validating general correctness properties such as absence of deadlock and infinite cycles, and also for validating specific correctness properties requested by the contracting parties. Results of the use of Spin for validating contracts have been reported in [5].

A model checker, Spin for instance, would immediately detect that the contract of Fig.2, fails to reach a final state because it makes no progress when the supplier forgets to send his offer. A corrected version of the contract is shown in Fig.6.

The sets of rights and obligations extracted from the contract are $R = \{R_1^P, R_2^P, R_1^S\}$ and $O = \{O_1^P, O_2^P, O_3^S, O_4^S\}$ respectively, and shown in Fig.7. The representation of this contract by means of FSM is shown in Fig.8.

Fig.6. Contract after correction.

Fig.7. Rights and Obligations after correcting the contract.

Fig.8. Representation of a contract by means of FSMs.

6. A realistic example of a business contract

In this section we use a realistic example to illustrate how the rights and obligations stipulated in a contract can be represented by means of FSMs, we will borrow the contract template provided in the appendix of [6]. This template is meant to be filled in and used as a contract for the purchase and supply of goods in conventional business. We are borrowing this template because it contains all basic clauses (commencement, place purchase
order, delivery, payment, rejection, etc.) most real life contracts would have; and also because it raises the issue of ambiguities in conventional contracts. As it appears in the original paper, the contract template contains several ambiguities. For example, the clause that deals with the rejection of goods does not specify when, after receiving an unsatisfactory item, the purchaser must reject the item so that he is entitled to receive a replacement or a refund. Nor does the clause specifies when, after receiving the rejected item, the supplier must send a replacement or a refund. Omissions like this would drive the x-contract into unexpected situations such as unreachability of certain states, thus they must be detected by the implementer at design stage, for example with the assistance of a model checker. We used Spin to check some correctness properties of the original version of our example. From among the most common correctness requirements of contracts of this type, we arbitrarily selected to validate that:

1. There are no unreachable states in the contract.
2. The contract is free from deadlocks.
3. The contracting parties do not receive unsolicited responses.
4. The delivery of e-goods cannot occur before receipt of correct payment.
5. The supplier will not proceed with any transactions if the e-goods are not downloaded in time after receiving of the payment.
6. The supplier will never reach a deal (end state) if a correct purchaser order (PO) has not been received, a correct payment has not been received or the e-goods have not been downloaded in time.
7. The purchaser will never reach a deal (end state) if the goods cannot be downloaded after sending a correct payment, or the supplier fails to provide a satisfactory remedy after receiving a notification of rejection of the e-goods, from the purchaser.
8. The purchaser will never infinitely often submit incorrect purchaser orders.

Our validation work resulted in a new version of contract. This version is shown in Fig.9. And the rights and obligations of the purchaser and supplier extracted from the text of the contract are shown in Fig.10.

The representation of the contract of Fig.9, by means of FSMs is shown in Fig.11, and Fig.12. In the figures Ri and Oi correspond to the numbers used in Fig.10, to enumerate the rights and obligations of the purchaser and the supplier.

It is important to notice that what Fig.11, and Fig.12, show is only one, out of several, possible realisations of the contract shown in Fig.9. A given contract can have different FSM representations depending on what and where we want to monitor and enforce. For instance, a purchaser’s obligation from our example can be represented in both the FSM of the purchaser and in the FSM of the supplier: the supplier’s FSM would detect if the purchaser fails to honour his obligation, whereas the purchaser’s FSM would remind his owner that he is about to breach the contract and wherever it is possible, execute automatically and on behalf of the purchaser, the operation needed to honour the obligation. This approach would make a contract a highly active entity; however, it results in complex FSMs and naturally, in complex implementations, as everything is represented and enforced twice. In the FSM representation shown in Fig.11, and Fig.12, we opted for simplicity, because of this, the obligations are represented in one FSM only; the result of this is that the FSMs do not actually enforce the contract they only monitor and detect when the contract is breached. The assumption here is that the FSMs notify the interested parties about the breach of contract they detect.
This Deed of Agreement is entered into as of the Effective Date identified below.

BETWEEN

[Name]             \[Name\]  

of [Address]       \[Address\]  

(to be known as the (Supplier))    \[to be known as the (Purchaser)]

WHEREAS (Supplier) desires to enter into an agreement to supply (Purchaser) with [Item] \[as defined in Section 2] (to be known as (e-goods) in this Agreement).

NOW IT IS HEREBY AGREED that (Supplier) and (Purchaser) shall enter into an agreement subject to the following terms and conditions:

1. Definitions and Interpretations
1.1 Price, Dollars or $ is a reference to the currency of the (Country).
1.2 All information (purchase order, payment, notifications, etc.), is to be sent electronically.
1.3 This agreement is governed by (Country) law and the parties hereby agree to submit to the jurisdiction of the Courts of the (Country) with respect to this agreement.
1.4 Commencement and Completion
1.5 The commencement date is scheduled as [date].
1.6 The completion date is scheduled as [date].
1.7 This whole may be modified by agreement as defined in Section 9.

2. Purchase Orders
2.1 The (Purchaser) shall follow the (Supplier) price lists.
2.2 The (Supplier) shall present (Supplier) with a purchase order for the provision of (e-goods) within 7 days of the commencement date.
2.3 The (Supplier) shall notify the (Purchaser) of acceptance or rejection of the purchase order within 7 days after the receipt of the purchase order.
2.4 If the purchase order is rejected, the (Purchaser) shall correct the purchase order within 14 days after the receipt of the notification.

3. Delivery
4.1 The delivery of the (e-goods) is the responsibility of the (Supplier). The (Supplier) shall keep the (e-goods) available for downloading at the specified e-address for at least 14 days after sending notification of acceptance of payment. The (Purchaser) shall download the (e-goods) within this period of time.

5. Payment
5.1 The payment shall be sent in full to the (Supplier) within 7 days after receiving a notification of acceptance of the purchase order.
5.2 The (Supplier) shall notify the (Purchaser) of acceptance or rejection of the payment within 7 days after the receipt of the payment.

6. E-goods rejection
6.1 If the (e-goods) do not comply with the order or the (Supplier) does not comply with any of the conditions, then the (Purchaser) is entitled to reject the (e-goods).
6.2 The (Purchaser) shall either (a) notify the (Supplier), of acceptance of the (e-goods), within 7 days after receiving them, or (b) return the (e-goods) to the (Supplier), within 7 days after receiving them.

7. Replacement and refund
7.1 The (Supplier) may, at its discretion, replace the (e-goods) according to the invoice or refund any monies paid.
7.2 The (Supplier) shall either (a) notify the (Purchaser) of refusal to replace or refund, within 14 days after the receipt of the rejected (e-goods), or (b) replace or refund any monies paid, within 14 days after the receipt of the rejected (e-goods).
7.3 In the case of a dispute in which the (Supplier) refuses to provide a replacement or refund, the (Supplier) shall, within 14 days after the receipt of the (e-goods) returning rejected (e-goods), notify the (Purchaser) to terminate the contract.

8. Termination
8.1 If (Supplier) of (Purchaser) fail to carry out any of their obligations and duties under this agreement, the offended party shall terminate the contract.

9. Disputes
9.1 (Supplier) and (Purchaser) shall attempt to settle all disputes, claims or controversies arising under or in connection with the agreement through consultation and negotiations in good faith and a spirit of mutual cooperation.
9.2 (Supplier) and (Purchaser) shall provide electronic evidences about breaches of the e-contract.
9.3 This method of determination of any dispute is without prejudice to the right of any party to have the matter judicially determined by a (Country) Court of competent jurisdiction.

10. Amendment
10.1 This agreement may only be amended in writing signed by or on behalf of both parties.

Fig.9 A contract for the purchase and supply of e-goods.

Fig.10 Rights and obligations of signatories.
Based on our own experiments we speculate that converting a contract into its x-contract will be a long and interactive process between the writer of the English text contract (for example, a lawyer) and the technical person in charge of the implementation of the x-contract. The technical person would receive the English text contract from the lawyer, convert it into FSMs and check it for ambiguities. Detection of ambiguities would mean bouncing the contract back to the lawyer for correction.

Only when both the lawyer and the technical person agree about the content of the English text contract and its
7. Middleware support

Next we investigate what middleware services are required to support a contract management system that guarantees that the rights and obligations stipulated in the contract are monitored and enforced. We are assuming that the organizations involved might not trust each other, so an important requirement from the middleware is that it should enable regulated interactions (as encoded in x-contract) between two or more mutually distrusting but autonomous organizations. It is clearly not possible to prevent organisations from misbehaving and attempting to cheat on their agreed contractual relationships. The best that can be achieved is to ensure that all contractual interactions between such organisations are funnelled through (a centralised or distributed) contract management system and that all other non-contractual interactions are disallowed.

We assume that each organization has a local set of policies for business interactions that is consistent with the overall business interaction rules encoded in the form of rights and obligations in the x-contract. Then, the safety property of the contract management system should ensure that local policies of an organization are not compromised despite failures and/or misbehavior by other parties; whilst the liveness property should ensure that if all the parties are correct (not misbehaving), then agreed interactions would take place despite a bounded number of temporary network and computer related failures.

Given the above observations, we can state that organizations will require (i) that their own actions meet locally determined policies; and that these actions are acknowledged and accepted by other parties; and (ii) that the actions of other parties comply with agreed rules and are irrefutably attributable to those parties. These requirements imply the collection, and verification, of non-repudiable evidence of the actions of parties who interact with each other.

For non-repudiable information sharing we propose to use the B2BObject middleware developed by us [7]. Assume that every organization has a copy of some shared information encoded in objects, then B2BObjects middleware provides non-repudiable coordination of the state of object replicas. State changes are subject to a locally evaluated validation process. State validation is application-specific and may be arbitrarily complex (and may involve back-end processes at each organisation).

Coordination protocols provide multi-party agreement on access to and validation of state. Fig. 13 presents four enterprises (E1, E2, E3, E4), sharing a state through three B2BObjects (A, B, and C). As shown in the figure, the logical view of shared objects in a virtual space (a) is realised by the regulated coordination of actions on object replicas held at each organisation (b).

Multi-party validation of state changes supports the notion of “joint ownership” of shared state. A state change proposal comprises the new state and the proposer’s signature on that state. The proposal is dispatched to all other parties for local validation. Each recipient produces a response comprising a signed receipt and a signed decision on the (local) validity of the state change. All parties receive each response and a new state is valid if the collective decision is unanimous agreement to the change. The signing of evidence generated during state validation binds the evidence to the relevant keyholder. Evidence is stored systematically in local non-repudiation logs. The B2BObjects middleware provides both the liveness and safety properties stated earlier.

With this background, we can hint at the overall implementation of an x-contract. The implementation of an x-contract that involves a purchaser and a supplier, is shown in Fig.14. Each party maintains a copy of the contract object, encoded as one or more B2BObjects (B2Bobj); operations on these objects are controlled by the contract FSMs. The dashed line that goes from the supplier to the purchaser shows what happens when the supplier sends an offer. When the offer is ready, the supplier invokes a send operation, and the supplier's FSM switches to its Waiting for response state and makes a SendOffer call to the local copy of a shared B2Bobj (that implements the operation). The local B2Bobj collects, and signs, evidence of the operation and requests coordination of the proposed update to its state with the purchaser's B2Bobj.

The purchaser's B2Bobj verifies the evidence provided and makes an up-call to the purchaser's FSM to validate the B2Bobj operation. Upon receiving the up-call, the purchaser's FSM switches to the Deciding to buy state.

The dashed line from the purchaser's FSM to the supplier's FSM shows how the purchaser's response is
transmitted to the supplier. The B2BObjects middleware ensures that all operations performed by the purchaser and the supplier are recorded and are non-repudiable. Thanks to this facility the purchaser of the example of Fig 12, can provide evidence, at a court for example, that he sent his payment within 7 days after receiving a notification of acceptance of his purchase order, even if the supplier denies receiving the payment. One of the major advantages of B2BObjects is that it ensures this without the need of involving centralized trusted third parties.

In this form of representation of business contracts has been the need of involving centralized trusted third parties. The B2BObjects middleware provides evidence, at a court for example, that he sent his payment within 7 days after receiving a notification of acceptance of his purchase order, even if the supplier denies receiving the payment. One of the major advantages of B2BObjects is that it ensures this without the need of involving centralized trusted third parties.

Contract management must be made part of the business processes of the organizations involved. An organization’s business processes can be divided into two broad categories. The business processes that are internal to the organization and the ‘contract management processes’ that involve interactions with trading partners. A difficult problem is that of coordinating multiple workflows in a decentralised manner. Most commercial workflow systems are inherently centralised. A way out is to use a workflow system with decentralised coordination (e.g., [8]) for managing just the inter-organizational workflows. This is left as a topic for further investigation.

8. Related work

Formal representation of business contracts has been identified as a crucial issue by several researchers. In this section we will summarise the essential ideas behind the works that we consider influential in this research field.

In the work of Milosevic et. al. [6] [9] [10] a contract is informally defined as a set of policy statements that specify constraints in terms of permissions, prohibitions and obligations for roles involved in the contract. A role (precisely, a role player) is an entity (for example a human being, machine, program, etc.) that can perform an action. Formally, each policy statement is specified in deontic logic constraints [9]. Thus each deontic constraint precisely defines the permissions, prohibitions, obligations, actions, and temporal and non-temporal conditions that a role needs to fulfill to satisfy an expected behaviour.

For example, a constraint can formally specify that, “Bob is obliged to deliver a box of chocolates to Alice’s desk every weekday except on Wednesdays for three years, between 9 and 9:15 am, commencing on the 1st of Jan 2004”. The expressiveness of deontic notation allows the contract designer to verify temporal and deontic inconsistencies in the contract. The authors of this approach argue that it is possible to build verification software to visually show that, Bob’s obligations do not overlap or conflict. Such verification mechanisms would easily detect a conflicting situation where Bob has to deliver a box of chocolates to Alice’s desk and to Claire’s who works miles away from Alice’s desk. Similarly, the verifier would detect that Bob is not obliged and prohibited to deliver chocolates to Alice during the same period of time.

Another research work of relevance to ours is the EDEE system. EDEE provides a framework for representing, storing and enforcing business contracts [11]. In EDEE a contract is informally conceived as a set of provisions. In legal parlance, a provision is an arrangement in a legal document, thus in EDEE a provision specifies an obligation, prohibition, privilege or power (a privilege or power is equivalent to a right in our work). An example of a provision is “Alice is obliged to pay Bob 20 cents before 1st Jan 2004”. Central to EDEE is the concept of occurrence. An occurrence is a time-delimited relationship between entities. It can be regarded as a participant-occurrence-role triple that contain the name of the participants of the occurrence, the name of the occurrence and the name of the roles involved in the occurrence. An example of an occurrence that involves Alice (the payer) and Bob (the payee) is “Alice is paying Bob 20 cents on 31st Dec 2003.” The formal specification of a contract in EDEE is obtained by translating the set of informal provisions derived from the clauses of the contract into a set of formal occurrences. Another basic concept in EDEE is query. A query is a request for items satisfying certain criteria (for example, “Payments performed by Alice before 31st Dec 2003”). At implementation level, the occurrences representing the
contract provisions are stored together with queries and new occurrences in an occurrence store in SQL views.

Business operations invoked by the contractual parties are seen as occurrences intercepted and passed through the occurrence store where they are analysed to see if they satisfy the contractual occurrences associated with the operations. EDEE has been provided with some means for detecting contract inconsistencies. To detect overlap between queries (a set of occurrences being both prohibited and permitted, a set of occurrences being obliged and prohibited, etc.) the authors of EDEE rely on a locally implemented coverage-checking algorithms.

Of relevance to our research is also the Ponder language [12]. Ponder is a declarative language that permits the specification of policies for managing a distributed system or contractual service level agreements between business partners. Ponder specifies policies in terms of obligations, permissions and prohibitions and provides means for defining roles and relationships. To detect and prevent policy conflicts such as conflict for a given resource or overlapping of duties, Ponder’s notation permits the specification of semantic constraints that limit the applicability of a given policy in accordance with person playing the role, time, or state of the system.

A common pattern of the related works discussed above is that all of them rely on elaborate logical notations that include temporal constraints and role players in their parameters. The expectation is that this notation should be able to specify complete and arbitrarily complex business contracts (see Section 1) and detect all kind of inconsistencies. This generality is certainly desirable; however, because of the complexity of the problem it might be rather ambitious. We believe that our modular approach that consists in dividing complex contracts into individual sub-contracts that can be formally described and verified separately is more realistic for detecting contract ambiguities. A crucial point here is the identification of the different sources of possible inconsistencies in business contracts. In our business model enterprises that engage in contractual relationships are autonomous and wish to remain autonomous after signing a contract. Thus a signing enterprise has its own resources and local policies. In our view each contracting enterprise is a black box where private business processes represented as finite state machines, workflows or similar automaton, run. A private business process interacts with its external environment through the contract from time to time to influence the course of the shared inter-enterprise business process.

From this perspective, we can identify two fairly independent sources of contract inconsistencies:

- Internal enterprise policies conflicting with contractual clauses.
- Inconsistencies in the clauses of the contract.

It is our view that these two issues should be treated separately rather than encumbering a contract model with excessive notation (details, concepts and information) that might be extremely difficult to validate. Such a separation is not considered in the papers discussed above. To simplify the issue, in our work we draw a clear line between the two possible sources of ambiguities. In this paper we are concerned only with the second issue, namely, with the cooperative behaviour of business enterprises and not their internal structure.

It is worth mentioning that the use of finite state machines for representing such interactions has other supporters. For example, it has been proposed for Web services (Web service conversation language, WSCL [13]. Likewise we note that inter-organisation business interactions, PIPs (partner interface processes) as specified in Rosettanet industrial consortium [14] can also be represented as finite state machines.

In our business model each contracting enterprise has the privilege and responsibility of verifying that its internal policies do not conflict with the clauses of the contract. Similarly, each enterprise exercises its independence to choose the roles players that would invoke operations on the contract and provide them with a proper contract role player certificate (a cryptographic key for example). Consequently, it is the responsibility of each enterprise to prevent inconsistencies with role players such as duty overlapping, duty separation, etc.

In our contract model we intentionally leave the notion of role players out of the game. However, we assume they are authenticated by the contract management system before they are allowed to perform operations on the FSMs. It can be argued that our FSM model lacks expressiveness in comparison with the related works discussed above. However we do gain in simplicity; as we described in Section 5, this simplicity gives us the alternative of using standard of-the-shelf model checkers like Spin [4] to validate general safety and liveness properties of contracts, relatively easily.

Electronic contracts have also been studied by Naftaly Minsky and his research group [15] [16] [17], under the concept of Law-Governed Interaction (LGI). The LGI mechanism is a message exchange software layer that allows a group of distributed agents to interact over a communication medium, honouring a set of previously agreed upon rules. An agent is an entity, for example, a computer program, with means for sending and receiving messages. As the term agent suggests, agents act on behalf of their enterprises. In the LGI paradigm, a business to business interaction involves a set of private laws and one interaction law: the private laws are internal to each enterprise and regulate the activities of the agents while operating as representatives of their enterprises whereas the interaction law is public to the members of the group and regulates the interactions
between the enterprises. To draw an analogy between the LGI and our FSM paradigms, it is worth mentioning that the interaction law is actually the business contract that the agents are expected to honour when they interact with each other on behalf of their enterprises.

A law can be regarded as a set of rules. An example of a rule contained in a private law would be “Agent E can place purchase orders without the approval of the manager only for purchases not exceeding 5000 pounds.”

Laws are enforced by controllers which are trusted entities conceptually placed between each agents and the communication medium. Thus the private law $L_A$ to be honoured by agent A is enforced by controller $C_A$ while the private law $L_B$ to be honoured by agent B is enforced by controller $C_B$. The law $L_{AB}$ that regulates the interaction between agent A and B is enforced by a mediator controller $C_{AB}$ which is conceived as working on behalf of a mediator agent that bridges the interactions between agents A and B. Every controller stores its law (formally represented as Prolog-like terms) and the current control state of its agent. When an event occurs (for example, “purchase order received”) the controller performs the corresponding operations stipulated in the laws (for instance, “send acknowledgement to business partner”, “notify the local manager”, etc.) to honour the private law, the interaction law, or both, and computes the new control state [17].

The LGI approach is similar to ours in that it suggests a separation of business to business laws from internal-to-enterprises ones. Likewise, the job of the mediator controller closely resembles the job of the FSMs of our approach. To the best of our knowledge, the LGI group has no reported results about validation of the laws or about how the controllers collect non-repudiable evidence of the operations performed by their agents.

### 9. Concluding Remarks

Converting an existing standard contract written in English or other human languages into an x-contract is a challenging yet achievable task. The result of the task should be a computer program that, when executed, performs, monitors and enforces the business operations stipulated in the original human oriented document. The ambiguities that are normally present in human oriented contracts make the conversion a difficult process aimed at correcting such ambiguities without changing the main goal of the original text contract. To find ambiguities in the text contract it is strongly advisable to convert it into formal notation with means for validating the correctness of the contract; in this paper, we proposed the use of FSMs for formally representing x-contracts. From our own experience we have learnt that FSMs are a simple yet expressive model for describing, validating, and implementing x-contracts. We have illustrated our ideas with the help of two simple, yet practical examples of a business contract.

In our ongoing work we are in the process of implementing x-contracts on top of the B2BObjects middleware service. B2BObjects is used to regulate the interaction between the contracting parties and to collect non-repudiable evidence of each of their actions. Using B2BObjects we can show that x-contracts can be monitored and enforced without requiring the involvement of independent trusted third parties.

### Acknowledgements

This work is part-funded by the UK EPSRC under grant GR/N35953/01: “Information Co-ordination and Sharing in Virtual Environments”, by the European Union under Project IST-2001-34069: “TAPAS (Trusted and QoS-Aware Provision of Application Services)”; and by the UK DTI e-Science programme under project “GridMist”.

### References


[13] Web Service Conversation Language (WSCL) 1.0 (http://www.w3.org/TR/wscl10/)


