
Further information on publisher website:
http://ieeexplore.ieee.org

Publisher’s copyright statement:
© IEEE, 2009

‘Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE.’

The definitive version of this article is available at:
http://dx.doi.org/10.1109/CEC.2009.77

Always use the definitive version when citing.

Use Policy:

The full-text may be used and/or reproduced and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not for profit purposes provided that:

- A full bibliographic reference is made to the original source
- A link is made to the metadata record in Newcastle E-prints
- The full text is not changed in any way.

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.
Exception Handling in Electronic Contracting

Carlos Molina-Jimenez, Santosh Shrivastava and Massimo Strano
School of Computing Science, Newcastle University, UK
{Carlos.Molina, Santosh.Shrivastava, Massimo.Strano}@ncl.ac.uk

Abstract

In a contract, exceptional clauses specify sanctions that come in force when the primary obligations are not fulfilled. An important aspect of exception handling is their resolution: determining which particular exception clause should be enforced when a violation is detected. This paper presents a specification and resolution technique for electronic contracts that can be used by a third party exception resolution service.

1. Introduction

In businesses, legal contracts form the basis to regulate the interaction between business partners. As businesses are increasingly being conducted electronically, there is a growing interest in exploring innovative ways of automating the regulation of business interactions using electronic contracting systems. By regulation we mean monitoring and/or enforcement of business-to-business (B2B) interactions to ensure that they comply with the rights (permissions), obligations and prohibitions stipulated in contract clauses. Electronic contracts need to specify both functional and non-functional requirements. Functional requirements refer to the terms and conditions of legal contracts, expressing what business operations the partners are permitted, obliged and prohibited to execute. They also stipulate when and in what order the operations can be executed. For instance, for a buyer-seller business partnership, the contract will stipulate when purchase orders are to be submitted and within how many days of receiving payment the goods have to be delivered, etc. Non-functional requirements (stated in terms of Service Level Agreements) are mainly concerned with the Quality of the Service (QoS). For example, a non-functional requirement might specify that an offered service is expected to provide a response time below 25 milliseconds during business hours. This paper focuses primarily on functional requirements.

Take the following clause in a buyer-seller contract as an example: “the seller is obliged to deliver goods within five days of receiving the payment”. Most likely, such a contract will also contain one or more exceptional (or contingency) clauses that come in force when the delivery obligation stated in the ‘primary clause’ is not fulfilled (breach or violation of the contract). An example is “the buyer is entitled to cancel the order and get a full refund if the goods are not received within five days”. The reader can appreciate that undesirable situations are possible (such as goods have been dispatched and the order cancelled); indeed, most contracts anticipate the likelihood of such situations and contain additional exceptional clauses. Situations that cannot be resolved because there are no suitable clauses in the contract pertaining to them must be handled outside of the contract.

Electronic contracts need to be made free from ambiguities that are frequently present in conventional contracts, where they are resolved by humans as the need arises. An important aspect of exception handling is exception resolution: determining which particular exception clause should be enforced when a violation is detected. In practical implementations, several factors combine to make this a hard problem.

(i) Precise and concise specification of clauses (exceptional ones, included) suitable for machine interpretation is a challenging task as contractual interactions can be very complex. Clearly, we need high level, easy to use notations for capturing the (often quite subtle) meaning of these clauses. At the same time, these notations should be implementable, that is, they should provide implementers with useful information on implementing a given functionality using the middleware technology employed in B2B messaging (e.g., ebXML [1], RosettaNet [2]).

(ii) Specifications must take into account the distributed nature of the underlying computations by paying due attention to the impact of software, hardware and network related problems (e.g., clock skews, unpredictable transmission delays, message loss, incorrect messages, node crashes etc.). There
must be some intuitively simple way of specifying the consequences of the above problems.

(iii) Business process executions at each partner obviously need to be coordinated at run-time to ensure that the partners are performing mutually consistent actions. However, the structure of business processes can be very complex containing many exception handling tasks and non-deterministic choices, making them inherently hard to coordinate, particularly in B2B settings where the partners are autonomous entities and loosely coupled. Indeed, there is a danger that business processes at interacting partners could get out of synchrony (state misalignment) with each other that could divert the processes from their normal business paths, eventually leading to contract violations.

Existing work on electronic contracts has not addressed exception handling by taking all of the above issues simultaneously, something that we rectify in this paper. We address (i) by developing a business rule based notation that takes due consideration of the underlying B2B messaging and is particularly convenient for specifying exceptional clauses; issue (ii) is addressed by developing an execution model for business conversations that enables mapping of software, hardware and network related problems on to just a small number of events (such as business failure and technical failure) at the rule level; issue (iii) is addressed by developing an exception resolution service based on a third party paradigm. Specific events arising from the execution of business conversations (including any business and/or technical failure events) are sent to the service that not only performs exception resolution but also supplies enough coordination information to business partners to prevent state misalignment. A salient advantage of the service is that it does not need to know the internal structure of partners’ business processes, this way it respects their autonomy. In the current design we assume that business partners are well-behaved (do not send misleading events to the service). The impact of relaxing this assumption, as well as the practicality of the approach developed here, are also examined in the paper. Major parts of the system described here have been implemented; brief description is presented in this paper (for addition details, the reader is referred to [3]).

2. Overall methodology

Exceptional clauses normally specify sanctions which are obligations that come in force when the primary obligations are not fulfilled (for this reason, sanctions are also referred to as contrary to duty obligations [4]). Exception resolution must be made accurate by identifying underlying causes for the violations so that sanctions are applied only when strictly necessary. In electronic contracting, it is particularly important to distinguish violations caused by infrastructure level problems: situations that arise primarily because of the inherently distributed nature of the underlying computations from those that are not and are mostly human/organisation related. Take a simple example: B fails to make a payment before the stipulated deadline. It makes sense to distinguish cases where the missing or delayed payment is owing to some infrastructure related problem (say the network was down) from cases where no such problems existed (so probably B was just late or deliberately avoiding payment); ideally, a sanction (such as a fine) should not be imposed on B under former cases, rather actions such as extending the deadline should be undertaken.

We therefore recommend that exceptional clauses in electronic contracts should be structured appropriately to take account of infrastructure level problems. Our study of messaging standards such as eBXML [1], RosettaNet [2], BizTalk [5] suggests that at the highest level of specification (e.g., legal English), such problems can be referred to as business problems (problems caused by semantic errors in business messages, preventing their processing) and technical problems (problems caused by faults in networks and hardware/software components). Fig. 1 shows a hypothetical contract with such a flavour.

---

1. To be entitled to 15% discount, buyer is expected to submit a purchase order within 7 days of receiving an offer.
   1.1 A delayed (not exceeding by 3 days) purchase order will be processed but granted no discount unless 1.1.1 or 1.1.2 apply.
   1.1.1 A delayed purchase order due to business reasons shall be granted only 10% discount.
   1.1.2 A delayed purchase order due to technical problems shall be granted 15% discount.
   1.2 Purchase orders delayed by more than 3 days will not be processed online.

2. Buyer is obliged to submit payment within 5 days of receiving invoice.
   4.1 Missing payments will incur 10% fine and, if submitted, not considered for online processing, except that:
   4.1.1 Failure to meet a payment deadline due to business or technical reasons will grant:
      a) a payment deadline extension of 5 days to the buyer.
      b) right of purchase order cancellation to the seller.
   4.3 The seller is obliged to refund payments received after cancellations.
   4.4 Buyer and seller are obliged to stop the purchase order transaction upon the detection of three failures to submit payment and process the payment offline.

---

Fig. 1. Contract with contingency clauses

Our overall approach to electronic contracting in general and exception handling in particular is then as follows: (i) we assume that legal contracts pay due consideration to the electronic nature of interaction, as suggested above; (ii) we make sure that B2B interactions concerned with exchange of electronic documents are structured to prevent state
misalignment; and produce events that correspond to success or business/technical problems (discussed in Section 3); (iii) we assume that a third party contract monitoring/coordination service receives these events and makes them available for analysis by a rule engine of the service (Section 5); (iv) legal clauses are translated to business rules that incorporate mechanisms for resolving exceptions (the underlying principles of resolution are explained in Section 4); (v) the service keeps track of the current sets of rights/obligations/prohibitions for each partner and makes sure that partners are aware of it; this information enables the partners to keep in synchrony and perform operations that are consistent with the business rules (and hence the contract).

3. A model for business operations

Fulfilment of some business function (e.g., order fulfillment) stated in the clauses of a contract requires partners to exercise their rights and/or obligations and this in turn requires them to take part in the execution of one or more shared business processes (also called public or cross organizational business processes), where each partner is responsible for performing complimentary business operations. We assume that these processes are composed of well defined primitive business operations (bos) or business activities, such as request purchase order, notification of invoice, etc. Each such bo involves exchange of one or more business documents, and is carried out by a business conversation. We assume that an electronic contract is also expressed in terms of the bos, thereby providing a way of establishing the correspondence between contract clauses and business activities.

Conversations use well-known network protocol techniques to deal with problems such as lost and corrupted messages, but there are additional problems that need special attention. Conversations have several timing and message validity constraints that need to be satisfied for their successful completion. A failure to deliver a valid message within its time constraint could lead to state misalignment (one party regarding the message as timely whilst the other party regarding it as untimely). Misalignment can also arise if a sent message is delivered on time but not taken up for processing due to some message validity condition not being met at the receiver (so that the sender assumes that the message is being processed whereas the receiver has rejected it). Synchronisation mechanisms to prevent state misalignment are therefore required [6,7]. As an example of real conversations, we show in Fig. 2, two RosettaNet conversations (referred to as Partner Interface Processes, PIPs).

A PIP performs two message validity tests on a message that is received on time: base validation (verification of a static set of syntactical and data validation rules) and content validation (documents must also be semantically valid: satisfy application specific correctness criteria); only base and content validated messages are processed.

Given the wide variety of events that can be generated at both sides when a conversation protocol takes place, it is worthwhile to examine if any aggregation can be performed to make only a few significant events visible to a party interested in observing the development of the business interaction. With this view in mind, we briefly present an execution model for a bo (Fig. 3) that incorporates four stages: initiation, synchronisation of initiation outcomes, actual protocol execution and outcome result synchronisation (more details are presented in [8]).

B2B messaging is typically implemented using Message Oriented Middleware (MoM) that permits loose coupling between partners (e.g., the partners need not be online at the same time). To guarantee that the conversation protocol is started only when both business partners are ready for the execution of a business operation, they execute an initiation protocol (Fig. 3-a). Once the init protocol is started, the initiator eventually produces either InitF or InitS to declare that locally the initiation was successful or failed. Similarly, on the other side, the responder produces either InitF or InitS. To guarantee that both parties always see the same initiation results, we execute a synchronisation protocol (initiation syn in Fig. 3-b). Naturally, the synchroniser declares InitS only when both partners declare success and InitF in any other possible combination of local outcomes.
failures detected at the middleware level, such as a late,
exceptional situations.

When, after a successful initiation, a party is unable to
model the (hopefully rare) execution outcomes at the
business level, e.g., the goods-delivery address
models semantic errors in a message detected at the
infrastructure. We make no assumptions about the
responsibility of the business partners to deploy them
irrelevant to their causes so that the most relevant exception clause
comes in force. We assume that contractual clauses stipulate deadlines to successfully complete (as
opposite to start) the execution of business operations.

4. Exception resolution

Contract clauses are constrained by several parameters; among them, time seems to be the most common and relevant; the reason is that a wide variety of potential problems that impact contractual interactions appear at the application level as overrun deadlines. Our analysis of exceptions will be grounded on this assumption. The key question we are trying to answer is when the execution of a given business operation results in a contract violation, how to relate it to its causes so that the most relevant exception clause comes in force. We assume that contractual clauses stipulate deadlines to successfully complete (as
opposite to start) the execution of business operations.

4.1. Fixed deadlines

Many contractual deadlines are stipulated as shown in
Fig. 4. The normal deadline represents the ideal time
to complete an execution; whereas the extended deadline is a contingency deadline that is normally granted when a normal deadline is missed. Executions that take place beyond the extended deadline are not handled by the online contracting process (and should be dealt with by some offline mechanisms).

![Fixed deadline extensions](image-url)

**Fig. 4. Fixed deadline extensions**

We assume that once successfully initiated, the maximum Execution Time (maxET) to complete an execution (that could produce any of Success, BizFail or TecFail outcome), is known a priori. Consequently, successful initiations that take place before the Latest Initiation Time (defined as LIT = NDL – maxET) will always complete before the expiry of the Normal Deadline (NDL). We use the LIT parameter to distinguish between missed deadlines caused by infrastructure related problems and those that are not, and are more likely caused by human/organization related reasons. This allows us to precisely resolve an exception. For instance, assume that purchase order is
delayed but within the three day deadline (Fig. 1); if there is evidence that the buyer attempted the operation before LOT but failed (say a TecFail event occurred), then clause 1.12 should be enforced granting 15% discount. A simplification is possible in applications where the value of maxET is insignificant in comparison to the length of the deadlines, in which case, maxET can be taken as zero.

4.2. Dynamically extended deadlines

The policies to grant deadline extensions stipulated in contract clauses can be quite complex. The simplest policy will grant a single, fixed length and unconditional deadline extension, as we discussed in the previous sub-section. However, in a general case (see Fig. 5), the extended deadline to complete execution can be regarded as composed out of \( N \geq 0 \) conditioned deadline extensions of possibly different lengths, where the conditions are formulated in terms of event patterns (of arbitrary complexity with and, or, etc composition operators) detected in previously missed deadline. The payment clause 4.1.1 is an example.

![Fig. 5. Dynamically extended deadlines](image)

The overall architecture of the exception resolution service is shown in Fig. 6.

**Fig. 6. Exception resolution service**

It is a coordination service that can be seen as an extension of the contract monitoring service described in [8] and is based on the Event Condition Action (ECA) paradigm. The only external events that it receives are those generated by the initiation and execution synchronisers as business partners execute their business operations. An event has several attributes: the name of the business operation, the names of the originator and the responder, a timestamp referring to the time of the occurrence and the event type. Referring to Fig. 3, the event type is one from the set \{InitS, InitF\} if it is an initiation event, or one from the set \{Success, BizFail, TecFail\} if the event is the outcome of the operation. These events are stored permanently in the event logger and temporarily (until they are processed by the relevance engine) in the event queue. The contract rules is the rule base repository used by the relevance engine and contains a list of rules that describe the contract in force with both primary and exceptional clauses. These rules specify what rights, obligation and prohibitions become active and inactive after the occurrence of events related to the execution of business operations. For each partner, the current set of business operations that the partner can execute are classified into rights, obligatory and prohibited and are explicitly stored in the current ROP set and available to the relevance engine for consultancy and update. The timer is used by the relevance engine to keep track of deadlines associated to each right, obligation and prohibition stored in the ROP sets. When a deadline expires, a timeout event is notified to both the event logger and event queue. The job of the relevance engine is to update the current ROP sets upon the arrival of events representing the execution of valid operations. Its algorithm is: (i) remove the event from the head of the event queue; (ii) consults the ROP sets to verify if the event corresponds
to a valid operation (rightful, obligatory or prohibited operation); (iii) if the verification is satisfactory, use a rule matching algorithm against the contract rules and execute all the rules found to be relevant to the current event. The execution of relevant rules normally results in updates of the current ROP sets and setting/resetting of deadlines in the timer. In this manner, the coordinator knows exactly what rightful, obligatory and prohibited operations the business partners can execute and their associated deadlines. Any change to the ROP set of a partner is automatically sent to the partner by the informer so that the partners always have an up-to-date view of the operations they can execute. Each partner uses this information to drive its business process. A correctly functioning B2B interaction coordinator should never trigger a deadline timeout for an operation that successfully executed within its deadline constraint or make similar misjudgments.

Correct functionality is guaranteed under the following assumptions: 1) All clocks are synchronised to a known accuracy. 2) Transmission and processing delays of events from the synchronisers to the resolution service are bounded and known. 3) Events are delivered exactly once to the resolution service in temporal order. 4) The resolution service is reliable, however, the buyer and seller infrastructure as well as the MOM that connects them might fail and recover.

5.1. Examples of rules with exception resolution

For the discussion of this section we will use the contract sample of Fig. 1. The actual rules are presented next, using the notation developed in [9]. We assume that they have been validated for consistency and that potential conflicts are solved by conventional prioritisation mechanisms. The typical structure of a rule consists of checking relevant attributes of the event (e.g., that the operation is in the ROP set) and then specific actions are taken (e.g., update the ROP set) depending upon the type (Success, BizFail, TecFail, InitF, InitS) of the event. InitS events are not explicitly processed in our examples of rules on the basis that an InitS event will always cause a Success, BizFail or TecFail event. We use the following acronyms: C—clause, e—event, orig—originator, obligs—obligations, d—day, ts—time stamp, BO—business operation, POsub—purchase order submission, POsubTO—PO submission timeout, Dist—discount. Some parameters are omitted and represented by ‘…’. We take maxET=0, so this parameter is absent from the rules.

5.1.1. Fixed deadlines. We will use the deadline model presented in Fig. 4 to analyse the clauses of our example.

```plaintext
#Buyer submits PO within 7d and gets 15%discount
rule POInTime #rule for C1
 when e is POsub && orig==buyer &&
   POsub in buyer.rights && e.ts<7d
 then
 Success: buyer.obligs+=Pay(Price-15%Disct),...  
 BizFail: pass 
 TecFail: pass 
 InitF: pass

#Buyer misses 7d deadline and gets 3d extension
rule POTimeout #rule for C1
 when e is POsubTO && orig==buyer &&
   POsub in buyer.rights
 then buyer.rights==POSub(3d)

#Buyer submits PO within 3d extension
rule LatePOsub #rule for C 1.1.1, C 1.1.2
 when e is POsub && orig==buyer &&
   POsub in buyer.rights && (7d < e.ts <10d)
 then
 Success: #PO successfully submitted

#Buyer misses 7d deadline and gets 3d extension
rule POTimeout #rule for C1
 when e is POsubTO && orig==buyer &&
   POsub in buyer.rights
 then buyer.rights==POSub(3d)

#Buyer submits PO within 3d extension
rule LatePOsub #rule for C 1.1.1, C 1.1.2
 when e is POsub && orig==buyer &&
   POsub in buyer.rights && e.ts<7d
 then
 Success: #PO successfully submitted

Otherwise: [#PO unsuccessfully submitted
 pass   #due to BizFail,TecFail,InitF
 #otherwise

Clause 1 represents the primary (expected) execution path and is achieved when the execution of the purchase order produces Success within the normal deadline (seven days in this example). InitF, BizFail and TecFail results produced within the normal deadline are only recorded. A Success event within the extended deadline (three days) brings the exceptional clauses 1.1, 1.1.1 or 1.1.2 into force. Similarly, any result produced beyond the extended deadline brings clause 1.2 into force. Clause 1.1.2 models infrastructure related exceptions and comes into force when a Success result achieved within the extended deadline is preceded by one or more TecFail or InitF events occurred within the normal deadline. Clause 1.1.1 is also concerned with infrastructure related exceptions except that it models business related problems. It comes into force when a Success result achieved within the extended deadline is preceded by one or more BizFail results occurred within the normal deadline. Clause 1.1 captures remaining exceptions that are likely to be caused by human/organisation related reasons. It stipulates that
the buyer will not be granted a discount if his purchase order execution produces Success within the extended deadline but there are no records of InitF, TecFail or BizFail within the normal deadline to indicate that the buyer tried previously to execute the purchase order in time. It is worth emphasizing that InitF, TecFail and BizFail events that occurred within the extended deadline are not mapped into infrastructure related exceptions but into human related as their time of occurrence suggests that the buyer initiated the execution after the latest initiation time.

In the example, rule POInTime shows how the four possible outcomes (Success, BizFail, TecFail, InitF) of a PO submission can be handled; pass means no action within the rule, yet the event is recorded in the event logger. Rule POTimeout grants three days extension to the buyer. Rule LatePO deals with late submissions; it verifies the existence (happened) and absence (!happened) of TecFail and BizFail records in the event logger, to grant or deny a discount to the buyer.

5.1.2. Dynamic deadlines. Clauses 4 to 4.4 grant deadline extensions conditioned to event patterns. To save space, we will only discuss some potential execution paths (see Fig. 7) that we consider illustrative of the complexity that exceptional situations introduce to business interactions. In the figure, d, NDL and extens. stand for day, normal deadline and extension, respectively; similarly, Inv, Pay, Can and Ref stand, respectively, for invoice, payment, cancellation and refund; in the same order, sub-scripts s, TF and BF stand for success, technical failure and business failure, respectively. Execution path 1) represents that ideal execution: no exceptions occurred. Execution path 2) will be covered by clause 4.1 where the buyer misses the normal time constraint for no reasons (no events showing intention to pay occurred) so it is taken as a human related exception and no deadline extension is granted; the successful payment operation that takes place after the normal deadline is ignored. The situation of scenario 3) is covered by clause 4.1.1: the buyer fails once due to a business failure (PayBF), so a 5d deadline extension is granted to the buyer, and the right to cancel is granted to the seller. The buyer succeeds in his second attempt (PayBC) while the seller decides not to cancel. In scenario 4), the payment fails three times (NoFail=3): a TecFail followed by two BizFail without cancellation from the seller, so the business transaction is stopped as stipulated by clause 4.4. In the last scenario the payment succeeds in the second attempt (PaySC) while the seller successfully exercises his right to cancel (CanC) after the buyer’s first attempt to pay fails produces a business failure (PayBF); if the executions of a successful payment (PayS) and successful cancellation (CanC) overlap, it is possible that (as shown in the figure) the event PayS is processed at the B2B interaction coordinator after CanC, if this happens, the seller needs to refund the payment (RefS) as stipulated by clause 4.3.

6. Implementation

The core components of the resolution service described in Section 5 (relevance engine, contract rules, event queue, event logger and timer) have been implemented [3]. The service relies on Drools (JBoss rule engine [10]), for the decision capabilities of the relevance engine and for rule management. Additional Java components for Drools implement the functionality required for the manipulation of ROP sets, historical queries and timer management, using Java statements within an augmented version of the Drools rule language. This augmented version of the Drools rule language has a more Java-like syntax, and is more verbose than the notation used in Section 5.1; it is therefore less human readable. A tool that takes in the high level notation of Section 5.1 and translates it to the augmented version of the Drools rule language is currently under implementation.

7. Assessment of the service

The exception resolution service presented in this paper is based on a trusted third party, conceptually located between the two business partners. Just as there are third party payment gateway services (e.g., [11]) that monitor and facilitate card payments involving buyers, sellers and banks, we speculate that in future, there could be third party contract-based services for monitoring, exception resolution, support for cross-organisation business process coordination etc. In this regard, our paper represents a first attempt at examining how exception resolution can be performed.
At current stage, the only functionality provided by the third party is the resolution of potential exceptions under the assumption that the business partners operate in good faith: they have to reason for generating and notifying malicious events (say with incorrect time stamp) to the synchronisers or for sending malicious messages to each other (see Fig. 3). These and other security related issues such as authenticity and non—repudiation could be addressed with the help of a non-repudiable interaction service as discussed in [12]; if needed such a service could be integrated into the exception resolution service. This enhancement would prevent malicious business partners from abusing the service. Another issue that needs further exploration is the constraints imposed on contract clauses; our discussion is built on the assumption that at application level most contract violations can be regarded as violation of deadline constraints. However, we are aware that contract clauses might contain a great variety of constraints (see [13]) that are not necessarily abstracted as deadline violations. This issue needs deeper exploration in particular with contracts with emphasis on non—functional requirements.

8. Related work

Exception handling mechanisms have been studied intensively in the field of fault-tolerant systems. Xu et. al. discuss the need for exception resolution when exceptions can be raised concurrently by different cooperating threads in an application [14]. Electronic contracts have been the subject of interest by several researchers; see [15] for a review of the state of the art. A common feature of much of the work on electronic contracting is their focus on the logical aspects of business interactions, without taking into account neither the impact of timing and validity constraints of B2B messaging, or the fault tolerant and concurrency issues, that we address. An exception is [16], where the authors highlight the complexity of handling exceptional situations such as the cancellation of a purchase order due to infrastructure or human related events; they argue that to be effective, a compensation mechanism should take into consideration the state of the two interacting applications. Although no solution is presented, the discussion presented is illuminating; our paper presents a concrete solution.

The idea of providing electronic contracts with discretionary mechanisms to react (as opposite to prevent) to contract violations in accordance with their causes (accidental, due to force majeure, intentional, etc.) is discussed in [17]; the focus of the paper is on settlement of potential disputes after the occurrence of contract violations related to non-functional requirements.

Law-Governed Interaction, LGI, is an early work on contract driven coordination [18]. LGI is a ‘law enforcer’ that regulates the interaction between two or more autonomous and distributed agents. Unlike our work, timing and message validity constraints that are an essential part of B2B messaging are not considered in LGI. The work on Business Contract Language (BCL) is based on the Deontic Logic operators, and has strong interest in the modeling of temporal constraints [19]. However, it does not specify what to do when a time constraint is violated. Exception handling in collaborative interaction is a subject of concern in the multi-agent community. In [20] for example, the authors suggest a taxonomy of exceptions that roughly matches our approach of distinguishing infrastructure related exceptions from human or organisation related. The paper does not discuss design and implementation details. A taxonomy of exceptions and of exception handlers within the context of workflow management is presented in [21]; the paper assumes the existence of a central point of control to allocate resources, activities and to handle exceptions when the execution of an activity completes abnormally. We regard this approach complimentary to ours since it can be used locally by business partners to manage their business processes. FCL (Formal Contract Language) is a Deontic Logic based contract language with the expressive power of the Deontic operators to express normative statements with obligations, prohibitions and reparations to violations [22]. In [23] an event—driven—architecture for cross organisational business processes is discussed; events are used to model normal and exceptional outcomes; however, exceptional outcomes cover only what we call business failures. This is also true for exception handling covered by contract enforcers discussed in [24] and [25]. The need of exception handling in Web service composition is recognized in [26]. However, the computation model is client--server whereas ours is peer--to--peer.

9. Concluding remarks

We have analysed exception handling in electronic contracting and presented an architecture of a third party exception resolution service. The service not only performs precise exception resolution but also supplies sufficient coordination information to business partners to prevent them from executing non-contract-compliant operations. Further, we presented notations that take due consideration of the underlying B2B messaging
and are particularly convenient for specifying exceptional clauses.

Acknowledgements

This work has been funded in part by UK Engineering and Physical Sciences Research Council (EPSRC), Platform Grant No. EP/D037743/1, “Networked Computing in Inter-organisation Settings”.

References