School of Computing Science,
University of Newcastle upon Tyne

A Method for Specifying Contract Mediated Interactions

Carlos Molina-Jimenez, Santosh Shrivastava
and John Warne

Technical Report Series

CS-TR-914

June 2005

A Method for Specifying Contract Mediated Interactions

Carlos Molina-Jimenez, Santosh Shrivastava and John Warne

School of Computing Science, Newcastle University, Newcastle-Upon-Tyne, UK
{Carlos.Molina, Santosh.Shrivastava, J.P.Warne}@ncl.ac.uk

Abstract. To form and automatically manage partnerships within a virtual organisation, it is necessary to have an electronic representation of the contract governing business relationships that can be used to mediate the rights and obligations that each interacting entity promises to honour. The paper describes a general method of representing business interactions using a widely used modelling language Promela and discusses how to represent permissions, obligations, prohibitions, actors (agents), time constraints, and message type checking; that is, all the basic parameters that compose most typical business contracts. Two levels of contract representations are described: implementation neutral, and implementation specific, that is a refinement of the former to include technical details such as acknowledgements and synchronization messages that form an important part of any implementation.

1. Introduction

An increasing percentage of eBusiness communication and computation activity in eCommerce, eScience, and eGovernment domains will be carried out by organisations participating in collaborative ventures called Virtual Organisations. We define a Virtual Organisation (VO) as a strategic alliance among a group of cooperating organisations that share services electronically – say using Web/Internet technology – for the accomplishment of a set of mutually beneficial business goals; these arrangements being made such that each organisation continues to maintain its own autonomy, except for the mutually agreed undertakings of the alliance.

A central requirement of VO operational management is to enable organisations to regulate access to their service resources in a manner, which honours their individual resource sharing policies both securely and with integrity. Regulating such access is made difficult since each potentially accessible organisation might not unguardedly trust the others. Accordingly, all organisations within a VO will require their interactions to be strictly controlled and policed. There will therefore be a need for all business process relationships to be underpinned by guarded trust management procedures.

A conventional business partnership is typically governed by rules laid down in a paper-based contract. These rules express what operations (actions) the business partner are permitted, obliged and prohibited to execute. In addition, the rules stipu-

late when and in what order the operations are to 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 the purchase order the goods have to be delivered, etc.

To form and automatically manage partnerships within a VO underpinned by guarded trust management procedures, it will be necessary to have electronic representations of contracts that can be used to mediate the rights and obligations that each interacting entity promises to honour. In the worst case, violations of agreed interactions are detected and notified to all interested parties (for this, an audit trail of all interactions will need to be maintained).

This requirement implies that the original natural language contract that is drawn by lawyers and other non-technical people has to undergo a conversion process from its original format into a piece of executable code or *executable contract* (*x-contract* for short) that works as a mediator of the business conversations. This conversion process involves the creation, with the help of a formal notation, of one or more computational models of the contract with different levels of details. This is the central topic of this paper. In particular, the contributions of the paper are the following.

- We describe a general method of representing business interactions using a widely used modelling language Promela [1] and discuss how it can be used to represent permissions, obligations, prohibitions, actors (agents), time constraints, and message type checking; that is, all the basic parameters that compose most typical business contracts. Our motivations for using Promela here is that such a representation can be validated with the help of the accompanying Spin model-checker tool [2].
- There can be several ways of deploying the business conversation mediator (e.g., reactive, proactive). Our representation method can be applied to all of these deployment models.
- We propose two levels of contract representation. (i) *Implementation neutral*: free of technical details related to technology-related interactions; in other words, specifying only business action interactions (for example, issue a purchase order, send payment, etc.). Such a description can be model checked and used for improving the original natural language contract to be free from various forms of inconsistencies as discussed in our earlier work [3]. (ii) *Implementation specific*: a representation (also amenable to model checking) that is a refinement of the former to include technical details such as acknowledgements and synchronization messages that form an important part of any implementation; the details will vary depending upon the implementation techniques and standards that are selected. As an example, the paper describes how an implementation neutral representation can be refined to Rosettanet [4] specific representation. Such a representation can be used by technical people for implementing the actual x-contract for business conversation mediator.

2. Deployment models for contract mediation

A question that arises is, how is the x-contract going to interact with the cross organizational business process and where can the x-contract be deployed? From the point of view of interaction, the x-contract can be reactive or proactive. A *reactive contract* intercepts messages exchanged between the business partners and reacts by approving or disapproving them; on the other hand, a *proactive contract* drives the cross organizational business process by inviting the business partners to send legal messages (right type, right sequence, time, etc.). Conceptually speaking, the x-contract is placed in between the two business partners so that it can regulate their business interactions. Deployment can be either *centralized* or *distributed*, this gives us four deployment models discussed below, where for illustration purposes we assume an interaction from buyer to seller:

(i) *Reactive Central*: The contract is deployed in a trusted third party (TTP), see Fig. 1(a). The x-contract is reactive in that it intercepts (1) and analyzes (2) the messages exchanged between the two business partners; legal messages are forwarded (3) to their final destination whereas illegal ones are dropped (3').

(ii) *Proactive Central*: The contract is deployed in a TTP, see Figure 1(b); the x-contract is proactive in that it coordinates the conversational interactions between the two organisations by invitation only. It sends (1) an invitation message to the business partner; the response is received (2) by the x-contract and analyzed (3); legal messages are forwarded (4) to the seller, whereas illegal ones are dropped (4').

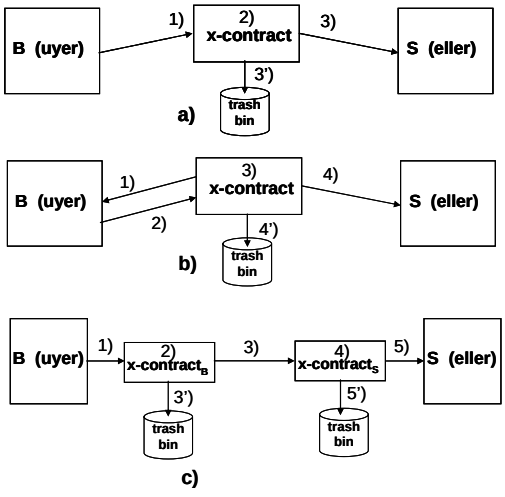

Fig.1. Deployment models (a) reactive central (b) proactive central and c) reactive distributed.

(iii) *Reactive Distributed*: Distributed version of reactive central: the contract is split and deployed in two TTPs, see Fig. 1(c).

(iv) *Proactive Distributed*: Distributed version of proactive central.

Which particular model is suitable in a given VO setting is a very interesting research problem, but not within the scope of this paper; we do not discuss either the actions to be taken by the mediator after dropping illegal messages. We note that distributed deployments face the difficult challenge of keeping contract state information synchronised at both ends. For example, a valid message forwarded by the buyer's x-contract could be dropped at the seller's end because intervening communication delays render the message untimely (and therefore invalid) at the seller side. State synchronisation is necessary to ensure that both the parties either agree to treat the message as valid or invalid. One approach that uses a non-repudiable state synchronisation protocol [5] is described in [6]. In the rest of this paper, due to space limitations, we will assume just the reactive central deployment model.

3. Representing business interactions

3.1. Business conversations

We assume that an x-contract is composed out of $N \geq 1$ conversations. We define a *conversation* as a small business activity executed between two business partners to perform a well defined task, such as issue a purchase order, process payment, refund money, cancel purchase order, etc. It is worth clarifying that by small in our definition of conversation we mean that the number of messages exchanged by the two business partners is small enough that one can reason about them with currently available tools; in practice the size of each conversation will be determined by the process specification standard used by the business partners.

For the sake of simplicity in this paper we consider only sequential composition of conversations.

3.2. Specifying conversations

The challenge here is to find a convenient formal notation that captures all or at least the most important parameters present in most business conversations. Business contracts can be abstracted as a set of permissions (P), obligations (O) and prohibitions (F) that are expected to be fulfilled by actors (also called agents or role players) for the benefit of others by means of performing (or not performing) operations (also called actions). We define a *permission* as an action that an actor, for example a buyer or a seller, is allowed to perform; for instance, "The buyer is allowed to use his discretion to send a purchaser order to the seller" is a buyer's permission for the benefit of the seller. Likewise, an *obligation* is defined as an action that an actor is expected to perform; an example of a seller's obligation for the benefit of the buyer is "The seller is obliged to respond to the buyer within three days after the receipt of the purchase order". A *prohibition* is an action that an actor is not expected to perform; an

example of a seller's prohibition for the benefit of the buyer is "The seller shall not send offers to the buyer unless they are requested". The execution of a permission operation is optional in the sense that there are no penalties for not executing it; conversely, a failure to execute an obligation operation and daring to execute a prohibited operation is considered a contract violation and the offending actor may be subjected to a *sanction*. A sanction can take different forms, for instance, it can grant the offended actor a permission, the offending actor can be refused a permission or the offending actor can be assigned a new obligation (for example, the obligation to pay a fine). In several applications, permissions, obligations and prohibitions are dischargeable as they become and cease to be in effect depending on the occurrence of events.

A promising approach for modelling the concepts we have just discussed is Deontic Logic, a formal notation that is sometimes referred to as the logic of permissions, obligations and prohibitions. The form of Deontic Logic that has been thoroughly studied is the Standard Deontic Logic (SDL) of von Wright [7]. However, as argued in [8], SDL can precisely describe impersonal ought ("it ought to be that the payment is sent") but it is not expressive enough to describe situations where actions are assigned to specific agents ("it ought to be that the payment is sent by Alice"). Another limitation of SDL is that it is static in the sense that it cannot describe permissions, obligations and prohibitions that become and cease to be in effect depending on the occurrence of time and other events. We are aware that currently there are several researchers exploring the possibility of enhancing SDL with additional logical constructs to overcome its limitations; for instance, there are suggestions to mix constructs from SDL with constructs from Modal Logic, Temporal Logic, Logic of Action or from their combinations. We discuss this work in the section on related work.

These combinations result in hybrid logic systems that can certainly express complex situations; unfortunately, as pointed out in [9], such logical systems have not yet been thoroughly studied and understood, consequently, the logical rigour of a contract expressed in such notations is questionable.

Our view is that it will take time for Deontic Logic approaches to reach a degree of maturity where the contract designer can automatically verify the correctness of his notation by proof-theoretical means or model checking. This fact discouraged us from using Deontic Logic notation to describe our contracts and motivated us to resort to Promela, perhaps a less elegant, yet a practical solution that is widely used for protocol specification and validation. In the next section we describe how Promela can be used for our purposes.

3.3. Implementation neutral representation: an example

In this section we will discuss an example of very small (hypothetical) business contract that stipulates business action interactions between a buyer and a seller for the purchase of goods.

1 Offer to buy

1.1 The buyer may use his discretion to send a purchase order to the seller.

1.2 The seller is obliged to confirm acceptance or rejection of the purchase order within 24 hrs of receiving the purchase order.

2 Payment

2.1 The seller is obliged to send an invoice to the buyer within 7 days of accepting the purchase order.

3 Invalid messages

3.1 The buyer and the seller are forbidden to send invalid messages.

4 Sanction

4.1 Failures to honour obligations and prohibitions will result in fines equal to 20% of the cost of the item. The offended party shall be granted permission to issue an invoice notification to the offending party.

4.2 Failure to respond to a fine shall be sorted out outside this contract.

5 Synchronization and handling of transaction failures

5.1 Should the buyer and/or the seller detect a technical failure that prevents them from continuing the normal course of a transaction, they are obliged to send a failure notification message by any other means.

Table 1 lists the permissions, obligations and prohibitions that compose the contract: P stands for permission, O for obligation and F for prohibition (forbidden). The number after P, O and F is the number of the clause in the contract from where the permission, obligation or prohibition was extracted. Notice that in the contract, clause 3.1 specifies a prohibition for the buyer and for the seller; to distinguish between these two cases we named them $F3.1_B$ and $F3.1_S$, respectively. Similarly, $P4.1_B$ and $P4.1_S$ stand for permission for the buyer and the seller, respectively, extracted from clause 4.1.

Permissions	**Subject**	**Beneficiary**	**Sanction**
P1.1 Send purchase order.	buyer	seller	none
$P4.1_B$ Issue invoice to fine.	buyer	seller	none
$P4.1_S$ Issue invoice to fine.	seller	buyer	none
Obligations			
O1.2 Send confirmation within 24 hrs.	seller	buyer	$P4.1_B$
O2.1 Send invoice within 7 days.	seller	buyer	$P4.1_B$
Prohibitions			
$F3.1_B$ Send invalid messages.	buyer	seller	$P4.1_S$
$F3.1_S$ Send invalid messages.	seller	buyer	$P4.1_B$

Table 1. Permissions, obligations, prohibitions and sanctions.

As it is, the above contract might not be detailed enough for the technical people in charge of creating its executable version, yet it contains information of great value for

this stage, for instance, it has enough information to begin reasoning about correctness of the contract. We believe that reasoning about the contract at this early stage is important because, text contracts are very likely to contain inconsistencies, to detect them we need to convert into a formal notation (a computational model of the contract) and validate it, perhaps with the help of automatic software tools. To illustrate our ideas, we will build a *reactive central* Promela model.

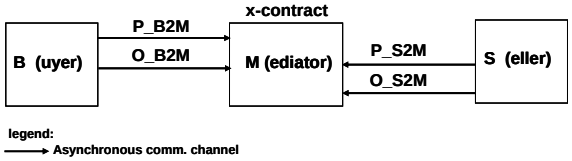

Fig. 2. Reactive central Promela model.

To make our paper self contained, we will very briefly discuss the main features of Promela [1]. Statements in Promela are either executable or blocked. A process trying to execute a blocked statement waits until an event that makes the statement executable occurs. Control flow in Promela is based on guarded commands which are represented by a double colon. Let us assume that $Stat_1, \ldots, Stat_6$ are Promela statements and analyze the following construction where the symbol "`->`" is a statement separator that can be interpreted as a casual relation between its left and right statements.

```
if
::(Pay == 100)-> Stat1 -> Stat3          /* option1 */
::(Pay != 100)-> Stat2 -> Stat4-> Stat6 /* option2 */
fi
```

In the above construction either the sequence of statements of $option_1$ or of $option_2$ executes. Guards are not necessarily mutually exclusive, if more than one guard is executable, one of them is selected randomly. Messages are transferred from one process to another over asynchronous channels. The statement *chan!msg* can be executed by a sending process to send the message *msg* over the channel *chan;* whereas the statement *chan?msg* can be executed by a receiving process to receive a message from the channel *chan*, and store it in the variable *msg*. Promela supports typed messages; the type of the message is a constant sent together with the message. A receiver will block until the expected typed message arrives through the channel. For example, the receiver executing the structure will block until either a message of type `T1` or `T2` is available from the channel so that it can be copied into the variable `val`.

```
if
::chan?T1(val)->…/*block until msg of type T1 arrives */
::chan?T2(val)->…/*block until msg of type T2 arrives */
fi
```

With this background in mind we can now present our contract represented as Promela code. The notion of obligation and permission as well as the notion of the role

player obliged to perform an operation and the beneficiary of the operation is captured in the name of the communication channel; thus a channel named O_B2M suggests that an obligation (O) is expected to be fulfilled by the buyer (B) for the benefit of the mediator (M); strictly speaking, the beneficiary of the operation is the seller, the name of the mediator appears in the name of the channel because the mediator is in control of the interaction; the receipt of the message at the mediator is taken as fulfilment of the obligation.

A complete Promela model for the system shown in Fig. 2 would include the Promela description of three processes (the buyer, mediator and seller) communicating over the two channels; by complete here we mean a model that can be validated with Spin. To save space, we will show only a simplified version of the mediator; we do not show the mediator forwarding messages to their final destination; this would involve two additional channels in Fig. 2, namely, a mediator to seller (M2S) and a mediator to buyer (M2B); in the same order, the Promela code would include mediator to seller (M2S!PO(val)) and mediator to buyer (M2B!ACCEPT(val)) send statements, the former is shown commented and in bold font on the OFFERTOBUY construction. We do not show either different types of ACK messages. We believe that this simplified code is explicit enough to help us understand (bare-eyed) the behaviour of the contract.

```
/* prefix/suffix B and S stand for Buyer and Seller   */
/* PO=purchase order,INVOICENOTIF=invoice notification*/
/* INVMSG=invalid message,val=value                   */
mtype={PO, ACCEPT, REJECT, INVOICENOTIF, INVMSG}
Proctype Mediator  (…)

B_OFFERTOBUY: /*B permitted to send PO to S */
  if
  ::P_B2M ? INVMSG(val)-> goto S_SANCTION_B /*S fines B*/
  ::P_B2M ? PO(val)->/* M2S ! PO(val)*/ goto S_CONF_PO
  fi

S_CONF_PO:/*S obliged to confirm PO within*/
  if      /* 24 hrs or pay fine          */
  ::O_S2M ? INVMSG(val)-> goto B_SANCTION_S /*B fines S*/
  ::timeout             -> goto B_SANCTION_S /*B fines S*/
  ::O_S2M ? REJECT(val)-> goto ENDCONTR_OK
  ::O_S2M ? ACCEPT(val)-> goto S_PAYINVOICE
  fi

S_PAYINVOICE: /*S obliged to send invoice within 7days */
  if
  ::O_S2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
  ::O_S2M ? INVMSG(val)-> goto B_SANCTION_S /*B fines S*/
  ::timeout             -> goto B_SANCTION_S /*B fines S*/
  fi
```

```
B_SANCTION_S:/*B permitted to fine S*/
  if
  ::P_B2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
  ::P_B2M ? INVMSG(val)      -> goto ENDCONTR_DISP
  fi


S_SANCTION_B: /*S permitted to fine B*/
 if
 ::P_S2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
 ::P_S2M ? INVMSG(val)      -> goto ENDCONTR_DISP
 if
ENDCONTR_DISP: /*end of contract: dispute*/
/* do something here */

ENDCONTR_OK: /*end of contract: success*/
/* do something here */
```

We refer the reader to [3], where we describe how correctness properties can be model checked. Once implemented as an executable code, the contract above will guarantee that only legal messages (right type, right sequence, and time) reach their final destination. This is a great advantage for the buyer's and seller's applications since they can blindly take incoming messages as correct and act upon them under the guarantee that they have already been approved by the mediator; furthermore, the applications are guaranteed that illegal messages sent accidentally will never reach their counterpart. A proactive version of the contract would also offer the buyer's and seller's applications the guarantee that they will be precisely instructed what actions to perform next.

4. Implementation specific representation

The contract shown in the previous section is not complete enough for technical people commissioned to convert into an x-contract: they will need to agree on the implementation related messages to be exchanged in order to enable business interactions to occur. The exchange of these messages will need to be expressed as additional permissions, obligations and prohibitions that the business partners are expected to honour.

To show what an implementation-oriented contract looks like, we will assume that the seller and the buyer have agreed to use the widely adopted Rosettanet process specification standard [4]. In Rosettanet, a buyer is expected to use the Request Purchase Order partner interface processes, PIP 3A4, to express its desire to buy. On the order hand, the seller is expected to use the Notification of Invoice PIP (PIP 3C3) to invoice the buyer. The specification of these two PIPs includes sending both business action messages and business signal messages. A graphical representation of interaction, which includes the two conversations, is shown in Fig. 3. Notice that the receiver of a business action message has the obligation to acknowledge it by sending a business signal message back.

The modified English text version of the implementation oriented contract is shown next. This version is different from the original one in that, it includes (in addition to the business actions messages) business signal messages to help the two business partners synchronize their interactions. The new clauses appear in bold font.

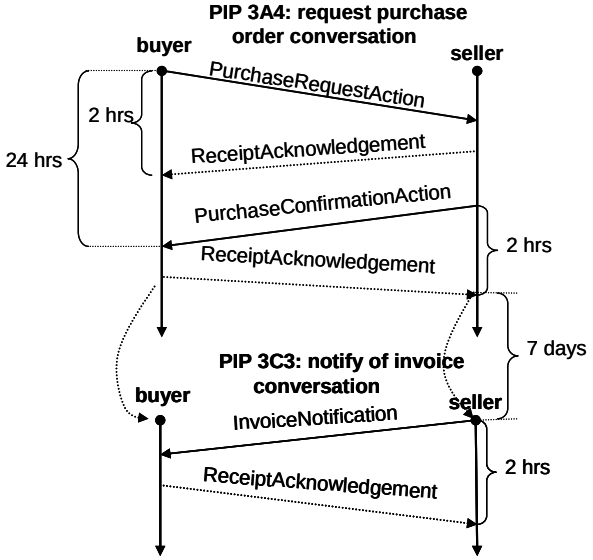

Fig. 3. Rosettanet PIPs.

1 Offer to buy

1.1 The buyer may use his discretion to send a purchase order to the seller.

1.2 The seller is obliged to acknowledge the purchase order within 2 hrs of receiving the purchase order.

1.3 The seller is obliged to confirm if the purchase order is accepted or rejected, within 24 hrs of receiving the purchase order.

1.4 The buyer is obliged to acknowledge the purchase order confirmation action within 2 hrs of receiving the message.

2 Payment

2.1 The seller is obliged to send an invoice to the buyer within 7 days of accepting the purchase order.

2.2 The buyer is obliged to acknowledge the invoice notification within 2 hrs of receiving it.

3 Invalid messages

3.1 The buyer and the seller are forbidden to send invalid messages.

4 Sanctions

4.1 Failures to honour obligations and prohibitions will result on fines equal to 20% of the cost of the item. The offended party shall be granted permission to issue an invoice notification to the offending.

4.2 ***The offender is obliged to acknowledge the invoice notification within 2 hrs of receiving it.***

4.3 Failure to respond to a fine shall be sorted out outside this contract.

5 Synchronization and handling of transaction failures

5.1 Should the buyer or the seller detect a technical failure that prevents it from continuing the normal course of a transaction, it is obliged to send a failure notification message by means outside this contract.

5.2 The counterpart is obliged to acknowledge the failure notification message within 2 hrs of receiving it.

We list the permissions, obligations and prohibitions of the implementation oriented contract in Table 2.

Permissions	**Subject**	**Beneficiary**	**Sanction**
P1.1 Send purchase order.	seller	buyer	none
$P4.1_B$ Issue invoice to fine.	buyer	seller	none
$P4.1_S$ Issue invoice to fine.	seller	buyer	none
Obligations			
O1.2 Ack purchase order within 2 hrs.	seller	buyer	$P4.1_B$
O1.3 Send confirmation within 24 hrs.	seller	buyer	$P4.1_B$
O1.4 Ack confirmation within 2 hrs.	buyer	seller	$P4.1_S$
O3.1 Send invoice within 7 days.	seller	buyer	$P4.1_B$
O3.2 Ack invoice within 2 hrs.	buyer	seller	$P4.1_S$
$O4.2_B$ Ack invoice to fine within 2 hrs.	buyer	seller	none
$O4.2_S$ Ack invoice to fine within 2 hrs.	seller	buyer	none
Prohibitions			
$F3.1_B$ Send invalid messages.	buyer	seller	$P4.1_S$
$F3.1_S$ Send invalid messages.	seller	buyer	$P4.1_B$

Table 2. Permissions, obligations, prohibitions and sanctions.

In Fig. 4., we show the mediated version of PIP 3A4. Notice that the mediator acts as a time stamping authority; for instance, the sending time of a purchase order is taken as the time when the mediator receives it; likewise, the receiving time of a receipt acknowledgement is taken as the time when such a message is received at the mediator. As one can imagine, conversation for PIP 3C3 produces a similar pattern; it is not shown here to save space. As we did with the implementation neutral version of the contract, we will now present the Promela specification of this Rosettanet specific contract.

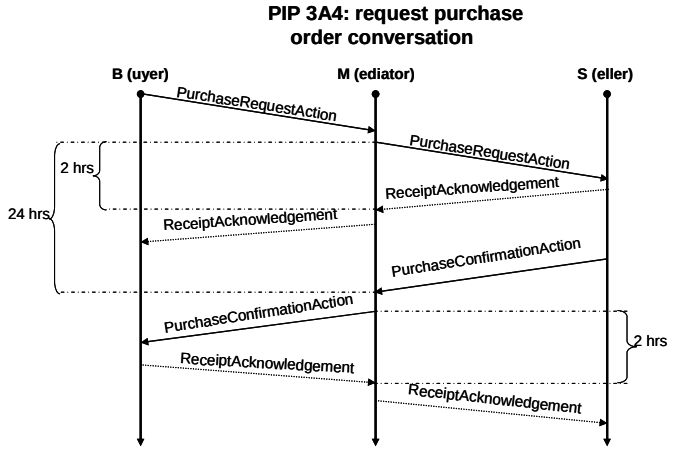

Fig. 4. PIP 3A4 mediated by a reactive contract.

```
/* prefix/suffix B and S stand for Buyer and Seller   */
/* PO=purchase order,INVOICENOTIF=invoice notification*/
/* INVMSG= invalid message, val=value                 */
mtype={PO,ACCEPT,REJECT,INVOICENOTIF,ACK,INVMSG}
Proctype Mediator  (…)

B_OFFERTOBUY: /*B permitted to send PO to S*/
 if
 ::P_B2M ? INVMSG(val)-> goto S_SANCTION_B /*S fines B*/
 ::P_B2M ? PO(val)-> /* M2S ! PO(val) */goto S_ACK_PO
 fi

S_ACK_PO: /*S obliged to ACK PO within 2 hrs*/
 if       /*or pay fine                        */
 ::O_S2M ? INVMSG(val)-> goto B_SANCTION_S /*B fines S*/
 ::timeout              -> goto B_SANCTION_S /*B fines S*/
 ::O_S2M ? ACK(val)->goto S_CONF_PO /*PO ACK rcv from S*/
 fi

S_CONF_PO:/*S obliged to confirm PO within*/
 if       /*24 hrs or pay fine              */
 ::O_S2M ? ACCEPT(val)-> goto B_ACK_ACCEPT /*PO accpted*/
 ::O_S2M ? INVMSG(val)-> goto B_SANCTION_S /*B fines S */
 ::timeout              -> goto B_SANCTION_S /*B fines S */
 ::O_S2M ? REJECT(val)-> goto B_ACK_RJECT  /*PO rejcted*/
 fi

B_ACK_RJECT:/*B obliged to ack REJECT within*/
 if         /*2 hrs or pay fine               */
 ::O_B2M ? ACK(val)    -> goto ENDCONTR_OK
 ::O_B2M ? INVMSG(val)-> goto S_SANCTION_B /*S fines B*/
 ::timeout             -> goto S_SANCTION_B /*S fines B*/
 fi
```

```
B_ACK_ACCEPT:/*B obliged to ack ACCEPT within*/
 if          /*2 hrs or pay fine             */
 ::O_B2M ? INVMSG(val)-> goto S_SANCTION_B /*S fines B*/
 ::timeout            -> goto S_SANCTION_B /*S fines B*/
 ::O_B2M ? ACK(val)   -> goto S_PAYINVOICE
 fi

S_PAYINVOICE:/*S obliged to send payment invoice */
 if          /*within 7 days or pay fine         */
 ::O_S2M ? INVMSG(val)-> goto B_SANCTION_S /*B fines S*/
 ::timeout            -> goto B_SANCTION_S /*B fines S*/
 ::O_S2M ? INVOICENOTIF(val)-> goto B_ACK_PAYINVOICE
 fi

B_ACK_PAYINVOICE:/*B obliged to ACK invoice within*/
 if              /*2hrs or pay fine               */
 ::O_B2M ? ACK(val)   -> goto ENDCONTR_OK
 ::O_B2M ? INVMSG(val)-> goto S_SANCTION_B /*S fines B*/
 ::timeout            -> goto S_SANCTION_B /*S fines B*/
 if

B_SANCTION_S:/*B permitted to fine S*/
 if
 ::P_B2M ? INVMSG(val)      -> goto ENDCONTR_DISP
 ::timeout                  -> goto ENDCONTR_OK
 ::P_B2M ? INVOICENOTIF(val)-> goto S_ACK_FINE
 fi

S_ACK_FINE: /*S acknowledges fine*/
 if
 ::O_S2M ? ACK(val)    -> goto ENDCONTR_OK
 ::O_S2M ? INVMSG(val) -> goto ENDCONTR_DISP
 ::timeout             -> goto ENDCONTR_DISP
 fi

S_SANCTION_B:/*S permitted to fine B*/
 if
 ::P_S2M ? INVMSG(val)      -> goto ENDCONTR_DISP
 ::timeout                  -> goto ENDCONTR_OK
 ::P_S2M ? INVOICENOTIF(val)-> goto B_ACK_FINE
 fi

B_ACK_FINE: /*B acknowledges fine*/
 if
 ::O_B2M ? ACK(val)    -> goto ENDCONTR_OK
 ::O_B2M ? INVMSG(val) -> goto ENDCONTR_DISP
 ::timeout             -> goto ENDCONTR_DISP
 fi

ENDCONTR_DISP: /*end of contract: dispute*/
 /* do something here */

ENDCONTR_OK: /*end of contract: success*/
 /* do something here */
```

5. Related work

Formal specification of business contracts has been studied by several authors. Of particular relevance is the work reported in [8, 10]. The formal notation used in this work is inspired by Deontic Logic. After arguing that the standard Deontic Logic is not expressive enough to specify personal agents and temporal constraints [8], the authors augment the standard Deontic Logic notation with operator to remedy the situation. The result of this is that obligations, permissions and prohibitions can be expressed as personal commitments with strict time constraints. Thanks to this notation one can naturally express obligations, permissions and prohibitions for a given role player within a certain period and reason about both deontic and temporal inconsistencies in the duties assigned to the role player. One can detect unwanted situation where a given role player is obliged and forbidden or permitted and forbidden during the same interval of time. Likewise one can detect unwanted situations where a role player cannot fulfil his duties due to time overlaps. The authors of this work claim that it is possible to build software tools to perform these verifications, however, no results have been reported in this direction yet.

A contract framework is presented in [11]. To overcome the lack of expressiveness of standard Deontic Logic to express agents and temporal constraints, the author introduces additional logic constructors; namely, the notion of subjects, beneficiaries and deadlines. A contract is conceived as a list of normative statements whose general format can be represented as $ns_i : \delta \rightarrow \theta_{s,b}(\alpha < \psi)$, where ns_i ($i \geq 1$) is a label (a name) that identifies the i^{th} normative statement of the contract; δ stands for a condition that might eventually become true; θ stands for permission, obligation or prohibition; *s* and *b* are the subject and beneficiary of θ, respectively; α is an action to be performed by *s* for the benefit of *b*; and ψ is a deadline; "$\rightarrow$ " is the conditional auxiliary symbol. If we assume that θ stands for obligatory, the above expression should be read as "it is obligatory that *s* performs α for the benefit of *b* given that δ, before the deadline ψ".

This paper does not discuss how the contract represented in Deontic Logic notation can be validated. A positive aspect of the paper is the explicit relationship between obligations and sanctions: violations result in sanctions which are represented as obligations or prohibitions; on this basis, signatory parties can use their discretion to fulfil their obligations or to pay sanctions, all within the normal course of the execution of the contract. The paper identifies the issue of synchronization of contract states shared between business partners and suggest that the business partners could possibly send, after or before the execution of a deontic normative statement, synchronization messages to synchronize views over shared states. From the point of view of their meaning, these messages are not different from the signal messages sent within Rosettanet PIPs. Whereas this work suggests that synchronization messages are optional, we argue that signal messages have to be explicitly stipulated in the text of the implementation oriented contract and consequently, in its formal specification.

An approach somewhat similar to ours in spirit is presented in [12], where permissions, obligations and prohibitions are mapped into ECA (even-condition-actions).

An executable contract becomes a set of ECA rules deployed within a trusted third party and placed between the two business partners to drive the interaction between the two business partners. Unlike our work, this paper does not use the concept of contractual conversations, so it is not clear how the specification can be mapped to specific conversations, such as PIPs.

In our paper we have been concerned with monitoring and enforcement of business operation clauses. Equally important aspect (not studied here) is monitoring the level of Quality of the Service (QoS) of services offered within a VO. This monitoring is concerned with the collection of statistical metrics about the performance of a service to evaluate whether a provider complies with the QoS that the consumer expects. This aspect is examined in [13], where related work in this area is also presented.

6. Conclusions

We have presented a contract specification technique that meets three requirements: i) Be expressive enough to capture most of the parameters (obligations, guarded execution, deadlines, etc.) that describe most business contracts. 2) Support automatic analysis of the contract model so that possible inconsistencies can be found and eliminated. 3) Produce a contract model that can be used as a sound skeleton structure on which the executable version of the contract can be built with minor difficulties. We also argued that from the interaction point of view, business contract are not different from communication protocols. On this basis we proposed the use of Promela (a widely available language for modelling communication protocols, provided with a model checker) for contract representation as it meets the requirements mentioned above.

References

[1] Gerard J. Holzmann: Design and Validation of Computer Protocols. Prentice Hall, (1991).

[2] Gerard J. Holzmann: The SPIN model checker, Primier and reference manual. Addison-Wesley, (2004).

[3] Ellis Solaiman, Carlos Molina-Jimenez, Santosh Shrivastava: Model Checking Correctness Properties of Electronic Contracts. In Proc. of the Int. Conference on Service Oriented Computing (ICSOC03).Trento, Italy, Dec. 2003. Lecture Notes in Computer Science Vol. 2910, Springer (2003).

[4] Rosettanet implementation framework: core specification, V2, Jan 2000. http://rosettanet.org

[5] Nick Cook, Santosh Shrivastava and Stuart Wheater: Distributed Object Middleware to Support Dependable Information Sharing between Organisations. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2002). Washington DC, Jun. (2002), pp.249-258.

[6] Carlos Molina-Jimenez, S.K. Shrivastava, E. Solaiman and J. Warne: Contract Representation for Run-time Monitoring and Enforcement. In Proc.of the IEEE Conference on Electronic Commerce (CEC'03). Newport Beach, CA, Jun. (2003), pp. 103-110.

[7] George Henrik von Wright: An Essay in Deontic Logic and The General Theory of Action. Acta Philosophica Fennica, North-Holland Publishing Company-Amsterdam, (1968).
[8] Peter Linington, Zoran Milosevic and Kerry Raymond: Policies in Communities: Extending the ODP Enterprise Viewpoint. In Proc. of The Second Int. Enterprise Distributed Object Computing Workshop (EDOC98). La Jolla, San Diego, Ca, Nov. 2-5, (1998).
[9] M. J. van den Hoven and G.J.C. Lokhorst: Deontic Logic and computer-supported computer ethics. *Metaphilosophy*, Vol. 33, N. 3, (2002).
[10] Olivera Marjanovic and Zoran Milosevic: Towards Formal Modelling of e-Contracts, In Proc. of The EDOC2001 Conference, Seattle, USA, Sep. (2001).
[11] Mathias Sallé: An Agent-based Framework for the Automation of Contractual Relationships. In Proc. of The AAAI-2002 Workshop on Agent-Based Technologies for B2B Electronic Commerce. Jul. (2002), Edmonton, Alberta, Canada (also, HP Technical report, HPL-2002-133).
[12] Oliver Perrin and Claude Godard: An Approach to implement contracts as trusted intermediaries. In Proc. of The First IEEE Intl. Workshop on Electronic Contracting (WEC04). San Diego, CA, Jul. (2004).
[13] Carlos Molina-Jimenez, Santosh Shrivastava, Jon Crowcroft and Panos Gevros: On the Monitoring of Contractual Service Level Agreements. In Proc. of The First IEEE International Workshop on Electronic Contracting (WEC). San Diego, CA, Jul. (2004), pp. 1-8.