Conversation-oriented Protocols for Contract Negotiations

James E. Hanson[1], Zoran Milosevic[2]

[1]*IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA*

jehanson@us.ibm.com

[2]*Distributed Systems Technology Centre, Brisbane, Qld 4072, Australia*

zoran@dstc.edu.au

Abstract

The expression of contracts in computer readable form, and the development of automated tests for completeness and well-formedness of contracts, has opened the door to significant advances in automating contract negotiations. To meet the needs of automation, such negotiations must follow explicitly specified message-exchange protocols. But to meet the needs of the negotiating parties, these protocols must be independent of the decision-making processes driving them as well as neutral to the outcome of the negotiations. In this paper we illustrate how both needs may be simultaneously met by a small set of conversation policies employed within a general purpose conversation support architecture.

1 Introduction

In e-business, as in the traditional business, contracts are a key governance mechanism for regulating activities between enterprises. A contract expresses a statement of promises of parties involved to act in a certain way, including the consumption of resources, fulfilling requirements and the communication of information.

The e-business setting however requires more comprehensive contract management support due to the increasingly dynamic, event-driven environment and the proliferation of contracts (both in terms of their sheer number and their complexity). This includes not only contract drafting facilities, possibly being the predominant contract management tools today (mostly word processor centered), but also tools to support other stages in contract life-cycle. This encompasses increasing levels of automation for activities such as: contract negotiation, legal validity checking, digital notarization and storage, notifications about upcoming milestones, performance monitoring, negotiation over variations during execution, enforcement, and contract analytics and re-negotiations.

To support automation of these contract management activities a formal expression of contract is needed. At present, there are several styles of such expression of contract. One approach is to specify contract in terms of expected behavior of parties involved in the contract, including the statement of sanctions associated with contact non-compliance. Examples of such approaches are [3], based on the Petri Nets and deontic logic formalism, and work reported in [2], making use of Genetic Software Engineering methodology.

Other approaches deal with the specification of contract semantics in terms of the conditions that need to be monitored in real-time, both to ensure the detection of existing or potential breaches to contract and to facilitate the incorporation of various corrective measures [1], [6]

The above two classes of approaches begin with a premise that a contract has been negotiated and the agreement is expressed in a form suitable for electronic processing. However, apart from some support in [3], these approaches do not consider the process of contract establishment and how the *objectives* and the corresponding *intentions* of parties involved in negotiation influence the structure and content of an agreed contract.

The recent work reported in [4] and [5] represents an initial attempt to address this problem. Both of these solutions express contract formation process in terms of speech act patterns [17]. These expressions are suitable to be used for contract negotiations because the speech acts formalism well reflects intentions of parties interested in collaboration and can be used to validate that their conversation is coherent and complete. However, neither of these two contributions provides support for a suitably conversational style of interaction to support exchange of multiple, correlated messages that for example carry speech acts semantics.

This paper can be regarded as a contribution in this direction. Namely, we focus on the problem of contract negotiation and in particular on the conversational

capabilities and protocols needed to facilitate any kind of negotiation in e-contracting processes.

In the paper we propose the use of generic conversation support model developed in [7] for the exchange of messages between parties during the contract negotiation stage. The central part of this conversation model is the use of pre-programmed patterns of message exchange called Conversation Policies (CPs). Examples of the messages used in contract negotiations and that are part of these patterns are offer, counter-offer, acceptance and rejection. We note that they can be expressed by using the speech acts formalism. When used in combination, e.g. offer and acceptance, speech acts provide the basis for establishing obligations and other deontic concepts typical to contract expressions.

The conversation support model draws on an extensive and growing body of work in the software agents community.[8] A key differentiator from that work is the attempt to span, in a uniform way, the broad range between highly structured programmatic processing of messages (as is found in Web Services, for instance), on the one hand, and the kind of dialogues humans engage in, on the other. The conversational model of ref. [7] fosters the evolutionary growth of hybrid systems in which there is sufficient structure for automated tools to be of use, but there is also sufficient flexibility and richness of feedback that people interact with them in a natural way.

We use the conversation model to support various levels of contract negotiation, namely at the level of *contract template* or at some lower level contract description such as individual *contract clauses* or *negotiable contract variables* e.g. price and quality of service (QoS). Thus the messages used in the course of negotiations above can be parameterized to carry any of these three levels of contract abstraction. These levels correspond to the contract representation developed as part of Business Contract Architecture (BCA) work [6]. In addition, this negotiation solution can be regarded as a specific realization of the BCA Contract Negotiator role.

We note that an important requirement for protocols supporting negotiations is that they must not try to channel the negotiation in one way or another. Rather, they provide a set of guidelines that negotiating parties can use in order to make their negotiations easier to carry out. In other words, the CPs provide a standardized "syntax" for the negotiation, which should be separate from specific business logic that entails the strategy of parties.

This is not necessarily true of protocols in general. The voluminous literature on mechanism design in economics [10] deals, in essence, with crafting protocols that will induce the negotiating parties to reach agreements that have some predefined property regarded as desirable by the mechanism designer--e.g., prices that divide the surplus equally, etc. The disadvantage of such things is that, if either of the parties doesn't want to be manipulated in this way, he'll simply choose not to use the protocol.

There are similarities between the CP based protocols and some aspects of ebXML BPSS specification [13]. Both models support exchange of messages that carry business documents. However, while BPSS mandates that two such messages should form a Business Transaction (i.e. messages enacted by the Requesting and Responding Business Activities) CPs are more open-ended, and can express more complex sequencing constraints than can BPSS choregraphy. On the other hand the specification of concurrency operators as part of BPSS orchestration is beyond the scope of CPs.

The remainder of this paper is organized as follows. In section 2 we review the conversation support architecture that provides the runtime support for contract negotiations. Section 3 describes the key contract negotiation requirements and highlights several contract parameters which are often subject of negotiations. These variables are discussed in the context of a business contract language of a broader scope. Section 4 describes a set of conversation policies designed to support contract negotiations, and section 5 discussion extensions to cover additional cases. Finally, in section 6 we conclude with a brief summary of some future work areas.

2 Conversation support architecture

The back-and forth, iterative nature of contract negotiations requires the adoption of the *conversational model* of interactions [7]. In the conversational model, two parties first set up a two-way, open-ended conversational session, and then proceed to send each other asynchronous messages within the context of that conversation. One common way to implement this is for one party to initiate the conversation by sending a message consisting of a request to start a conversational session, and including a globally unique conversation ID to be used by both parties for the duration of the session. Thereafter, each message sent in the conversation contains, in its header, that conversation ID.

Once the conversational session has been established, the conversational model specifies that the two parties begin following a common conversational protocol, or *conversation policy* (CP), which specifies a set of messages, sequencing constraints, and timeouts defining the message-exchange protocol. Each party executes its own copy of the CP, using it to carry out its own side of the conversation by assuming one of the roles defined in the CP.

The execution of CPs is most naturally factored into a separate *conversation management* subsystem. This sybsystem mediates between the *messaging* subsystem defining the "plumbing" by which messages are exchanged, and the decision logic (a/k/a business logic), which drives the interaction by selecting which of the allowed messages, as defined by the CP, to send in any given instance, and as well supplies the values of the business data to be sent in the message. This is shown in Fig. 1.

One likely scenario is for the messaging system to make use of Web Services technology [11] for both sending and receiving messages. The conversation management system validates and classifies incoming messages, translates them into the appropriate decision data (a/k/a business objects), and hands the classified decision data to the decision logic. Similarly, for outputs of the decision logic (i.e., decisions it has made), the conversation management system verifies that the decision is in conformity with the CP being executed (i.e., that it's "allowed"), translates the decision data into the appropriate message, and hands it over to the messaging system for delivery. This separation of concerns naturally insulates those aspects of the interaction which must be common to the participants-- messaging formats and sequencing constraints--from those which need not be--business objects and decision logic.

The decision logic in this case includes a subsystem for establishing the completeness and well-formedness of contracts in a similar way as was proposed in [2]. In addition, the decision logic could include support for checking legal validity of the contract under negotiation, as for example was discussed in the BCA context [16]. This subsystem may involve some automated tools and is shown in the upper right box in Figure 1.

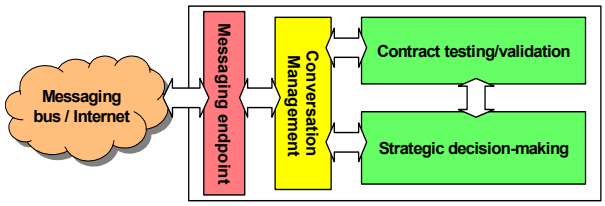

Figure 1 Conversation support architecture and link to contract negotiation business logic

Another aspect of the decision logic is related to a strategic decision maker of any character. This may be a human interacting directly with the conversation-management system, or it may be a semi-automated workflow system in which humans play predefined decision-making functions, or role, or may, in the future, approach full automation. This is depicted by the bottom right box in Figure 1.

Of essence here is the fact that every aspect of the decision logic is hidden from the other party, which only "knows" about the CPs being used. This feature permits businesses to adopt any internal development plan, i.e., to automate in any way, according to any timetable, without depending on the other parties involved.

3 Contract Negotiation

The characteristics of the conversation policy model described above are suitable to be exploited to support conversation-oriented protocols for contract negotiation. We begin with a summary of key requirements for contract negotiation infrastructure, followed by the description of key contract constructs that are usually subject of contract negotiations.

3.1 Contract negotiation requirements

We define contract negotiation informally, as a process of making and adjusting offers between potential parties willing to be involved in an economic transaction - until an agreement is reached or the process is terminated without an agreement.

In an electronic form negotiation can be implemented by sending electronic messages between parties. These *negotiation messages* carry contract related offers and counteroffers. Examples of negotiation message types include proposeTemplate, counter-proposeTemplate, addClause, replaceClause, accept, reject and so on.

Since negotiation infrastructure can be linked to both automated and human decision makers it is desirable that such an infrastructure supports a varied degree of structuredness of negotiations. This refers to both the structure of *negotiation process* and the structure of *negotiation message content* exchanged.

To support predictable negotiation processes, an automated negotiation infrastructure needs to be able to interpret the incoming negotiation messages and consequently negotiation message types should belong to a predefined message set. These message types are used to fully define the negotiation process and they are the basis for the design of negotiation engines - such as the one we propose in the next subsection. Therefore, the negotiation process should be fully structured irrespective whether humans or automated entities are involved in negotiations.

As far as the structure of negotiation messages is concerned there needs to be more flexibility regarding the variability of their contents. This is to provide a certain degree of freedom for decision makers involved in negotiations, in particular human negotiators, allowing them to send any negotiable variable, be it chosen from a predefined set or created ad hoc. In the former case for example the message contents can be restricted to support some specific e-commerce standards data types or organization specific contract clauses. In the latter case the content can be left unspecified to allow any data to be inserted within (again, predefined) negotiation message types, e.g. any kind of new clause to be proposed and counter-proposed.

In summary, negotiation engines require a structured negotiation process while the content of the negotiation messages can be unstructured. These two principles are fully supported in the contract negotiation engine proposed in section 4.

Contract negotiation can involve two or more parties. In case of multiparty negotiations a great deal of such negotiations can be reduced to multiple bilateral negotiations. For example two organizations can agree to jointly respond to a tender issued by a third organization and in that case these two organizations can engage in prior negotiations and subsequently choose prime contractor for the tender. The prime contractor may be involved in some negotiations with the issuer of the tender. There are also some other negotiations that are true multi-party negotiations as when all parties have the same status in a submission as for example is case in a response to some technical standard RFP.

When considering electronic contract management applications, in general, contract negotiations can be treated independently from the contract performance monitoring activities. This is currently the approach taken within BCA - mostly driven by the pragmatics of our system development - and reflected in the formalism of our Business Contract Language described below.

3.2 Business Contract Language

Business Contract Language (BCL) defines a number of concepts for the specification of contract conditions necessary for the support of real-time contract management activities. The BCL is part of our recent work related to BCA and the language is evolving as we learn more from the business problems associated with contracting.

For the purpose of this paper we describe a subset of the BCL of particular relevance to contract negotiation. For readability purposes, we chose to explain this BCL semantics in slightly informal manner – using natural description of the key structural language constructs rather than providing XML schema for the language. We note that we use XML as a basis for the description of our contract language, both in terms of its structural aspects of relevance for the negotiation and for the behavioural aspect of the language, as in part presented in [2].

The outermost language construct in BCL is the *contract template*. Each contract template consists of a list of contract clauses (which can reference each other if needed) and an element reserved for signature details. A contract template is introduced to reflect the fact that many business contracts are predefined – expressed in terms of standard contract forms. These contract forms can be regarded as partially completed contracts and they serve as a basis for the creation of specific contract instances that reflect objectives of each trading partner.

Each *contract clause* contains a human readable text and can also include one or more fields that are placeholders for data to be filled for a contract to be instantiated. Typically, these fields are the items that each party to the contract can negotiate, and we refer to such an item as *NegotiatedItem*. Examples are start and end date of contract, quantity of goods, QoS parameters, roles etc. These negotiable items can be compared to the formal parameters and the actual parameters are the values that have been agreed after the negotiation process.

Each clause can also contain one or more *subclauses*. Our language currently supports recursive expression of sub-clauses. Another option is hierarchical representation. Further, each clause has a corresponding type, such as preamble clause, ordinary clause or termination clause.

A *signature* element of the contract template contains a list of two or more *SignatureDetails* elements providing a placeholder for two or more <person, digital signature> pairs.

Finally, every contract can have association with one or more *executable policies* which we introduced mainly for real-time monitoring aspects of contracts. These policies specify conditions to be monitored and they are referenced by the BCA contract monitor role. Although we envisage a possibility of negotiations of these executable policies, this negotiation is more related to the agreement on the deployment options for the contract monitoring and this problem is beyond the scope of this paper. Thus the focus of the BCL description is mostly related to the structural aspects of contract. Other BCL aspects, such as those concerned with expression of policies are described elsewhere [2].

We believe that the BCL constructs of template, clause and negotiated items are the most frequently used

constructs as subject of negotiations. They are key semantic constructs that can be used as part of messages sent via the negotiation engine introduced in section 2. In addition, the exchange of signatures typically confirms the negotiation process resulting in a legally binding agreement.

This negotiation process is explained in section 4 below, but before we describe it in detail, we make a few remarks regarding the nature of BCL.

Current BCL, driven by contract monitoring needs was developed without taking into account negotiation perspective. However, there can be additional benefits by establishing a better link between these two aspects of contract management, and in particular a link between the negotiation messages and the obligation monitoring activities. For example, the history of negotiations could be used to determine certain characteristics of trading partners – which can be exploited say in dispute resolutions and in determining reputation characteristics of the parties.

One way of establishing a semantic link between negotiation messages and the obligations stated in the agreed contract is to consider speech acts semantics [17], long used by the software agents community. Speech acts express units of communication between people or agents. Essentially, every offer exchanged in contract negotiation is an instance of a speech act. Speech acts usually have a meaning when they come in pairs, resulting in some kind of social effect, such as obligations, prohibitions, authorizations or accomplishments [4]. Thus, the motivation behind the consideration of speech acts is that they establish the basis on which obligations and other policies in the contract are expressed. For example when a supplier of a service registers their services and their price structure in a service registry, this action is regarded as provider's statement of promise – a special kind of speech act. When subsequently a buyer accepts this offer, this places obligations for the supplier to deliver the service according to the promise and for the buyer to do payment according to the service contract.

4 Conversation policies for negotiation

We represent the conversation policies for contract negotiation in a multi-level scheme, in which one conversation policy serves to frame the negotiation as a whole, making use of a sequence of other conversation policies to deal with negotiations at different levels of detail. These detail levels are, in order of decreasing generality:

1. Negotiation of the contract template.
2. Negotiation over clauses.
3. Negotiation over variables within the clauses.

The overall evolution of the conversation is from general to specific, though as we will see, both parties have opportunities for revisiting a more general level after having operated on a more specific level.

In addition to specifying the sequencing constraints on messages, CPs may place restrictions on the time period over with the negotiations may continue. Time limits may be specified for the negotiation as a whole, for any of its phases, or for individual messages. This is described further in section 5.2; for brevity, timeouts are omitted from the rest of the discussion in this section.

In the subsections below, we describe the structure of the main contract negotiation policy and of the "child" policies that represent negotiations at the different levels of detail. The policies themselves are specified in cpXML [14], an experimental XML dialect for specifying state-machine based conversational protocols. We have chosen to present them here in the form of state-transition diagrams in order to emphasize the structure of the protocols rather than the details of the cpXML schema.

4.1 NegotiateContract CP

The NegotiateContract CP is the main or "framing" conversation policy, that lays out the different options available for use by the negotiating parties. It structures the conversation as a sequence of negotiations over increasingly specific aspects of the contract, each with its own CP. First, the two parties negotiate over which contract template to use; then they move on to negotiations over the clauses of the contract; and finally, they negotiate over the detailed values of the NegotiatedItems specified by the clauses. Both parties have the opportunity to renegotiate both clauses and values. Finally, after both parties have signaled that they are done negotiating, the conversation moves into a confirmation phase, which is also specified by a CP.

The state diagram for the NegotiateContract CP is shown in Fig. 2, which should be interpreted as follows. Both parties start their interaction in the state with the double line on its border (in this case, the oval state labeled `start`). Thereafter, the party playing role `A` sends a `StartNegotiation` message to the party playing role `B`, as shown by the label on the edge exiting the start state. The CPs of both parties then enter state labeled `LoadChild: "NegotiateTemplate"`. This in an "in-child" state, shown as a hexagon, and means that both parties, upon entering the state, begin executing the named CP (i.e., NegotiateTemplate; cf. below), while keeping the state of the parent CP (i.e., NegotiateContract) unchanged. When the child CP is done executing it returns a string identifying which terminal state it ended up in,

and control reverts to the parent CP. Executing a child CP is analogous to making a subroutine call.

The transitions from an in-child state correspond to the possible return codes from the child CP: in this case, indicating that either a contract template has been agreed upon, or the negotiation has been cancelled. In the former case, the NegotiateContract CP then enters another in-child state, this time executing the NegotiateClauses CP. This pattern is repeated again for the NegotiateVariables CP. Following the return from the NegotiateVariables CP, both parties either signify that they are done negotiating, or reopen the negotiation, either of the clauses or of the values. Finally, if both parties are done negotiating, the ConfirmContract CP is executed.

The approach shown is sufficiently flexible that it can handle any degree of negotiability, such as those illustrated by examples below.

Many business contracts, especially contracts between a business and a consumer (e.g., contract for personal credit cards, financial contracts such as Direct Debit/Credit contracts, real-estate contracts prescribed by local regulatory authorities etc) are such that neither the contract template nor the clauses are negotiable. Hence, the negotiations in these contract types would move quickly to the NegotiateVariables CP, to settle on the values of negotiable variables defined in the clauses of the given template. Attempts by one party to enter into an extended negotiation over the template or the clauses would simply be vetoed by the other. In some cases, even the negotiation over values would be relatively short, as in the case where one party fills in the form, and the other party merely checks that the form has been filled out.

Another class of business contracts, typically found, e.g., in the construction industry, involves a significant amount of negotiation of the clauses that go into it. Here, in certain cases there is a set of predefined clauses available– maybe taken from a repository of standard clauses--and in such cases the middle level of negotiations can apply.

At the outermost level of the CP, the contract template level, there is a scope of negotiation, when for example different regulatory bodies (e.g. state legislations) prescribe different templates for the same contract type. This can either come under the simple contracts as in the first example or more complex as in the second.

One feature of the nested structure of the NegotiateContract CP is that it modularizes the protocols for the different phases of the negotiation. Since the details of each phase are encapsulated within a child CP, changing any phase is merely a matter of loading a different child CP.

In addition, by splitting the negotiation into distinct phases, the NegotiateContract CP helps reduce the computational (or intellectual) effort involved in any one decision. For example, although the conversational protocol would be much simpler if all proposals and counterproposals consisted of complete contracts, partially or even completely filled out, such a protocol would require complete reanalysis of the contract with each offer, effectively treating each proposed contract as a completely new artifact.

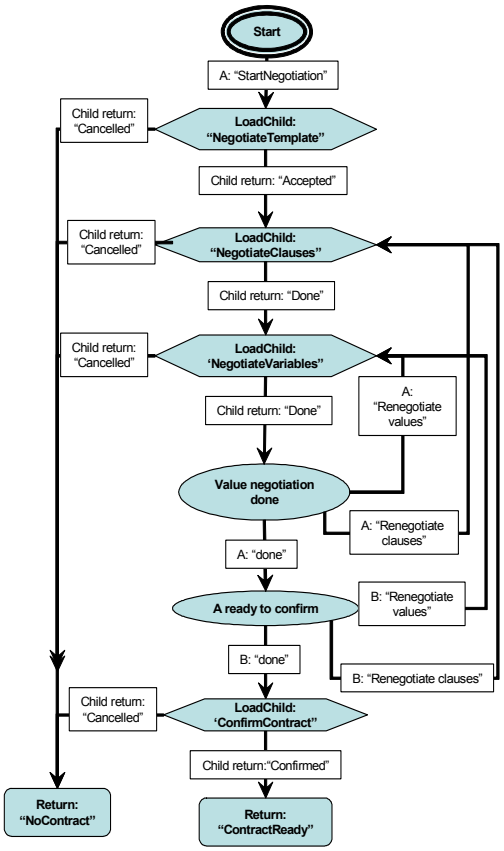

Figure 2. NegotiateContract conversation policy

4.2 Negotiation of the template

For template negotiation, the NegotiateTemplate CP shown in Fig. 3 is appropriate. Here the two parties exchange proposals for the contract template to adopt as a starting point: either by exchanging unambiguous names (i.e., URIs) of published templates, or by sending an actual text of the proposed template. The NegotiateTemplate CP's structure is partly derived from the fact that there can be only one template, which must be agreed upon by the two parties before the negotiation can proceed.

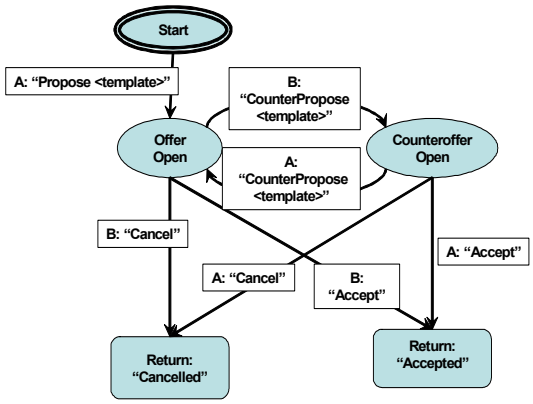

Figure 3. NegotiateTemplate conversation policy

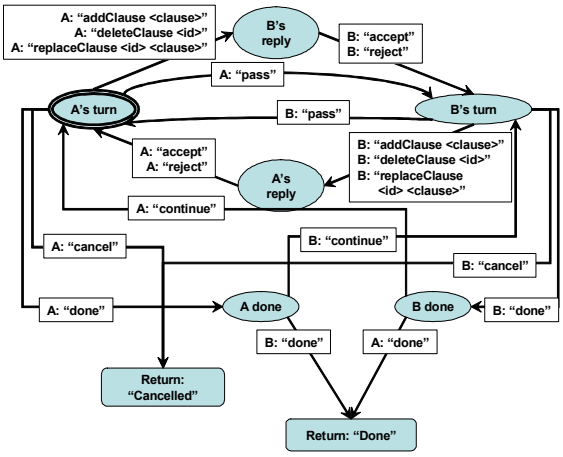

Figure 4: NegotiateClauses conversation policy

As is shown in Fig 3, the two parties exchange offers and counteroffers until a template is agreed upon, or until the negotiation is cancelled. In either case, the NegotiateTemplate CP enters a terminal state, at which time control reverts to the parent CP, NegotiateContract.

4.3 Negotiating clauses

In negotiating clauses, it is natural for the two parties to take turns proposing modifications, which may be either adding a new clause, removing an extant one, or replacing one. Since the number of such modifications is indeterminate, it is necessary for the entire process to be repeatable any number of times. These two features are both evident in the NegotiateClauses CP, shown in Fig. 4. In the figure, the transitions containing multiple messages, such as the one connecting the state labeled `A's turn` to the state labeled `B's reply`, are shorthand for multiple parallel transitions, each with its own message.

There are two obvious scenarios for clause negotiation. In the first scenario, the clauses themselves are predefined, e.g., as part of the contract template. In that case, the two parties, after agreeing to use a particular template, choose to limit the set of possible clauses to those defined in the template. The proposal message would contain the ID of the clause being proposed.

The other scenario is when entirely new clauses are proposed by one or the other party. For these "unstructured" contracts, the proposal message contains the text of the clause itself. The CP specifies the syntax of the clause, but not its content. I.e., it would prescribe that the proposal must contain a well-formed XML *clause* element.

The latter scenario is clearly more difficult for automated systems to cope with. Thus we expect that, where predefined clauses are available, automated negotiators would prefer them.

Note, however, that the NegotiateClauses CP is applicable to both scenarios, and even permits negotiation in situations where one party chooses to propose entirely new clauses, and the refuses even to consider them, e.g., because they are new.

A variation on the NegotiateClauses CP, in which proposals may contain more than one clause (e.g., the `addClause <clause>` message would be replace by `addClauses <list-of-clauses>`) is also acceptable here, and may prove to be preferable, since it helps the negotiating parties deal with constellations of interdependent clauses.

4.4 Negotiating variables

Finally, we come to the most specific conversation policy: the negotiation of variables in clauses. Since, at this level, correlations between variables may become important, it is not sufficient to take up variables one at a time. Rather, offers consist of sets of variables.

Figure 5 below shows a conversation policy for carrying out these negotiations. Note that the NegotiateVariables CP itself consists of two phases: first a mutually agreed-upon set of variables (i.e., *NegotiatedItems*) is found, and then the values for those variables are established. Following that, both parties have an opportunity to go

back to the beginning, and start the selection process again for a new set of variables.

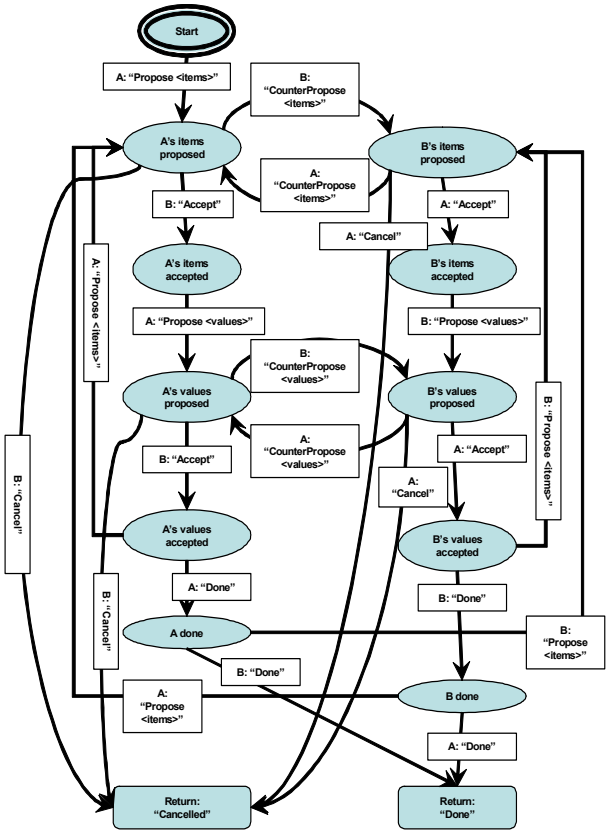

Figure 5: NegotiateVariables conversation policy

This separation is convenient from the point of view of automating decision logic, because it separates out to inherently different activities: defining the attribute space within which one is negotiating, versus exchanging offers that consist of points in that space. Human decision-makers may not have difficulty combining these two, but automated negotiation strategies generally presume that the space is fixed, and furthermore tend to work best when it is low-dimensional. (For an illustration of the state of the art in automated negotiation, see for example [12] and the references therein.)

Another CP which may sometimes be used instead of the NegotiateVariables CP shown in Fig. 5, is the Asymmetric Multi-Attribute Bidding (AMAB) CP described in [9]. In the AMAB CP, one party (acting as the "seller") presents sets of alternative offers, from which the other party (the "buyer") selects his favorite. The seller may then confirm the buyer's selection, or present another (presumably further refined) set of offers to the buyer for consideration. This sequence continues indefinitely. The AMAB CP is appropriate in cases where the negotiation is inherently asymmetric—e.g., because the seller has detailed knowledge of the range of options available, while the buyer does not.

4.5 Confirming

Once both parties have signaled that they are done negotiating, the NegotiateContract CP enters a state in which it executes a confirmation CP.

A very simple CP for confirmation is the ConfirmContract CP, shown in Fig. 6. This CP merely provides for one party to send the contract as a whole to the other party for verification. More advanced versions of this CP would include messages for electronically signing the contract, exchanging signed copies, and so forth.

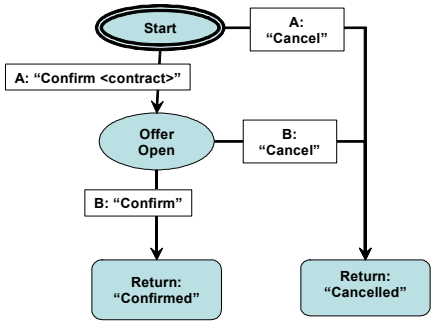

Figure 6. The ConfirmContract conversation policy

5 Additional considerations

Conversation support can be extended to cover a number of situations beyond the negotiation scenarios described above. In this section, we briefly mention a few of the more obvious cases.

Having the conversation-support architecture in place means that the overhead of adding these new protocols is drastically reduced, because the apparatus for specifying and for executing these CPs is already there. The only things that need to be supplied are those things that will necessarily be specific to each individual business-i.e., the decision logic required to drive the CPs.

5.1 CPs for renegotiation and variances

The arguments in section 1 above, which advocated an "outcome neutral" approach to protocol design, apply to protocols for contract renegotiation and for dealing with potential or actual violations. For example, we do not assume that a business will choose to exercise compensatory actions spelled out in a contract, even when

it can, according to the contract, do so, This reflects a common fact of business relationships: frequently, businesses *agree to violate* contracts, for any of a wide variety of reasons. In order to reach such an agreement, the parties involved must negotiate over what parts of the contract to keep in force, what parts to disregard, etc.

Thus we advocate defining CPs for renegotiating already agreed-to contracts; CPs for negotiating variances or exceptions to contracts; and so forth, to be used at the discretion of the parties involved. The former are similar in form to the NegotiateContract CP, except that the template is not negotiable, since it is presumably determined by the actual contract in force. The latter can be represented as the negotiation of a temporary or short-term contract subsidiary to the main one.

5.2 Timeouts in CPs

Any non-terminal state in any CP may be assigned a timeout. Timeouts limit the amount of time the conversation may stay in that state. When timeouts are present on normal states, they impose a limit on the allowed delay before one or the other party must send a message. When present on "in-child" states, they have the effect of limiting the amount of time for the execution of the entire child policy.

For each state for which timeout is specified, there is an associated transition that specifies the new state to change to when the specified time has elapsed. In the CPs above, we have omitted these timeouts and timeout transitions for clarity. In general, we expect the timeout transitions typically to lead to a state in which the negotiation is cancelled, or, in more elaborate situations, to a state in a child CP is started to exchange messages about the timeout itself. E.g., one party would notify the other that a timeout has occurred, and that the negotiation will be cancelled unless a message is received on short order. This latter threat would then be enforced by means of another timeout transition, this time to a "negotiation cancelled" state.

It is often the case that there is some external deadline, which will govern the overall time over which the contract negotiations can be carried out. This can be achieved by executing the entire NegotiateContract CP as a child of a another CP, and placing the deadline on the in-child state within which the NegotiateContract CP is executed.

5.3 Multi-party negotiations

The conversational approach can also be extended to multi-party negotiations. As noted in section 3.1, in many cases a negotiation involving more than two parties can be reduced to multiple bilateral negotiations, conducted serially or in parallel. However, there are still situations in which it would be preferable for the several parties involved to all negotiate together in a single, multi-way conversation.

In a multi-party conversation, the conversation policies define roles for all participants, such that each participant plays a unique role. All messages are multicast to all participants. Care must be taken to avoid ambiguities in state-synchronization that could occur if multiple participants send messages at the same time (this is true of two-party conversations as well, though the problem is less serious there.) For example, one simple technique is for the CP to have participants take turns in a round-robin fashion.

Another feature of such an extension is the need for a multi-party session management CP, which would describe the way in which multiple parties arrive at the start of the conversation, how they drop out without ending it, and so forth. A multi-party version of the NegotiateContract CP would also have to be developed. Beyond the obvious extensions, such as requiring confirmation from all negotiating parties, this latter would also define the ways in which some parties would simply decide to withdraw from the negotiations, leaving the others to continue as far as possible, search for a replacement, etc.

6 Conclusion

In this paper, we have laid out a set of conversation policies for use in contract negotiations, specified in a way that promotes the use of automated or semi-automated negotiators.

In addition to looking at providing appropriate integration points between BCA and the conversation infrastructure architecturally, one of the interesting research directions would be to study links between negotiation processes and the rest of contract management systems. As part of this, it would be interesting to investigate dependences between deontic concepts and speech act formalism and position these in relation to our current business contract language. This is field of significant future research.

7 Acknowledgements

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources)

The authors would like to thank James Cole who has provided XML Schema for service contract, to Glen Nolan for his input into the pragmatics of contract negotiations, and to Prabir Nandi and Kalyani Deshpande for their work on the cpXML specification.

8 References

[1] P. Linington and S. Neal, "Using Policies in the Checking of Business to Business Contracts", *IEEE 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003),* 2003, to appear.

[2] Z. Milosevic and G.Dromey, "On Expressing and Monitoring Behaviour in Contracts", In *Proc. 6th IEEE Conf. on Enterprise Distributed Object Computing (EDOC-2002)*, IEEE Computer Society, 2002, pp. 3-14.

[3] R. Lee, "A Logic Model for Electronic Contracting", *Decision Support Systems*, 4, 27-44, 1988.

[4] W.-J. van den Heuvel and H. Weigand, "Cross-Organizational Workflow Integration using Contracts", *Business Object Component Workshop, OOPSLA 2002*.

[5] P. Mathieu and M-H. Verrons, "A generic model for contract negotiation", www.lifl.fr/SMAC/publications/aisb02-ants.pdf

[6] Z. Milosevic, A. Josang, T. Dimitrakios, and M.A Patton, "Discretionary Enforcement of Electronic Contracts", In *Proc. 6th IEEE Conf. on Enterprise Distributed Object Computing (EDOC-2002)*, IEEE Computer Society, 2002, pp. 39-50.

[7] J. Hanson, P. Nandi, S. Kumaran, "Conversation Support for Business Process Integration", In *Proc. 6th IEEE Conf. on Enterprise Distributed Object Computing (EDOC-2002)*, IEEE Computer Society, 2002, pp. 65-74.

[8] See, for example, M. Greaves and J. M. Bradshaw, eds., *Proc. Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies*, 1999.

[9] J. Hanson, G. Tesauro, J. Kephart, E. Snible. "Multi-agent implementation of asymmetric protocol for bilateral negotiations", *ACM Conference on Electronic Commerce (EC-03),* San Diego, 2003, to appear.

[10] H. Varian, "Economic Mechanism Design for Computerized Agents", In *Proc. First Usenix Conference on Electronic Commerce*, New York, 1995.

[11] http://www.alphaworks.ibm.com/webservices

[12] G. Tesauro, "Efficient Search Techniques for multi-attribute bilateral negotiation strategies", In *Proc. Third International Symposium on Electronic Commerce (ISEC-02)*, IEEE Computer Society, 2002.

[13] "ebXML Business Process Specification Schema", www.ebxml.org/specs/ebBPSS.pdf

[14] "cpXML: Conversation Policy XML Version 1.0" , www.research.ibm.com/convsupport/papers/cpXML-v1.htm

[15] M. Rebstock, "An Application Architecture for Supporting Interactive Bilateral Electronic Negotiations", In K. Bauknecht, S. K. Madria, and G. Pernul (eds.), *Electronic Commerce and Web Technologies*. Springer, New York, pp. 196-205.

[16] Z. Milosevic, D.Arnold, and L.O'Connor, "Inter-enterprise Contract Architecture For Open Distributed Systems:Security Requirements", *WETICE'96*, Stanford Univeristy. 1996.

[17] J. Searle, *Speech Acts*, Cambridge University Press, 1969.