InterNeg Research Papers INR 06/04

Revised version appeared in:

E-Commerce and Web Technologies
K. Bauknecht, M. Bichler and B. Proll (Eds.),
Lecture Notes in Computer Science, Vol. 3182
Springer, New York, 2004, (106-115).

Protocols for Electronic Negotiation Systems: Theoretical Foundations and Design Issues[*]

Gregory E. Kersten, Stefan Strecker and Ka Pong Law

John Molson School of Business
Concordia University
Montreal, Canada
{gregory, strecker, kplaw}@jmsb.concordia.ca

Abstract

Existing electronic negotiation systems (ENSs) typically implement a single, fixed negotiation protocol, which restricts their use to negotiation problems that were anticipated and established a priori by the system designers. The single-protocol restriction limits ENSs' applicability in experiments and in many real-life negotiation situations. ENSs that allow for the use of different protocols also allow for the customization to users' needs and abilities. We present theoretical foundations for the design of flexible and highly customizable protocol-driven ENSs. Our design enables researchers and negotiators to map negotiation activities onto system components and to construct their own negotiation protocols by creating a sequence of layout programs invoking components and rules.

[*] This work was supported with grants from the Initiative for New Economy of the Social Sciences and Humanities Research Council Canada, and the Natural Science and Engineering Research Council Canada.

1. Introduction

The term *e-negotiation systems* (ENS) has been used to describe software that employs Internet technologies, is deployed on the World Wide Web, and capable of supporting, aiding or replacing one or more negotiators, mediators or facilitators [1]. A number of ENSs have been designed, implemented and applied to various negotiation problems. Some systems facilitate negotiation of documents and their joint preparation, e.g., contract negotiations [2], others use email, chat and streaming video software [3, 4]. There are also systems that allow the negotiators to enter offers, which are subsequently sent to human experts who suggest agreements (e.g., http://electroniccourthouse.com). Some ENSs require the installation of a special client, such as SmartSettle (http://smartsettle.com), while other (hybrid) systems combine auction and negotiation elements, such as NegotiAuction [5] or, negotiation and workflow elements, such as GNP [6]. Several of these systems are currently investigated within the SSHRC project on e-negotiation (http://interneg.org/enegotiation). An overview of different ENSs can be found in Shim [7] and Neumann et al. [8].

With two exceptions—SilkRoad [9] and INSS [10]—existing ENSs implement only one fixed negotiation protocol [11]. This restricts the use of ENSs to types of problems and interactions that were assumed and established a priori by their designers. This, in turn, imposes limitation on the behavioural research of the ENSs' use and their efficiency and efficacy, on the ENSs' applicability to support evolving negotiations, and those conducted by users who have different needs, cognitive abilities, and cultural and professional backgrounds.

Ongoing behavioural research in ENS focuses on (i) technology adoption by negotiators, and (ii) the impact of different systems on the negotiation process and negotiated outcomes [12, 13]. Both research directions utilize experimental and empirical methodologies. From this perspective (in particular in experimental studies of ENSs' use and adoption), the assessment of the impact of different system features on the process and outcomes of negotiations requires the use of systems, whose differences and similarities can be easily controlled by the researcher. From a negotiator's point of view, the limitation to a single fixed protocol restricts the use of a particular ENS to the supported class of negotiation problems, which may not include their problem at hand. If, on the other hand, ENS implement negotiation protocols, which apply to a large class of negotiation problems, i.e., are very general, they impose significant cognitive and informational demands on the users who need to make decisions about the selection of tools and features.

Users who use a system to negotiate need to concentrate on the problem and process, and make decision about the concessions rather than compare different tools and decide about system features. It is thus advantageous that: (1) a protocol be constructed for the negotiators based on their characteristics and the negotiation problem and context, or (2) the negotiators decide on a negotiation agenda, which sets a particular formal protocol.

The purpose of this paper is to present the theoretical foundations and discuss design and implementation issues of negotiation protocols. These protocols can be implemented in a software platform [14] and then used for construction of various ENSs. The remainder of this contribution is organized as follows. Section 2 briefly reviews vital elements of a negotiation methodology. Section 3 introduces the theoretical foundations of negotiation protocols and their properties. The design and implementation issues of mapping negotiation activities to system components are discussed in Section 4, and Section 5 presents ongoing and future work.

2. Negotiation methodology

Negotiation methodology describes the methods, procedures, and techniques used to collect and analyze information used in negotiation, the process of communication, exchange of offers and concessions, and arrival at an agreement or deadlock. It is important that these methods and techniques match the negotiator's capabilities, complement each other, do not produce contradictory information and—when used—contribute to the negotiation effectiveness.

2.1 Negotiation process and activities

The use of a methodology has been advocated by negotiation experts, but this advice is often neglected in many face-to-face, email and unstructured negotiations. One of the important contributions of an ENS is to provide a methodology, which matches the negotiators' requirements and is appropriate to their problem. The use of a methodology in an ENS is also required for the tractability of the process and its ease of use.

For the purpose of this work, we consider only two key components of the negotiation methodology: (1) the *negotiation process model,* and (2) the *negotiation protocol*. The process model provides a framework for negotiations; it organizes the activities undertaken by negotiators by grouping them into negotiation phases and by assigning different activities to each phase. It serves as a starting point for the software design and draws its significance from imposing a methodologically sound approach to negotiators [15].

The protocol is a *formal* model, often represented by a set of rules, which govern software processing, decision-making and communication tasks, and imposes restrictions on activities through the specification of permissible inputs and actions [16, 17]. Negotiation protocols are further discussed in the next section.

To our knowledge, there are no behavioural studies on e-negotiations and, therefore, no process model specific to e-negotiation has been developed. For the purpose of designing and implementing an ENS, we use a five-phase model based on Gulliver's eight-phase model [18], which allows for the consideration of a wide range of negotiations, including those supported by ENSS.

Each negotiation phase has its own purpose and set of activities, which are concrete actions undertaken by each negotiator. The purpose of the different negotiation phases is to provide the participants with a framework and rationale for activities conducted in each phase. The consideration of phases helps to specify negotiation activities undertaken and the relationships among them. The phases' main activities are briefly described below.

Planning comprises activities that negotiators undertake individually and jointly. The negotiators formulate their representation of the negotiation problem including the specification of issues and options. In this phase, the negotiators specify their objectives and preferences, and such negotiation-specific constructs as the best alternative to the negotiated agreement and reservation levels. The joint activities in this phase include the selection of the negotiation location and time, and the communication modes the negotiators will use.

Agenda setting and field exploring include the negotiators' discussion about the negotiated issues and their meaning. The result of the discussion may be that new issues and options are added and/or some are deleted. The negotiators may also discuss the protocol they will follow, the timing of the exchanges, the deadline and—in some negotiations—their objectives, priorities and constraints. The result of these discussions is that the negotiators may have to revise the problem, objectives and preferences, and also their strategies and initial tactics.

Exchanging offers and arguments allows the parties to learn about the others' limitations, and to identify the key issues and critical areas of disagreement. During this phase, the parties realize the potential of a compromise and can assess its main features. The analysis of a negotiation may focus on the modification of strategies, the determination of concessions and revision of aspiration levels, and on the restriction of efficient solutions to those which may be acceptable to the parties.

Reaching an agreement means that the parties realize that the negotiation will be successful. Having identified the critical issues, the parties may develop joint proposals or soften their individual limitations. The parties may also identify a limited number of possible compromises.

The negotiation concludes when the negotiators reach an agreement. They evaluate this compromise and consider possible improvements. They may also discuss additional issues which, however, have no impact on the negotiations (e.g., about the agreement implementation).

The negotiation process model provides a framework, but it does not impose any restrictions on the negotiators concerning the sequencing of phases. In any given phase, the negotiators may revisit previous phases and then return to initial phase. Moreover, it often occurs in real-life negotiation that negotiators skip or ignore one or more phases. Although negotiation experts suggest that all phases should be considered, we leave this issue to the protocol designer as there may be specific situation, in which one or more phases should be bypassed.

2.2 Negotiation protocols and activity types

Any negotiation supported by an ENS requires that the software designers precisely define the activities and their sequence using a negotiation protocol [13, 16]. The negotiation protocol defines the activities that are permissible in every state of the negotiation, their sequence as well as input and output requirements. The key concepts used to define the activities and to specify their sequencing are presented in Figure 1. Behavioural theory posits that activities depend on the negotiators' characteristics and the negotiation context (including power distribution, relationship and the relative importance of outcomes). These characteristics determine the negotiators' approaches, their strategies and tactics leading to the selection of specific activities from the negotiation phases.

Behavioural research cannot provide sufficiently precise insights regarding sequencing of activities within each negotiation phase. This is because of the number of possible combinations of the negotiator's characteristics, interdependencies between characteristics of the negotiators, dependence of the negotiators' behaviour on external factors (e.g., relationship with other stakeholders, competing decision problems and the consideration of future situations), and the complexity of the problem and process.

With the exception of well-defined and highly structured negotiations, such as those taking place in procurement of standardized goods, the negotiators cannot follow a strict set of rules defining the activity's sequence. The above mentioned complexities introduce the requirement for providing the negotiators with some degree of freedom in the selection of activities. During the process, the negotiators may wish to review the problem, modify their preferences, add or remove issues etc., which imposes the requirement of some activities to be optional and/or exchangeable for other activities. Also, the negotiators may be forced to undertake certain activities in order to move to the next activity. For example, they should learn about the negotiation problem, consider their own objectives and preferences and evaluate the counterpart's offer before making their own offers. To accommodate these requirements, we distinguish between *mandatory activities* and *optional activities* (see Figure 1).

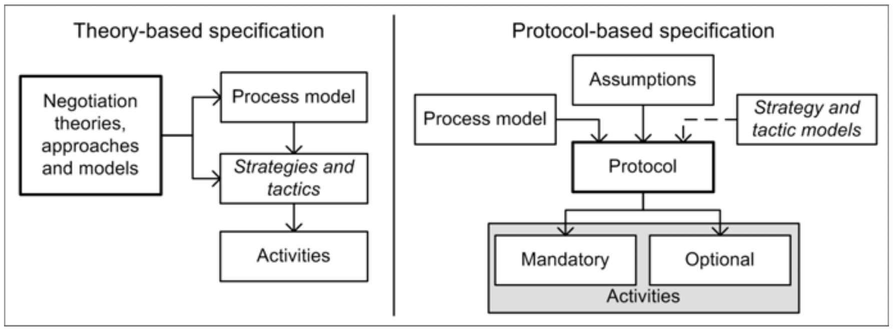
Theory-based specification
Negotiation theories, approaches and models
Process model
Strategies and tactics
Activities
Protocol-based specification
Assumptions
Process model
Strategy and tactic models
Protocol
Mandatory
Optional
Activities

Figure 1. Theory- and protocol-based activity specification

The distinction between mandatory and optional activities is necessarily context-dependent; the same activity may be mandatory in one situation and optional in another. For example, when the user enters the system for the first time, he is required to learn about the negotiation problem; at this time this activity is mandatory. When he logs in the second and subsequent times, learning about the problem should become not mandatory but an optional activity. It is the negotiation protocol that, based on the process model and the assumptions of the protocol designer, categorizes some activities as mandatory or as optional, and modifies this categorization as the negotiation progresses.

The assumptions underlying a specific protocol may reflect the negotiators characteristics (e.g., culture and profession), the type of negotiations (e.g., distributive, integrative, and mixed), and the complexity of the negotiation problem (e.g. one or more issues; constrained or unconstrained). In the research environment, these assumptions may also reflect the needs of the researcher studying the users' behaviour and the system's efficacy.

Different negotiators and negotiation situations require that different protocols be used. The protocols may differ in the sequencing of the same set of activities. Conversely, the same set of activities may be (re-)used in the construction of many different protocols. Negotiation strategies and tactics also may require the use of different protocols. Moreover, different types of negotiations, e.g., single or multiple issues, and different roles of the negotiators, e.g., buyer or seller, require different protocols. Another need for different negotiation protocols derives from the requirements and demands that different stakeholders have regarding their use of an ENS. To meet the requirements of negotiators and researchers, it is essential to equip an ENS with the flexibility to carry out several different protocols and to provide the user or researcher with the possibility of designing new negotiation protocols.

2.3 Process model and negotiation states

The framework provided by the process model is implemented in the negotiation protocol, which is represented by a sequence of activities and rules imposed on the execution of the sequence. Additionally, the execution of a protocol depends on the context of a negotiation, or more precisely, on the current state of a negotiation a user is currently involved in and on the user's earlier actions in that negotiation. The process model reflects the progression of a negotiation as it tracks the completion of phases and activities. An example of the process model, and its phases and activities is given in Table 1. We use these phases and activities to illustrate, in Section 3, formal protocol construction and manipulation.

Table 1: Example of the process model, activities and states

Negotiation phase and activity	**State**	**Abbr.**
1. Planning		
- Negotiation problem	Negotiation case	*NC*
- Preferences and rating	Utility construction	*UC*
- Assessment of alternatives	Alternative construction	*AC*
2. Exchanging offers and arguments		
- Offer and/or message construction	Offer message	*OM*
- Counter-offer assessment	Counterpart's offer	*CO*
3. Reaching agreement		
- Agreement	Agreement reached	*AR*
	Agreement assessment	*AS*
- Closing negotiation	End	*EN*
4. Concluding negotiation		
- Agreement improvement	Agreement improvement	*AI*
- Offer and/or message	Offer message	*OM*
- Counter-offer assessment	Counterpart's offer	*CO*
- Closing negotiation	End	*EN*

Each negotiation activity is associated with an ENS state (see Table 1); however the reverse is not true: The system may be in a state that does not correspond to any negotiation activity. For instance, state AS involves agreement efficiency analysis and does not correspond to any activity. The difference between a negotiation activity and an ENS state is that the former describes a user action, while the latter denotes a user and/or a system action.

3. Negotiation protocols

Every ENS implements a negotiation protocol—even though some system designers do not specify the protocol explicitly—the protocol can be derived from the required and possible interactions between the negotiators and the system. It is sensible to formulate the negotiation protocol explicitly, because it specifies the users' interactions and thus the users need to determine if the system conforms to their requirements. In addition to the interaction transparency introduced by explicit protocols, it also allows for mapping protocols onto negotiation processes. Furthermore, it is also possible to assess the protocols' underlying assumptions and characteristics. Formulation of protocols should follow rules and procedures that correspond to negotiation methodologies.

3.1 Preliminaries and conditions

Following the distinction between mandatory and optional activities (see Figure 1), we distinguish two types of ENS states: *mandatory* or *optional*.

Let:

$S = \{s_1, \ldots, s_N\}$ be the set of all possible states;

M be the set of mandatory states ($M \subset S$);

O be the set of optional states ($O \subset S$);

$s_{\text{start}} \in S$ be the first state of the protocol; and

$s_{\text{end}} \in S$ be the last state of the protocol; it is the protocol termination state.

We assume that $s_1 = s_{\text{start}}$ and $s_N = s_{\text{end}}$.

Every state is associated with at most one mandatory state, which defines a partial protocol sequence, i.e. a sequence is a *pair* of two states:

$$s_i \rightarrow s_j,\ s_i \neq s_j \text{ with } i, j \in I, \qquad \textbf{(1)}$$

where $s_i \in S$; $s_j \in M$; and $|I| = N$ is the number of states.

Using the states given in Table 1, we can formulate several sequences, including the following two sequences: $NC \rightarrow UC$, $UC \rightarrow AC$. State UC is mandatory for NC; the user can move to state AC only after completing activity associated with UC. Similarly, AC is mandatory for UC.

Optional states allow the user (or system) conducting activities within a sequence prior to moving to the next sequence.

$$s_i \rightarrow s_j\ opt\ O_i, \text{ with } s_i \neq s_j,\ s_i \in S,\ s_j \in M,\ O_i \subset O, \qquad \textbf{(2)}$$

where opt is the operator of state association and O_i is the set of optional states associated with state s_i.

Formula (2) is interpreted as follows: The user who is in state s_i can visit states $s_l \in O_i$ multiple times and return from these states to s_i, but he cannot move to any state s_k ($s_k \neq s_j$, $s_j \in S \setminus O_i$), before he visits state $s_j \in$ M. For example, the user who assesses alternatives (AC) can return to the description of the negotiation problem (NC) and revise his preferences (UC), but he cannot move to any other state listed in Table 1, unless he formulates an offer, that is: $AC \rightarrow OM\ opt\ \{NC, UC\}$.

A state may have a null state associated as mandatory state as long as the set of optional states is non-empty. The resulting sequence with a mandatory null state is denoted as:

$$s_i \rightarrow 0\ opt\ O_i \text{ with } s_i \in S,\ O_i \subset O, \qquad \textbf{(3)}$$

where 0 is the null state. Similarly, a state may have a mandatory state with an empty set of optional states ($O_i = \varnothing$):

$$s_i \rightarrow s_j \ opt\ 0 \text{ with } s_j \in M. \qquad \textbf{(4)}$$

Based upon these preliminaries, negotiation protocols are required to meet certain general conditions.

Condition 1: If state s_i has null mandatory state, then at least one optional states has to be associated with s_i, i.e.,

$$s_i \rightarrow 0 \ opt\ O_i \Rightarrow O_i \neq \varnothing. \qquad \textbf{(5)}$$

Condition 1 is required in order for the user to be able to move from the state with which no mandatory state is associated to one or more optional states. Moves between optional states are not considered sequences.

Assume that set O_i in (5) has three elements $O_i = \{ s_j, s_l, s_k \}$. The user can move from any optional state to any other optional state, e.g., he can move from s_j to s_l to s_j. One implication is that all optional states in (5) are elements of O_i.

The second implication of (5) is that to move from s_i or one of its optional states to a state, which is not an element of O_i, is possible only if there is a state in O_i, which is a part of another sequence of type (2) or (3). This requirement is formulated in the following condition.

Condition 2: With every state s_i, which has a null mandatory state, at least one optional state has to be associated, which is part of another sequence, that is:

$$\forall\ s_i : s_i \rightarrow 0 \ opt\ O_i : \exists\ s_l, s_l \in O_i \text{ and } (s_l \rightarrow s_j, s_j \in M \text{ or } s_l = s_{\text{end}}).$$

This condition assures that the user can move from any state to either a state with which a mandatory state is associated or to the termination state s_{end}. Condition 1 and 2 do not assure that every state can be accessed by the user; this is achieved if the following condition is met.

Condition 3: With the exception of the starting state (s_{start}), every state $s_i \in S$ is mandatory and/or optional, i.e.,

$$\forall\ s_i, s_i \neq s_{\text{start}}, s_i \in S : s_i \in M \cup O.$$

Condition 4: A mandatory state may appear only in one sequence of the protocol, i.e.,

$$\forall\ s_i, s_j, s_l \in S,\ s_i \rightarrow s_j \ \Rightarrow \neg\ \exists\ \ s_l \rightarrow s_j, s_i \neq s_l, s_j \in M.$$

The above four conditions define the *required* characteristics of every protocol.

Definition 1: $q(i,k)$ is the *list* of k sequences that begin with state s_i and end with the mandatory state s_{i+k}, such that:

1. s_i is an optional state for s_{i-1};
2. The mandatory state for s_{i+k} is the null state.
3. No state, other than s_{i+k}, in $q(i,k)$ has mandatory null state; and
4. With the exception of s_{i+k} every mandatory state is the first state in *one* other sequence in $q(i,k)$.

The list of sequences $q(i,k)$ is:

$$q(i,k) : s_i \rightarrow s_{i+1},\ s_{i+1} \rightarrow s_{i+2},\ \ldots,\ s_{i+k-1} \rightarrow s_{i+k}\ \wedge\ s_i \notin M \wedge\ s_{i+k} \rightarrow 0. \qquad \mathbf{(6)}$$

List $q(i,k)$ can be represented as a graph starting at state s_i and ending at s_{i+k}. Every state starting a sequence has a mandatory state which starts another sequence, with the exception of the last state s_k. The sequences in the list may or may-not have associated with optional states (see Figure 2).

Let:

J be the index set of lists of sequences; $q(i_j,k_j)$ denotes the j-th list in the protocol;

$Q = \{q(i_j,k_j),\ (j \in J)\}$ is the set of all lists of sequences of the type given by (6) in the protocol;

$P = \{s_i \rightarrow 0\ opt\ O_i,\ i \in I\}$ be the set of all sequences in which the mandatory state is the null state;

S_i ($S_i \subset S$) be the set of states that are elements of the mandatory and optional sets associated with s_i and the states preceding s_i, i.e., $S_i = \{s_{\text{start}}, s_2, \ldots, s_i; O_{\text{start}}, O_2, \ldots, O_i,)$; and

S_{i+} ($S_{i+} \subset S$) the set of states that are elements of the mandatory and optional sets associated with s_{i+1} and states following s_{i+1}, i.e. $s_{i+2}, \ldots, s_{\text{end}}$.

Note that $S_{i+} \cap S_i$ is not necessarily an empty set because some states may appear in more than one optional sets both preceding, including and following s_i.

Definition 2: Negotiation protocol $\wp$ is the 5-tuple:

$$\wp = (S, O, M, P, Q) \qquad \mathbf{(7)}$$

3.2 Protocol completeness

Protocol $\wp$ has a number of states; an obvious requirement is that these states can be visited, that is an activity corresponding to a state can be undertaken. This protocol characteristic is its *completeness*.

Theorem 1: Negotiation protocol $\wp$ is complete if Conditions 1-4 hold and:

1. The state s_{start} occurs only once in the protocol and s_{start} is neither a mandatory nor an optional state for any state $s_i \in S$.
2. The ending state $s_{i_j+k_j}$ ($j \in J$) of each *list* of sequences $q(i_j,k_j)$ is the first state in a *sequence* with a mandatory null state and its set of optional states is non-empty:

$$\forall\ q(i_j,k_j) \in Q : s_{i_j+k_j},\ (ij+kj) \in P \wedge O_{i_j+k_j} \neq \emptyset. \qquad \mathbf{(8)}$$

3. With the exception of the protocol starting state s_{start}, every state s_{i_j} that begins list $q(i_j,k_j)$ belongs to at least one set of optional states:

$$\forall\ q(i_j,k_j) \in Q\ \ \exists\ l \in P :\ s_{i_j} \in O_l. \qquad \mathbf{(9)}$$

4. For every state which is associated with null mandatory state either:

$$\forall i \in P\ \exists\ s_j \in O_i : s_j \rightarrow 0\ opt\ O_j,\ O_j \cap S_{i+} \neq \emptyset \qquad \mathbf{(10)}$$

or

$$\forall i \in P\ \exists\ s_j \in O_i : s_j \rightarrow s_l,\ s_l \in M \cap S_{i+} \qquad \textbf{(11)}$$

5. The protocol termination state s_{end} occurs only once in the protocol and $S_{\text{end}+} \setminus S_i = \emptyset$.

<u>Outline of the proof:</u>

A. If s_{start} occurs in several sequences or optional sets, then—according to the protocol $\wp$–the user can enter the protocol in each of these sequences or optional sets. Then there is no guarantee that all the states can be visited. An illustration of this situation is given in Figure 2. If s_{start} is added to any of the optional sets or one of the states s_i (i = 1, …, 13) is replaced with s_{start}, then the number of states cannot be visited. The only exception is if s_{start} is added to set O_{start}, but this operation makes little sense as then one sequence has two of the same states. For example, replacing s_1 with s_{start} causes that states included in O_{start} cannot be visited.

B. If s_{start} is a mandatory state then the starting state for the sequence, in which s_{start} is mandatory cannot be visited, unless it is an optional state in some other sequence. This situation cannot take place because of 1A.

1. From the definition of the list of sequences (6), it follows that every end state has a mandatory null state. If its optional set is empty, then no other state can be visited from this ending state, and Condition 1 is violated.

2. From the list definition (6), it follows that the state that begins list $q(i,k)$ is not mandatory. If it does not belong to any optional state, then this state cannot be accessed from any other state, and Condition (8) is violated.

3. From (5), we know that every state which has null mandatory state also has nonempty optional set. Assume that for some i ($i \in P$), condition (10) is violated. If this is the case, then there is no state in O_i ($i \in P$), which is an optional state for any of the states which are elements of S_{i+}. Similarly, if condition (11) is violated, then there is no state in O_i ($i \in P$) that is associated with a mandatory state that has not yet been visited. In effect no state in S_{i+} can be accessed from s_i.

4. If s_{end} occurs more than once in the protocol, then there are states that are in between these occurrences. Selection of the first occurrence causes the protocol to terminate without the possibility for visiting states that follow this occurrence. The s_{end} state may be optional (see Figure 2) or mandatory, but it cannot have a mandatory state and thus it cannot have optional states. This is because from state s_{end} no other state can be visited. Thus we have $S_{\text{end}+} = \emptyset$.

An example of a complete protocol is given in Figure 2. 13 states (including s_{start} and s_{end}) are distinguished. There may be many other states that are included in the sets of optional states.

The protocol depicted in Figure 2 has three lists of sequences. In one list, two optional sets are indicated (one associated with s_{start} and the other with s_3). Two other lists have only mandatory states ($s_5 \rightarrow s_6$ and $s_8 \rightarrow s_9$, $s_9 \rightarrow s_{10}$). Following (5), termination states for these sequences have nonempty optional states (O_4, O_6 and O_{10}). Following (9), with the exception of s_{start}, the two states that start a list (s_5 and s_8) are also optional states in, respectively, O_4 and O_6.

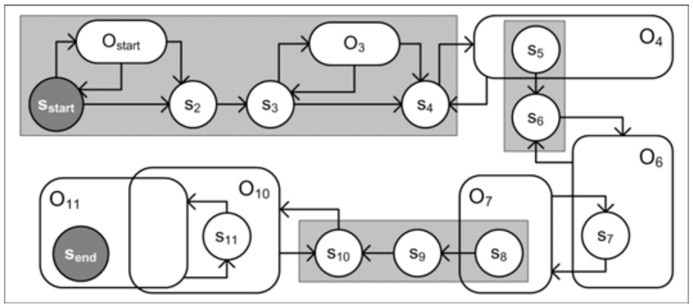

Figure 2. An example of a complete protocol

Following (10), states s_6 and s_{10} have null mandatory sets but their set of optional states (O_6 and O_{10}, respectively) contain states, respectively s_7 and s_{11}, which have nonempty optional sets. Following (11), the nonempty set of optional states of states s_6 and s_{10}, includes the set optional states, O_7 and O_{11}, respectively, which in turn include states, s_8 and s_{end}, respectively, which are part of the sequences.

Note that some optional states may never be visited by a user during the negotiation process. In the protocol depicted in Figure 2 only the 13 states that are identified have to be visited; other optional states may but need not be visited, e.g., no state in O_{start} and O_3 have to be visited, and only state s_5 in O_5 has to be visited.

3.3 Protocol modifications

The discussion of different protocol states and their relationships given in Sections 3.1 and 3.2 does not take into account the dynamics of the negotiation process. We formulated Conditions 1-4 and Theorem 1, which specify protocol properties and, in particular, introduce the concept of a complete protocol. This is not to say that there may not be situations in which the protocol designer may want to violate one or more of these properties. One such example is the condition 5 given in Theorem 1 stating that termination state s_{end} occurs only once in the protocol. For practical reasons this condition may be purposefully violated, so that the user has an option to terminate the negotiation at every state rather than be forced to follow the protocol until he reaches state s_{end}.

The distinction between mandatory and optional states partially takes into account context-dependency; it only allows to access different optional states before moving to a mandatory state. Protocol $\wp$ defined with (7) may be used in highly structured protocols, which follow the rules implemented in $\wp$. The rules governing the moves between states are static; they do not depend on the states visited. A stronger requirement reflecting protocol context-dependency is to allow for the states' rearrangement during the negotiation. In many negotiations, the permissible states depend on states previously visited.

To illustrate this form of context-dependency consider the example given in Table 1. At the beginning of the negotiation, the user is required to read the negotiation case *(NC*), construct utility function (*UC*), and assess alternatives displayed by the system (*AC*). While in state *UC*, the user may wish to read the case again. Hence we have the following two sequences.

$$NC \to UC \; opt \, 0, \; UC \to AC \; opt \, NC. \qquad (\mathbf{12})$$

If the user is in state *UC* and decides to read the case (select optional state *NC*), the part of the protocol given in (12) forces him to construct the utility function once again. This makes little sense; the user may wish to do this but state *UC* should not be mandatory anymore in the first sequence. This we can achieve through the following modification of (12):

$$S_{UC} \Rightarrow NC \rightarrow 0 \ opt \ UC. \qquad \textbf{(13)}$$

Modification (13) replaces mandatory state *UC* with null state and makes it an optional state for *NC*.

In order to represent this and similar forms of protocol context-dependency, we need to consider the protocol *instance*, which is the protocol that at any given time of the negotiation specifies the accessible mandatory and optional states.

Let $\wp_i$ be the instance of protocol $\wp$ in state s_i ($s_i \in S$) and

$$\forall \ s_i \in S \quad \wp_i = (S_i, O_i, M_i, P_i, Q_i); \ S_i \subseteq S, \ O_i \subseteq O, \ M_i \subseteq M. \qquad \textbf{(14)}$$

The states and sequences, which are elements of sets defining $\wp$ but which are not elements of $\wp_i$, are used in the modification of $\wp_i$. This modification may involve the introduction of new lists of sequences and breaking up existing lists in $\wp_i$. Similarly, a null state in a sequence may be replaced with a mandatory state. Therefore, in (14) sets P_i and Q_i are not subsets of, respectively, P and Q.

In $\wp_i$ a state may be both mandatory and optional. To illustrate this we use the example (12)-(13). The modification of the first sequence in (12), ($NC \rightarrow UC \ opt \ 0$) to the sequence given in (13) implies that we removed a mandatory state and added an optional state, i.e.:

$$S_{UC} \Rightarrow M_{UC} = M_{NC} \setminus \{\mathrm{UC}\} \ \wedge \ O_{UC} = O_{NC} \cap \{UC\} \qquad \textbf{(15)}$$

That is, in $\wp_i$ some states may be both *potentially* optional and mandatory, that is, $M \cap O \neq \emptyset$. However, $M_i \cap O_i = \emptyset$.

The dynamics of the negotiation process are reflected in context-dependent modifications of the negotiation protocol at run-time, which means that some mandatory states may became optional, some optional states may be added and others removed depending on the user and system actions. We allow for context-dependent system modification by introducing the following four operators:

1. *rmv-ma* removes the mandatory state $s_j \in M^R$ from state $s_i \in S$, when state $s_k \in S$ is visited; where M^R ($M^R \subset M$) is the set of mandatory states that can be removed, when the user is in state s_k:

   $$\textit{rmv-ma}(k,i,j){:} \ \forall \ s_k \in S : s_i \rightarrow s_j \ opt \ O_i, s_i \in S, s_j \in M^R, k \neq i, k \neq \mathrm{j}, i \neq j \Rightarrow s_i \rightarrow 0 \ opt \ O_i.$$

2. *add-op* adds an optional state $s_j \in O^{ji}$ to the set O_i, when state $s_k \in S$ is visited; O^{ji} ($O^{ji} \subset S \setminus O_i$) is the set of optional states that can be added to O_i, when the user is in state s_k.

   $$\textit{add-op}(k,i,j){:} \ \forall \ s_k \in S : s_i \rightarrow s_l \ opt \ O_i, \ O^{ji} \neq \emptyset, \ s_j \in O^{ji} \Rightarrow s_i \rightarrow s_l \ opt \ O_i \cup \{s_j\}.$$

3. *rmv-op* removes an optional state, $s_j \in O_i$, from the set of optional states, O_i, when state $s_k \in S$ is visited:

   $$\textit{rmv-op}(k,i,j){:} \ \forall \ s_k \in S : s_i \rightarrow s_l \ opt \ O_i \wedge O_i \neq \emptyset, \ s_j \in O_i \Rightarrow s_i \rightarrow s_l \ opt \ O_i \setminus \{s_j\}.$$

4. *add-ma* adds the state $s_j \in S$ as mandatory state to state $s_i \in S$, when state $s_k \in S$ is visited:

$$\textit{add-ma}(k,i,j)\text{: } \forall\ s_k \in S : s_i \rightarrow 0\ \textit{opt}\ O_i,\ s_j \in S,\ s_k \neq s_i,\ s_k \neq s_j,\ s_i \neq s_j \Rightarrow s_i \rightarrow s_j\ \textit{opt}\ O_i.$$

Using the above operators it is possible to modify different protocols, for example, after the user visited state *UC*, the sequence $NC \rightarrow UC$ is replaced with $NC \rightarrow 0\ \textit{opt}\ UC$. This modification involves two operators: *rmv-ma*(*UC*, *NC*, *UC*) and *add-op*(*UC*, *NC*, *UC*). Note that the operator's parameters (*k*,*i*,*j*) can be read as (when, where, what), e.g., *rmv-ma*(*UC*, *NC*, *UC*) means when the user is in state *UC* remove in state *NC* the mandatory state *UC*.

3.4 Intervening states

The characteristic that distinguishes negotiations from individual decision making is exchange of information between the negotiators (e.g., offers, messages and offer acceptance). Information sent by one negotiator affects his counterpart's activities. Therefore, the system has to display this information at the earliest possible time and irrespectively of the state he wants to visit. The states which contain and process information sent by the counterpart are the *intervening states*.

In face-to-face negotiations, the negotiators may exchange messages even during the planning phase. The synchronous aspect of these negotiations causes that they rarely formulate offers before both sides are ready to negotiate.

Asynchronicity of negotiations conducted via an ENS causes that one party may learn about the problem and be ready to exchange offers and messages, while the other party does not yet know the negotiation problem. This may require that the user be moved from some states to an intervening state, but not from other states. For example, if the user is in state *NC* and his counterpart makes an offer, this offer should not be displayed to the user prior to his specification of preferences in state *UC*. We therefore associate with every intervening state a set of *permissible states*; these are the states from which the user can be moved to the intervening states.

The move to an intervening state is defined with:

$$\forall\ s_i \rightarrow s_j\ \textit{opt}\ O_i,\ s_l^*,\ s_l \in V,\ s_i \in V_l\ \Rightarrow \bullet \rightarrow s_l^* \qquad \textbf{(16)}$$

where: s_l^* means that s_l is active (e.g., the counterpart sent message); V ($V \subset S$) is the set of intervening states; V_l ($V_l \subset S$) is the permissible set for state s_l; and $\bullet \rightarrow s_l^*$ means that the user selection of any state from the sequence ($s_i \rightarrow s_j\ \textit{opt}\ O_i$) moves him to s_l^* rather than to the selected state.

After an intervening state s_l was activated (e.g., the counterpart made an offer) and the user was moved to this state, this state becomes inactive. Once the intervening state is activated for the first time it becomes optional for all its permissible states, i.e.,

$$\bullet \rightarrow s_l^*\ \Rightarrow \forall\ s_i \rightarrow s_j\ \textit{opt}\ O_i \cup \{s_l\},\ \ s_l \in V,\ s_i \in V_l \qquad \textbf{(17)}$$

Formula (17) allows users to review information sent earlier by their counterparts.

4. Design and implementation issues

ENSs, like other web-based systems, interact with their users through pages. A page provides information to the user (e.g., about the negotiation problem or an offer) and/or may require that the users enter information (e.g., determine problem issues or construct an offer). This requires the separation of every page into its components. A component is the basic building block of Invite; it comprises text, graph, form, header, links or footer.

Components are invoked by page composer programs. These programs set the page layout and position the input/output interface of the components on the page. Components are reusable; the same set of components can be invoked from different page composers. This is especially useful, if the page layout itself is the subject of experimental research (e.g., to study the positioning of components on a page on the negotiators' efficiency), but also for the internationalization of negotiations (e.g. in different languages and numeric systems).

Activities are concrete actions undertaken by the negotiator, while states are actions undertaken by the user or by the system. If an action for a given state is completed, the system transitions to another state (mandatory, optional or internal to the system). The state of the execution of an instance of a negotiation protocol corresponds to a page composer. From its current state, a negotiation protocol instance provides transitions to a set of next states, which are invoked through hyperlinks and/or buttons embedded in the page.

An example illustrating the relationship between user activities, page composers and components is given in Figure 2. Component 1 produces output C1 and Component 2 requires the user's input; both correspond to Activity 1. A component may use data stored in databases and one or more external applications, e.g., for creating a graphic or calculating utility values. Not every component corresponds to an activity; Component 3 produces an output, which is not associated with any activity. Examples of such components are headers, footers or the information about the status and context of the negotiation.

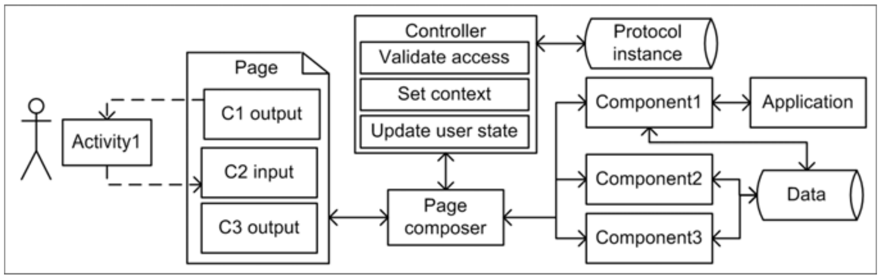

Figure 3. Activities, components, page composers, and pages

The component that is critical for the protocol instance implementation is the context-setting component; its output is present on every page as a set of links associated with the mandatory and optional states. This component provides the context-dependent association between the negotiation activities and the negotiation protocol.

This conceptualization of the negotiation framework reflects the model-view-controller (MVC) or mediator design pattern [19], where a page composer corresponds to a view and a component reflects a model. Following the MVC paradigm, the controller governs the sequencing of page composers.

However, the conventional MVC pattern does not contain provisions for executing negotiation protocols as described in the previous paragraphs. Several extensions to the MVC pattern have been proposed in order to account for the application flow required for negotiation protocols, e.g. numerous workflow patterns [20] and the hierarchical MVC pattern [21]. It remains, however, unclear how to deploy those design patterns to web environments [22]. It is therefore necessary to design a "negotiation" controller, which executes instances of negotiation protocols.

The negotiation controller runs a protocol instance assigned to users' negotiations. The context, in which each instance of a negotiation runs, is the state in which the negotiators are at a given point in time and is defined by the activities undertaken so far. It selects and activates the appropriate page composer program according to a protocol instance. It is important to note that the negotiation controller has to be independent of the sets of components, page composers and rules. This enables to introduce into the system additional components, page composers and rules without changing the implementation of the negotiation controller. In effect this separation of the controller from other elements of the systems provides essential flexibility; the negotiation controller executes a protocol instance, without being "aware" of any internals specific to a component, page composer or rule. The available sets of components, page composers and the rules constrain the design of negotiation protocols and the execution of a protocol instance, but do not modify the controller itself. As a result, the negotiation controller is capable of executing different protocol instances of different protocols at the same time.

5. Summary and future work

A negotiation protocol is a generic specification of a sequence of page composers. Its instance is derived by assigning a protocol to a specific user, who participates in a specific negotiation. This approach allows for the construction of different ENSs based on the components and page composers implemented in a software platform. In this paper, we lay out theoretical foundations for negotiation protocols based on a negotiation methodology derived from behavioural research. The foundations facilitate the design and implementation of negotiation protocols and allow for the construction of ENSs based on these protocols. A negotiation controller allows the software platform to execute different negotiation protocols in different ENSs and thus simplify the use of e-negotiations in real-world and in research environments.

We are currently designing and implementing a software platform Invite, and are predominantly concerned with the design and implementation of a negotiation controller. Earlier research has shown that there are several ways to design and implement a negotiation controller [6], e.g. by a rule-based expert system, by logic programming or by a database. For the purpose of Invite, we have chosen the database approach in order to avoid the additional complexity of a rule engine. More importantly, a database approach allows for building a controller that is simple and, as an inference engine, independent of the database content.

The construction of negotiation protocols may be a highly complex task. The theory in Section 3 will allow for the design of a software tool that supports protocol designer and automatically verifies, if the particular protocol meets the formulated conditions. Our future work includes the implementation of several bilateral negotiation protocols, the components and page composers which are used by these protocols, and a respective protocol designer support tool. We also plan to extend the Invite platform to multi-bilateral and multilateral e-negotiations.

References

1. Ehtamo, H., R.P. Hämäläinen, and V. Koskinen (2004). An e-learning module on negotiation analysis. *Hawai'i International Conference on System Sciences*. Hawai'i: IEEE Computer Society Press.
2. Schoop, M. and C. Quix, (2001). DOC.COM: A Framework for Effective Negotiation Support in Electronic Marketplaces. *Computer Networks*, **37**(2): 153-170.
3. Lempereur, A., (2004). Updating Negotiation Teaching Through The Use of Multimedia Tolls. *International Negotiations Journal*, **9**(1): (to appear).
4. Moore, D., et al., (1999). Long and Short Routes to Success in Electronically Mediated Negotiations: Group Affiliations and Good Vibrations. *Organizational Behavior and Human Decision Processes*, **77**(1): 22-43.
5. Teich, J.E., et al., (2001). Designing Electronic Auctions: An Internet-Based Hybrid Procedure Combining Aspects of Negotiations and Auctions. *Electronic Commerce Research*, **1**(1): 301-314.
6. Benyoucef, M., et al. (2000). Towards a Generic E-Negotiation Platform. *Sixth International Conference on Technologies for Information Systems*. Zurich, Switzerland.
7. Shim, J. and N. Hsiao (1999). A Literature Review on Web-Based Negotiation Support System. Documentation for Web-based Negotiation Training System (WNTS): cpol.albany.edu/wnts/WNSS_Literature_Review.pdf. Accessed: April 23, 2002.
8. Neumann, D., et al., (2003). Applying the Montreal taxonomy to State of the Art E-Negotiation Systems. *Group Decision and Negotiation*, **12**(4): 287-310.
9. Ströbel, M. (2003). *Engineering Electronic Negotiations*, New York: Kluwer.
10. InterNeg (1997). INSS. http://interneg.org/interneg/tools/inss. Accessed: March 4, 2004.
11. Ströbel, M., (2001). Design of Roles and Protocols for Electronic Negotiations. *Electronic Commerce Research Journal*, **1**(3): 335-353.
12. Starke, K. and A. Rangaswamy (1999). *Computer-Mediated Negotiations: Review and Research Opportunities*, eBusiness Research Centre: University Park. 37.
13. Bichler, M., G. Kersten, and S. Strecker, (2003). Towards the Structured Design of Electronic Negotiation Media. *Group Decision and Negotiation*, **12**(4): 311-335.
14. Kersten, G.E., S. Strecker, and K.P. Law (2004). *A Software Platform for Multiprotocol E-Negotiations*, InterNeg Working Papers INR04/04, InterNeg: Ottawa. http://interneg.org/interneg/research/papers/.
15. Lewicki, R.J., D.M. Saunders, and J.W. Minton (1999). *Negotiation*. Boston, MA: McGraw-Hill.
16. Kim, J. and A. Segev (2003). A Framework for Dynamic eBusiness Negotiation Processes. *IEEE Conference on E-Commerce*: http://groups.haas.berkeley.edu/citm/.
17. Kersten, G.E. and G. Lo, (2003). Aspire: Integration of Negotiation Support System and Software Agents for E-Business Negotiation. *International Journal of Internet and Enterprise Management (IJIEM)*, **1**(3): 293-315.
18. Gulliver, P.H. (1979). *Disputes and Negotiations: A Cross-Cultural Perspective*, Orlando, FL: Academic Press.
19. Gamma, E., et al. (1995). *Design Patterns: Elements of reusable Object-Oriented Software*, New York: Addison-Wesley.
20. Aalst, W.v.d. (2003). Patterns. http://tmitwww.tm.tue.nl/research/patterns. Accessed: February, 16, 2004.
21. Cai, J., R. Kapila, and G. Pal, (2000). HMVC: The layered pattern for developing strong client tiers. *Java World*.
22. (2003). Barracuda - Framework Comparisons. http://barracudamvc.org/Barracuda/docs/barracuda_vs_struts.html. Accessed: February 16, 2004.