Invite: A Multi-protocol Negotiation Platform

Ka Pong Law

A Thesis

In

The Department

Of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

September 2005

Your file Votre référence
ISBN: 0-494-10290-X
Our file Notre référence
ISBN: 0-494-10290-X

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: Ka Pong Law

Entitled: Invite: A Multi-Protocol Negotiation Platform

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____Chair

_____Examiner

_____Examiner

_____Co-supervisor

_____Co-supervisor

Approved_____
Chair of Department or Graduate Program Director

_____ 20____ _____

Dr. Nabil Esmail, Dean
Faculty of Engineering and Computer Science

Abstract

Invite: A Multi-Protocol Negotiation Platform

Ka Pong Law

Existing electronic negotiation systems implement a single, fixed negotiation protocol. Therefore, their use is restricted to these types of negotiation problems and these kinds of interactions which had been assumed a priori by designers. Recent research focuses on software which can be configured to suit specific objectives of their designers and/or to match particular styles and approaches of the negotiators. Such software also allows studying the impact of different features of negotiation systems on the negotiation process and outcome.

This thesis explores the design and implementation of Invite, a multi-protocol negotiation platform. The platform is a flexible and customizable e-negotiation software platform enabling negotiators to map negotiation activities to components and construct their own protocols by creating sequence of layout programs. It allows for multiple different negotiation protocols being run simultaneously under one system. A negotiation protocol is used to configure Invite resources and construct a negotiation engine. The Invite negotiation platform, two protocols, and two negotiation engines are presented in this document.

Acknowledgments

I would like to thank those people gave input and support for the completion of this thesis. I am indebted to my thesis supervisors, Dr. Kersten and Dr. Shiri for their valuable guidance, suggestion, input during the entire process. Their motivation brought this thesis alive.

I thank the Social Science and Humanities Research Council, Natural Sciences and Engineering Research Council, for their financial support for this research.

I would also like to express my gratitude to the entire e-negotiation group, especially to Stefan, JinBaek, Norma, and Eva, for their valuable input.

Finally, I would like to thank my family, especially my parents, for their support. Without their assistance and persistence, I wouldn't have the opportunity to study in Canada.

Table of Contents

List of Figures

List of Tables

1. Introduction

The evolution of using software to support negotiation processes began in late 1970s [KM78]. In early eighties, researchers expanded decision support systems (DSS) to aid negotiators. This led to the development of negotiation support systems (NSS) [KMWZ86, JJS87, JF89].

NSSs are based on the modelling approaches coming from decision sciences and, since the mid nineties, also from negotiation analysis. Other approaches used in the design of NSSs come from computer science and artificial intelligence[JF89]. Recently, Internet and computing technologies provided new opportunities for the design and development of software capable of supporting negotiators in an electronic format. Internet allows people to negotiate virtually from any place and at any time while browsers provide uniform and easy to use interface.

Early NSSs used a stand-alone system in a local network. With the advance of information and communication technology (ICT), recently developed NSSs support various communication methods such as email, chat, and video-conferencing. The term *e-negotiation systems* (ENSs) has also been used to describe software that employs Internet technologies; ENSs are deployed on the web, and one capable of supporting, aiding or replacing one or more negotiators, mediators, or facilitators .

Companies increasingly use internet in conducting business negotiations [RRM03]. With the increase of the use of Internet, ENSs would replace some forms of face-to-face

negotiations in a near future. An example of such systems is eBay, where the marketplace occurs on the Internet.

This thesis is organized into nine chapters. Following this introductory chapter, Chapter 2 provides an overview of E-negotiation, electronic negotiation system (ENS) and a review of some existing ENSs. Chapter 3 presents the objectives of this thesis. Chapter 4 provides a foundation for building ENSs, and chapter 5 presents statechart as a formal representation of this foundation. Chapter 6 describes the software engineering methodology and database model used in the project. Chapter 7 presents the design and implementation of the *Invite* platform and two complementary ENSs. Chapter 8 illustrates execution of an ENS running under *Invite*. Chapter 9 includes conclusion and suggestions for future work.

Electronic negotiations are becoming an important research subject not only in the area of electronic commerce but also in area such as cross-culture and dispute and resolution. Researchers have built several electronic negotiation systems such as Inspire, SimpleNS, WebNS, Negoisst and others. Each of them has different purpose of research study. However, some software components they use are very similar. For example, existing electronic negotiation system uses communication tools whether is text-based, real-time chat or voice and video technology. Existing electronic negotiation systems are case- and domain-dependent and limited to one single protocol. Every time researchers expand their research studies, they build new software or redesign an existing one. If all electronic negotiation systems are similar in some extend, why we don't build a platform that has a compilation of most of electronic negotiation system features. The approach is

to build negotiation platform software which is able to host many *electronic negotiation systems.* Whenever negotiation software needs to be built, we just pick and choose components and assemble them like playing *Lego*.

2. E-negotiation and ENSs

Negotiation support systems emerged from the field of decision sciences in an attempt to construct systems that are capable to aid negotiators [KMWZ86]. With the developments of negotiation analysis and ICTs, negotiation support systems gave rise to e-negotiation systems. Kersten, Strecker and Law [KSL204] give the following definition of ENSs:

> "E-negotiation system (ENS) defines software that employs Internet technologies, is deployed on the web, and capable of supporting, aiding or replacing one or more negotiators, mediators or facilitators".

ENS can be classified into two categories: (1) Preparation and evaluation systems and (2) Process support systems [MSKM87]. A system of the first category includes a pre-negotiation phase whose purpose is to study and organize the negotiation problem, determine negotiation issues and assign preferences to each issue. A process support system on the other hand helps users to communicate and exchange offers. Its purpose is to maximize the outcome by providing electronic bargaining tables and tools for assessment of the compromise efficiency [MSKM87].

2.1 Classification of e-negotiations

Several attributes can be used to describe negotiations. Attributes that are relevant to the construction of ENSs have been selected for the purpose of this thesis.

Distributive or integrative negotiations.

In a distributive negotiation, the gain of one party is the loss of the other party. In other words, one party's objective is to minimize the issue value while the other party's objective is to maximize this value. Distributive negotiation is often considered as a single issue negotiation where the negotiation resembles a division of a fixed-size "pie": the bigger the portion of the pie one party gets, the smaller the portion for the other party can get. For these reasons, distributive negotiations are also called win-lose negotiation or, using the game theory concept, a zero-sum game [RWBW89].

Integrative negotiations usually describe processes that involve multiple issues with the possibility of one party not being interested in what the other party would like to get. These processes are considered non-zero-sum games because not all negotiation parties have mutually exclusive goals. The outcome of an integrative negotiation usually is a win-win result [EHK04]. Integrative negotiations demand exploration of negotiators' interests and requirements with the purpose of expanding the "pie" [KNT00].

Participants

The number of participants is a factor of choice of the negotiation protocol. We classify negotiations as bilateral, multi-bilateral, and multilateral. Figure 1 shows a bilateral negotiation. It refers to two parties negotiating with each other. Multi-bilateral refers to multiple bilateral negotiations in which one party negotiates with two or more parties over the same problem (set of issues). Figure 2 illustrates a multi-bilateral negotiation. An example of this kind of negotiation is a car dealership where the seller negotiates with

several potential buyers. Another example is EBay where one seller negotiates with multiple buyers in the auction. Figure 3 shows a multilateral negotiation which every party negotiates with everybody at the same negotiation. Most existing ENSs are bilateral. Auction systems are usually multi-bilateral.

Figure 1. Bilateral negotiation

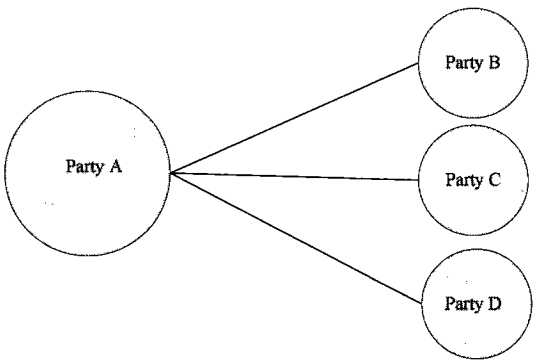

Figure 2. Multi-bilateral negotiation

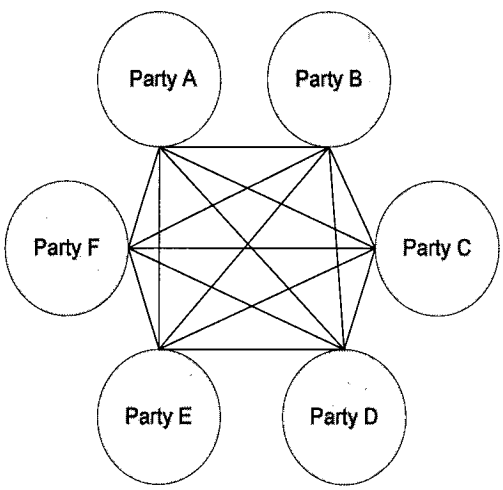

Figure 3. Multilateral negotiation

Negotiation Protocol

A negotiation protocol determines which *activities* are permissible and specifies the order in which they can be performed. Execution of the permissible *activities* moves the parties from one *ENS state* of the negotiation process to another. An a*ctivity* is a task for a negotiator and an *ENS state* is the group of *activities*. *Activity* and *state* are defined in Chapter 4. A negotiation protocol may include syntax and semantics as well as the timing and sequence of offers . Protocols can be fixed prior to the negotiation with no changes allowed, or they may be flexible so that activities which were not considered a priori can be undertaken during the negotiation [BKS03]. Thus, fixed protocols do not allow the negotiators to change the flow of the sequence of *activities*. Flexible protocols allow negotiators change the sequences, that is, remove some *activities* and/or add new one.

Issues

This attribute refers to the number and flexibility of the subjects of the negotiations. Negotiation issues are attributes of the goods or services and terms of conditions of the potential agreement (i.e. price, delivery date, date of payment). Single issue negotiations are usually used in auction systems such as EBay where the only issue is price. However, negotiation systems are often multi-issue, in that the parties are interested in more than one attributes of the good and/or service. Flexible protocols allow users to modify the number of issues by adding or removing them. A fixed protocol does not allow users to modify the set of issues during the negotiation.

Technology

Information technology has created new opportunities for e-negotiation. Email-based systems use email as the media for communication. Web-based systems employ a web browser or client-side software which communicates with a negotiation server through the Internet or intranet.

2.2 E-negotiation process model

A negotiation process model describes a structured set of phases. Each phase is broken down into activities. The phases are not always in sequence. They can be omitted, combined, or revisited during a negotiation.

In the negotiation literature, several process models have been suggested; the three-phase process model which consists of pre-negotiation, conduct of negotiation, and post-settlement [MPKT99]; an eight-phase model proposed by Gulliver [G79]. Gulliver decomposed the negotiation process into a search for arena, agenda definition, exploring field, narrowing differences, preliminaries to final bargaining, final bargaining, ritualization of outcome, and execution of outcome.

In order to develop software capable of supporting and aiding negotiators, we need to specify the activities that are undertaken by the negotiators and those performed by the software. In negotiations, as in other processes that require interactions among people, the *activities* are undertaken in certain order. Therefore, we need to take into account this order when we design software. This would also allow us to include prescriptions coming

from behavioural studies regarding the *activities* that negotiators should undertake and also regarding the ordering of the *activities*.

In this thesis, we use a five-phase model proposed by Kersten [KSL204]. The five-phase model is based on the Gulliver's model and it was adapted to model e-negotiations. These five phases are (1) Planning, (2) Agenda setting and exploring the field, (3) Exchanging offers and arguments, (4) Reaching agreement, and (5) Concluding the negotiation and post negotiation. Figure 4 shows the phases of this process model.

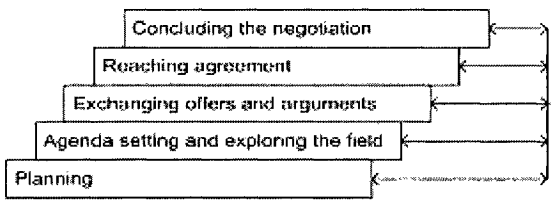

Figure 4. The five-phase negotiation process model [KSL204]

In the following section, we explain the negotiation process through an example of selling a used car. John would like to sell his red Toyota Corolla 2002 and Mary would like to buy John's car.

2.2.1 Planning

Planning is the phase prior to any interaction between negotiation parties. It involves negotiation participants and their thoughtful considerations of the problems. The parties need to consider what they want and what they can achieve. They also need to assess their strengths and weaknesses.

Cost, time, and potential possible outcome are variables which the negotiator takes into account before engaging into negotiation [KSL04]. A party decides to enter a negotiation if it is possible to reach an agreement which is better than the current situation.

When parties decide to negotiate, each first identifies the negotiation issues to be discussed and their associated ranges of values. They also identify the issues that cannot be negotiated or issues that are interdependent.

The negotiators assess their strengths and weaknesses, possible negotiation styles and negotiation strategies that might influence a desired outcome. They also identify their issue preferences; which issues are more important than others, and which ones could be traded off for a more favourable offer. The negotiators should also determine their Best Alternative To the potential Negotiated Agreement (BATNA). BATNA is a break even point from which a negotiator references whether to accept a negotiation settlement or reject it. BATNA is an intricate part of the planning process, in that if a negotiator sets a high value to BATNA then it is unlikely that she would engage in a negotiation [B04].

The negotiators try to get as much information as possible about their counterpart. This information includes culture, values, custom, norms, practice, and an estimation of the counterpart's reservation value and aspirations levels. Reservation value is the amount at which a negotiator is indifferent between reaching a deal and walking away; it is the worst possible deal the negotiator is willing to accept. Aspiration level is an agreement that can reasonably be achieved and one that satisfies the negotiator [K98]. The more a party knows about her counterpart, including the counterpart's aspiration and reservation

levels, the more advantage she has during the negotiation.

In our car example, both John and Mary might want to engage in a negotiation because they might expect to be better-off buying directly from the owner or selling directly to the buyer without an intermediary. The issues that could arise are the price, delivery charge, condition, color, and mileage. However, condition, color, and mileage are not negotiable between the two negotiators because neither of the negotiators has control over these attributes. Also, delivery charge could be included in the price. John knows that red is one of the favourite color among drivers but he also knows that his car has a factory defect. His BATNA is to stay with the car for another year and save up more money to buy a nicer car.

2.2.2 Agenda setting and exploring the field

Once the negotiators become aware of the negotiation problem, they decide on the time and place to negotiate, and the agenda. In e-negotiations, the selection of place is setting-up a marketplace. A disagreement might arise at this point. A negotiator might start practicing her negotiation tactics by showing her toughness by not agreeing to the time or place. They also need to decide on the communication mode which could be synchronous or asynchronous. Depending on the negotiator's personality, choosing the right communication mode could take a competitive advantage over her counterpart [G79].

During the planning phase, the negotiators should decide on their own set of issues. The selected issues may not agree with those selected by the counterpart. When a party introduces an issue that the counterpart is not willing to expose, the negotiation stalls

until the parties reach some sort of agreement regarding the issues they wish to negotiate about.

In the planning phase, the negotiation parties also estimate the strength of their opponent and determine a strategy and initial tactics to use. They may discuss the terminology and the specifics of issues to be negotiated. This preliminary discussion is critical for the negotiation because it is used to gather information about the counterpart. Upon this discussion, a party might change her initial strategy based on the attitude of the counterpart's attitude.

Following our car example, John might want to negotiate at his place whereas Mary might want to meet in a restaurant. If John works at night and Mary works during the day, then who would actually give up a day of work to satisfy the other party or they might decide to negotiate by email. If during their first meeting, John finds that Mary does not like red cars, he might want to soften his initial strategy and attempt to persuade her to accept the color of his car.

2.2.3 Exchanging offers and arguments

In this phase, the parties formulate and present their offers and arguments. Their proposals and also the questions they ask and clarifications they seek are for testing and adjusting their assumptions about the other party.

Negotiators gather more information about their counterpart each time they exchange an offer or argument. They try to find out their counterparts reservation levels and BATNA.

They make concessions to show that they are willing to make a progress in the process and get closer to a compromise. Thus, after several rounds of offer exchange, the negotiation -if successful- narrows to an agreement zone. Agreement zone is the set of all feasible alternative solution for the negotiation problem that are preferred over BATNA. The key point here for each negotiator is to try to minimize her counterpart's demand and to gain advantage over the agreement zone. It is also a good practice that an offer is followed by an argument to support why an offer is good [KSL04] (see Figure 5).

There are two types of offers exchanged during the negotiation and they correspond to closed negotiations and opened negotiations. In closed negotiations, the issues are fixed and no additional issues can be added. In an opened negotiation, on the other hand, parties might introduce new issues at any given point of time during the offer exchange.

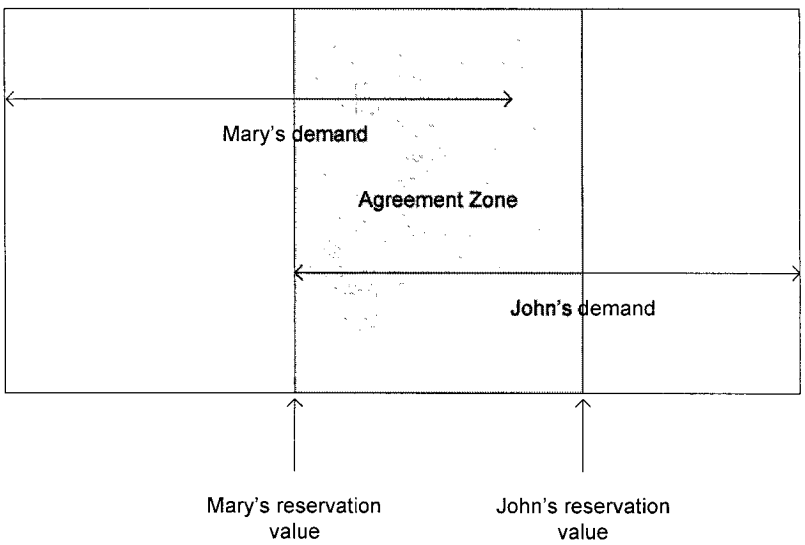

Figure 5. Agreement zone

In our example, John might demand $15,000 for his car knowing that his car is worth only $6,000

on the market. This offer allows Mary to determine John's maximum demand, i.e., it is $15,000. However, Mary would walk away if she knew the car is worth $6,000 because she might think that John is not serious about selling it. If she decides to continue, then she sends a counter-offer, e.g. $2,000, usually it is her maximum offer for the car. The offer exchanges, which are also known as negotiation rounds, may continue for some time. At some point, Mary offers John $6,000 to buy his car. John would agree if he realizes that Mary is not willing to pay more and his reservation value is $5,800. This is because Mary's offer is $200 more than his reservation level.

Reaching an agreement

Reaching an agreement means that the parties realize that the negotiation is 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.

Sometimes, a negotiation can no longer continue because of the parties' unwillingness to make concession. There are several ways to circumvent such deadlocks including coercive power to impose the other party to do what you want, try to compromise with your counterpart, slow down the conversation and give each other more time to think about reaching a solution [KSL04].

In our car example, if John refuses to negotiate with Mary because he does not like her tone in the negotiation, she might want to compromise and soften her tone in order for the negotiation to continue.

2.2.4 Concluding the negotiation

The negotiation concludes when the negotiators reach an agreement. The parties may evaluate the compromise they achieved and consider possible improvements. They may also discuss additional issues which have no impact on the negotiation. If what is achieved is inefficient, the parties may go back to re-negotiate until an efficient outcome is reached. An inefficient agreement can be identified using analytical tools to measure the efficiency for both parties [KSL04].

2.3 Review of ENSs and negotiation platforms

A number of ENSs have been designed, implemented, and applied to various negotiation problems. In this section, a brief review of some existing ENSs is provided.

2.3.1 *Inspire*

Inspire is a bilateral ENS developed by the InterNeg group based on decision and negotiation analysis [BKS03]. Its main purpose is to investigate cross-cultural negotiations and to provide a teaching tool in negotiation courses [K98]. *Inspire* views a negotiation as a process that occurs in a particular context. The system uses a simplified 3-stage process model: pre-negotiation analysis, conduct of negotiation, and post-settlement analysis.

The pre-negotiation phase comprises planning and agenda setting phases of the five phase model discussed in Section 2.2. It involves an analysis of the situation, problem and opponent, specification of preferences, reservation levels, and strategy. An important

activity undertaken in these phases is preference elicitation and construction of a rating function. The preference elicitation is a simplified utility which standardizes users' preferences over attributes into one unit of measurement. The construction of rating function is done in three steps. First, the user specifies her preferences over pre-defined issues by distributing 100 points among issues. Once the user assigns preference over issues, she proceeds to assign option[1] preferences. The next step of the pre-negotiation is generation of packages[2] followed by the calculation of ratings for these packages and by the user's verification of these ratings. If the user changes the displayed rating values the least-square procedure propagates these changes to all remaining ratings.

The negotiation phase involves exchange of messages and offers, evaluation of offers, and the assessment of the progress of the negotiation. In support of the offer exchange activity, the system presents a drop down menu for each issue, so that user can select for each issue only one option. In this way the user constructs a complete package and submits it as an offer. *Inspire* also provides a real time rating calculator. Once a package offer has been constructed, the rating is displayed based on the rating function, earlier constructed from user's preferences.

The post-settlement phase involves the evaluation of the negotiation outcomes generated by, and after, the negotiation activity [K98]. The system first determines the rating values of the achieved compromise for both users. Then it runs an optimalization program to

---

[1] Option is one of the alternative values that an issue can take. For example, the issue "Tolerable product failure rate" may have the options "3%", "5%", and "10%".

[2] Package is a particular combination of options that has been selected across all the issues.

find out whether the compromise is efficient (i.e., Pareto-optimal). If the compromise is inefficient, that is, there is a package that can make at least one party better-off without making another worse-off, the system searches for up to five efficient packages. These packages are displayed to the users who have the possibility to re-negotiate and improve inefficient compromises.

A negotiation is terminated when (1) an agreement has reached among both parties, (2) one party terminates in any phase during the negotiation, or (3) the period allocated for talk is expired.

2.3.2 WebNS

The *WebNS* system is a Java-based ENS developed at McMaster University in Hamilton [YDRA99]. It focuses purely on the conduct of negotiation through the facilitation of communication and it does not offer analytical negotiation support [YDRA99].

WebNS is based on a negotiation process model derived from Gulliver [KSL04].It divides the negotiation process into two main phases: preparation and offer exchange. Preparation is supported by tools such as the session description and private notes. The main support of *WebNS* is in providing a platform for users to engage in communication. The system uses real-time chat and video conferencing to exchange offers and counter-offers as well as short messages. The protocol underlying *WebNS* treats every issue separately and, hence, does not explicitly support discussion of tradeoffs among the issues [KNT00].

2.3.3 *Negoisst*

The *Negoisst* system is an ENS developed by the Electronic Negotiation Group at RWTH in Aachen [S02]. Its primary purpose is to enable complex dynamic negotiations between human negotiators, to guide them through the complex negotiation process, and to provide support rather than automation of negotiations.

Negoisst takes a document-oriented approach, as opposed to a communication-oriented approach, towards negotiation support and uses speech act theory to model communication between negotiators. The system architecture is a Java-based 3-tier client/server architecture based on the DOC.COM framework [SQ00]. The negotiation protocol used in *Negoisst* distinguishes between different types of messages, e.g., a question or clarification. The negotiation protocol of *Negoisst* is based on a deterministic finite automaton with symmetric states and constraints on some states [S02].

2.3.4 *SimpleNS*

As a part of this research project, this researcher has developed the *SimpleNS* system (http://mis.concordia.ca/SimpleNS). The system has been developed for teaching and comparative studies on the use and effectiveness of different ENSs. It provides a virtual negotiation table which allows users to exchange offers and messages.

SimpleNS system displays the negotiation case and other information required to conduct the negotiation, presents a form in which users write messages and offers, and shows the negotiation history in which all messages and offers are displayed in one table together

with the time they were made. *SimpleNS* is based on a three-stage process model similar to *Inspire*. However, it does not offer analytical support to the negotiators. *SimpleNS* has been used in teaching at the University of Ottawa, Concordia University, Vienna University, Austria and National Sun-yat Sen University, and Taiwan.

This brief overview illustrates the breadth of approaches towards ENSs design. Some systems are grounded in negotiation theory, some draw on one or more models from negotiation research, and some do not have any methodological foundation at all. However, most ENSs make very limited use of existing theories and are restricted to a single negotiation protocol. Hence, the user can neither adapt the ENS to her needs nor follows a consistent methodology.

2.4 ENS software platforms

Internet and intranet technologies have provided new opportunities for commercial and other transactions. These include individual, and group decision making, auctions, and negotiated decisions. Initially, systems supporting individual and group decisions and negotiations have been designed for stand-alone and local area networks. These are now ported to the Web. Emerging organizational models (e.g., used in e-commerce and e-marketplaces) and the Internet technologies gave rise to the design and development of new types of systems, including ENSs.

There are many types of negotiations, including single-issue and multi-issue, bilateral,

multi-bilateral and multilateral. Many simple, well structured and multi-bilateral negotiations that involve a single issue or several issues which are known from the outset, maybe replaced with on-line auctions. Multi-issue bilateral and multilateral negotiations during which issues may be added, modified, or removed require ENSs.

Every specific ENS has a negotiation protocol built in either implicitly or explicitly. The protocol guides the execution of the system. A negotiation protocol not only imposes rules on permissible input and output generated by the system but it also allows for communication between the system and its users. Currently, most of the existing ENSs implement only one negotiation protocol and hence they can be used for a specific type of negotiations. These systems are not adequate to support complex and evolving negotiation processes in which the change of the protocol may be necessary. For example, expansion of the process from two to several negotiators and heterogeneous process which starts as an auction and after the selection of few counterparts is transformed to a multi-bilateral negotiation.

Few systems allow multiple protocols. The SilkRoad platform is an example of a multi-protocol system, but it was designed to support only auctions [S03]. Similarly GNP is an auction-oriented system [BK00]. INSS is the only system that was designed to support several types of negotiations [K98]. This system, however, is not functional and its protocols are not explicitly formulated, hence making its extensibility difficult. Below, we give a brief description of the above mentioned platforms.

2.4.1 SilkRoad

SilkRoad is a platform which facilitates the design and implementation of ENSs [S03]. Its design methodology is based on reusable negotiation support components, e.g., for matching offers and mediating compromises. To set up a specific ENS, a negotiation designer uses *SilkRoad* to define a negotiation (case, issues and options), and to define a negotiation protocol. On the basis of a complete negotiation design, run-time specifications are generated. These specifications are then deployed to a server running the *SilkRoad* run-time environment. Through the activation of these run-time specifications, the server is now customised to support an electronic negotiation among agents in exactly the way specified by the negotiation designer.

2.4.2 GNP

Generic negotiation platform (*GNP*) is proposed by Benyoucef [BK00]. It supports running combined negotiation. It is designed to negotiate two or more items as a bundle. Negotiations that run on this system are based on auction type protocol in which workflow management system is used to deal with interdependency.

2.4.3 INSS

INSS was an attempt to improve *Inspire* [K98]. It is based on a "workbench" approach which provides a limited number of tools to construct an ENS. The tools that it offers are coming from decision and negotiation analysis, and experiences with the use of the *Inspire* system [K98]. These tools include, parallel negotiations, sequential negotiations,

unstructured messages, BATNA and reservation levels specification, new values for continuous and discrete issues, and new issues.

Parallel bilateral negotiation means that the two negotiators discuss about all issues at the same time. This type of negotiation requires that the user submits a complete package as an offer. A package is considered complete if it contains values for all negotiated issues. In contrast, in the sequential negotiation, a user may submit an incomplete package, that is, a value for one, two or more issues but not necessarily for all issues. In the sequential negotiation process the negotiators present one or more issues at a time. When they reach an agreement with the discussed issues, they proceed to deal with others. Unstructured message allows users to send messages such as argument along with his offer or counteroffer. *INSS* allows introduction of new issues at any point during the negotiation but prior to achievement of an agreement [K98].

INSS follows a three-phase negotiation process approach. Preparation, in which users study the negotiation problem and identify all issues and alternatives involved. The actual conduct of negotiation takes place when negotiators exchange offers and arguments. Once negotiators reach an agreement, the system analyses whether there is at least one package that can optimize the utility for both negotiation parties. If that is the case, the system allows negotiators to re-negotiate until they reach a better outcome [K98].

2.5 Summary

To summarise, because most ENSs deal only with specific negotiation types, i.e., single user interface and support tools that cannot be selected by their users, their applicability

to different negotiation types and different user requirements is severely limited. These systems are inflexible placing a major restriction on negotiation from both research and practical perspectives. In order to expand the experimental studies of ENSs and provide field studies with their potential users, we develop an ENS platform that is capable of dealing with different protocols and user interfaces. The primary goal of this thesis is:

the design and implementation of a multi-protocol e-negotiation systems.

To achieve this goal, we have:

- Conducted a thorough study of related work on foundations of e-negotiation development, integration, classification, and protocols, as well as methodologies for flexible systems design.
- Developed of a multi-platform support system, that includes:
    - protocols and protocol controller;
    - platform schema;
    - platform database; and
    - complementary interfaces

In other words, our goal is to build a software application for e-negotiation which is capable of hosting several types of negotiations, including auction or bargaining, provides different user-interfaces, allows for negotiating multi-issue problems, and supports single and multiple negotiators.

3. E-Negotiation Foundation

Any negotiation supported by an ENS requires that the software designers define precisely the activities and the sequence of these activities. For this, a negotiation protocol can be used. A protocol defines activities that are allowed in certain *ENS state* of a negotiation. An *ENS state* is a set *activity* that a negotiator undertakes at a particular moment during the use of an ENS. Protocols also state the flow of a negotiation and its input and output requirements.

3.1 Process model mapping

The framework provided by the process model, discussed in section 2.2, 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, depends on the current *ENS state* of negotiation, which 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 a process model, including its phases and activities is given in Table 1.

Table 1. Example of a process model, activities, and states

Negotiation phase and activity	ENS State
1. Planning	
- Negotiation problem	Description of negotiation case
- Preferences and rating	Utility construction
- Assessment of alternatives	Alternative construction
2. Exchanging offers and arguments	
- Offer and/or message construction	Offer message
- Counter-offer assessment	Counterpart's offer
3. Reaching agreement	
- Agreement	Agreement reached
	Agreement assessment
- Closing negotiation	End
4. Concluding negotiation	
- Agreement improvement	Agreement improvement
- Offer and/or message	Offer message
- Counter-offer assessment	Counterpart's offer
- Closing negotiation	End

Each negotiation *activity* is associated with an *ENS state* (see Table 1). However, the reverse is not true. For example, the system may be in a state that does not correspond to any negotiation activity. The difference between a negotiation *activity* and *ENS state* is that the former describes a user action, whereas the latter denotes a user and/or a system action.

3.2 Phases, ENS states, and activities

A negotiation can be interpreted as a sequence of *activities*. Given a particular negotiation

protocol, some *activities* are mandatory and some are optional. For instance, in a protocol used to conduct experiments reading about the negotiation problem is a mandatory *activity*. In another protocol, the parties may be required to exchange offers in a sequence, that is, after one party makes an offer, the second party has to either reply with a counteroffer or accept the offer. In this protocol, making a counter offer after the offer receipt and offer accept are two mandatory *activities* but displaying and reading earlier offers are optional *activities*. Sending a support message along with an offer would typically be an optional *activity* but it would be mandatory in protocols in which the users are required to justify their offers.

A *phase* is a set of tasks in a negotiation process. Following Section 2.2, we assume that the process comprises six *phases*. Each *phase* can be decomposed into a set of *ENS states*. An *ENS state* consists of the information available at a particular time and the *activities* that can be performed at this time. Reading an offer, and preparing and sending a counteroffer are examples of *activities*. Some *activities* can be combined and some are treated separately.

The output of an activity or a set of *activities* is a *construct*. An example of constructs includes a package that the user considers, an offer sent to the counterpart, and a message. The same constructs can be the output of the *activities* undertaken in different *phases*, for example, an offer made before an agreement and an offer made after the parties realized that the negotiated agreement is inefficient.

An *activity* can appear in more than one *phase*. For instance, reading a negotiation

problem can appear in the planning *phase* and also in exchanging offers and arguments *phase*. This provides the negotiator an option to go back and review an *activity*; he has done in the past. As stated earlier, an *ENS state* is a set of *activities*. A completion of an *ENS state* allows negotiator to move to a next state. If a *state* is the last step of a *phase*, the negotiator proceeds to the next *phase*. The sequence terminates when there is no further *state* or *sequence* ahead. Figure 6 illustrates the relationship among *phases*, *ENS states*, *activities*, and *constructs*.

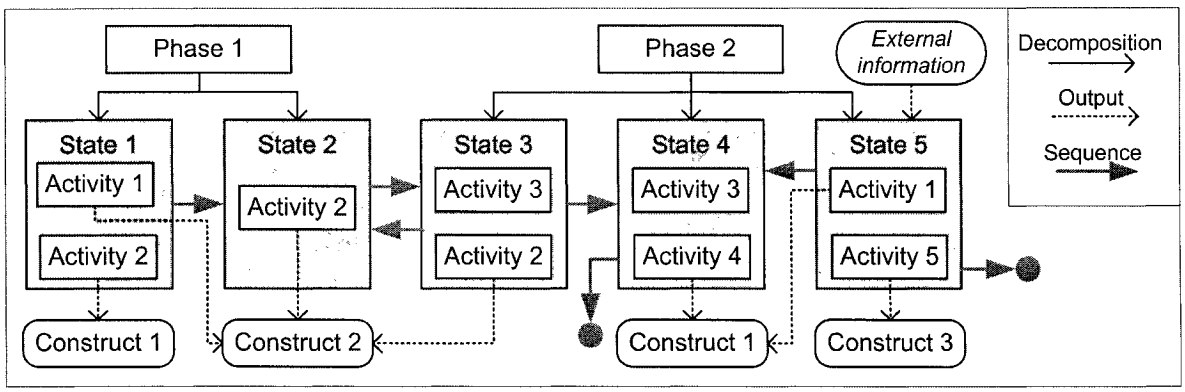

Figure 6. Phases, ENS states, activities, and constructs (KL05)

Some *ENS states* have to be undertaken and other may not. A *state* that appears in a negotiation for the first time is often mandatory and the next time it appears is often optional. This is because when the user undertakes one *state*, she should be able to access results of the past *states* in order to refresh her memory.

3.3 Sequence

During the negotiation, the set of *ENS states* changes; when new information becomes available at one *state*, it can be accessed at later states. This leads us to introduce

mandatory and *optional ENS states*. After all activities in the *mandatory ENS state* are performed the user may undertake other *activities*. That is, the *mandatory ENS state* directs the user to *another ENS state* which may be *mandatory* or *optional*.

We can distinguish the *ENS states* which are accessible at any given time and which the user can visit many times prior to completing all *activities* associated with the *mandatory ENS state*. This set together with the relationships between its *ENS states* is called a *sequence*. A *sequence* includes an *initial ENS state*, which is the entry point of a *sequence*, a *mandatory ENS state* which is the exit point to a *sequence*, and also, includes zero or more *optional ENS states*.

When a user enters a *sequence* she may move from its *initial ENS state* to any *optional* or *mandatory ENS state*. She can repeat the *activities* associated with the *optional*, *initial* and *mandatory ENS states* until she decides to exit to another *sequence*. To exit from one *sequence* and enter another, the user must first complete the *activities* of the *mandatory ENS state*.

When the user is in a given *sequence*, she has to be able to move to at least one other *sequence*. The *sequences* in which the user can move to are directly reachable with the exception that when a *mandatory ENS state* concludes the negotiation. This means that every *sequence* has to be connected to one or more *sequences* and all the *ENS states* inside a *sequence* are also "interconnected" to each other. Figure 7 describe the *sequence* properties.

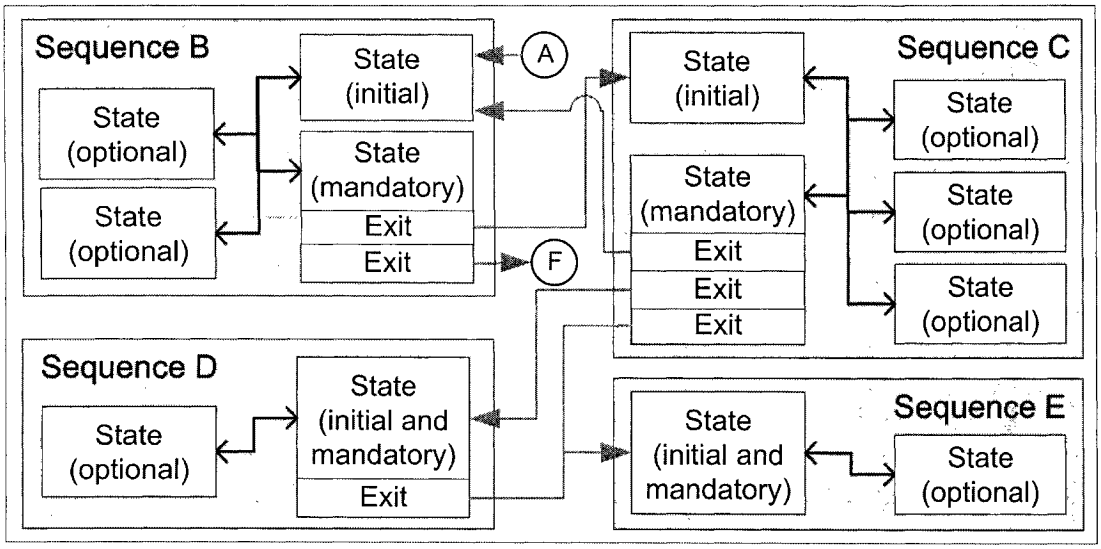

Figure 7. An example of sequences and states (KL05)

In this figure, the user first enters the *initial state* of *sequence* B, after completing *sequence* A as indicated with the circle. While in *sequence* B, she may move to any other optional or mandatory *ENS states*. If she chooses to move to another *ENS state*, she can always go back to the *initial ENS state*. When the user moves to the *mandatory ENS state*, there are two exit points, one of which directs the user to *sequence* C and the other to *sequence* F. If the user decides to exit B and proceed to C, she immediately arrives at the *initial ENS state* of *sequence* C. This process repeats until she reaches to the *mandatory ENS state* and chooses to exit C and enters D. There are *sequences,* such as D, in which the *initial* and *mandatory ENS states* are combined. When user enters *sequence* D, she may exit to another *state* immediately right after its completion. In *sequence* D, there is only one exit point, which leads to *sequence* E. User arrives at the *initial ENS state* of *sequence* E. In that *sequence*, there are only one *initial-mandatory* and one *optional ENS state*. The user may move back and forth between the two states. However,

the *mandatory ENS state* does not have an exit point. This means that the user has entered a "completion" *sequence*.

Note that a termination sequence is different from a completion *sequence* (*sequence* E). A *termination sequence* must exist in any negotiation protocol and any sequence can link to it. This is because at any given point of time, the negotiators should be allowed to terminate the negotiation in which they are currently involved.

Sequences may have hidden *initial* and *mandatory ENS states*. They can only be activated by triggering *intervening events*. The concept and purpose of *intervening event* are discussed in section 4.4.

3.4 Intervening Event

A negotiation protocol of one single negotiator can be constructed using sequences and states. However, negotiations always involve two or more negotiators. At one point during a negotiation, the negotiators' activities must be synchronized. For example, if negotiator A accepts negotiator B's last offer, A moves to the *Agreement* state by accepting the offer. However, B needs to be notified about A's acceptance. At the same time, B also needs to be moved to the *Agreement* state. However, this action is forced by the system because after A's acceptance, both negotiators should be synchronized to go to the *Agreement* state. In order for the ENS to move B to the *Agreement* state, the ENS needs an event to occur in order to identify and trigger the action. This event is called

intervening event. Intervening event defines as any events during an electronic negotiation that is not controlled by the user. This event is often activated by the user's counterpart of the negotiation. The construction of the theory behind intervening event was revised during the implementation of Invite. We found that the initial proposed theory [KSL04] could not be implemented.

Intervening event is activated by an event that is external to the system and is controlled by neither the user nor the system. If the user enters a sequence that has hidden *mandatory ENS state*, the only way to move on to another *sequence* is to wait for an *intervening event* to occur. The intervening event occurs when an activity is completed by an external entity. The external entity could be the negotiator's counterpart or an external system component such as negotiation software agent. When the system acknowledges the presence of external information, it immediately activates the *intervening event* for the negotiator. Activation of the *intervening even* causes one of the following three functions to take place:

1. One or more hidden *ENS states* become optional in the current *sequence* and subsequent *sequences*, which could be an *optional ENS state* of the current *sequence*.

2. A *mandatory ENS state* becomes visible in at least one of the *sequences* following the given *sequence*; or

3. An *ENS state* becomes an *initial* and/or *mandatory ENS state* for the current *sequence*.

To motivate and describe the functionalities of the *intervening event*, consider a negotiator who begins the negotiation later than his counterpart.

Functionality 1 is illustrated as follow: While the user reads the negotiation problem (planning phase), her counterpart sends a message. The system acknowledges the message and activates the "message display" *ENS state* in the negotiator's current *sequences* and subsequent *sequences*. This allows negotiator to access her counterpart's message and its follow up messages.

To illustrate the functionality 2, consider the following scenario. When the negotiator reads the negotiation problem, her counterpart sends an offer because in this particular protocol she is not allowed to read an offer without completing the phases prior the "Exchange offers and arguments" *ENS state*. Once the system receives the external information, it proceeds to inform the negotiator that an offer is waiting to be read. It also explains why she is not allowed to read the offer at this particular moment, by making a hidden *optional ENS state* visible ("offer received remainder") to every *sequence* prior the "exchange offers and arguments" *ENS state*.

To illustrate the functionality 3, we follow the scenario of functionality 2 example. Once the negotiator reaches to "exchange offers and arguments" *sequence*, the "offer display" *ENS state* becomes *initial* and *mandatory*. This forces the negotiator to read her counterpart's offer as soon as she is permitted to do so. The intervening event forces an *ENS state* to be *mandatory* because the negotiator has to either accept or reject the offer sent by her counterpart before she can continue in the negotiation. Accepting the offer

moves the negotiator to the agreement phase.

The intervening event provides flexibility to the system by allowing dynamic changes in the protocol instance at any point in time. However, it creates another level of complexity to implement such flexibility into the protocol. Note that an *intervening event* consists of a set of rules. The question is "how to store and use *intervening event* rules in an efficient manner?"

4. State Diagram

In the previous chapters, we reviewed research in negotiation process theory and process transformation. Some have adopted the Finite State Machine (FSM) approach to describe negotiation process [KS99]. The advantage of using statecharts to model negotiation process is that existing tools such as Statemate [HLNP90] are available for simulation and analysis to help validate and render the model. Figures 8 and 9 illustrate e-negotiation process in statechart.

The high-level protocol description given in Figure 7 is decomposed into two parts to describe the transformation to state diagram. A *sequence* can be viewed as a superstate. Each superstate contains a set of states. An *ENS state* is the equivalent to a state in the statechart. The transition from one state to another is either a user-action or an *intervening event*. In *ENSs*, which are web-based applications, user-action means the user clicks on a web-link. *Intevening event* transition is always triggered by the system (*ENS*).

Figure 8 illustrates the state diagram of a protocol without *intervening event*. When a user enters superstate B, she goes directly to state 1 which is the initial state of that superstate. From the state 1, users are allowed to navigate to any of the states inside the superstate. However, when she decides to exit that superstate, she must go to State 4, which is the equivalent to *mandatory ENS state*, causing the user exits from B and enter the superstate C. Some superstates have states that contain both *initial* and *mandatory*. Superstate D is an example of such superstate where a user can exit the superstate right after entering.

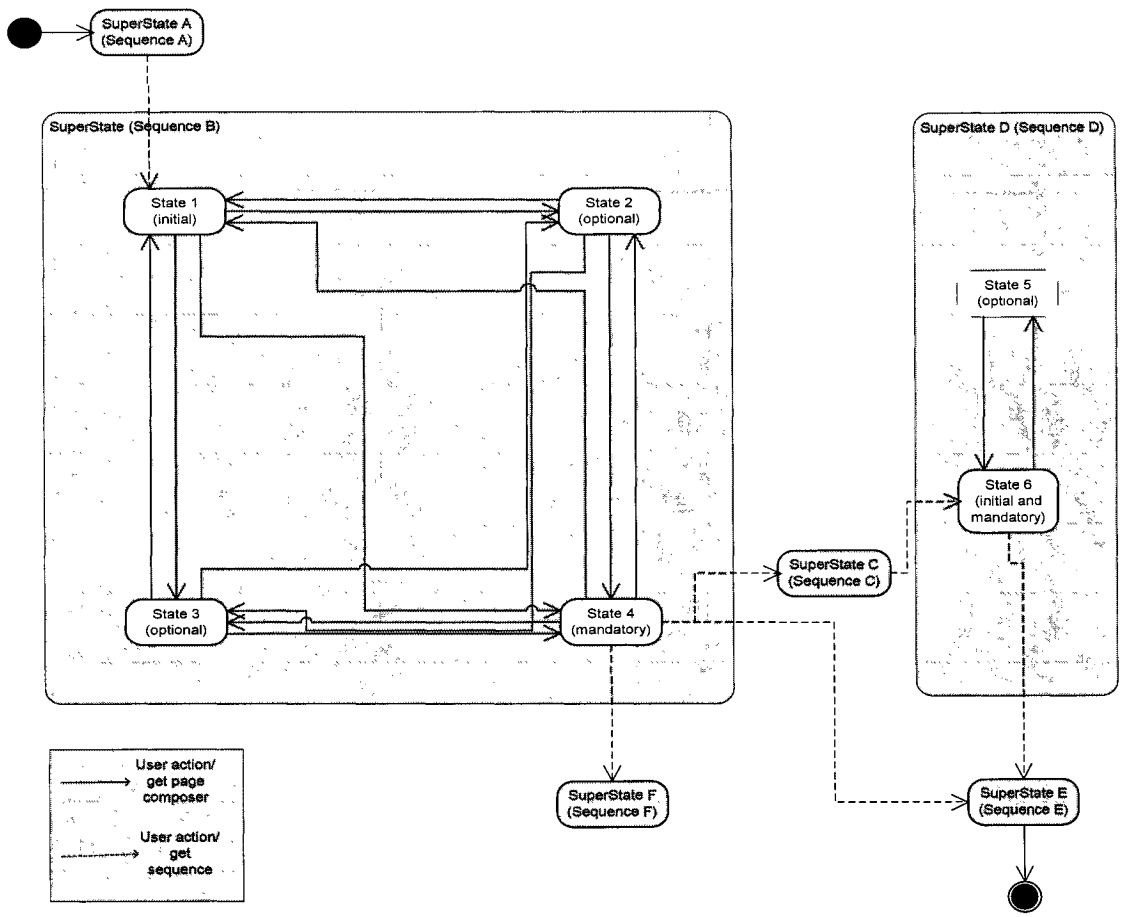

Figure 8. Protocol without intervening information

Figure 9 illustrates a protocol that includes an *intervening event.* It describes scenarios that could happen when an *intervening event* occurs. A user can exit superstate B by her own action or by her counterpart's action. The decision maker, shown by a diamond in the figure, in the exit point of superstate B decides which state in superstate C should be visible to the user based on the exit action. If it is triggered by the user, state 7 is visible. However, if it is triggered by the user's counterpart, then state 8 is visible to the user.

When a user is inside a superstate, she can be forwarded to another state by his counterpart action. Suppose that a user is in state 8. If her counterpart triggers an

intervening event that affects the user flow in the protocol, the user is automatically forwarded to state 5. From state 5, the user can either exit the current superstate into superstate D by going though state 10, or by her counterpart triggering *intervening event*. In another scenario, a user may exit from her current sequence to different sequences. Suppose now that the user is in superstate B. She can exit this state and move into superstate D. However, she can exit superstate B by the *intervening* action by her counterpart. The counterpart's action will trigger the *ENS* to forward the user to superstate E.

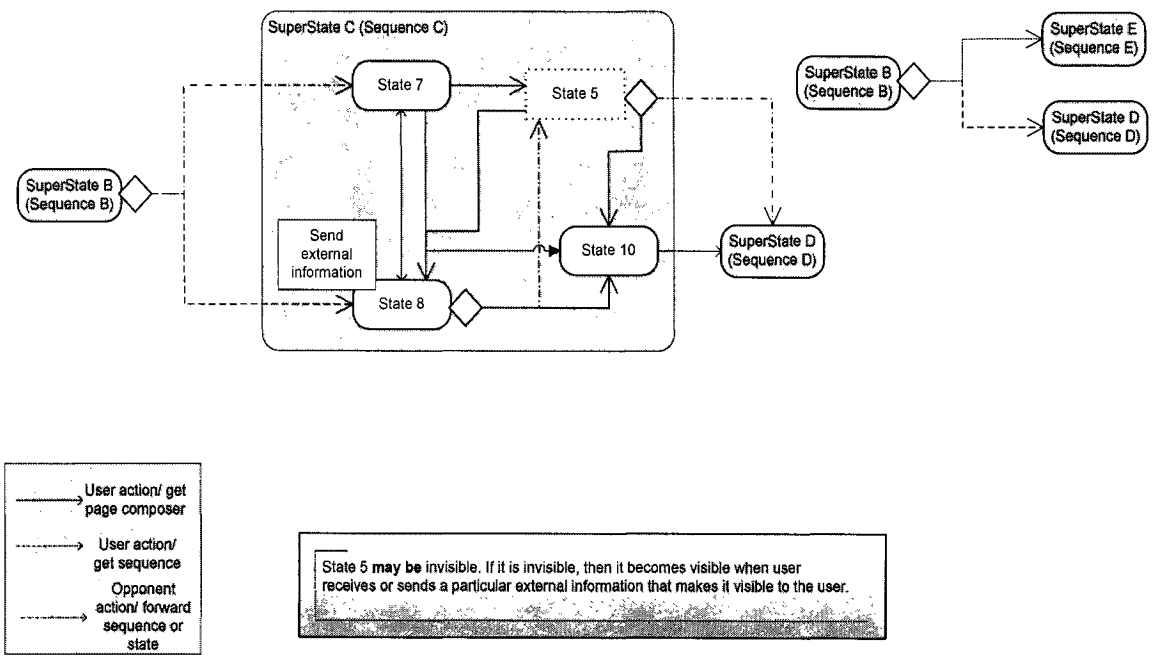

Figure 9. Protocol with intervening information

5. Software Engineering Models

5.1 System development life cycle

System development life cycle (SDLC) is a widely used software development methodology. SDLC is a process that consists of 4 phases [P01]. SDLC has several models to develop software systems. Some of these life cycle models includes: waterfall, rapid application development (RAD), joint application development, prototyping, incremental, synchronize-and-stabilize, and spiral models. In this thesis, the prototyping model is used. The phases in this model and their key purpose are as follows.

- Requirements specification and analysis identifies the needs of the end users. To identify the requirements, mock-ups and prototypes are used;
- Architectural design identifies the roadmap, resources, and plan for the system;
- Coding and debugging creates the system; and
- System testing evaluates the functionality and usability of the system.

Requirements analysis attempts to gather all the prerequisites and needs to build a system. Once the requirements are gathered, they are organized into groups. A prototype is built to clarify the requirements or design uncertainty. This organization helps analyzing the requirements and filters these which are unrelated, ambiguous and unnecessary for the system and its users. The requirement specification phase attempts to

describe the system to be constructed as complete as possible.

The architectural design phase identifies the resources (i.e. servers, databases, and interfaces), followed by the concrete designs of the structure of the system. The main characteristics of a good web design include simplicity, support, familiarity, clarity, encouragement, satisfaction, availability, safety, versatility, personalization, and affinity [K98]. The user interface should be simple and straightforward; users must have control by not restricting the sequence of the flow; system is build from past experience that users are familiar with; figure of icons should represent the feature of its use; users should feel comfortable in using the system; all objects should be available at all time; the system must be bug-free; and allow users to customize the interface [BS00].

The coding and debugging phase involves mapping the requirements and design into program codes and components. Also, the developers should have a clear understanding of the system based on the client's feedback of the prototype. Tools selection, coding guidelines, development environment are parts of this phase. The final product of this stage is a system that meets the clients' requirements.

Once the code has been created, the internal and external features of the system are tested. Internal testing makes sure that all the program statements function correctly and the external testing deals with the usability of the system for the user [B04].

5.2 ENS Requirements analysis

The analysis phase is a critical step in the development of a system. The main purpose of

analysis is to determine the need and the requirement of a multi-protocol e-negotiation platform that supports multiple ENSs.

Based on the observation in previous experience building ENSs (i.e. *SimpleNS*, *Inspire*), the major requirements of a multi-platform ENSs are as follows:

- Provide a wide variety of protocols
- Allow easy construction of user-define negotiation protocol
- Support managing activities undertaken by negotiators
- Support re-usable negotiation components
- Support easy customization of user-interface

A way to identify the main challenge is the architectural design. It addresses the issue of discovering functionality of the system and the resources used. Compared to existing ENSs, *Invite* delivers a wide range of functionality to improving usability and performance. Invite introduces the concept of customizable, user-defined negotiation protocols. It not only implements single, fixed negotiation protocol, but a set of pre-defined components using which a negotiator can compose a desired negotiation protocol, in addition to a set of pre-defined negotiation protocols already defined in *Invite*.

Based on the requirement analysis, *Invite* is designed in two parts;

1. *Invite* platform which controls negotiation protocols and ENSs; and

2. *Invite* ENSs which contains different run-time ENS engines.

Invite platform consists of a negotiation controller that manages ENS engines and negotiation users. The negotiation controller gets user and negotiation information from the platform database. The user and negotiation information provides the negotiation controller with each of the negotiation instance, the parties that are involved, the negotiation case and the side of a case that a party is assigned to. ENS engines in *Invite* refer to negotiation protocols which allow adding and removing components for activities easily. *Invite* ENSs contains all necessary components to construct an ENS. Dividing *Invite* into components makes the system modular and its components reusable.

In this work, we use the Fusebox Lifecycle Process (FLIP) as the software engineering method to guide the development of the system in various stages. We thus give an overview of Fusebox and FLIP in the next section.

5.3 Fusebox and FLIP

Fusebox provides a framework and methodology for creating web applications [TKLC+03]. It divides a web application into components, called *fuses*. The framework supplies a web application engine to link and manage fuses.

Fuses classify the components based on their tasks and assign each a prefix. *Fuses* whose names have prefix *qry_* are used to obtain some data from the database and create a record set. *Fuses* whose name has prefix *dsp_* display mostly HTML files including the

record set created by *qry_* fuse. *Fuses* with prefix *act_* process data input or perform file processing. *Fuses* with prefix *lay_* display the structure of a web interface layout that would be used to assemble a web page.

Once the components are classified, they are registered in a circuit under a fuseaction. A Fusebox circuit consists of a single directory file. It does a few related tasks. An analogy of a circuit is any circuits in a house. Circuit for kitchen is separated from the circuit for the living room. In other words, Fusebox circuits are sorted by the task they perform. Inside a circuit, there is fuseaction, which puts all related fuses together. For example, to display a search result, we need to include a *qry_ fuse* to get data from the database, a *dsp_ fuse* to display the output and a lay *fuse* to organizing the layout of the data. Each user request is interpreted as a fuseaction. It is passed to the server file index.cfm by a URL variable. The index.cfm parses the fuseaction variable and triggers the action that user has requested. Note that index.cfm is the manager file. Every request from the user has to pass through this file in order to be processed (see Figure 10).

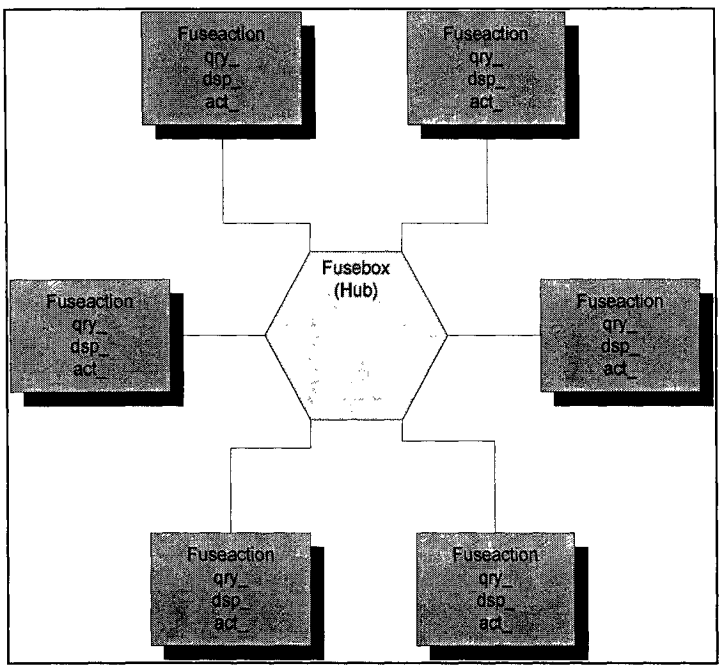

Figure 10. Logical view of Fusebox

Fusebox also provides a software development methodology called FLIP. It consists of the following steps: (1) wireframing, (2) prototyping, (3) application architecting construction and (4) coding.

5.3.1 Wireframing

Wireframing constructs mock-up prototypes using key activities only. It is the simplest prototype using plain text. The result of the wireframing does not have the look of final product but provides a logical view of the system for the software architect. Figure 11 shows a wireframe of a Registration form page. It shows what fields the page contains in the page (i.e., firstname, lastname, date of birth, username and password), and the "submit button" links to a validateData.cfm page, and a home link to index.cfm.

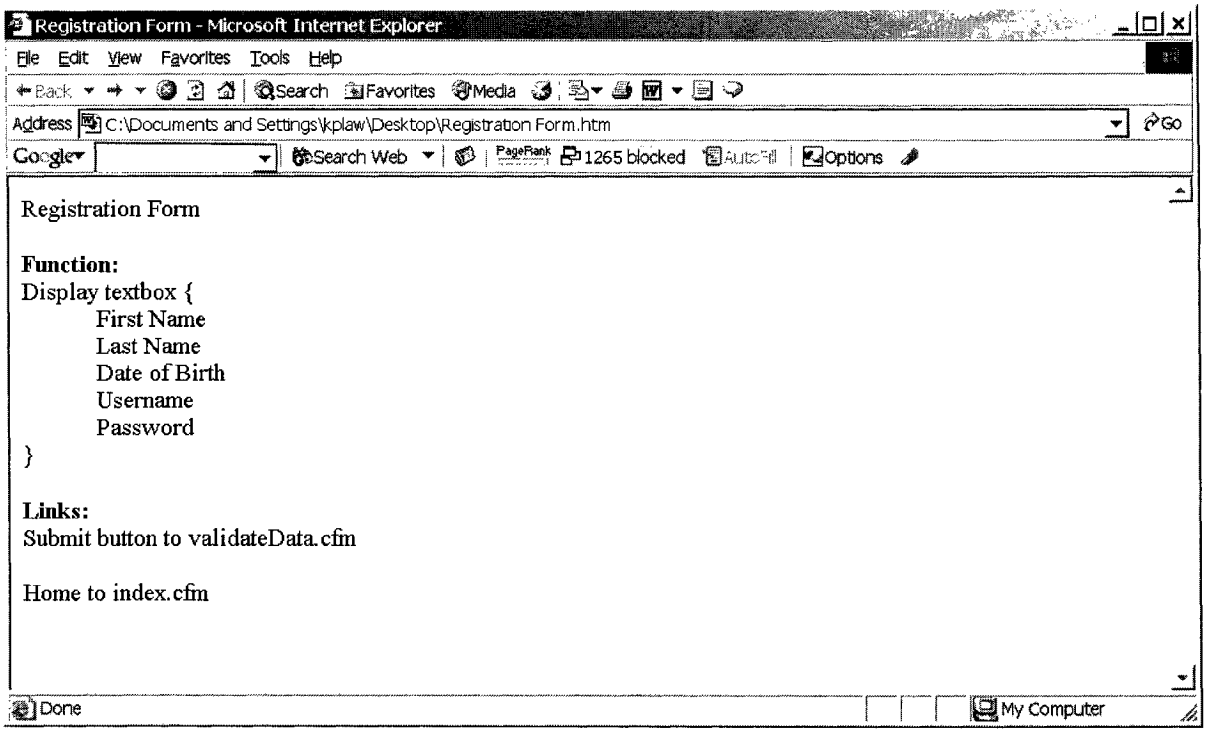

Figure 11. Wireframe example of registration

5.3.2 Front-end prototyping

Fusebox Prototyping is a fully clickable end product of the system. However, it does not have back-end to process user inputs. The purpose is to show the final layout of the product to the client to apply changes in the earliest stage of the development. The prototype is usually done in html, based on the wireframe specification. A prototype of the registration based on wireframe example is illustrated in Figure 12.

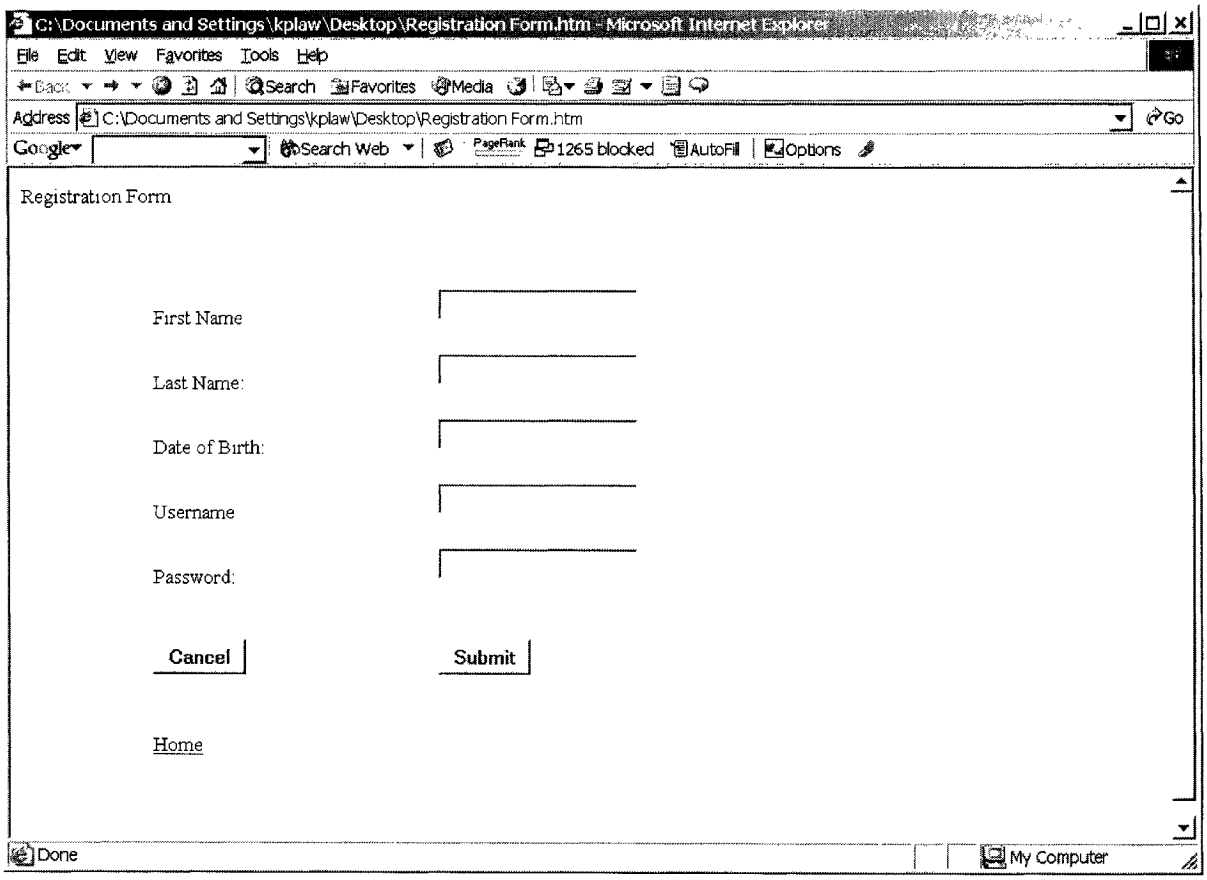

Figure 12. Front-end prototype of the registration example

5.3.3 Application architecting

This module constructs the back-end design and Fusebox schema. It identifies the system required fuseactions and organizes them into circuits. Architecting also breaks down the fuseactions into individual fuses based on their tasks. For each fuse a document called FuseDoc is created. FuseDoc is an XML documentation for fuse prior its construction. FuseDoc provides the developers the necessary information to code a fuse without any information of where the fuse belongs to. It includes a general task of the fuse, all input and output variables, display format, etc. Figure 13 displays an example of a FuseDoc.

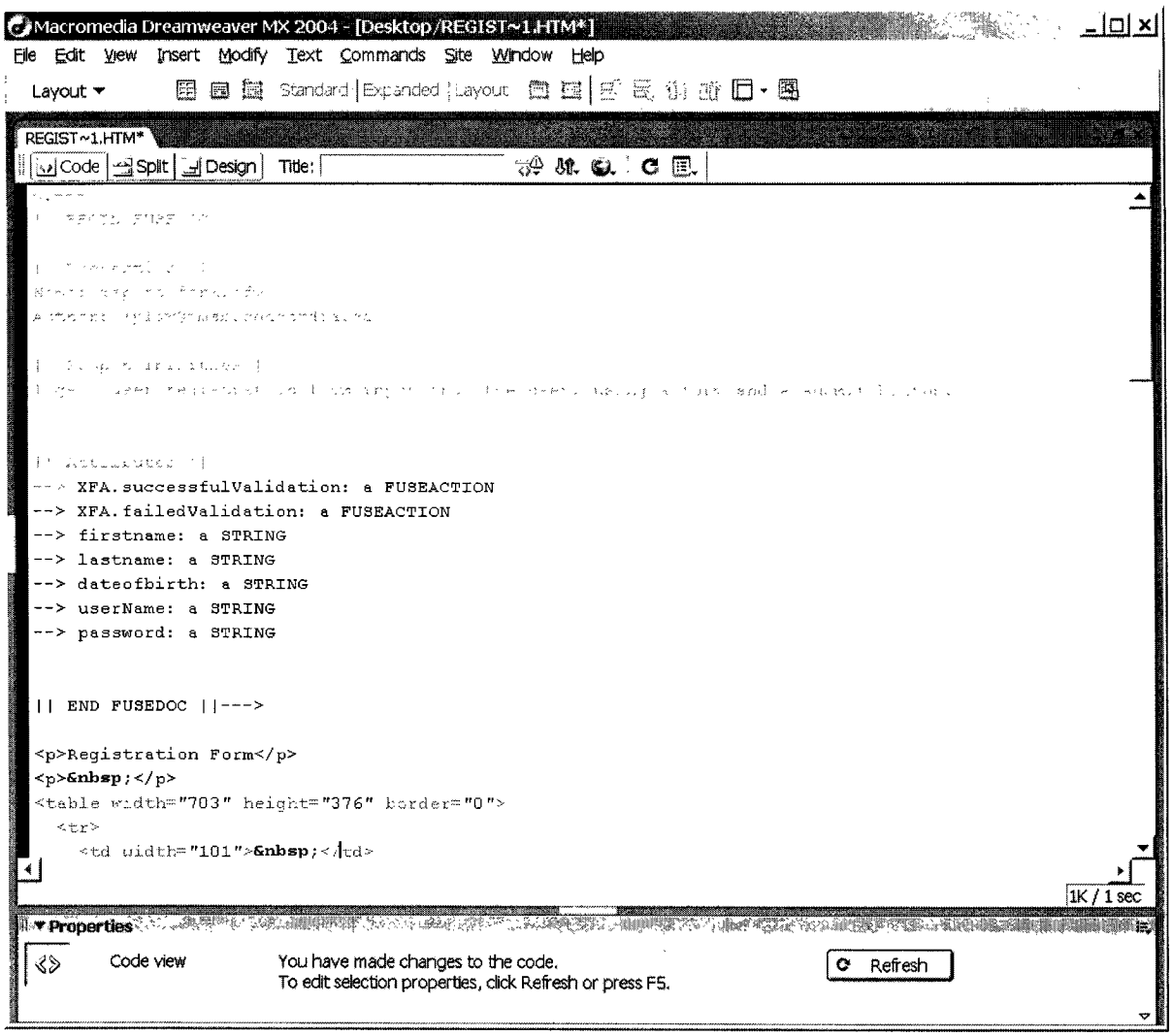

Figure 13. FuseDoc

5.3.4 Construction and coding

A FuseDoc is a recipe for building a fuse. A programmer writes a fuse according to the FuseDoc. Each fuse can be tested independently as there should be no dependency among the fuses. Once all related fuses are coded, the Fusebox proceeds to assemble them together in a circuit and tests them.

5.3.5 Unit testing

Fuses can be tested individually for correctness because each fuse is independent of others. This allows many programmers to work on the same project concurrently. For instance, a simple application that displays some data from the database to a web browser, requires three fuses; a query fuse (*qry_)* that contains a SQL statement that inserts data into a recordset variable, a display fuse (*dsp_*) that interprets the recordset to html, and a layout fuse to provide the display structure of the data. Testing of query fuses (*qry_*) does not depend on the display fuse because its input or output does not affect any other fuse. It is true that the display fuse (*dsp_*) input is from a query fuse (*qry_*); however, the task of the display page is to present a variable which is the output of a query fuse and displays to the web browser. Because its input is a variable, the display fuse does not need to be tested along with the query database. Its main task is the display the data properly.

5.4 Model-View-Controller

The Model-View-Controller (MVC) concept is a programming design pattern invented by Xerox PARC [TKLC+03]. It decomposes software into three types of objects; models, views, and controllers. View objects manage the graphical output of an application. Model objects manage the action and the data of the application domain. Controller manages the interactions, including, a mouse and keyboard input from the user to the system.

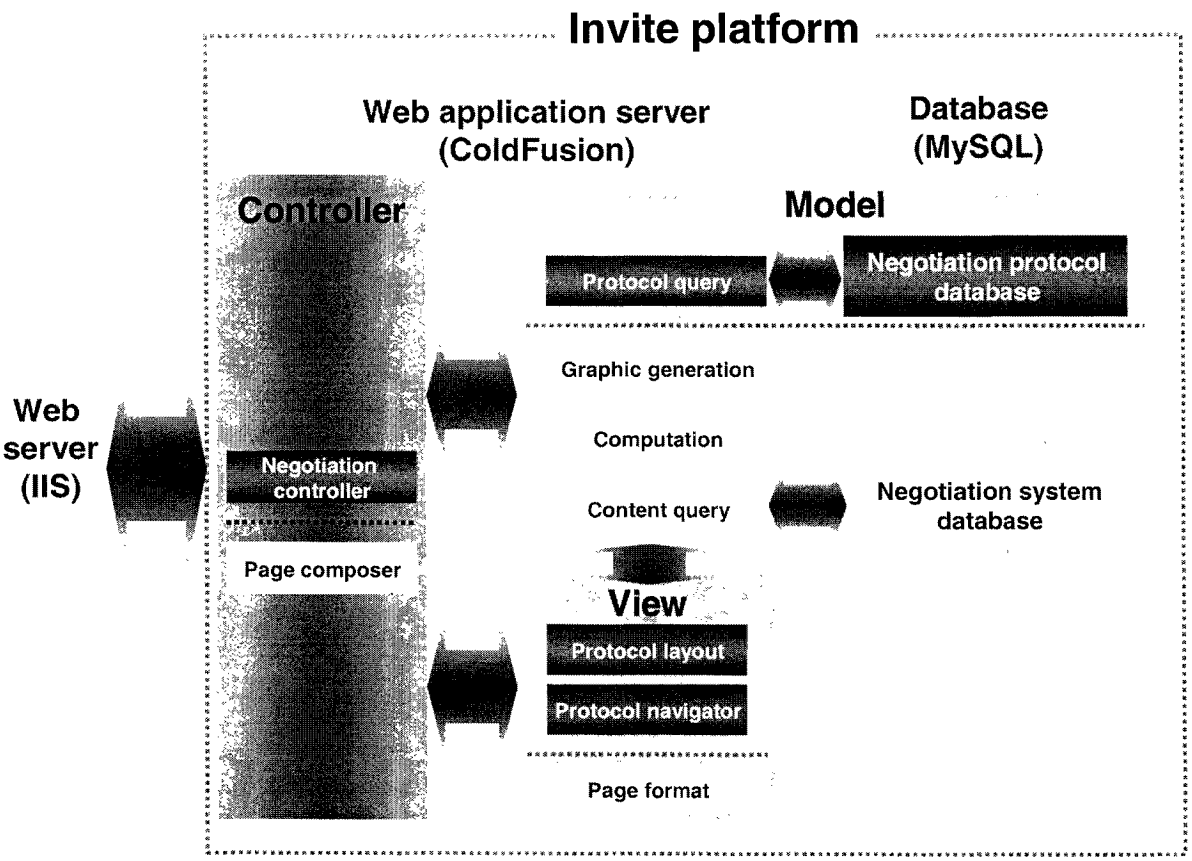

Figure 14. Invite MVC architecture

Figure 14 illustrates the MVC architecture of *Invite*. In Invite, the MVC controller includes two parts: negotiation controller and *page composer*. MVC controller gets a GET or POST HTTP request from users. A HTTP request includes a negotiation protocol state number which a user would like to visit. Negotiation controller and *page composer* takes the request and proceeds to select the appropriate model and view. Negotiation controller commands the model to validate and update the users' state using protocol query against the negotiation platform database. It then pushes the output to the view to display it to the users. The view for the negotiation controller consists of a protocol

layout which displays a customized layout according to the negotiation system a user uses, and protocol navigator which displays states a user is allowed to visit.

On the other hand, *page composer* commands the model to execute the content of the requested state. The content may include the need for using graph generation, computation, content query and negotiation database. The output of the model is pushed to the page format, which is the view for *page composer*, to display it to the user.

At the implementation level, we store the application files into three different folders, one for each of the layers. Because Fusebox methodology already separates application files into display, action, query and layout, it is straightforward to adapt Fusebox to the MVC model. All display (*dsp_*) and layout (*lay_*) files are stored in the View folder, and all action (*act_*) and query (*qry_*) files are stored in the Model folder. The controller folder contains a circuit along with all fuseactions called by the users.

5.5 Database modeling

The process of designing a database starts with gathering and analysis of the data which identifies the data as well as the relationships among data. A structure (schema) of a database, that is, its schema, is identified during the modeling and normalization of the data. Several database modeling approaches have been proposed. (i.e. ODL, network, hierarchical, and E/R). The most widely used approach is the Entity-Relationship (E/R) model which is a graphical language.

An E/R diagram is a graphical representation of the structure of the database [U89]. It has three main components: *entity*, *relationship* and *attributes*. The entity set component is a set of entities (objects) with similar properties. Attributes describe the properties of entity set. Relationships describe the type of the connection between entities. An example of E/R diagram is presented in Figure 15. This diagram has two entity sets: UserInfo and History. UserInfo has six attributes and History has three attributes. There is a relationship that links these two entities together. The diamond at one end and the circle at the other means there is a 1-to-many relationship from UserInfo to History. If both ends have are diamond, the relationship is a 1-to-1. If both ends are circle, the relationship is many-to-many.

The underlined attribute or set of attributes, in general, indicates the key of an entity. Every entity set or relationship set has a unique primary key. However, an entity may also have a foreign key used to relate its attributes to the attributes of an entity from which the foreign key (FR) comes from. This situation is also illustrated in Figure 15. The User_id attribute is the foreign key in entity History taken from entity UserInfo. This foreign key may have repeated values in History and it takes place when there are many "histories" for one user (uniquely identified with User_id).

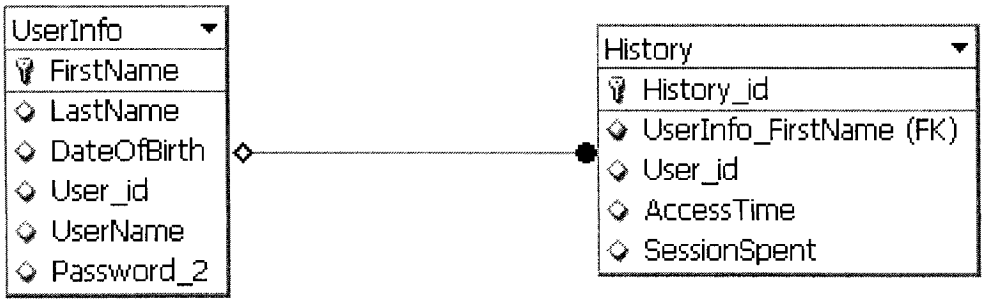

Figure 15. E/R diagram

6. Design and Implementation

6.1 Design and implementation challenge

The difficulty in the design and implementation of *Invite* is to translate from a high-level negotiation process model described in Section 2.2 into a very concrete and detailed negotiation protocol which is represented as a set of rules controlling the system execution and interaction with its users. Although a framework for the construction negotiation protocols have been proposed by Kersten, Strecker and Law [KSL04; KSL204], during the development a number of issues required further specification and modification. In particular this work led to the revision of the concepts—discussed in the chapter—of a controller, page composer and intervening event.

The interdependence of the negotiating parties requires synchronization of exchanged messages and offers. This is a difficulty that has to be addressed. Even though this prototype of *Invite* is not meant for real-time negotiation, the system has to deal with situations in which both negotiation parties are logged in and are sending offers to their counterpart at the same time. Another challenge is to keep track of the states of the users. For instance, a user who was in state 4 right before she logged-out, should begin in state 4, once she logs in again. Yet another is to keep the system modular, so that components can be reused in different protocols.

6.2 System architecture

Invite is a 5-tier web application that includes a web browser technology in the

presentation or the GUI tier, a web application server in the presentation logic, business and data access tiers, and a database management component in the data tier.

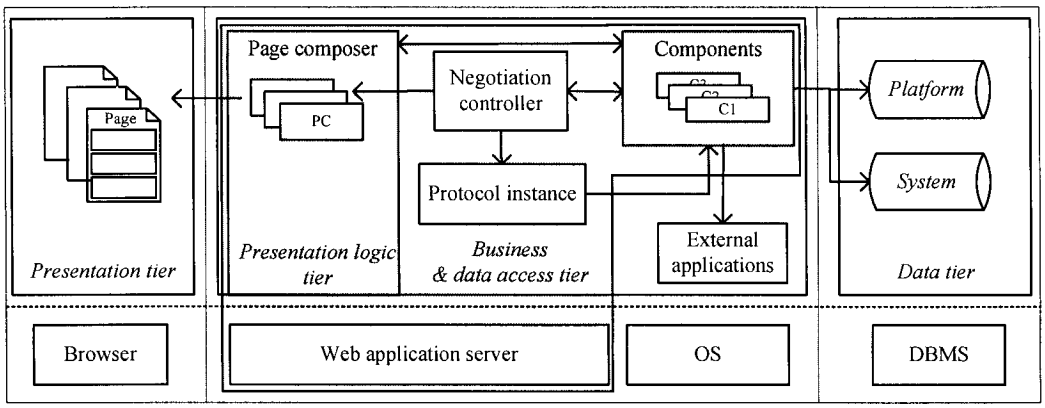

Figure 16. Prototype system architecture

Figure 16 illustrates the architecture of the *Invite* prototype developed for the purpose of this thesis. The browser under the presentation tier displays HTML pages, formatted by the presentation logic tier, to the user and receives user input which it passes to the web application server.

Business tier is where we place all the custom business logic. It is divided into four parts: *negotiation controller* (presented in Section 7.2.1), *page composer* (Section 7.2.2), *components* (Section 7.2.3), and *protocol instance*. *Negotiation controller* and *components* compose the business tier of the system. *Negotiation controller* uses components to control the execution of negotiation protocol instances and executes requests from *page composers*. *Page composers* compose the presentation logic tier that delivers the output to the presentation tier. It gathers and formats the display layout of *components*. The *components* contain business logic of the platform and *ENSs* which run

on the platform. Components also serve as the data access tier to communicate with the data tier.

The DBMS is the data tier of the system manages two or more databases. One database is dedicated for the *Invite* platform which different protocols reside and the other(s) are complementary databases to store *Invite* application (negotiation engine). Because *Invite* is a multi-protocol platform, it allows two or more different ENSs (e.g., *SimpleNS*-like, *Inspire*-like) run at the same time. Each ENS stores its users' data separately. This facilitates the analysis and collection of data from negotiators.

6.2.1 Components

A *component* is the basic building block of *Invite*. Each component corresponds to an activity described in Section 4.2. A component is a piece of code that receives input and generates output. There are two categories of components in *Invite*;

1. The *platform component* which is associated with the general functionality of the system, process model and its protocol translation (i.e. login component, user update component, activate *intervening event* component), and

2. The *Invite ENS component* which is used to build negotiation engines. (i.e. send message component, negotiation problem display)

A *component* is also an application interface. In this case, it connects *Invite* to external programs, such as graphic generator and utility calculator. The flexibility obtained by dividing the system into *components* makes the system extensible. In addition to using

the *components* provided by *Invite*, the system can also use *components* from other applications.

6.2.2 Page composers

Page composers are used to gather and organize *components* for the purpose of constructing a single web page. A *page composer* is a program that sets the layout and defines the order of the execution of the *components*. Because *components* are reusable, different *page composers* can invoke the same *components*. This contributes to reusability and increases efficiency of the system maintenance.

The modular architecture of *Invite* and its reliance on protocols allow for the use of protocols that differ in a few components and have slightly modified *page composers*. This is useful in experimental studies. For example, one can study the impact of graphical elements of the interface on the negotiation process and outcome. In one study, a *page composer* may invoke a graphing component and in another it may not. This feature is also useful for internationalization of the system. It allows using the same components but in different language.

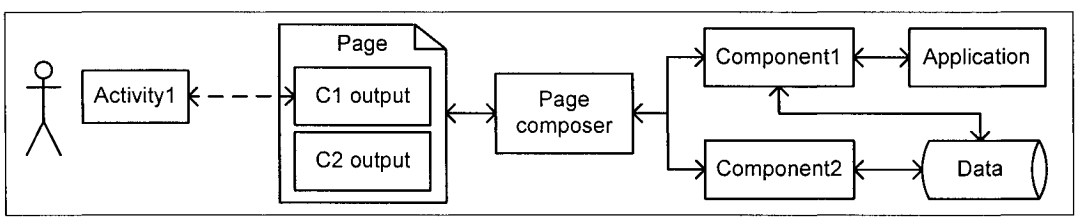

Figure 17. Component *page composer*

Figure 17 illustrates *components* and *page composers* association. *Components* process

input and generate output. They also access and use data stored in the database. *Components* are invoked by the *page composer* and the output they produce is inserted on the web page which is sent to the user. Note that *components* do not always correspond to a single activity. For example, in Figure 16, only component 1 could be a system *component* that links to an activity. Component 2 could be a platform component, such as footer or header.

6.2.3 Negotiation controller

The *negotiation controller* is an intermediary program that updates and executes protocol instances. The *negotiation controller* runs a copy of an initial protocol, that is, a protocol instance. This protocol instance is constructed prior to the negotiation and it is updated dynamically to account for all activities of the negotiators, which are undertaken at any given moment. The *negotiation controller* obtains information, such as the current *sequence* and *ENS state* of the negotiation from the URL variables of a requested HTTP protocol. Based on this information the *negotiation controller* invokes the proper *page composer* for execution. *Page composers* are visible only to *negotiation controller.* They can be added to the system without changing the implementation.

The design of the *components/page composers/negotiation controller* allows easy expansion of negotiation systems. It does not impose coupling among the *components. Components* can only be invoked by *page composers*, and *page composers* can only be invoked by the *negotiation controller.* This design allows the system architect to introduce new *components* or *page composers* at any time with minor modification to the

system. All *components* and *page composers* are indexed in the Fusebox circuit.

Negotiation controller also controls access to *page composers*. For example, suppose a user logs into the system for the first time, so she is in the *readPublicCase* state under sequence 1 shown in Table 2, and then the user logs out. Next time, when the user logs into the system and tries to access the *readPrivateCase page composer*, the system prevents her from accessing *readPrivateCase* because her last *ENS state* visited was *readPublicCase*. The reason is that every HTTP request is handled first by the application server, which forwards the request to the negotiation controller, which in turn checks the request against the current *ENS state* of the user in the database. Her attempt fails since the record shows that she is still in sequence 1 while she is requesting to go to some *ENS state* in sequence 2. The *negotiation controller* displays an error page to inform the user. Another function of the *negotiation controller* is to get from the database all possible states and exit points in a sequence and to interpret them in the user interface, represented by links.

6.3 Protocol example

A simplified instance of the protocol representation in the database is shown in Table 2. For simplicity, we assume that the initial *ENS state* is a mandatory state so that as soon as the user enters a *sequence*, she is allowed to go to other *sequences* if there is an exit. Each row in the table represents a *sequence* in the protocol. We have the *sequence_id*, the *ENS states*, and *exits*.

Also in the database, we have an indexed pool of all available *page composers*. For

example, *readPublicCase* is a *page composer* which contains a text field *component* to display a case. In the first *sequence* (sequence 1), the *initial ENS state* invokes *readPublicCase page composer*. In that *sequence*, there is no other *optional ENS state*; however, there are two exits, *exit_2* and *exit_6*. That means, the user can only exit to *sequence* 2 or 6 and if she chooses to remain in the *sequence* 1, she can only see the *page composer readPublicCase*. Suppose, she chooses *exit_2*, then she is moved immediately from *sequence* 1 to *sequence* 2.

Table 2. SimpleNS protocol

User_id	Sequence_id	Initial_State	Optional_State	Exit_point
1	1	readPublicCase	N/A	exit_2, exit_6
1	2	readPrivateCase	readPublicCase	exit_3, exit_6
1	3	sendOffer / Message	readPublicCase, readPrivateCase	exit_4, exit_6
1	4	readOffer / Message	readPublicCase, readPrivateCase, history, agreement	exit_3, exit_5, exit_6
1	5	Agreement	readPublicCase, readPrivateCase, history	
1	6	terminateNego	readPublicCase, readPrivateCase, history	

Because the *initial ENS state* of *sequence* 2 invokes *readPrivateCase*, the content collected and formatted by the *readPrivateCase* composer is displayed to the user. From here, she has an option to go to *readPublicCase page composer* or to go to sequence_3 or

sequence_6. Note that the *readPublicCase* in sequence_2 is not associated with the same *ENS state* as *readPublicCase* in sequence_1 even though they are constructed by the same *page composer*. The latter is *readPublicCase* state in *sequence* 1 whereas the former is *readPublicCase* in sequence_2. Continuing the negotiation, the user chooses exit_6 which terminates the negotiation. She is then moved to sequence_6, which has no exit. This means that the user is forced to remain in the *sequence* and the negotiation terminates there.

6.4 The database design

The data model of *Invite* platform is shown in the entity-relationship (E/R) diagram in Figure 18, 19, and 20. There are twenty tables in the platform database to manage and support the negotiation systems. The database contains users' information, initial protocol schemata, and the actual user protocol instance tables. For clarity of the presentation, we omit the initial protocol tables. Initial protocol contains an original core model of a protocol. As discussed in Section 7.2.3, the initial protocol is copied to the user protocol instance tables when users request a negotiation.

At the beginning of a negotiation, the user protocol instance is an exact copy obtained from the database, which is then modified and adapted during the negotiation based on her interaction to suit the user.

The database does not contain any table related to specific negotiation. For example, the table that stores a history of exchange offers between two negotiators is not stored in this database. Each ENS has its own database to store such information. The platform

database is designed purely to support and control negotiation process model, protocols, and protocol instances.

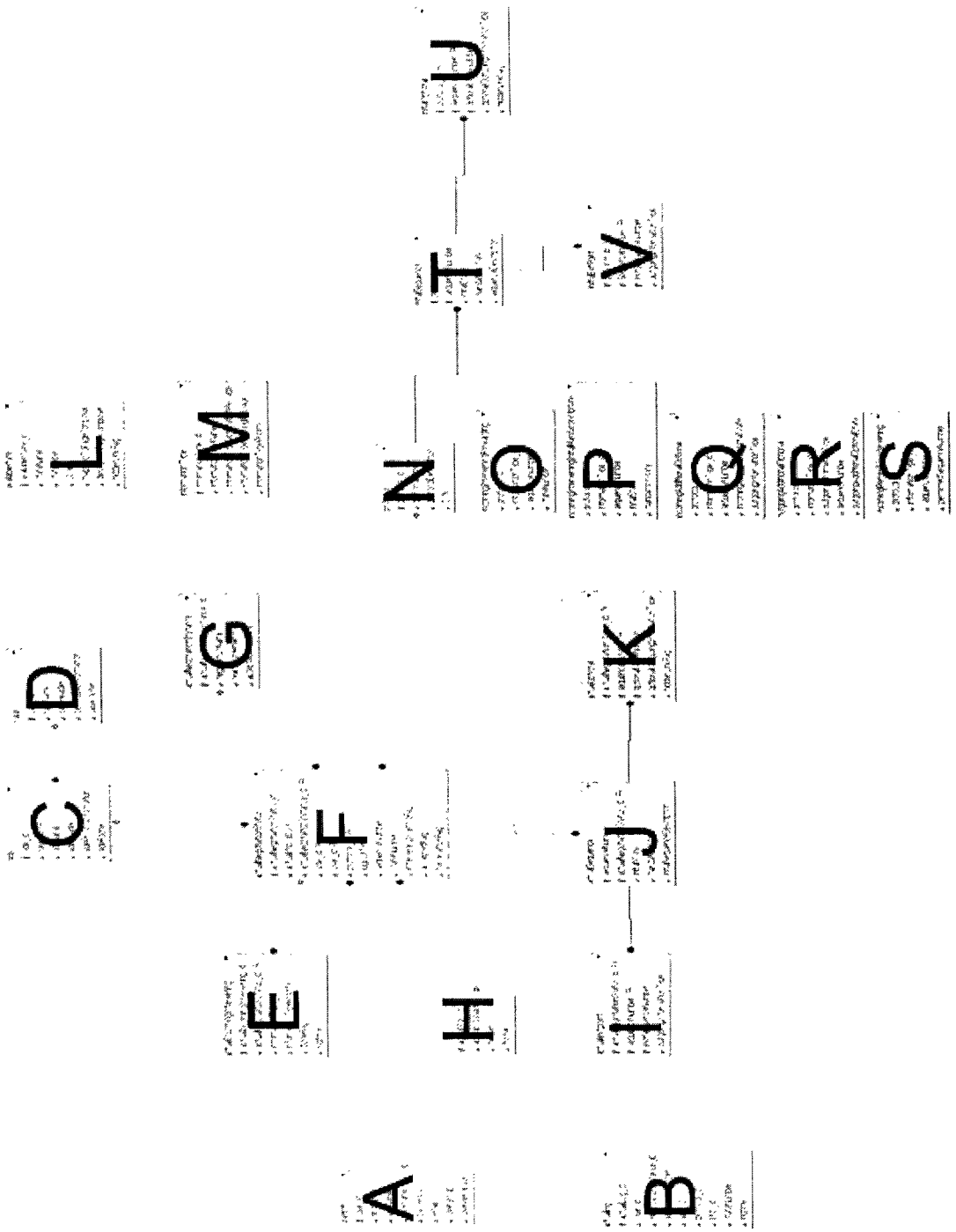

Figure 18. Invite data model

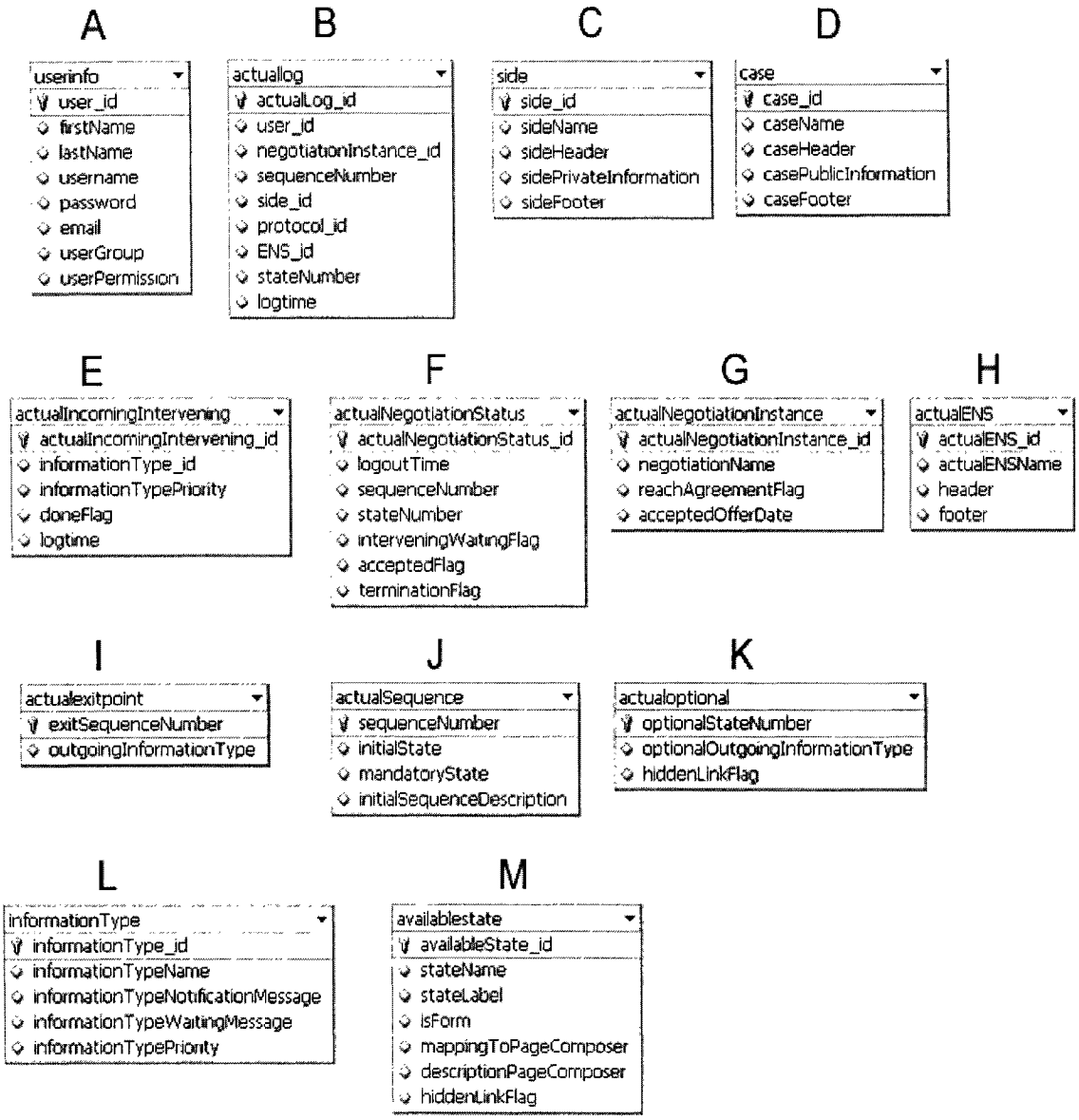

Figure 19. Invite data model (cont')

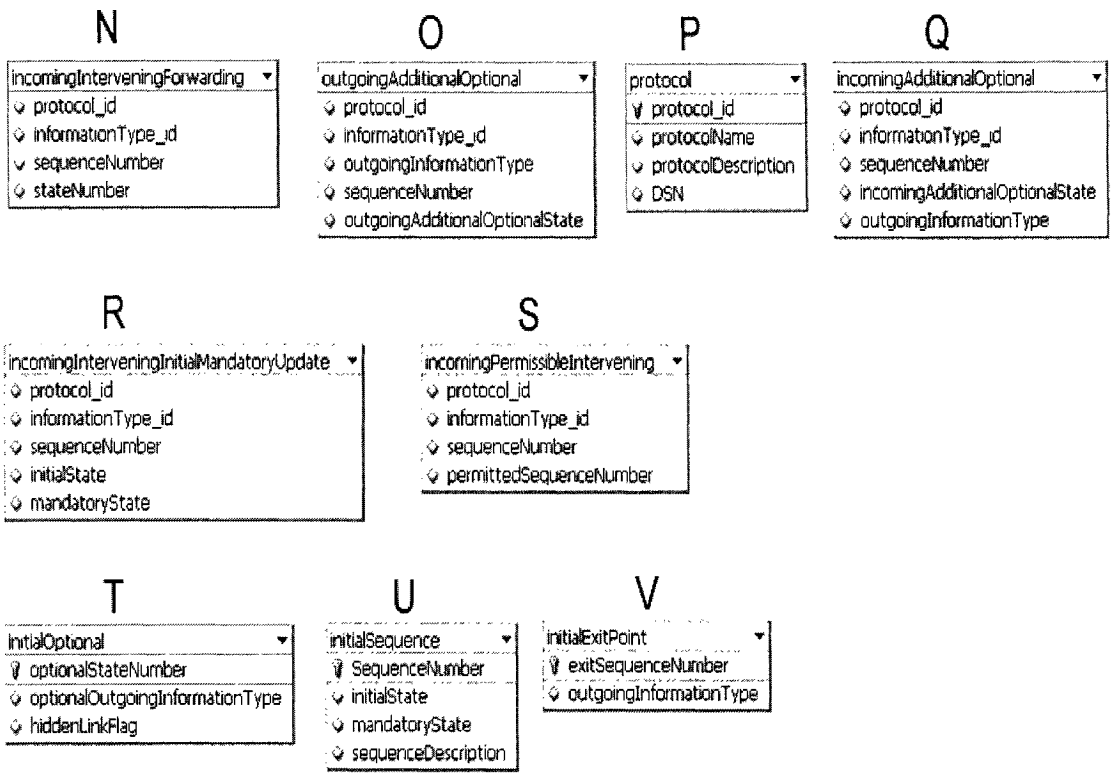

Figure 20. Invite data model (cont')

Every table in the database has a primary key (PK) define unique identifiers for the records. Tables also contain foreign keys that relate them to other, tables. We next briefly describe the data stored in the platform database tables is briefly discussed.

The *UserInfo* table contains personal information such as username, last name, first name, password, and email of all negotiators in the system. The primary key is user_id.

The *ActualNegotiationInstance* table stores information about the assignment of negotiation cases, side_case, protocol and ENS to users. The primary key of this table includes the attributes *actualNegotiationInstance_id*, and *protocol_id*. This is because a

negotiation instance is assigned to two or more users and they can be assigned to different protocols. This table is linked to *User* table by the *user_id*. This table, representing a relationship, also assigns a negotiation case and the *side* to a user. The attribute *dsn* is used to identify the ENS that the negotiators are negotiating in. If a negotiation is complete, an attribute *end* assumes value 1. Otherwise, its value is 0.

The *actualNegotiationStatus* table contains the run-time status of a side of negotiation. Note that once a user is assigned to a negotiation case, she is referred by her *side_id* rather than by her *user_id*. This facilitates future ext*ens*ion of the system to deal with multi-bilateral negotiations in which a side has more than one user. Attributes *side_id*, *negotiation_instance_id*, and *protocolState_id* form the key for this table. *sequenceNumber* records the current negotiation sequence the user is in and the *stateNumber* attributes records the current *ENS state* in this *sequence*. When Intervening event occurs, *InterveningWaitingFlag* is assigned a number greater than 0. This number is the *sequence_id* that the system should send the user to a particular sequence when she is permitted.

Table *actualSequencel* is the main negotiation protocol instance table. It is decomposed into the following tables after we normalized it, *actualOoptional*, and *actualExitPoint* after normalization. The key includes the attributes *side_id*, *acutalNegotiationIinstance_id*, *sequenceNumber*, and *protocol_id*. *sequenceNumber* corresponds to a *sequence* in the process model theory. Each *sequence* has one *initial ENS state* and one *mandatory ENS state*. Note that this table links to itself by the *mandatoryState_id*. This occurs when a user exits one *sequence* to enter another.

Table *Availablestates* stores an index of all available *page composers* (see section 7.2.2). Its primary key is *generic*. Table *actualNegotiationStatus* links to this table to get the proper display *page composer*. Attribute *stateName* stores the name of the *page composer*. The names should allow protocol designer to identify the content of the *page composer* whereas the *mappingToPageComposer* attribute stores the actual fuseaction names of the *page composers*.

Table *actualOptional* table stores the *optional ENS states* of *sequences* in protocols. This table is the result of the normalization of relation *actualSequence* normalization. It uses the set of *side_id*, *actualNegotiatonIinstance_id* and *protocolState_id* as its primary key.

Table *actualExitPoint* stores one type of the exit point (see section 4.3). It links to the *actualNegotiationStatus* table to go from one sequence to another. The table primary key is the combination of *side_id*, *negotiaton_instance_id* and *protocolState_id.*

Table *Protocol* table stores a list of available protocols. The attributes *starting_state* and *ending_state* stores the starting and ending states of each protocol in the *UserInstanceProtocol* table. Its primary key consists of the collection of attributes *side_id*, *negotiaton_instance_id* and *protocolState_id.*

IncomingInterveningForwading, *incomingInterveningInitialMandatoryUpdate*, *incomingAdditioanlOptional*, *outgoingAdditionalOptional*, and *incomingPermissibleIntervening* are lookup tables of intervening event. They indicate when to trigger intervening event and what to trigger.

Table *actualLog* stores the history of a user protocol instance. It records every *sequence* the user has got into during a negotiation in a chronological order. Its primary key is the combination of *side_id*, *negotiatonInstance_id,* and *actualNegotiationStatus_id.* Attribute *accesstime* stores a timestamp, with the obvious meaning.

6.5 Protocol construction steps

The protocol construction in *Invite* consists of the following four steps:

1. Identification of the negotiation stages;
2. Separation of the negotiation stages into *activities*;
3. Mapping of the activities to existing list of *page composers,* and
4. Population of the *Invite* database.

These four steps are briefly described below.

The first step in constructing protocols identifies the negotiation process and stages. Recall that according to the adopted process model (see Section 2.2); all negotiations follow a process which consists of 5 phases. Not these stages described in section 2.2 are always used in a negotiation. To construct a negotiation protocol, the protocol designer selects the stages needed for that protocol.

Step two separates negotiation stages into *activities*: negotiation process stages can be

divided into *activities*. Negotiation *activities* are the tangible components of a negotiation, In *Invite*, these activities are represented in *Invite* by at least one *component*.

In step three, *activities* are mapped to list of existing *page composers*. Once the protocol designer identifies all the *components* needed, the designer proceeds to map them to existing *page composers*. A *page composer* is a set of *components*. The list of existing *page composers* is an index of *components* in a particular *page composer*. If a *component* needed in a protocol is not found in the list, new *components* and/or *page composers* can be created and added to the list.

In the fourth step, the *Invite* database is populated. Once all *page composers* are identified, the last step is to populate the database. The design of the database in *Invite* is such that it supports the negotiation theory discussed in section 4. Each of the elements, *sequence*, *initial*, *mandatory* and *optional ENS states* and *exit points* in Table 3, has its corresponding table in the database. *Initial*, *mandatory states* link to their respective *page composer* whereas *exit points* link to a *sequence* in the *AvailableState* table.

6.6 Prototype

In this section, we introduce the prototype system we have developed for *Invite*. Recall that according to the FLIP development methodology, a prototype comprises only the user interfaces. A full scale of the prototype with all functionalities was built after we were satisfied with the user interface, the look and like of the system. In our implementation, the user interface consists of web pages for which we have used Macromedia Dreamweaver MX 2003. Dreamweaver is a web design software tool for the

development of graphical user interfaces. Figures 21 and 22 illustrate two different *page composers* that include the same *component*. The snapshots show one of the benefits of a *page composer*, namely the *reusability* of the components.

Figure 21 shows two *Invite* components: *send offer component* (on the left) and *send message component* (on the right). The same two *components* are invoked by a different *page composer* and therefore they are differently presented to the user. That is, the use of two different *page composers* in Figure 21 and 22, respectively results in different page layout. In Figure 21, as compared to Figure 21, the send message *component* is moved to the left-hand-side and the send offer *component* is moved to the right-hand-side of the screen.

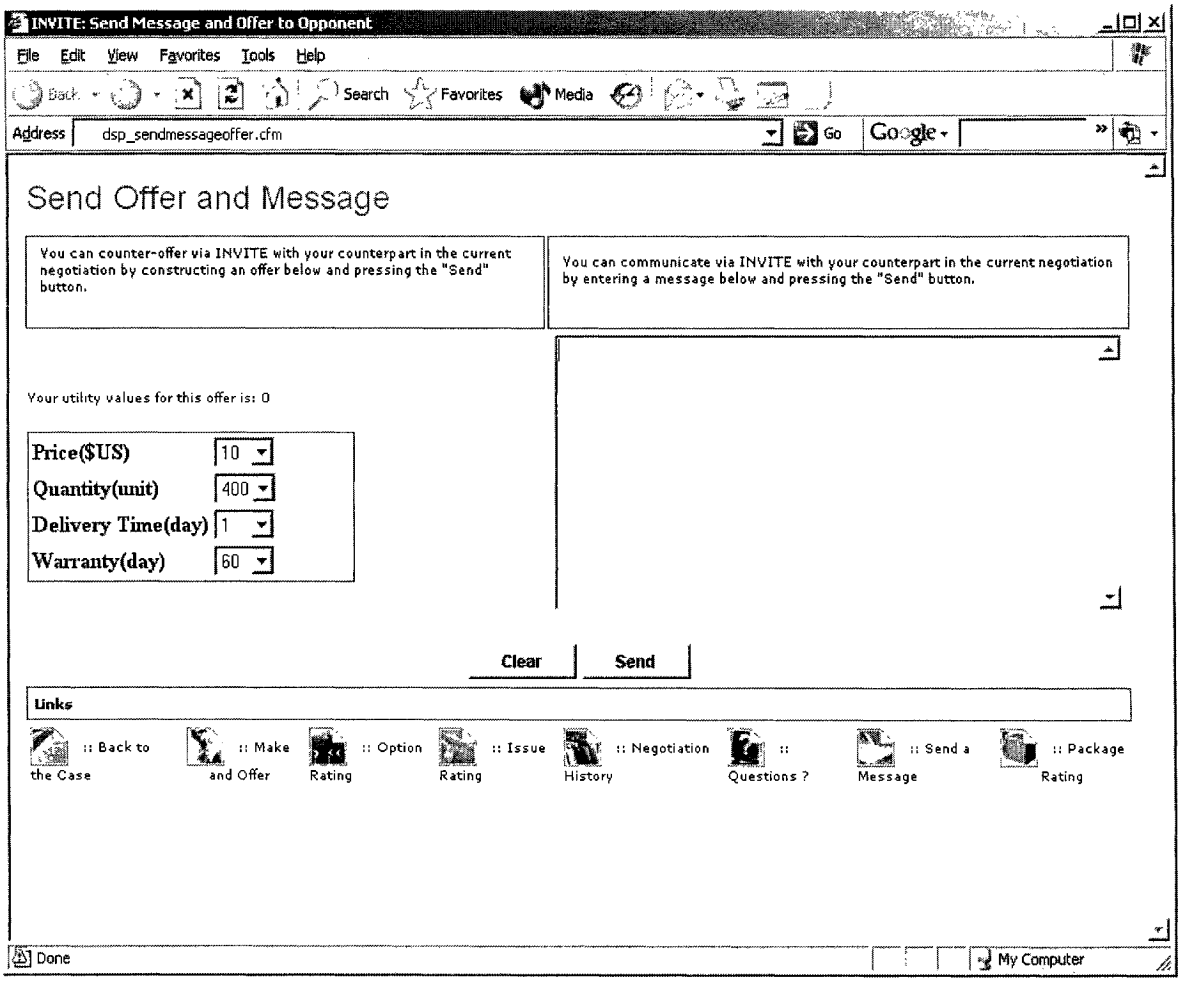

Figure 19. Send offer/message *page composer*

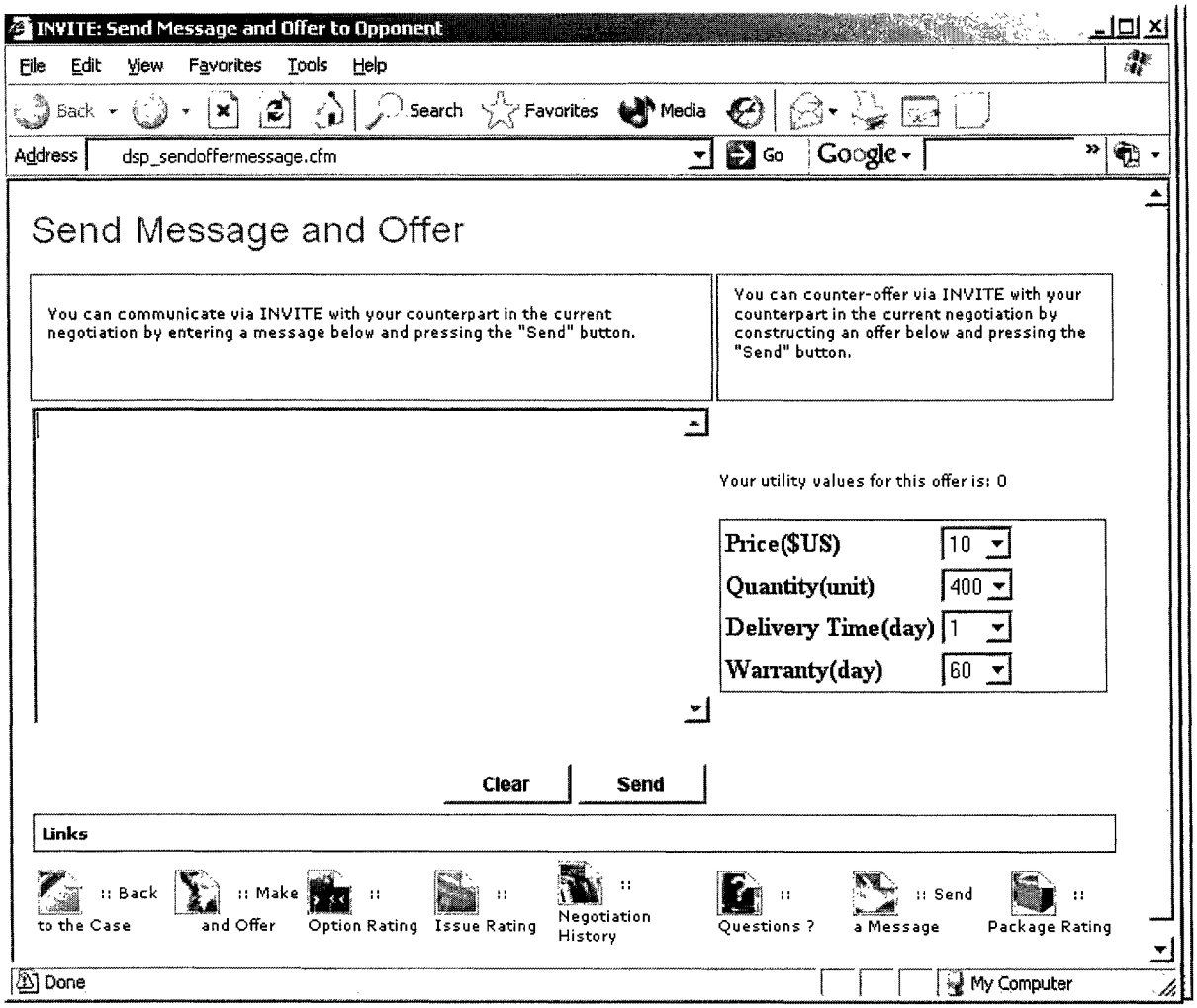

Figure 20. Send message/offer ***page composer***

In the prototyping process, we realized that a *page composer* cannot have more than one html form such as the one shown in Figures 23 and 24, because every form needs a submit button to send user's input to the system. Send offer and Send message *components* have their own submit button. If they are shown on the same *page composer*, only one input from the *components* can be submitted. To overcome this problem, we divided a form file to contain three sections (files), the form header, form content, and form footer. The form header section contains the opening state of a HTML form

(<FORM action="xx" name="xx">). The section form content stores the actual *component* to be used such as send offer and send message *components*. The form footer includes standard form buttons such as reset and submit. To construct a *page composer* that contains two or more forms, we proceed to include the header file, the actual *Invite components*, and then we close the form with the footer file. This way, we can include as many forms in a *page composer* as required, rather easily.

6.7 Invite ENSs

Each ENS has its own database. The database stores negotiation cases in a particular ENS and the data that are used solely by the ENS. For instance, a *SimpleNS-like* ENS database contains tables such as *Case*, and *Side_Case* to store the negotiation cases, and tables *Offer* and *Message* to store negotiation offers and arguments sent by users, in addition to the *Case*, *Side_Case*, *Offer*, and *Message*, an *Inspire-like* ENS database includes, *User_issue_preference* and *User_option_preference* tables.

In order to set up a negotiation instance, a negotiation protocol and a negotiation case are assigned to the parties involved. Each of the ENSs in the *Invite* platform is assigned to a particular protocol. For instance, a *SimpleNS-like* ENS protocol is different from an *Inspire-like* ENS protocol. Moreover, each ENS has its own set of negotiation cases. This is because the content of a negotiation case is often related to the features of the ENS. For instance, a negotiation case that highly depends on an analytical tools *component* does not work on an ENS that has no support for analytical tool such as *SimpleNS*.

6.7.1 *SimpleNS*-like

SimpleNS-like ENS is a replica of the *SimpleNS* system developed by the Interneg Negocourse. Negocourse is an online negotiation course offered to students around the world. *SimpleNS* is a simple *ENS* that has no analytical support. It provides only the media to exchange offers and messages between negotiators.

6.7.2 *SimpleNS* protocol

The *SimpleNS* protocol described in Section 4.1 is produced below again for convenience.

Table 3. SimpleNS protocol (reproduction from section 4.1)

Sequence_id	Initial_State	Optional_State	Exit_point
1	Pre-questionnaire		2
2	readPublicCase	N/A	3,7
3	readPrivateCase	readPublicCase	4,7
4	sendOffer / Message	readPublicCase, readPrivateCase	5,7
5	readOffer / Message	readPublicCase, readPrivateCase, history, agreement	4,6,7
6	Agreement	readPublicCase, readPrivateCase, history	8
7	terminateNego	readPublicCase, readPrivateCase, history	8
8	Post-questionnaire		

6.7.3 *SimpleNS* database

Figure 21 illustrates the *SimpleNS* database. The database is linked to the *Invite*'s platform database through the attributes *side_id*, *negotiation_instance_id*, and *user_id*. The region in white represents the database in *SimpleNS* engine database and the region in dark represents the *Invite* platform database.

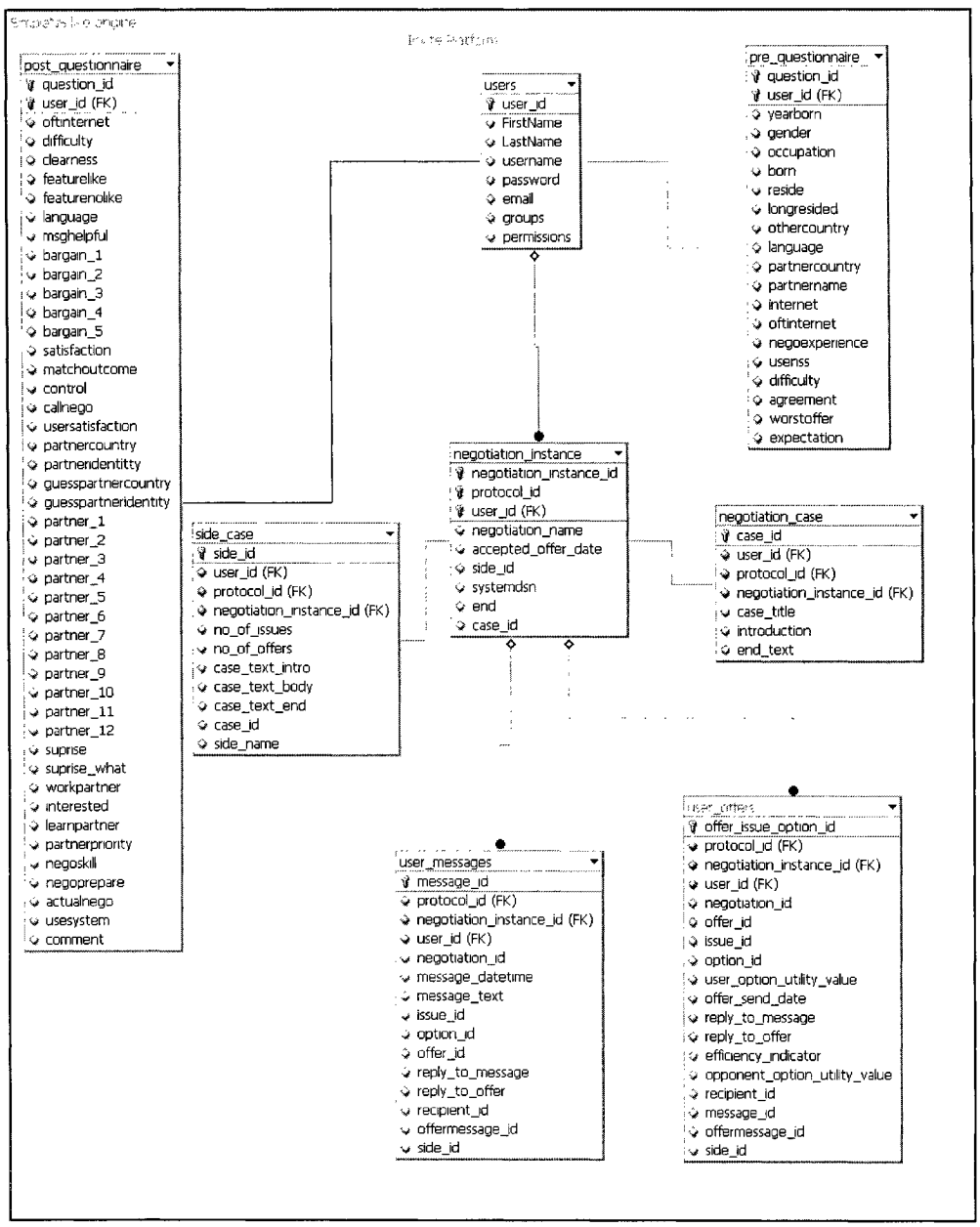

Figure 21 *SimpleNS*-like database

Table *Pre_questionnaire* stores the pre-questionnaire filled by the user at the beginning of a negotiation. The questionnaire is about user's negotiation and internet experience. It uses *question_id* and *user_id* as its primary key

Table *Post_questionnaire* stores the post-questionnaire filled by the user at the end of a negotiation. This questionnaire is about user's experience with the *ENS*. It uses *question_id* and *user_id* as its primary key

Table *Negotiation_Case* stores the description of cases. Every negotiation instance is assigned to a case. A case description contains public information for negotiators. This information gives the negotiators a general background of the case. It is shared among all participants. The primary key is *case_id.*

Table *Side_Case* stores the description of each side of the case. For instance, a bilateral negotiation has two sides whereas a multi-lateral negotiation has n sides. Each side has its own private information that affects their negotiation judgement of negotiators' preferences. The primary key of this table is *side_id.* It is linked to attribute case_id in table *Negotiation_Case.*

Table *User_Message* stores the messages during the exchange of offers and arguments phase. It records the message sent, the id of the user who sent the message, and the time that was sent. Its primary key is *message_id.*

Table *User_Offer* stores the offers during the exchange of offers and arguments phase. It records the offer sent, the user who sent the message, and the time it was sent. This table

can store the offer as a text message or it can be linked to the option table which contains a pre-defined set of options to choose from.

6.7.4 *SimpleNS*-like components and page composers

SimpleNS-like ENS is a simple *ENS*. It uses *components* such as *pre-questionnaire*, *post-questionnaire*, *send message*, and *send offer* components. To organize the *components*, we use the following *page composers*: *readPublicCase*, *readPrivateCase*, *pre-questionnaire*, *sendOffer-message post-questionnaire* and *history*.

The ReadPublicCase page composer displays general information of a negotiation case that is shared by all negotiation parties.

The ReadPrivateCase page composer displays private information of a negotiation case that is given to a particular negotiation party. This information is not shared to other negotiation parties.

The Pre-questionnaire page composer displays the content of a questionnaire in a form. It then submits the user input to the verification *component* to check for errors. If an error is detected, the system sends an error message to the user. If the input information is validated, the system proceeds to insert it into the database. The output of this *page composer* is the success or failure flag. This *page composer* uses the following *components*, *dsp_prequestionnaire*, *act_prequestionnaire*, *act_verifyprequestionaire* and *qry_prequestionaire*.

The SendOffer-message page composer displays two text boxes, one for composing

message and the other for composing offer. Once the message and/or offer are submitted, it proceeds to verify whether the text is empty. If that is the case, it returns with an error. If there is no error, it passes the data to a component which inserts it into the database. The output of this *page composer* is the success or failure flag. This composer also uses *agreement* and *terminateNego components* represented by buttons. The agreement is triggered when either side of the negotiation accepts his counterpart offer or counter-offer. Once the agreement button is pressed, the negotiation is terminated. At this point, no other exchange of offers can be made. The *teminateNego component* is activated if either of the negotiation parties terminates the negotiation without reaching an agreement. This page uses the following *components*, *dsp_formHeader*, *dsp_sendMessage*, *dsp_sendOffer*, *dsp_formButton*, *dsp_formFooter*, *act_verifySendMessage*, *act_verifySendOffer*, *qry_InsertMessageOffer*, *dsp_agreement*, *dsp_terminateNego*, *act_agreement*, and *act_terminateNego*.

The Post-questionnaire page composer displays the content of the post-questionnaire in a form. It then submits the user input to the verification component to check for errors. If an error occurs, the system sends an error message to the user. If the input information is validated, the system proceeds to insert it into the database. The output of this *page composer* is the success or failure flag. This *page composer* uses the following *components*: *dsp_postquestionnaire*, *act_postquestionnaire*, *act_verifypostquestionaire,* and *qry_postquestionaire*.

The History page composer displays the history of exchange messages and offers in the chronological order. This *page composer* has the *qry_history* and *dsp_history* as its

components.

6.8 *Inspire*-like ENS

Inspire is an advance version of *SimpleNS*. It not only includes the communication support but also the analytical support. It offers a combined analytical tool to help negotiators formulate preferences, reservation levels, and strategies. The combined analytical tool is composed of four Invite components; (1) Issue rating, (2) Option rating, (3) Package generation, and (4) Package rating. The analytical tool components are discussed in section 7.8.3.

6.8.1 *Inspire*-like protocol

The *Inspire*-like ENS includes not only all negotiation sequences shown in *SimpleNS* protocol but also the rating negotiation *sequences*. The *Inspire-like* protocol is shown in Table 4.

Table 4. Inspire protocol

Sequence_id	Initial_state	Optional_state	Exit_point	Intervening event trigger	Intervening event
1	readPublicCase	N/A	2,6		
2	readPrivateCase	readPublicCase	3,6		
3	Issue rating	readPublicCase, readPrivateCase	4		
4	Option rating	readPublicCase, readPrivateCase,	5		
5	Package rating	readPublicCase, readPrivateCase, history	6		
6	sendOffer / Message	readPublicCase, readPrivateCase, history	7,8	6	6
7	Agreement		9	7	7
8	Terminate negotiation			8	8
9	Post-settlement		10		
10	Post-settlement sendOffer/ Message	readPublicCase, readPrivateCase, history	11,8		
11	Agreement				

6.8.2 *Inspire*-like database

Inspire-like ENS is illustrated in Figures 24 and 25. The database, as in *SimpleNS*, is linked to the database in the *Invite* platform through the attributes *side_id*, *negotiation_instance_id,* and *user_id.* The region in light represents the database in *SimpleNS* ENS database and the region in dark represents the *Invite* platform database.

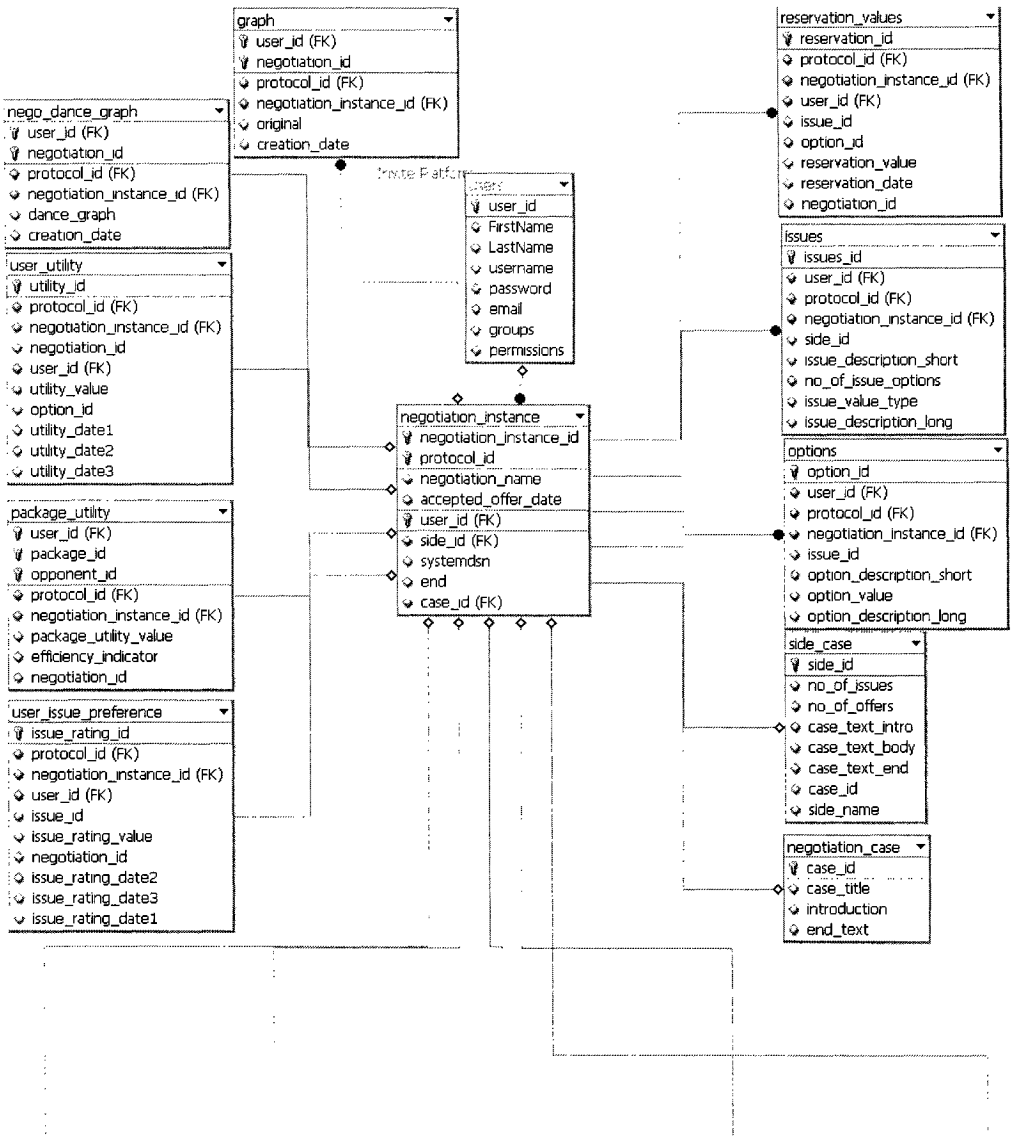

Figure 22. *Inspire*-like database

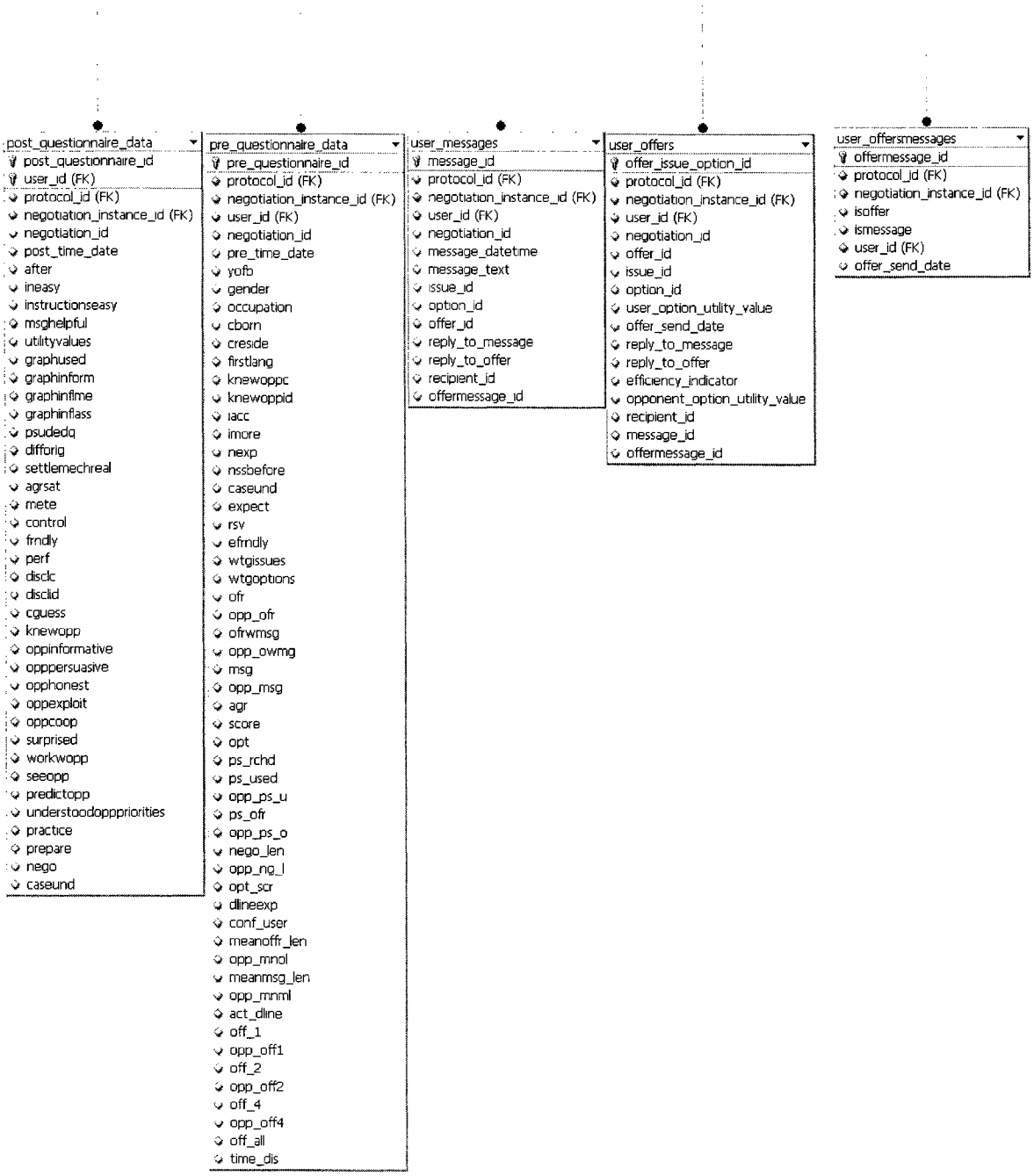

Figure 23. *Inspire*-like database (cont.)

The tables: Pre-questionnaire, Post-questionnaire, Negotiation_Case, and *Side_Case,* are reused from the database of *SimpleNS*. This is because *SimpleNS* and *Inspire* have similar negotiation process.

Table User_Message stores the messages exchanged during the exchange of offers and arguments phase. It includes the message sent, the user who sent the message and the time it was sent. Its primary key is *message_id*.

Table User-Offer stores the offers given during in the exchange of offers and arguments phase. It includes the offer sent, the user who offered, and the time it was done. The offer is either stored as a text message in this table.

Table Issues stores pre-defined *issues* for a negotiation case. It includes attributes such as *issue_description_short*, *issue_description_long*, *issue_value_type*, and *no_of_issue_options*. Attributes i*ssue_description_short* and *issue_description_long* record the short and long format of the issue description. Attribute i*ssue_value_type* stores the type of the *issue*. An issue can be qualitative, categorical, or quantitative. *No_of_issue_options* stores the number of *options* in each issue. Its primary key is *issue_id*, and the attributes *side_id*, *protocol_id*, and *negotiation_instanca_id* are foreign keys.

Table Option stores pre-defined options for issues in a negotiation case. It includes attributes such as *option_description_short*, *option_description_long*, and *option_value*. Attributes o*ption _description_short* and *option_description_long* store the short and long format of the option description. *Option_value* stores the value of the *option*. Its

primary key is *option_id.* Attributes *issue_id*, *side_id*, *protocol_id*, *negotiation_instanca_id* are foreign keys.

Table User_Issue_Preference stores the user preference values over an issue in the issue rating activity. It has three attributes, issue_rating_date1, issue_rating_date2, issue_rating_date3, to store the date of a change in issue preference value from the user.

Table User_Option_Preference stores the user preference values over an option in the option rating activity. It has three attributes, *option _rating_date1*, *option _rating_date2*, *option _rating_date3*, to store the date of a change in option preference value from the user.

Table Package_Utility stores the user preference value over a package in the package rating activity. It has three fields, *package_rating_date1*, *package_rating_date2*, *package_rating_date3*, to store the date of a change in package preference value from the user.

Table Graph stores the directory path of the graph that shows the history of the exchange of offer and counteroffer of a party.

Table Nego_Dance_Graph stores the directory path of the *negotiation dance graph,* a graph which marks the changes in the rating accepted by each negotiator *that* shows the history of the exchange of offer and counteroffer of both parties.

6.8.3 *Inspire*-like components and page composers

Inspire-like page composers include not only all *SimpleNS-like page composers* but also additional *page composers* that serve as analytical tools for the negotiators.

The *ReadPublicCase page composer* displays general information of a negotiation case that is shared by all negotiation parties.

The *ReadPrivateCase page composer* displays private information of a negotiation case that is given to a particular negotiation party. This information is not shared by other negotiation parties.

The *Issue rating page composer* displays a table with pre-defined negotiation issues. There is a textbox next to each issue that a user can use to input his preferred issue. The negotiator has to distribute 100 points among all issues. Whenever a negotiation enters or leaves an issue rating textbox, the distribution rating point calculator shows, at the bottom of the rating table, the remaining points to be distributed. A submit button is located below the rating table. Once the negotiator submits her rating, a JavaScript checks whether the textbox fields are empty, negative, or include non-numeric values, and the sum of all ratings is different than 100. If any of the above conditions is true, the page displays an error message to the negotiator. The Javascript prevents negotiators to submit faulty information. When the server receives the issue ratings from the user, it proceeds to update the appropriate fields in the database. This *page composer* uses the following *components*: *dsp_issueRating, act_issueRating*, *dsp_distributionRatingCalculator*, and *qry_issueRating.*

The *Option rating page composer* displays a table with pre-defined negotiation options. There is a textbox next to each option which a user can use to enter the value preferred for that particular option. The negotiator has to distribute 100 points among all the options related to an issue. Whenever a negotiation enters or leaves an option rating textbox, the distribution rating point calculator shows, at the bottom of the rating table, the remaining points to be distributed. A submit button is located below the rating table. Once the negotiator submits her rating, a JavaScript checks whether the textbox fields are empty, negative, or non-numeric values, and the sum of all the ratings is different from 100. If any of the conditions above is true, the page displays an error message to the negotiator. When the server receives the option ratings from the user, it proceeds to update the appropriate attributes in the database. This *page composer* uses the following *component: dsp_optionRating*, *act_optionRating*, *dsp_distributionRatingCalculator*, and *qry_optionRating.*

The *Package rating page composer* displays a selected set of packages. A package is a combination of all options in an issue. The total number of all packages grows exponentially in the number of issues and options. For that reason, only selected packages are shown in the page. The selection of the packages is done by a java class. The program represents the option with 1 and 0 in a matrix, where 1 means an option is selected and 0 means otherwise. The program generates a square matrix in such a way that the number of packages selected is equal to the sum of the number of options in all the issues. For instance, if a negotiation case has 4 issues and each issue has 3 options, the script generates 12 packages. A square matrix is required in order to compute the

hybrid conjoint analysis process that required using the least square method. The selected packages are displayed along with their respective rating. This package rating displays initially in the *page composer* is calculated by the sum of the option rating taken from the issue and option rating. Then, the negotiator has an option to modify the initial rating of a package. Once the negotiator submits her rating, a JavaScript checks whether the textbox fields are empty, negative, or have non-numeric values. If any of these conditions is true, the page displays an error message to the negotiator. When the server receives the package ratings from the user, it proceeds to readjust the option rating based on the *hybrid conjoint analysis process* and proceeds to update the appropriate attributes in the database. This *page composer* uses the following components: *dsp_packageRating*, *act_packageRating*, *act_matrixGenerator*, *act_hybridAnalysis*, and *qry_packageRating*.

The *Pre-questionnaire page composer* displays the content of a questionnaire in a form. It then submits the user input to the verification *component* to check for errors. If an error occurs, the system sends an error message to the user. If the input information is validated, the system proceeds to insert it into the database. The output of this *page composer* is the success or failure flag. This *page composer* uses the following components: *dsp_prequestionnaire*, *act_prequestionnaire*, *act_verifyprequestionaire* and *qry_prequestionaire*.

The *Send offer-message page composer* displays 2 text boxes, one for composing messages and the other for composing offers. Once the message and/or offer are submitted, it proceeds to verify whether the text is empty. If that is the case, it returns with an error. If there is no error, it passes the data to a component that inserts the data

into the database. The output of this *page composer* is the success flag or the failure. This composer also uses *agreement* and *temrinateNego* components shown as buttons. The agreement is triggered when either side of the negotiation accepts the offer or counter-offer by the counterpart. Once the agreement button is pressed, the negotiation is terminated. At this point, no other offers can be exchanged. The *teminateNego* component is activated if either negotiation party decides to terminate the negotiation before reaching an agreement. This page uses the following components: *dsp_formHeader, dsp_sendMessage, dsp_sendOffer, dsp_formButton, dsp_formFooter, act_verifySendMessage, act_verifySendOffer, qry_InsertMessageOffer, dsp_agreement, dsp_terminateNego, act_agreement,* and *act_terminateNego*

The Post-questionnaire page composer displays the content of the post-questionnaire in a form. It then submits the user input to the verification component to check for errors. If an error occurs, the system sends an error message to the user. If the input information is validated, the system proceeds to insert it into the database. The output of this *page composer* is the success or failure flag. This *page composer* uses the following components: *dsp_postquestionnaire, act_postquestionnaire, act_verifypostquestionaire,* and *qry_postquestionaire.*

The History page composer displays the history of exchange messages and offers in the chronological order. *Inspire-like* history *page composer* has a graphical output of the offers and counter-offers made by the negotiator. The graphical output has two-axis charts: the vertical axis displays the user preference value of a package and the horizontal axis displays the time it was submitted. The *components* of this *page composer includes*:

dsp_history, *ac_ graph*, *dsp_graph*, and *qry_history*.

7. Example of *Invite* Negotiation

In this chapter, we describe an example of *SimpleNS-like* ENS in *Invite.* Two users negotiate for Millennium and Halcion. Millennium represents an insurance company and Halcion represents an advertising company. Halcion is seeking compensation for the losses of not being able to backup its clients' data because Halcion's server went down for four days.

Halcion accesses *Invite* from the login page described in Figure 26. She inputs the pre-assigned username and password.

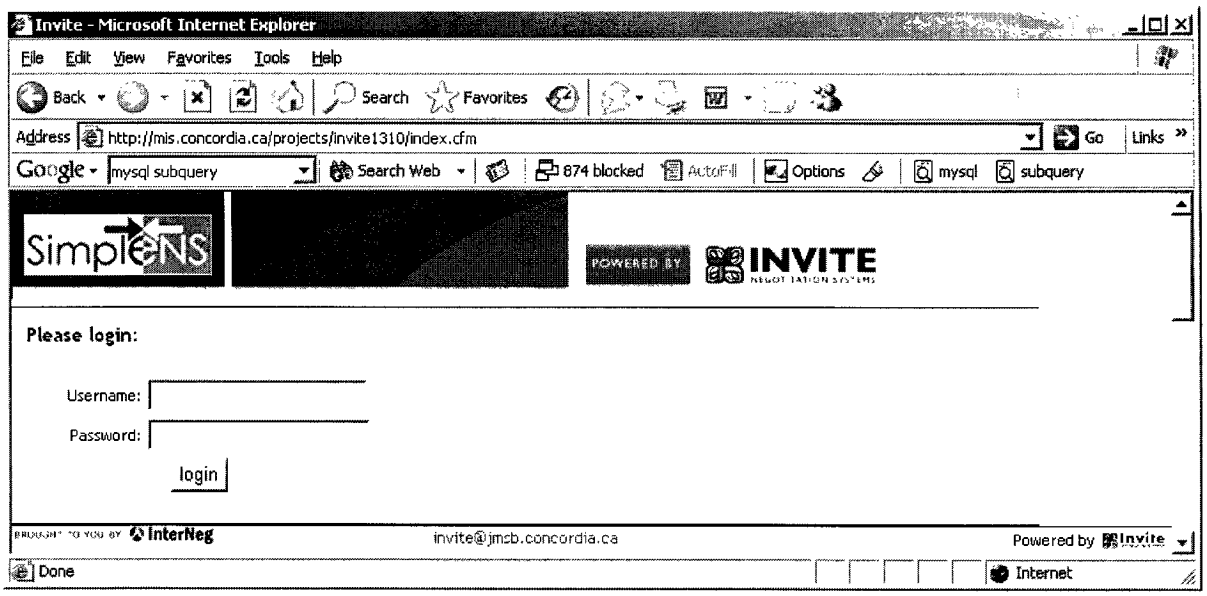

Figure 24. *SimpleNS* login page

Once Halcion is authenticated to access *Invite*, the system acknowledges that she is using *SimpleNS* as shown in Figure 27. If she has two or more negotiation instances currently assigned to her, a list of assigned ENSs would be shown in the drop down menu.

However, if she has no current negotiation instances, the system would not display any ENS names in the drop-down menu.

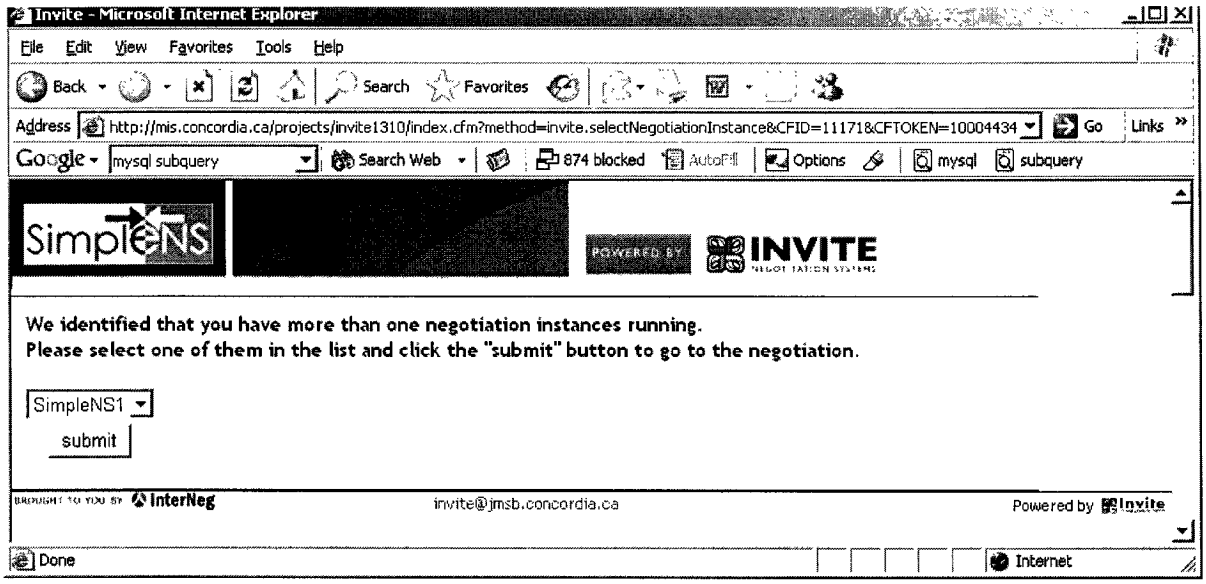

Figure 25. ***SimpleNS*** **negotiation selection page**

After she selects the ENS, *Invite* presents to her the current negotiation status as shown in Figure 28. The negotiation status describes the negotiation process. In *SimpleNS*, there are three phases: Negotiation Preparation, Offer Exchange, and Agreement. Each phase may also contain activities. For example, General Information and Private Information are sub-tasks in the Negotiation Preparation phase.

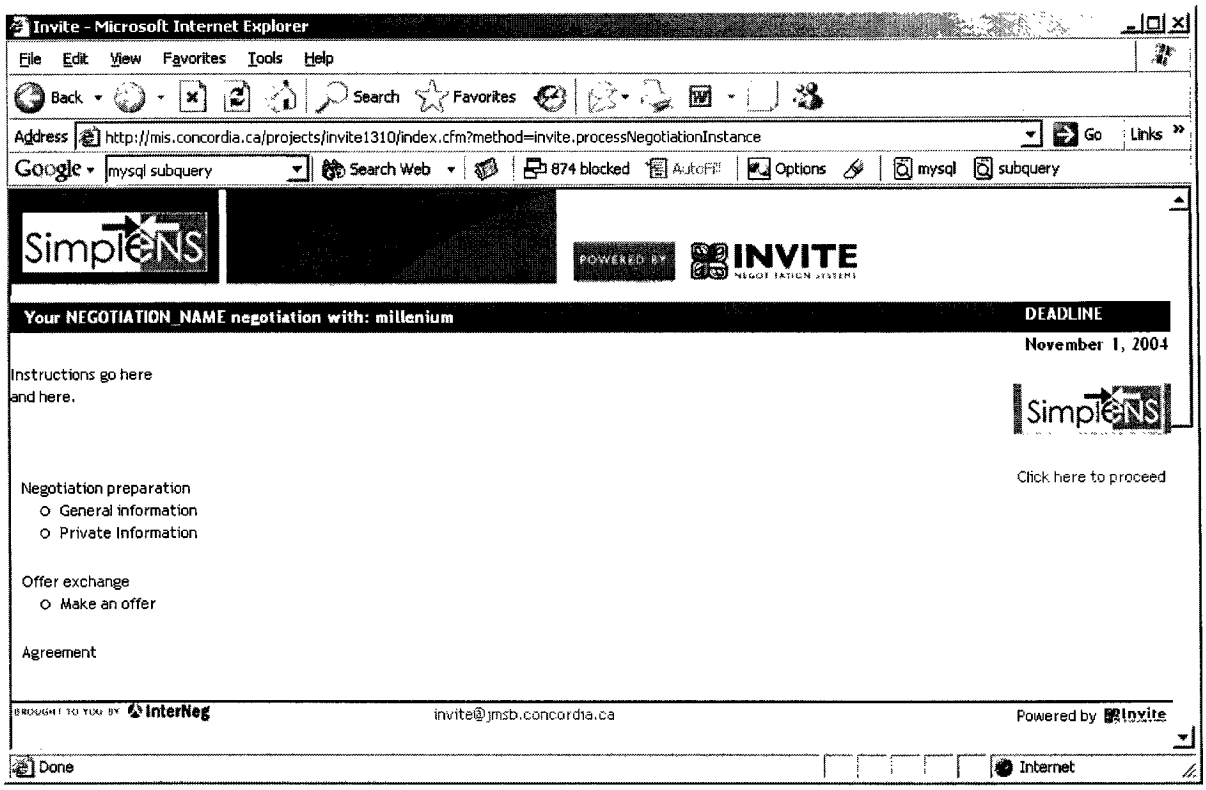

Figure 26. ***SimpleNS*** **status page**

On the next screen, Halcion is presented the *Public Case*, the first *ENS state* in the negotiation illustrated in Figure 29. *Public Case* presents the general information of the current problem between the negotiators. This information is shared by both parties. On the right-hand-side of the screen, links to other *ENS state* are shown. However, at this moment, Halcion cannot send any offer to Millennium yet because according to the protocol in *SimpleNS*, the negotiator needs to read the *Public Case* and the *Private Case* before she can send an offer. Halcion now moves to *Private Case*.

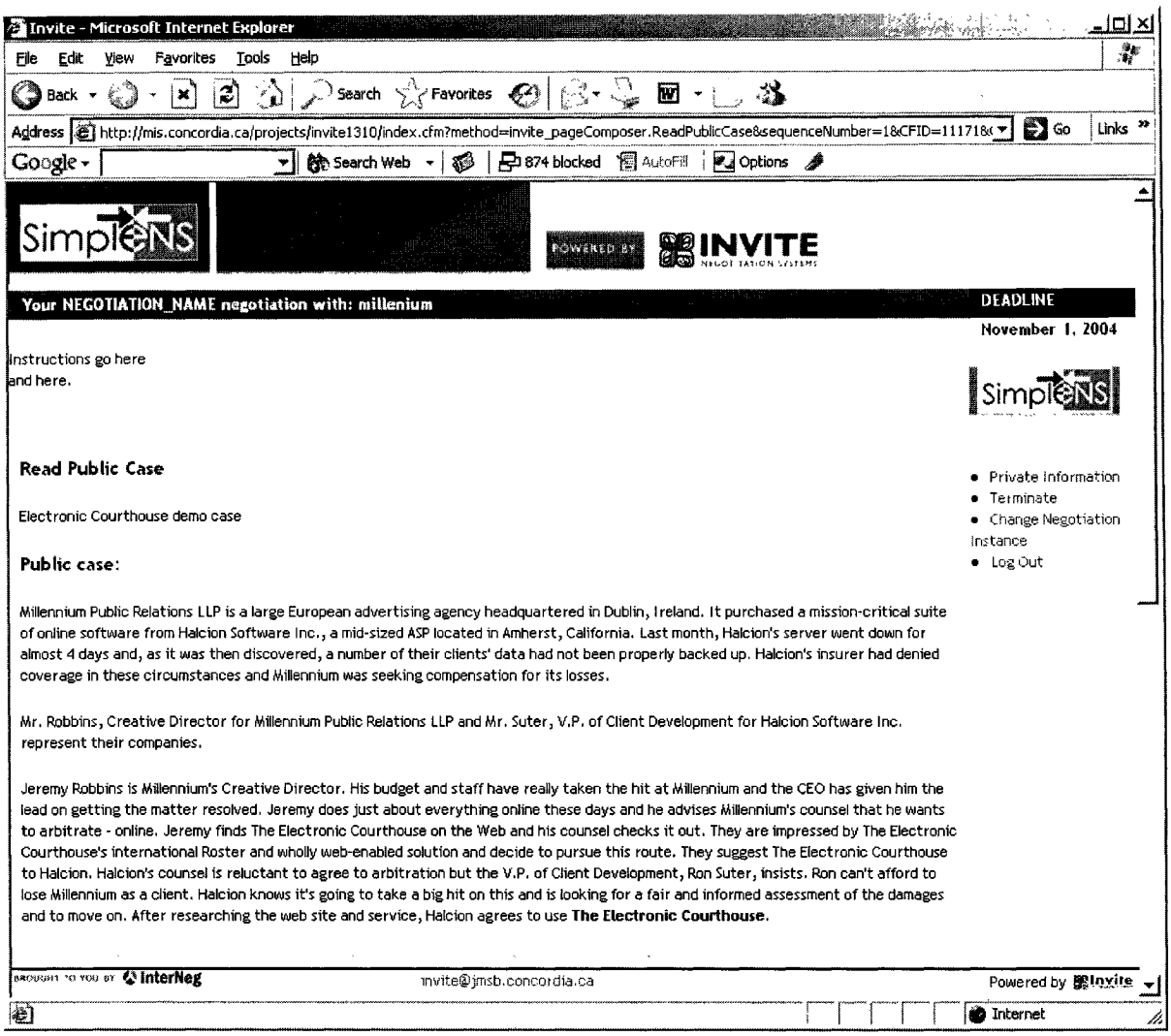

Figure 27. *SimpleNS* public case page

In the *Private Case activity*, private information of the problem, shown in Figure 30, is presented to Halcion. This information is not shared with her counterpart, Millennium. As Halcion moves to this *ENS state*, *Invite* adds two extra links to the menu: *Send Message or Offer*, and *View History*. The generation of these links are triggered by the protocol. Halcion moves to *Send Message or Offer ENS state*.

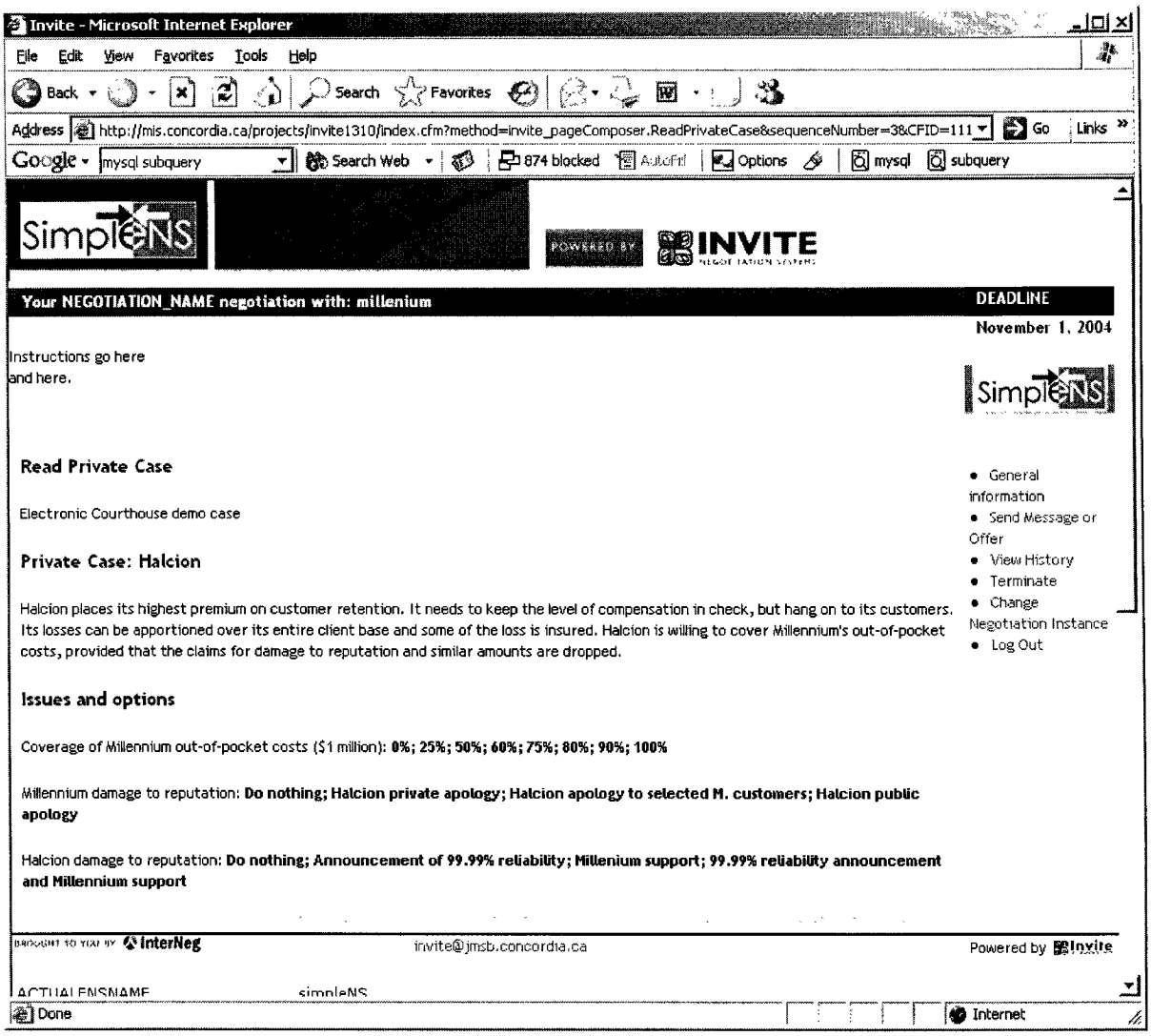

Figure 28. *SimpleNS* private case page

In the *Send Message or Offer*, shown in Figure 31, a form and two text areas, *Message*, and *Offer*, are presented to Halcion. She introduces herself to Millennium in the *Message*, text area on the left and constructs her first demand to Millennium in the *Offer*, text area on the right. She submits the first message and offer. A confirmation of the submission is displayed in Figure 32.

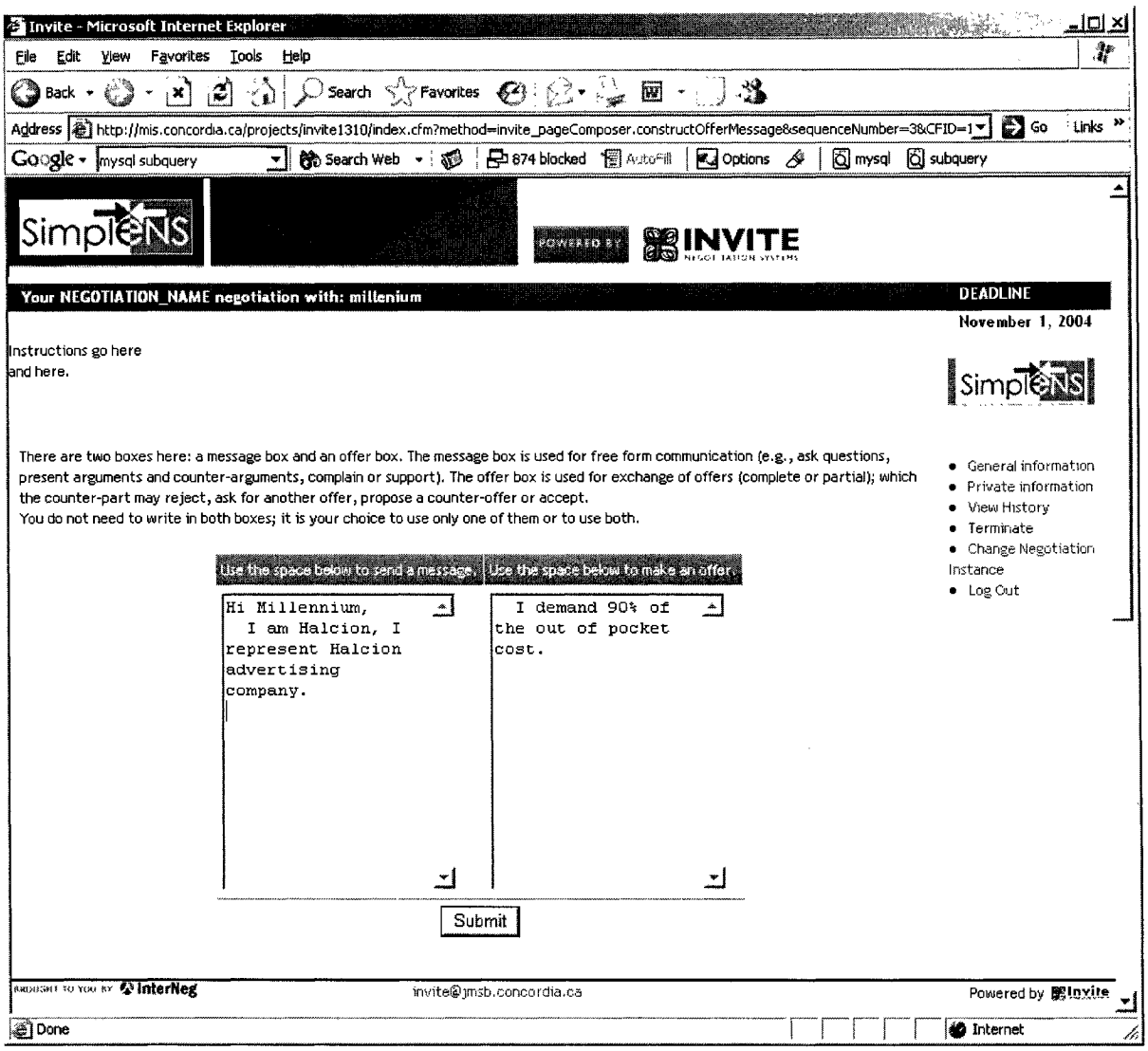

Figure 29. *SimpleNS* send offer and message page

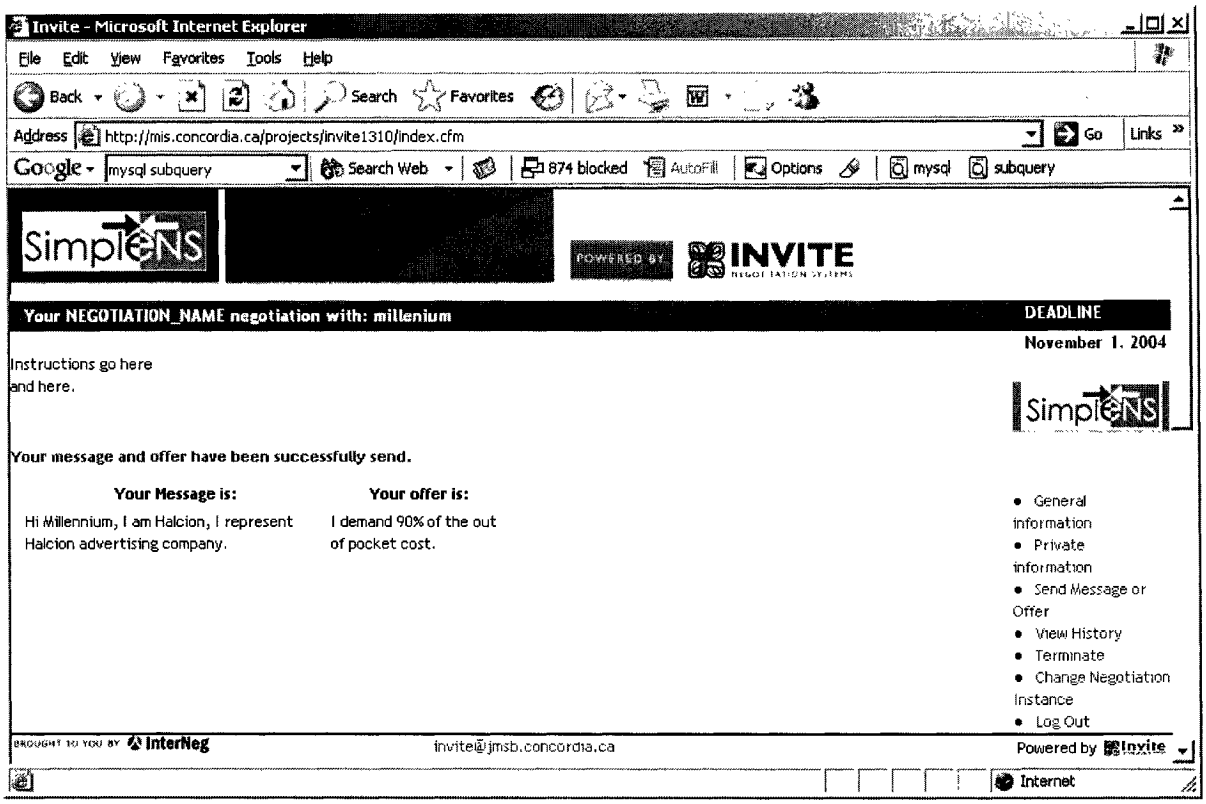

Figure 30. ***SimpleNS*** **send offer and message confirmation page**

At this point, Millennium logs into *Invite*. *Intervening event* from Halcion is triggered because Halcion sent an offer and message to Millennium. There are two scenarios which could happen. (1) If Millennium is prior the *Send Offer and Message ENS state*, *Invite* notifies Millennium that she received an offer and/or message from her counterpart; however, she cannot read them before she completes the current *ENS state*. This is shown in Figure 33. Millennium has to finish *Public Case* and *Private Case* before she can read Halcion's offer and message. (2) If Millennium is currently in the *Send Offer and Message* activity, Invite presents to her Halcion's offer and notifies her. This is shown in Figure 34.

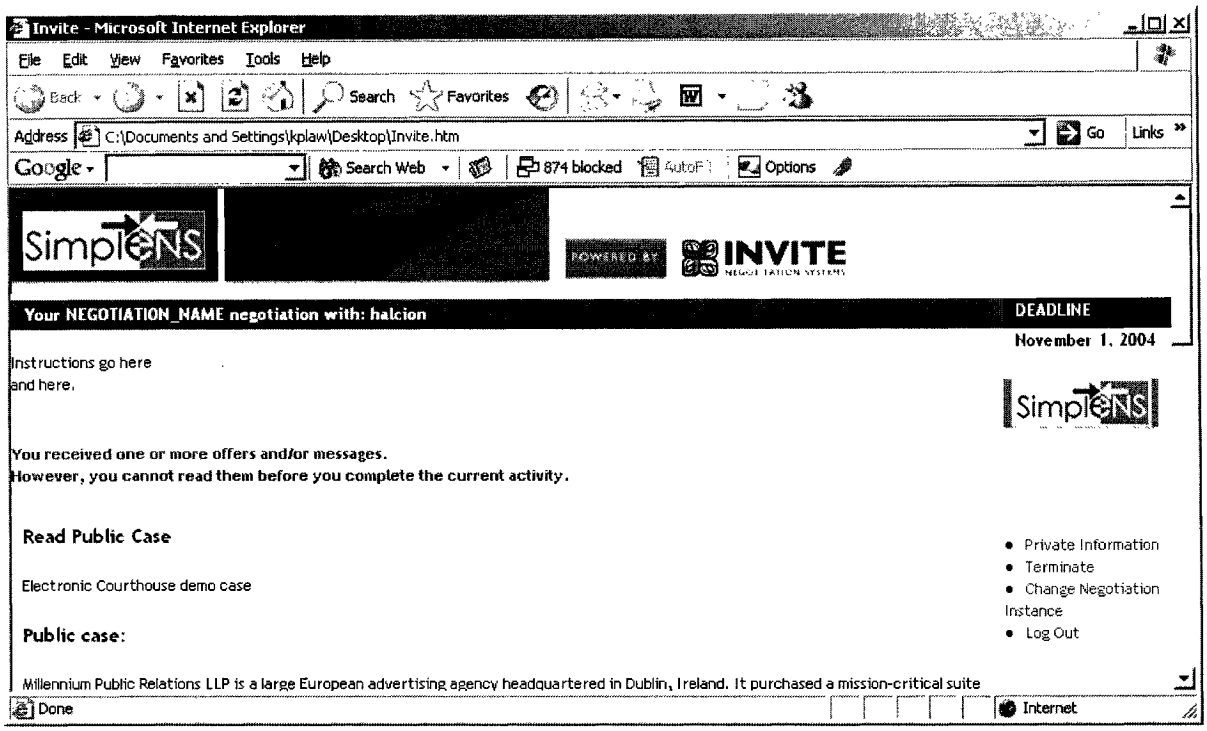

Figure 31. *SimpleNS* intervening event 1

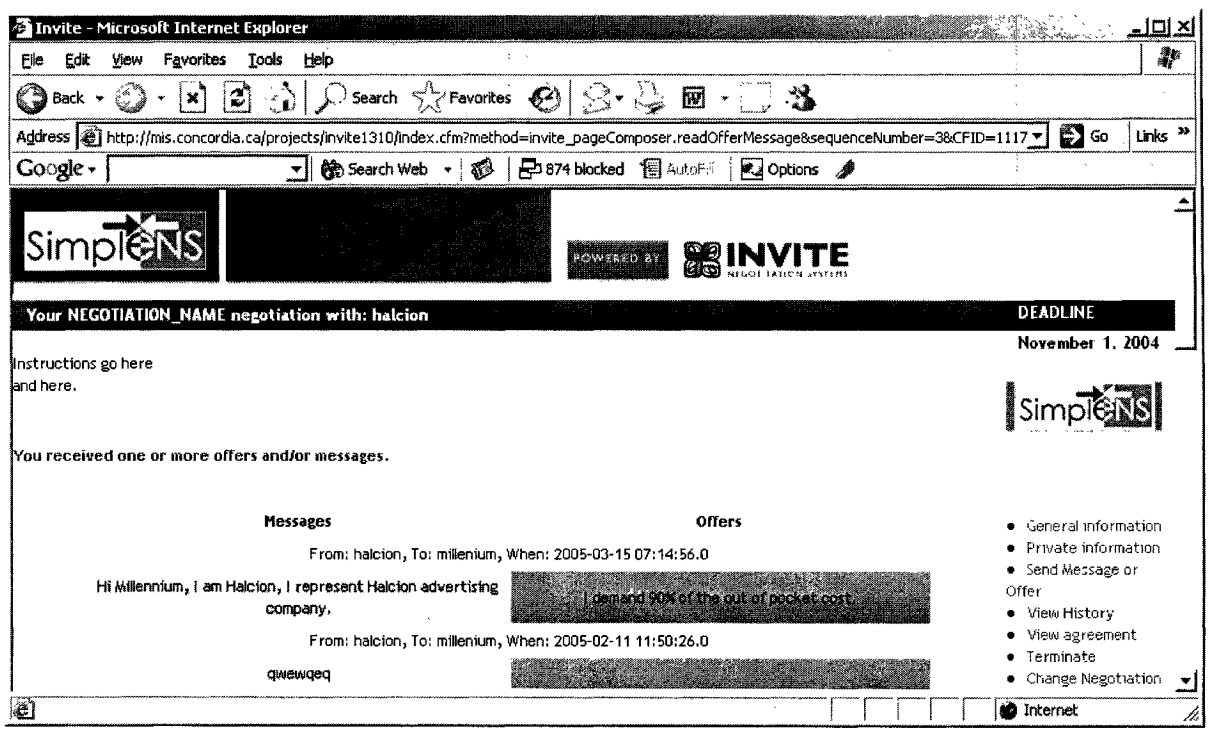

Figure 32. *SimpleNS* intervening event 2

After Millennium reads Halcion's offer. Millennium decides to terminate the negotiation. Both negotiators are moved to the *Termination ENS state* shown in Figure 35. *Termination* is *intervening event*. When Millennium triggers *Termination*, *Invite* automatically moves Halcion to *Termination ENS state* as well.

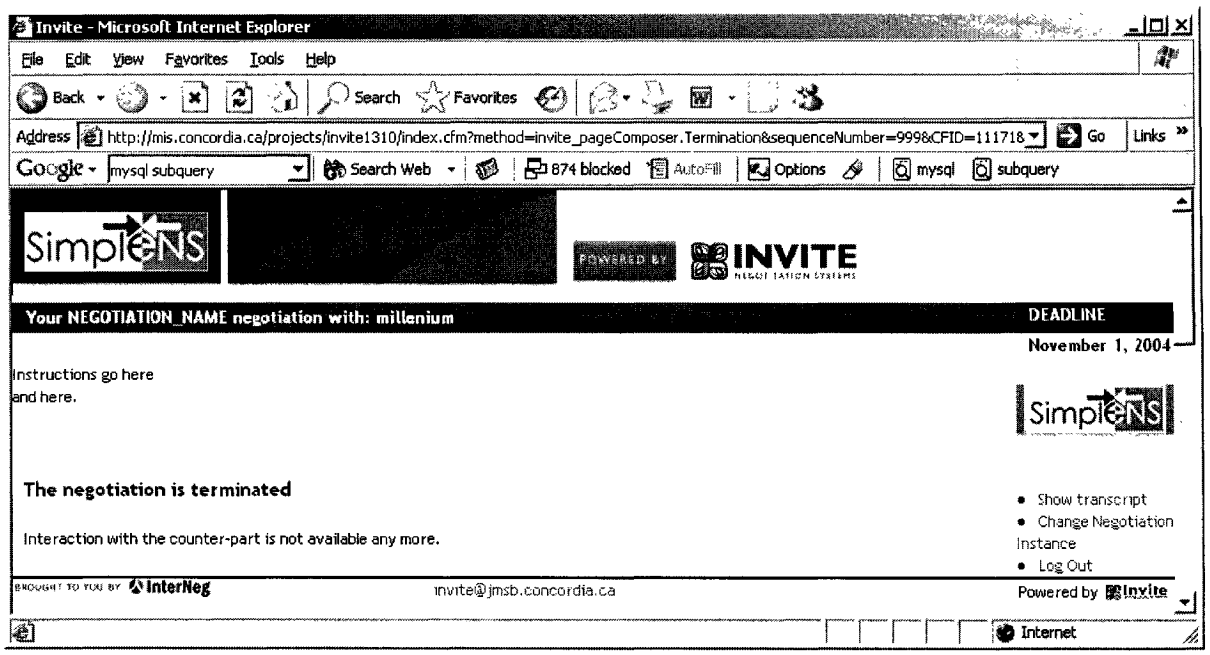

Figure 33. Termination

8. Conclusion and Future Work

8.1 Conclusion

In the conclusion of Chapter 1, a postulate was made that both real-life negotiations and experimental studies require a software platform which would be capable of hosting many ENSs so that whenever negotiation software needs to be built, we could pick and choose components and assemble them like *Lego* blocks. The research described here is an attempt to prove that such a concept is possible; that indeed one can build an ENS from earlier constructed components. We have shown that building such a system requires a detailed plan which describes the role and place of every component in the negotiation process. This plan is the negotiation protocol.

In Chapter 2 on related work, we discussed the fact that the existing ENSs do not provide enough flexibility. They have been proposed to mainly serve a single and fixed protocol. When comparing different ENSs' features, the use of multiple systems is not practical because their different graphical user interfaces may disturb users' perspectives on using a particular feature. This makes it difficult to isolate a particular feature of different ENSs to conduct a comparison. Our goal in this thesis has been to overcome those limitations, and thus a multi-protocol negotiation platform *Invite* has been proposed. *Invite* is capable of running multiple and different protocols concurrently. In addition, the components of *Invite* can be easily reused in different protocols.

The main contributions of this thesis can be summarized as follows.

This research established a link between negotiation processes comprising phases and activities and e-negotiation protocols. Negotiation process models are well known but ill-defined. They identify main activities, which people undertake, and group them into phases. The use of software and the division of labour between the negotiators and software requires formal and detailed representation of all the activities and their relationships.

The representation of negotiation processes has been achieved by breaking the process into sequences, negotiation states and transitions between them. Through the decoupling of individual ENS user activities we achieved higher degree of flexibility, because the users do not need to coordinate their activities. This, however, required that the coordination be undertaken by software. In effect we had to revise the theory behind the interaction between negotiators by adding intervening events.

The high level representation of protocols was implemented as a set of rules into a database schema. A prototype system was designed and implemented in MySQL database. Apart from the platform database, each of the ENSs running under *Invite* has its own database to store negotiators' negotiation data. This approach makes addition of new ENSs simpler.

Invite, a multi-protocol negotiation platform, was coded based on the database schema. Both the content and flow, which governs the interaction between users and the system, are database driven. The reuse of coding was carried out by taking advantage of Fusebox

framework and methodology of building web-based applications.

The replication of the two existing ENSs, *Inspire* and *SimpleNS*, under *Invite* was developed as a proof of concept for the use of multi-protocol negotiation platform. These systems have their own negotiation protocols, yet they share some of the components they use.

In summary, this project involved:

1. Conceptualization of the principles of a software platform for e-negotiations;
2. Relating the structure of face-to-face negotiation processes and the interactions between its components to the software requirements and interactions mediated by software;
3. Formulating the principles of e-negotiation protocols and representing them as database structures;
4. Proposing an e-negotiation platform architecture and specification page composers, platform components and the *Invite ENS* components;
5. Implementing and testing the *Invite* platform;
6. Formulating two e-negotiation protocols based on *SimpleNS* and *Inspire*, and designing and implementing the *ENS* components for these protocols; and
7. Testing the *Invite* platform with its two *ENSs* systems.

The result of this project is *Invite*, an experimental e-negotiation platform capable of hosting many e-negotiations, which are based on different protocols. Recently, this platform has been modified to support multiple e-negotiations processes conducted simultaneously. The modifications also included setting-up negotiations, user registration, and the consideration of privacy and security issues. While these changes are necessary for the use of the *Invite* platform, the platform itself and its page composers and components are being used in the experimental setting. The result of this work is now being used by researchers. Issues discussed in the following sections will allow for the platform to be tested in real-life e-negotiations.

8.2 Future work

Currently, *Invite* is limited to running bilateral negotiations only, and also restricted to two ENSs running under it. In addition, it is a long and tedious process to set up negotiation instances due to lack of platform administration features. The following section will provide some suggestions for future work based on the current state of *Invite.*

1. Expand support to multilateral negotiations

In order to support more protocols and ENS, Invite needs to be able to support multi-lateral and multi-bilateral negotiations. The design and implementation of *Invite* was meant for running only bilateral negotiations. Our database model and the protocol controller could be improved in several ways.

2. Adding new negotiation systems

While we incorporated two ENSs under *Invite*, more ENSs should be supported, in general. A suggestion would be building an ENS with support of agents. In designing the platform, we did not consider incorporation of intelligent agents to play a role into ENSs. The role of an agent is a question should it be part of an ENS, or whether it is as a mediator or purely a helper for a particular negotiation party only. It requires more work to determine how these roles should be implemented in *Invite* to fit its existing platform database and implementation.

3. Add administration page

Administration page is request in order for end-users to setup easily negotiation instances. Currently, administrators need to manipulate directly the platform database to setup a negotiation instances. This process consists of four steps: First, a record needs to be input in the *User* table to register a new user in the platform. Second, the chosen protocol for the negotiation instance needs to be copied from the repository tables: *initialOptional*, *initialExitPoint*, and *initialSequence* to the *actualExitPoint*, *actualOptional*, and *actualSequence* tables. Third, an entry needs to be inserted in the actualNegotiationInstance to match negotiation parties involved in the instance. Finally, a record needs to be inserted in the actualNegotiationStatus in order to keep track of the negotiator progress.

Administration page will make the setup process faster and easier. It will contain a web interface which can be used by administrators to create, delete, and modify users and

negotiation instances. After administrators input the required information, the setup process can be done by running a SQL script which will automatically generate required records in the database.

4. Tune database query

Our implementation uses MySQL version 4.0.2 which does not support sub-queries. It requires a *loop* function in order to simulate a subquery which slows down the overall execution time of the processes in the system. MySQL version 4.1 supports subqueries. Once the development database server upgrades to version 4.1, we will rewrite current queries, using *loop*s to allow sub-queries

5. Improve the GUI

Invite's GUI has basic functionalities; it requires more work to make the interface more attractive and user friendly. We believe a "good" graphic design will make the users interaction with the negotiation system a more pleasant experience.

6. Add more pre-fabricated *page composers*

Currently, we have built *page composers* for *SimpleNS* and *Inspire*. In order to expand the use and creation of new ENSs under Invite, new *page composers* need to be built. This will make the construction of new ENSs more flexible and convenient when users want to create their own protocols.

References

[BS00] Ben-Natan, R. and O. Sasson, IBM Web Sphere Starter Kit. New York: McGraw Hill, 2000.

[BK00] Benyoucef, M. and R. K.Keller. Towards A Generic E-Negotiation Platform. in Proc. of the Sixth Conf. on Re-Technologies for Information Systems. 2000.

[BKS03] Bichler, M., G. E. Kersten and S. Strecker. "Towards a Structured Design of Electronic Negotiations." Group Decision and Negotiation, 12(4): pp. 311-335, 2003.

[B04] Burgess, G., International Online Training Program On Intractable Conflict. Conflict Research Consortium, University of Colorado, USA, 2004.

[EHK04] Ehtamo, H., R.P. Hämäläinen, and V. Koskinen. An e-learning module on negotiation analysis. in Hawaii International Conference on System Sciences. Hawaii: IEEE Computer Society Press, 2004.

[G79] Gulliver, P.H., Disputes and Negotiations: A Cross-Cultural Perspective. Orlando, FL: Academic Press, 1979.

[HLNP+90] Harel D., Lachover H., Naamad A., Pnueli A., Politi M., Sherman R., ShtullTrauring A., Trakhtenbrot M. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. IEEE Trans. on Software Engineering, 16(4): pp. 403-414, 1990.

[KM78] Keen, P.G.W. and M.S.S. Morton, DSS: An Organizational Perspective., Reading: Addison-Wesley, 1978.

[K98] Kersten, G.E. Negotiation Support Systems and Negotiating Agents. in Modèles et Systèmes Multi-Agents pour la Gestion de l'Environement et des Territoires, Cemagref, ENGREF. Clermont-Ferrand, France, 1998.

[K98] Kersten, G.E., Negotiation Support Systems and Negotiation Agents. Modèles et Systèmes Multi-Agents pour la Gestion de l'Environement et des Territoires Cemagref ENGREF, Clermont-Ferrand, France, pp. 307-316, 1998.

[K93] Kersten, G.E., Rule-based Modeling of Negotiation Processes. Theory and Decisions, 34(2), 1993.

[KL05] Kersten, G.E. and H. Lai. Satisfiability and Completeness of Protocols for Electronic Negotiations. InterNeg Research Paper INR 01/05. European Journal of Operational Research (to appear).

[KNT00] Kersten, G., S.J. Noronha, and J. Teich. Are All E-Commerce Negotiations Auctions? in COOP'2000: Fourth International Conference on the Design of Cooperative Systems. Sophia-Antipolis, France, 2000.

[KSL04] Kersten, G., S. Strecker, and K.P. Law, A Software Platform for Multiprotocol E-negotiation. http://interneg.concordia.ca/enegotiation/resources/reports.html, 2004.

[KSL204] Kersten, G., S. Strecker, and K.P. Law, Protocols for Electronic Negotiation Systems: Theoretical Foundations and Design Issues. EC-Web, pp. 106-115, 2004.

[KMWZ86] Korhonen, P., Moskowtz H., Wallenius J., and Zionts S. An Interactive Approach to Multiple Criteria Optimization with Multiple Decision-Makers. Naval Research Logistics Quarterly, 33: pp.589-602, 1986.

[KS99] Kumar, M. and Stuart I. Feldman., Internet Auctions, in IBM Research Division, T.J. Watson Research Center. 1999.

[JJS87] Jarke, M., M.T. Jelassi, and M.F. Shakun, MEDIATOR: Toward a Negotiation Support System. European Journal of Operational Research, 31(3): pp. 314-334, 1987.

[JF89] Jelassi, M.T. and A. Foroughi, Negotiation Support Systems: An Overview of Design Issues and Existing Software. Decision Support Systems, 5: pp. 167-181, 1989.

[LSM97] Lewicki, R.J., D.M. Saunders, and J.W. Minton, Essentials of Negotiations. Boston, MA: McGraw-Hill, 1997.

[MPKT99] Mahadevan, A., Ponnundurai K., Kersten G., Thomas R. Knowledge Discovery in Databases for Decision Support, in Decision Support Systems for Sustainable Development in Developing Countries, G.E. Kersten, Z. Mikolajuk, and A. Yeh, Editors. Kluwer: Boston, 1999.

[MSKM87] Matwin, S., Szpakowicz S., Kersten G., and Michalowski W. Logic-Based System for Negotiation Support. in Proceedings of the Symposium on Logic Programming. San Francisco: IEEE Computer Society Press, pp.

499-506, 1987.

[P01] Pressman, R.S., Sofware Engineering. A Practitioner's Approach. 5th edition Boston: McGraw Hill, 2001.

[RRM03] Raiffa, H., J. Richardson, and D. Metcalfe, Negotiation Analysis. The Science and Art of Collaborative Decision Making. Cambridge: Harvard University Press, 2003.

[RS97] Rangaswamy, A. and G.R.Shell. Using Computers to Realize Joint Gains in Negotiations: Toward an "Electronic Bargaining Table". Management Science, 43(8): pp. 1147-1163, 1989.

[RWBW89] Rangaswamy, A., Eliasberg J., Burke R., and Wind J. Developing Marketing Expert Systems: An Application to International Negotiations. Journal of Marketing, 53: pp. 24-39, 1989.

[S02] Schoop, M., Electronic Markets for Architects - The Architecture of Electronic Markets. Information Systems Frontiers, 4(3), pp 285-302, Kluwer, 2002.

[SQ00]. Schoop, M. and C. Quix. Towards Effective Negotiation Support in Electronic Marketplaces. in Tenth Annual Workshop on Information Technologies & System (WITS), Brisbane, Australia, 2000.

[S03] Ströbel, M., Engineering Electronic Negotiations. New York: Kluwer, 2003

[S99] Strobel, M., On Auctions as the Negotiation Paradigm of Electronic Markets. IBM Zurich Research Laboratory: Ruschlikon. pp. 7, 1999.

[TKLC+03] Tivadar, J.Q., Kotek B., LeRouz B., Clark S., and Woodin P. Discovering Fusebox 4 with Coldfusion. 2nd edition, Techspedition, 2003.

[U89] Ullman, J.D., Principles of database and knowledge-base systems, volume 2, volume 14 of Principles of Computer Science. Computer Science Press, 1989.

[YDRA99] Yuan, Y., H. Ding, J. B. Rose, and Archer N. "Multi-party Interaction in a Web-Based Negotiation Support System", WebNS, 1999. https://ecomlab.mcmaster.ca/webns/papers/Interact.html, Accessed: April 23, 2004.