Towards a Generic E-Negotiation Platform

Morad Benyoucef, Rudolf K. Keller[*],
Sophie Lamouroux, Jacques Robert, Vincent Trussart[†]

Abstract

To investigate the various research questions concerning e-negotiations, an appropriate software infrastructure is needed. As part of our project on electronic marketplaces, we have developed the Generic Experimentation Engine (GEE). GEE supports game-oriented experimentation and allows for the study of human behavior under various game situations. The more recent Generic Negotiation Platform (GNP) is a more focused version of GEE that will support experimentation with alternative market designs and with various types of negotiations. GNP is being re-engineered from GEE in that important concepts and technologies developed for GEE are carried over into GNP. The paper provides a detailed overview of GEE. The re-engineering of GEE into GNP is addressed by presenting a list of lessons learned from the GEE development and by providing a vision for GNP. Furthermore, the paper addresses the formalization of auction rules, one of the prerequisites for making GNP truly generic.

1 Introduction

The rapid and significant progress of information technologies is profoundly changing the structure of the economy, in particular the way economic negotiations are undertaken and exchanges of goods and services are organized. Electronic marketplaces as nexus of services for the largest possible network of businesses will be at the core of future e-commerce. Negotiations will be supported by open negotiation servers where deals are struck and prices determined, and where evaluation, matching, and advising services are being offered.

The creation and efficient operation of electronic marketplaces raises many challenges. E-commerce should do more than just replicate what is being done today, and gains may only be obtained through re-engineering the way decisions are made and markets operate. In the TEM (Towards Electronic Marketplaces) project, a joint industry-university project started in early 1999, we aim to make the above vision possible. Thus, we address market design issues in respect to resource allocation and control and reward mechanisms, investigate open protocols for electronic marketplaces, and explore concepts and tools for e-negotiations. E-negotiations may be re-engineered from traditional negotiations, or designed expressly for electronic marketplaces.

[*]Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128 Succursalle Centre-ville, Montréal, Québec, H3C 3J7, Canada, {benyouce, keller}@IRO.UMontreal.ca.

[†]Centre Interuniversitaire de Recherche en ANalyse des Organisations (CIRANO), 2020 rue University, 25ème étage, Montréal, Québec, H3A 2A5, Canada, {lamouros, robertj, trussarv}@CIRANO.UMontreal.ca.

As part of the initial phase of the TEM project, we have built GEE (Generic Experimentation Engine), a software prototype for game-oriented experimentation. In its current version, GEE supports continuous double auctions [21], and is fit for the study of human behavior under various game situations. In the recently incepted phase II of the project, we are developing GNP (Generic Negotiation Platform). Whereas GEE is a game-oriented engine, GNP is an auction-oriented platform, that is, GNP will be larger in scope and will support many different types of negotiations (auctions are considered a special case of negotiations). The different types of negotiations are described in a uniform and formal way, and the descriptions can be serialized for exchange in the e-marketplace. From an engineering point of view, GEE is being re-engineered into GNP, which will exhibit a layered architecture based on objects, statechart diagrams [18], and scripts. Our development process is incremental and follows the feature development approach described in [10]. It is the re-engineering at the infrastructure level which is the subject of this paper.

The contributions of this paper are threefold. First, the paper provides a detailed overview of GEE. Second, the re-engineering of GEE into GNP is addressed. This has been done by factoring out and parameterizing the application-independent components of GEE so that new applications can be developed easily and quickly. Our approach is similar to that taken by expert system shells and by high-level programming environments such as 4GL and SQL, yet it has not, to our knowledge, been applied to the e-negotiation domain. Third, the formalization of auction rules is discussed, and the approach adopted in TEM is explained.

In Section 2 of this paper, the requirements for TEM's experimentation engine and negotiation platform are discussed. Section 3 gives an overview of the implementation of GEE. In Section 4, the lessons learned from the GEE development are explained. Then, in Section 5, the need for formalizing auction rules is discussed, and the TEM approach is presented. Section 6 provides our vision of the GNP functionality and architecture. Finally, we wrap up the paper with some concluding remarks.

2 Requirements for experimentation engine and negotiation platform

The level of abstraction required by a generic e-negotiation platform is quite high. The negotiation engine should be able to support a wide variety of disparate negotiation rules and algorithms. The common ground shared by these algorithms is that they can be viewed as an economic game according to game theory.

Game theory, an important branch of micro-economics, provides a mathematical structure to study social interactions among rational agents (forward-looking, bayesian and optimizing). An extensive-form game is an exhaustive description of the social interaction under consideration. In order to be well-defined, a game must specify:

1. the set of players;
2. a sequence of decisions;
3. the precise structure of the information flow;
4. a representation of the players' preferences over the set of all possible outcomes of the game.

Hence, it must specify what each player knows at the beginning of the game, in particular what information is public and what is private; by whom and when each decision is made and what information the decision-makers have when making their decisions; and finally, what are the gains (monetary or otherwise) accruing to the players as a result of the history of the game.

The precise definition of the extensive representation of a game can be found in graduate micro-economic textbooks (see, for instance, [16]-Chapter 7, or [12]-Section 2). In order to implement a game, we can proceed through a finite number of rounds. In each round, one or many participants are provided with some information (in the form of a table or a small text) and are invited to select an action. This generic structure is sufficiently general to implement any finite game. We can implement simultaneous-move games in which players are asked to select their actions simultaneously, or sequential games in which players are informed of past plays and are asked to choose moves one at the time. The key is that we are able to control both the information that participants receive and the set of actions which they are allowed to select.

Our objective was to build a generic experimentation engine that is flexible, so it can implement any conceivable game. In order to do so, GEE's unique responsibility is to orchestrate a sequence of rounds. In each round, including the initial one, each player receives sets of public and private information and of selectable actions. Depending on the specific design of the game, a new round is initiated either when a predefined number of players have submitted their choices or when a specific delay has expired. At the beginning of every round, the sets of public and private information and of selectable actions are updated according to the rules of the game. Finally, at the end of the game, each player receives a score that can eventually be translated into a monetary gain. To help players keep track of how well they are doing, points are added to or subtracted from their score after each round. The flexibility of GEE comes from the fact that it imposes no specific restrictions. How the private and public information, the selectable actions and the gains are initially generated and updated after each round is not part of the GEE design. It is handled as configuration information for the specific instance of the experimentation engine.

Market negotiation games form a special class of economic games. The major characteristic of market negotiation games is that their outcome mainly specifies two things: (1) a final allocation which determines who gets the item or items on trade; and (2) prices or monetary transfers specifying who pays or receives what. The negotiation process includes, depending on the specific design, rounds of offers and counter offers (quotes, bids, asks, etc.) and a closure rule which ultimately leads to an "allocation and prices" outcome.

GEE was developed in order to have an open electronic environment that could reproduce any mechanism involving several players interacting with each other. It was done with two objectives in mind: (1) to be a functional and flexible tool to validate economic theory by making electronic experimentations on small groups of individuals; and (2) as we focus on negotiations, to include some kinds of negotiation rules in GEE to help extract the common concepts found in negotiation processes.

The objective of GNP is to offer a more focused version of GEE that will enable us to experiment with various market designs. Using GNP, it should be easy to create and deploy various types of negotiations. For such easy deployment to be possible, as is already the case with GEE, no client software installation should be required (except for a web browser).

The development of GNP as a platform supporting multiple negotiation algorithms will lead us to formalize and

validate common concepts and similarities and to develop a clear and consistent terminology for negotiations. Such knowledge will help us create an open protocol for negotiation information exchange and a formalized representation of negotiation rules. Eventually, GNP should be powerful and flexible enough to allow further research and development in a wide range of fields, including operations research and decision support systems.

To reach these goals, we adopted an iterative approach based on feature development [10]. The choice of this software engineering process is guided by the fact that our requirements are evolving continuously. Our first prototype, GEE, is an implementation focusing on the creation of games according to game theory principles and on flexibility. GNP is meant to be larger in scope and to address the specifics of negotiations. Finally, concepts and base technologies developed for GEE should be reusable when building GNP.

3 Implementation of GEE

In this section, we present our first prototype of GEE. The section is organized into the subsections user interface, dynamic behavior of games, data management and scripting, and architecture and technology.

3.1 User interface

GEE is an application accessible via the Internet. If GEE is activated and some games are already on, a player newly logged on can choose and participate in these games. A game process is divided into rounds. Arriving at any time, a player will get an HTML page that will not change during the current round. The page is composed of four sections (see *Figure* 1, left). Section 1 contains information about the current round (time, round number, etc.). Section 2 shows information in read-only mode. This can be public information about the state of the game given to all players and/or private information given only to individual players. The information in Section 2 is an HTML fragment generated by a script and inlined at the beginning of a round. Section 3 displays questions in the form of a questionnaire. By completing and sending out the questionnaire, a player participates in the current round. The responses of the players will only be treated at the end of the current round, and feedback will be given at the beginning of the next round, by displaying updated information and new questions in the HTML page. A game is over when no more questions are generated and instead a “final message” is issued to all players. Section 4, finally, is an *icon bar* that provides access to other useful functions, such as help files and transactions history.

Figure 1, right, shows a screenshot taken during one of the rounds of a game. The game in this example is a continuous double auction. The player can submit a bid (buy order) or an ask (sell order). Section 3 lets the player choose to add or delete an order, and to buy or sell, and enter the quantity and the price. Section 2 displays the orders submitted by all the players (orders submitted by the player interacting through the HTML page are tagged with an asterix *****) and the current price (last trade). Furthermore, Section 2 shows the wallet of the player, that is, the number of units and the amount of money of the player. Finally, it also depicts the wallet when the engagements made by the player are taken into consideration, and it displays the details of the last transaction. Sections 1 and 4 are not shown in the figure.

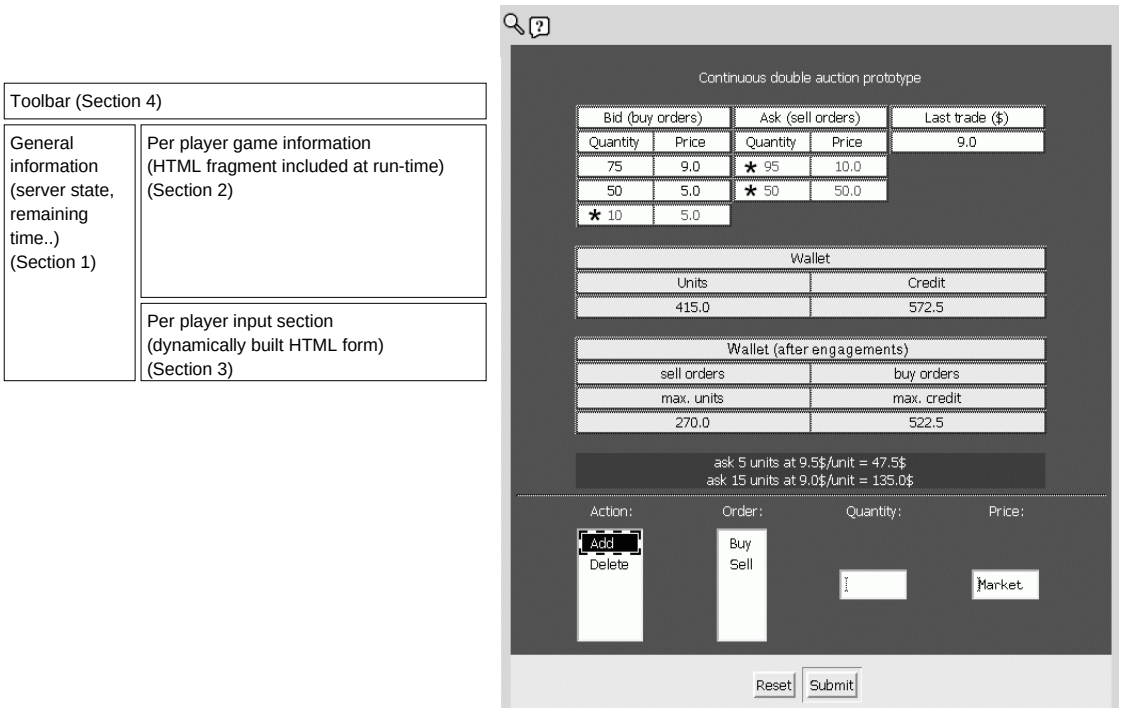

Figure 1: **User interface of GEE: schema (left) and screenshot (right)**

3.2 Dynamic behavior of games

The statechart diagram shown in *Figure* 2 captures the behavior of the games supported by GEE. Games are based on an iterative pattern that we call a round. During a round, the responses sent by the different players are queued. A round ends on a particular event, which can be either the number of responses sent, a timeout, or both. When such an event occurs, a file containing a script program is called which processes the queued responses and generates the messages (information, questions) sent to the players for starting the next round.

Rounds, information, questions, and responses are the basic concepts built into GEE. The game designer can use these concepts in a flexible way, having full control over the way data is computed from one round to the other. We define a game designer as the person who sets up the rules of the game and implements them in a script.

3.3 Data management and scripting

The data of GEE are managed in files and directories. Specifically, GEE is in charge of the output files (information and questions) to be displayed on the player's screen and of the input files (responses) to be written in the directory assigned to the current round. Once an end condition is met, an executable file, that is, a script file written in JPython [6], is called to complete the current round. The script processes the input files produced during the ending round and generates output files into the directory assigned to the next round.

A round's end condition may vary from round to round. End conditions are stored in dedicated files that can be manipulated by the scripts. The activity diagram [18] shown in *Figure* 3 summarizes the life cycle of a script

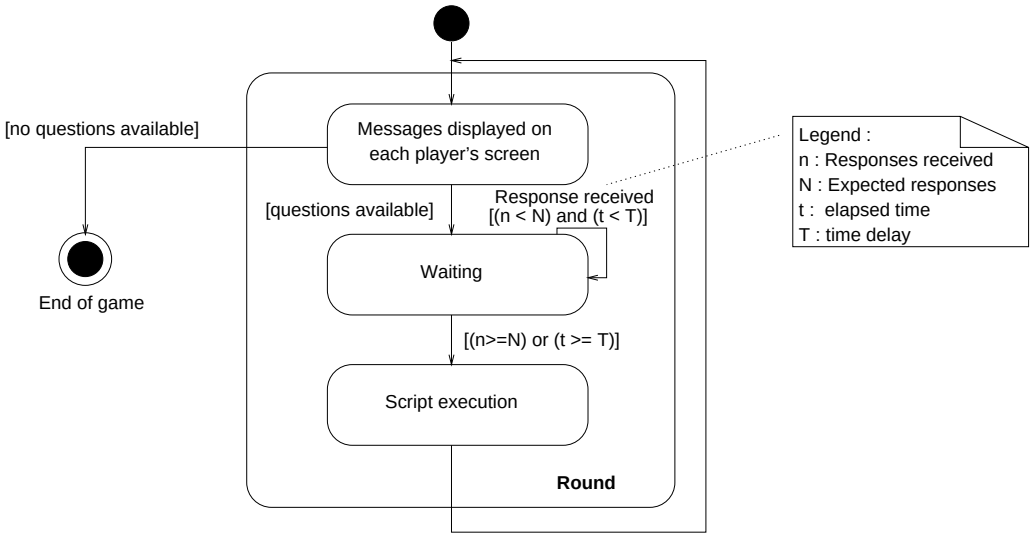

Figure 2: **Statechart diagram of games supported by GEE**

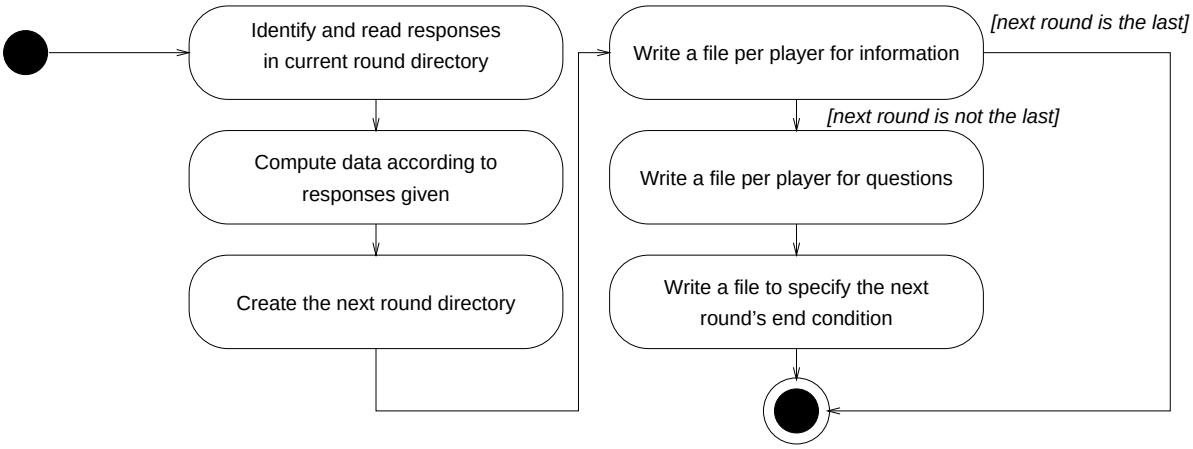

Figure 3: **Activity diagram of GEE script for a given round**

for a given round.

As building a directory hierarchy and scripting are under the control of the game designer, anyone with knowledge of JPython should be able to set up a game with her own rules. During scripting, the game designer has to decide on which round will be the last. In the second but last round, the script should not write any questions to the players, nor specify a round's end condition. The absence of questions will be interpreted by GEE as the end of the game.

3.4 Architecture and technology

To ensure easy deployment, the negotiation engine is built as a *web application* using server-side Java technologies that is accessible in three ways (see *Figure* 4):

- Pull mode: the information is sent to the player's web browser only upon request. In order to keep the

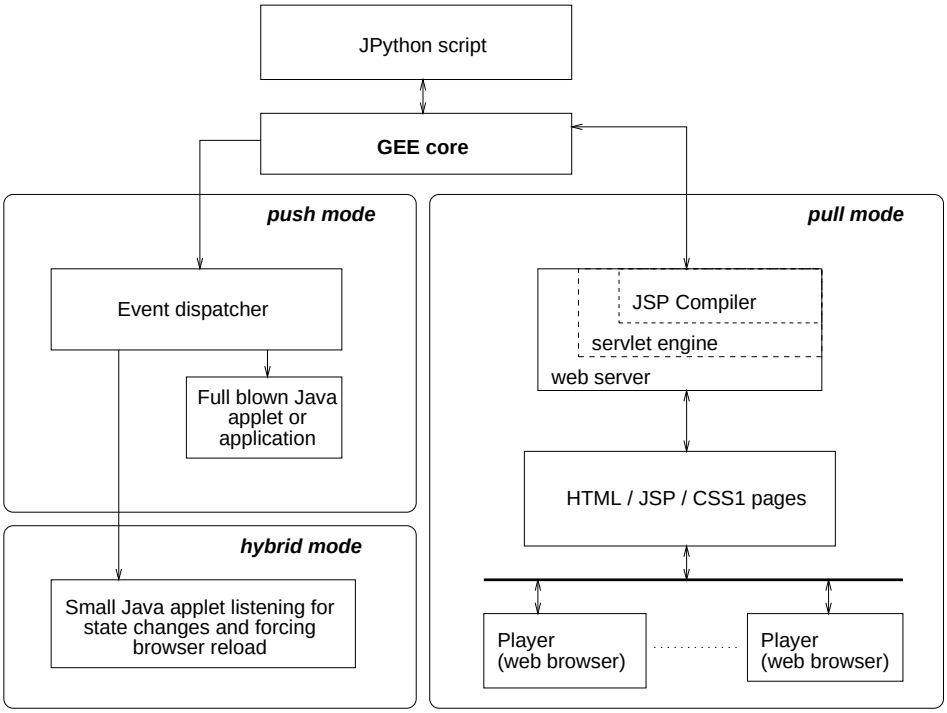

Figure 4: **GEE architecture**

displayed information up-to-date, this mode relies on polling a dynamically generated web page. The pull mode is lightweight; however, it is little responsive and imposes the task of browser reloads onto the player.

- Push mode: in this mode, the information is sent from the server to the client when needed. This allows for a responsive and full-featured user interface at the expense of a heavy download (Java applet) and uncertain reliability (Java 1.1 is currently not well supported in the major web browsers).
- Hybrid mode: this mode sits in between the two previous ones. It relies on a very small Java applet to perform *on demand* web page polling by listening for events on the server (through a TCP/IP socket) and performing a browser reload only when needed.

The selection of the access mode is a task performed at deployment time by the operator of GEE. The preferred access mode is the hybrid mode because of its lightness, responsiveness, easy development and customizability.

The web pages for the players are generated dynamically (cf. *Figure* 1). Since these pages are coded using the *Java Server Pages* (JSP) specification [4], multiple deployment alternatives are provided.

4 Lessons learned from GEE development

The GEE prototype is currently being used for experimentation of economic games. Our development experience and the ongoing field tests have shed light onto the strengths and weaknesses of the design and implementation of GEE.

In this section, we present some of the shortcomings of GEE and how we intend to overcome them in GNP. We introduce the platform services that will be provided, a new approach to time management, a negotiation toolkit that will be made available, and a high level interface for easing the task of the game designers.

4.1 Platform services

GEE only provides basic error handling. In GNP, we envision pervasive error handling in respect to the players, the game designer, and the person operating GNP. Furthermore, a registration tool to regulate and manage access to GNP and to customize negotiation rules according to the profiles of players will be provided. Finally, persistence management in GEE is done manually by the game designer. In GNP it will be a built-in feature supported by appropriate technology. This will allow recovery after game failures and for tracing for recording and analysis purposes.

4.2 Fine-grained time management

Time in GEE is managed merely at the round level, limiting every round with a time delay. As time is a decisive factor in negotiation processes, it should be dealt with in a more flexible way. First, a delay should limit the overall duration of a negotiation. Second, this overall delay should be divided into several limited time periods. Those periods would represent different phases of a negotiation process, e.g., a number of consecutive rounds, in which the same rules could be applied. Such fine-grained time management would allow for the more precise definition of negotiation processes. Moreover, relative time delays as well as absolute dates should be supported.

4.3 Negotiation toolkit

The architecture of GEE is independent of any specific negotiation type, with data files and scripts being the only entities describing the rules of the games. In complex games and negotiation processes, the size of the files and scripts may become rather large. Especially, scripts may become difficult to write and maintain. Since the scripts are written in JPython, we were naturally inclined to build many generic reusable classes, persistent objects, and files for use in different negotiation processes. A sample of these is:

- initial data of a negotiation: announce;
- information sent to all players: market quote;
- information sent to some players: private message;
- choices that the players make in each round: bid;
- matching of bids and asks according to negotiation rules: transaction.

Whereas in GEE this collection of classes was built in an ad hoc manner, GNP will provide them from the outset organized as a toolkit.

4.4 High-level interface for game designers

In GEE, a game designer who wants to implement a game has to manage all the data and is responsible for writing the scripts. In the case of negotiations, she would need substantial knowledge of both programming and economic market design. Even if a powerful, simple and easy to learn language such as JPython is used, programming may become a burden. A good knowledge of market design should be the only prerequisite for effectively using GNP.

We have identified a number of operations that are common to different negotiation processes. Some of these operations are:

- define attributes and default values for the formalized concepts;
- setup the end conditions for rounds, phases and the whole negotiation;
- define the information to be displayed to or hidden from the players.

Such operations may be provided to the game designer as a high-level interface. In the medium-run, however, we want to free the game designer from directly dealing with this high-level interface and any negotiation toolkit. To this end, we envision to leverage the formal descriptions of auction rules, as described in the section below.

5 Description of auction rules

5.1 Need for formalization

If we are to make buying and selling decisions based on automated auctions, then it is important that we have high confidence in the software involved in this activity. Cass [9] suggests that an automated negotiation must have four necessary properties. It must be:

- Correct: an automated negotiation must not contain errors such as deadlocks or incorrect handling of exceptions;
- Reasonable: it must allow adequate time for bids to be made (or for decisions to be made in general);
- Robust: it should continue to function properly after an improper action by the user;
- Fast: it has to execute and respond quickly.

We believe that there should be a fifth property. An automated negotiation must also be traceable. In order to be trusted, the software must be able to justify its actions to the user if necessary. This could be done by tracing the execution and by showing the decisions taken together with their rationale [11].

In order to achieve these properties, we need to formally describe negotiation processes. To our knowledge, this issue has not been explored yet in any depth. In the literature, we found few papers addressing such formalization. Typically, natural language is used to describe the negotiation and auction types under consideration. Some papers try to classify them while others try to find similarities so that common parts can be isolated [8]. We agree with Cass [9] that "a notation with well-defined, well-formed semantics can facilitate verification of desirable properties".

Another issue is the need to separate the process of negotiation from the other parts of the software. We believe that the rules governing the negotiation should not be hardcoded. The software tool supporting the negotiation should be able to pick one type (style) of negotiation from a repository and use it. Separation permits efficient implementation, easy testing and, last but not least, encourages reuse. We can actually benefit from the fact that negotiation processes contain parts that are common to all of them.

Another reason we need a formal way to describe the auctions is that the auction rules should be known to the user (human or software). They should be available to anyone who wants to consult them before engaging in the auction. We think that the rules are as important as, or perhaps even more, than the other information describing the good or service that is the object of the auction. We agree with Wurman [22] that "implementing auction mechanisms in a larger context will require tools to aid agents in finding appropriate auctions, inform them about auction rules, and perform many other market facilitation functions".

5.2 Defining a formalism for auction rules

We need a mechanism that enables us to formally describe the rules governing an auction, visualize this description when necessary and serialize it in order to transfer it over the network. A review of the formal methods used to describe auctions can be found in [8].

Wurman et al. [22, 21, 19, 5] categorize auctions as either single or double, sealed-bid or open-cry, and ascending or descending. They use messages and activities to describe an auction. The three main activities are: receive-bid, clear, and supply-information. Parameters are used to complete this description. These parameters indicate for example if a new bid should be greater than the previous one, what are the closing conditions, etc. Natural language is used to describe the auctions.

Kumar and Feldman [13, 14] use a Finite State Machine (FSM) to model an auction. The states of the FSM are the states of the auction. Its input alphabet is the set of messages that can be sent by the participants. A message is expressed as a pair < p , m > where p is the sender of the message and m is the message sent out. The output alphabet is the set of messages sent to the participants. These messages are expressed as pairs << p , m >> where p is the subset of all the participants that will receive the message and m is the message itself. The process flow of the auction maps into the transitions of the FSM. The messages make the auction go from one state to another.

The FSM alone is not sufficient to describe an auction process. Therefore, the description is fine-tuned according to the following policy decisions: anonymity, restriction, rules for closing the auction, and services provided to the participants.

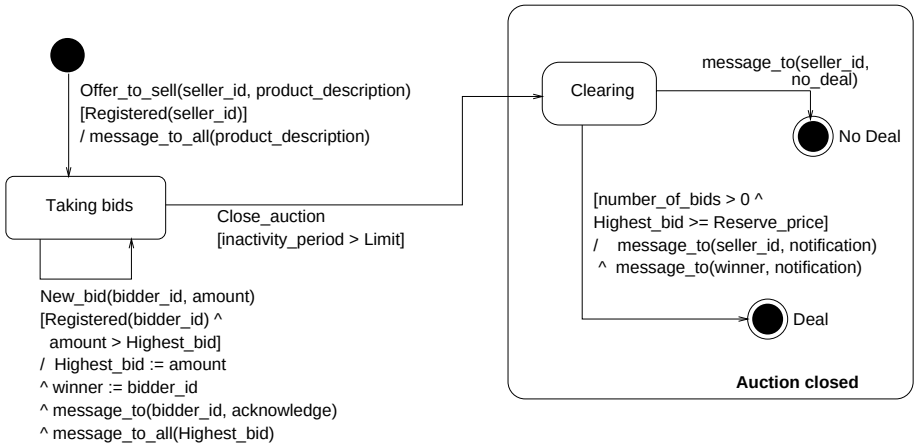

Figure 5: **Statechart diagram of an English auction**

Little-JIL [20] is a graphical agent coordination language realized at the University of Massachusetts at Amherst. A program in Little-JIL has a root that represents the entire process. The root itself is decomposed into steps that describe the process. Visually a step is a rectangle with many tags associated to it (the name of the step, the resources necessary to execute it, etc.). The execution of a step and its sub-steps is controlled by the step's sequencing (sequential, parallel, conditional, etc). Little-JIL was used to describe a set of auction processes [9]. This way of visualizing an auction process simplifies reasoning by following the flow of control and the flow of data in the tree of steps. Since an auction involves multiple participants, Little-JIL may be used to decompose the auction into steps to be carried out by the participants and by providing coordination and communication between them. This notation is unfortunately quite exotic, and the programs are hard to understand.

The only formalism that satisfies most of our requirements is Kumar's and Feldman's FSM. However, the FSM alone can not capture the complete auction process. We propose to extend it by using UML's statechart diagrams. They are well established and widely used, and they are semantically rich enough to formally describe and visualize processes. An important feature statechart diagrams have and that we will be relying on is the possibility to be serialized in XMI [7]. Various types of auctions can indeed be described using statecharts, including the Continuous Double Auction (CDA) which is implemented in GEE, and the English auction which is described in Figure 5. The main states of the English auction are "Taking bids" and "Auction closed". When the auction is closed it goes to the "Clearing" state where the auctioneer has to determine if there is a deal or not. The two possible final states are "Deal" and "No Deal". In the first case the seller and the buyer are notified. In the second case only the seller is notified. The transitions are labeled with the string *event[guard-condition(s)]action(s)*.

6 GNP vision

GNP builds upon the lessons learned from GEE and shifts focus from games towards electronic negotiations. As GNP will be designed especially for negotiations, implementing a negotiation through scripting will be even simpler than creating a game with GEE.

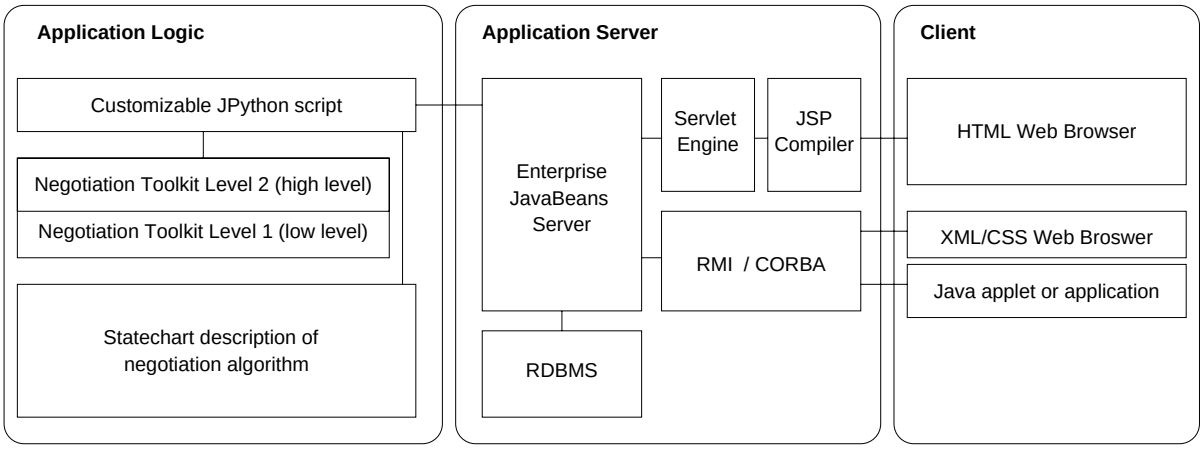

Figure 6: **GNP architecture**

Two new *services* will be provided to the game designer: a negotiation rule description analyzer that can process a formalized description of negotiation processes and a toolkit (a collection of classes) implementing frequently needed concepts (wallet, information storage, etc.). For example, a game designer could create (or reuse) a formalized description of a Dutch auction using a statechart diagram serialized using the XMI format and script a negotiation using this description (see *Figure* 6).

To further improve the openness of the negotiation platform, an information exchange protocol based on XML [3] will be developed. Such a protocol will allow for the creation of software agents that can be used as smart advisors, and for setting up a simulation environment to investigate optimal strategies for complex negotiation algorithms.

The architecture of GNP will reuse several concepts and technologies that were already adopted in GEE: it will be a multi-tier web application using servlets, JSP pages and JPython scripts. However, GNP will need to support large scale deployments and features required by an enterprise level electronic negotiation platform, such as security, scalability, reliability, persistence, and transactional behavior. To meet these requirements, GNP will be a reusable server component executing in an *Enterprise JavaBeans* (EJB) [2] application server using a Relational Data Base Management System (RDBMS) for information storage.

GNP will be *document oriented*, that is, a negotiation is treated as a document, and every interaction is a manipulation of that document. This approach eases the scripting task since the internals of the system are open and accessible. To help manipulate these internal documents, a two level API will be provided which we call *the Negotiation Toolkit*. The first part of the toolkit (referred to as NTK-1) consists of storage management functions, such as loading, searching, and saving information about the negotiation. The second part (NTK-2) will provide functions and objects that are frequently needed by the game designer, according to our experience with GEE. For example, the following objects will be provided by NTK-2: Bid, Negotiation, Round, Player, Wallet, etc. Another benefit of the *document oriented* approach is that obtaining an XML formatted document containing information about a negotiation is trivial; this will allow us to easily create an open protocol based on XML.

Since the most recent generation of web browsers can process and format XML documents combined with

CSS style sheets, the XML protocol for negotiation information exchange will lead to more responsive user interfaces, without affecting their low cost of development and deployment. For example, a small Java applet loaded in the player's browser could listen for events on the EJB server (using RMI or CORBA [17]) and update only portions of the DOM tree [1] contained in the browser without requiring a complete web page reload.

7 Conclusion

In this paper, we detailed GEE, a game-oriented experimentation engine, and described how GEE will be evolved and adapted to become GNP, a generic negotiation platform. We have shown that in this endeavor several important concepts and technologies of GEE will be reused, allowing us to come up with a GNP prototype within just a few months.

There is already some evidence from experimentation with 10 students that indeed GEE constitutes a flexible, powerful, and useful infrastructure for studying the behavior of players under game conditions. After several iterations, the design of GNP is now stable enough to start implementation.

Once GNP is available, various research questions can be investigated. A sample problem from operations research that will be addressed using GNP is determining the winner in a combinatory auction. In combinatory auctions, the user may bid on a combination of items, and the problem for the auctioneer is to determine a winner, that is, the bid that maximizes the auctioneer's gains. Another class of negotiations, combined negotiations, will be refined using GNP. Combined negotiations allow an agent (human or software) to engage in and manage many negotiations at the same time. This latter type of negotiation will lead us to explore the role of decision support systems and workflow management systems [15] at the infrastructure level. For instance, in the case of business-to-business e-commerce, such systems may carry out after-the-deal activities.

The wide range of the above research questions underlines the value of providing GNP. The dynamic nature of this research suggests a feature development approach, and an ongoing re-engineering of GNP. Given the positive experience of evolving GEE and designing the first GNP prototype, we are confident that GNP will prove flexible and useful throughout the project.

Acknowledgment:

This research is supported by Bell Canada, BCE Emergis, NSERC (National Sciences and Engineering Research Council of Canada), and CIRANO. We would like to thank Robert Gérin-Lajoie for his technical assistance in the design of GEE and GNP.

References

[1] Document Object Model (DOM) level 1 specification, version 1.0. `http://www.w3.org/TR/REC-DOM-Level-1/`.

[2] Enterprise JavaBeans Specifications. `http://java.sun.com/products/ejb/docs10.html`.

[3] Extensible Markup Language (XML) 1.0 specification. http://www.w3.org/TR/1998/REC-xml-19980210.

[4] Java Server Pages Specifications. http://java.sun.com/products/jsp/techinfo.html.

[5] The Michigan Internet Auctionbot. http://auction.eecs.umich.edu/.

[6] The JPython Language. http://www.jpython.org.

[7] XMI Metadata Interchange Specification. ftp://ftp.omg.org/pub/docs/ad/98-10-05.pdf.

[8] Morad Benyoucef and Rudolf K. Keller. A survey on description techniques for auction rules in e-commerce. Technical Report GELO-101, Montreal, Quebec, Canada, August 1999.

[9] Aaron G. Cass, Hyungwon Lee, Barbara Staudt Lerner, and Leon J. Osterwei. Formally defining co-ordination processes to support contract negotiation. Technical Report UM-CS-1999-039, University of Massachussetts, Amherst, MA, June 1999.

[10] Peter Coad, Eric Lefebvre, and Jeff De Lucas. *Java Modeling in Color with UML*. Prentice-Hall, 1999.

[11] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 13(3), June 1988.

[12] David M. Kreps and Robert Wilson. Sequential equilibria. *Econometrica*, 50(4):863–894, July 1982.

[13] Manoj Kumar and Stuart I. Feldman. Business negotiations on the internet. In *INET98 Conference of the Internet Society*, Geneva, Switzerland, July 1998.

[14] Manoj Kumar and Stuart I. Feldman. Internet auctions. Technical report, IBM Institute for Advanced Commerce, Yorktown Heights, NY, November 1998. Available at http://www.ibm.com/iac/.

[15] Heiko Ludwig and Keith Whittingham. Virtual enterprise co-ordinator - agreement-driven gateways for cross-organisationl workflow management. pages 29–38, San Francisco, CA, 1999.

[16] Andreu Mas-Collel, Michael Whinston, and Jerry Green. *Microeconomic Theory*. Oxford University press, 1995.

[17] Gopalan Suresh Raj. A detailed comparison of CORBA, DCOM and Java/RMI, September 1998. Available at http://www.execpc.com/ gopalan/misc/compare.html.

[18] Rational Software Corporation. UML Documentation Set Version 1.1, 1998. Santa Clara, CA. Available at http://www.rational.com/uml/.

[19] Michael P. Wellman and Peter R. Wurman. Real time issues for internet auctions. In *IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, Denver, Colorado, June 1998.

[20] A. Wise. Little-JIL 1.0 language report. Technical Report 024, Department of Computer Science, University of Massachussetts at Amherst, 1998.

[21] Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998.

[22] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The michigan internet AuctionBot: A configurable auction server for human and software agents. In *Second International Conference on Autonomous Agents (AGENTS-98)*, pages 301–308, Minneapolis, MN, May 1998.