Designing the Market Game for a Trading Agent Competition

The authors discuss the design and operation of a trading agent competition, focusing on the game structure and some of the key technical issues in running and playing the game.

TAC Team
University of Michigan
North Carolina State University

Michael P. Wellman
Peter R. Wurman
Kevin O'Malley
Roshan Bangera
Shou–de Lin
Daniel Reeves
William E. Walsh

Trading in electronic markets is a topic of increasing interest within the artificial intelligence (AI) and electronic commerce research communities. As Internet marketplaces proliferate, programs that monitor and bid in these markets automatically—what we call "trading agents"—will play a significant role.

Various AI research communities have held competitions to compare different approaches to common problems, and showcase the state of the art.[1-3] Inspired by their successes, we organized a trading agent competition (TAC), held at the most recent International Conference on Multiagent Systems. The competition pitted agents from 20 teams around the world against each other in a market game. The agents ran from participants' home institution using TCP/IP socket connections to exchange messages with the auction server at the University of Michigan.

In this article, we discuss general criteria for the design of a market game using trading agents and present our competition as a model problem description for this domain; we also describe the competition's genesis, its technical infrastructure, and its organization. The article, "Autonomous Bidding Agents in the Trading Agent Competition"[4] (page 52), describes the competition from a participant's perspective and describes the strategies of some of the top-placing agents. A visualization of the competition and a description of the preliminary and final rounds of the TAC are available in *IC Online* (http://computer.org/internet/tac.htm). Further information is available on the TAC Web site (http://tac.eecs.umich.edu/) and in other articles on TAC agents.[5]

Designing the Competition

Once we had resolved to organize a competition for trading agents, our first charge was to design a market game, that is, a well-defined scenario that lets agents interact through a market. We approached this design with several criteria:

- *Strategic challenge.* The game should present difficult issues in real-world bidding strategy. In particular, we were interested in scenarios that required traders to deal in multiple interrelated goods.

- *Multiplicity of issues*. Trading is inherently a multifaceted problem. We attempted to balance the kinds of issues the agents faced to avoid enabling an agent with a particular competence in one area to dominate the game.
- *Realism*. The scenario should plausibly resemble what automated traders might be practicing within a few years.
- *Simplicity*. To avoid prohibitive barriers to entry, the competition rules and interfaces should be as simple as possible. (We were perhaps least successful on this criterion, as the features we introduced to diversify the game and make it challenging also contributed to its complexity.)

In games where there are many agents, it is possible to reason about others only in the aggregate.

We used these criteria to evaluate several candidate market games.[6] For example, we ruled out an abstract, continuous double auction (CDA) tournament (employed in a pioneering competition at the Santa Fe Institute ten years ago[7]) primarily because it lacked multiple interrelated goods. We ruled out others, including some based on problems in our research group, because they seemed too parochial.

The early design document was circulated among an international organizing committee comprising active researchers in the field. Their deliberations, as well as suggestions from other sources, led us to settle on a shopping scenario. In addition to offering the challenges of interrelated goods and multiple issues, automated shopping is an active concern of agent researchers, and is plausibly realizable in nontrivial form in the relatively near term.[8]

Another issue in trading games is predicting or influencing the behavior of other agents. In a game with only one or two other agents, effective reasoning about one's counterparts can be a dominant factor. In games with very large numbers of (similar-sized) agents, it is possible to reason about others only in the aggregate, and attempting to influence outcomes (for example, prices) is fruitless. To avoid either extreme, we chose to include eight agents in each game instance.

Travel Shopping Game Basics

In the TAC scenario, each trader is a travel agent charged with arranging trips for its clients. At the start of a game, or *game instance*, the agent receives requests from eight clients for trips from TACtown to Boston during a specified five-day period. The travel agent's objective is to maximize the total satisfaction of its clients (the sum of the client utilities) relative to the money expended procuring travel resources in the TAC markets.

A client travel package consists of round-trip flights, a hotel room for each night, and tickets to various entertainment events. There are obvious interdependencies, as the traveler needs a hotel for every night between arrival and departure, and can attend entertainment events only during that interval. In addition, clients have individual preferences regarding the days they are in Boston, the type of hotel, and modes of entertainment.

Bidding

There are three types of auctions in the TAC: flight, hotel, and entertainment (described in the sidebar "TAC Auction Types," next page). Agents submit bids to the auctions, expressing offers (in dollars) to buy or sell the corresponding goods. The auction server admits offers in the form of *discrete demand schedules*—that is, sets of quantity-price points:

$$\{(q_1, p_1), \ldots, (q_n, p_n)\}.$$

A point (q_i, p_i) with $q_i > 0$ means that the agent is willing to buy q_i units of the good for no more than price p_i per unit. If $q_i < 0$, then the agent is willing to sell $|q_i|$ units of the good at a price p_i or greater. For example, submitting the offer {(-2, $40), (3, $20), (1, $10)} means, "I am willing to sell two units if the price (per unit) is $40 or more. I am willing to buy three units if the price is $20 or less, and to buy one additional (or four total) if the price is $10 or less."

The rules for a particular auction specify any restrictions on admissible bids, possibly as a function of bids received up to that point. Once a bid is admitted to a TAC auction, the auction server immediately updates its price quote, a summary indicator of the current price. The rules for a particular auction specify when, or under what conditions, the auction will match the bids and record the transactions.

Client Preferences and Initial Endowments

At the start of each game instance, the agent queries the auction server for its client travel requests and initial endowment of entertainment

TAC Auction Types

TAC agents assemble trips for their clients by acquiring travel goods in three different markets: flights, hotels, and entertainment tickets. The agents must deal with distinct types of auctions, implementing qualitatively different trading rules for the three markets.

Flight

TACair is the only airline to fly between TACtown and Boston, operating one flight each way per day. Tickets for the flights are sold in single-seller auctions—one auction for each day and direction (in or out). Since all clients must stay at least one night in Boston, there are no flights to Boston on the last day, nor return flights on the first day. TACair is represented in the marketplace by a bidder defined by the competition to set prices according to a stochastic pricing policy. The process used to update flight prices is a random walk, starting between $250 and $400 and perturbed by -$10 to $10 every 30 to 40 seconds. (Values are chosen uniformly in their respective ranges.) Prices are adjusted if necessary to remain in the range of $150 to $600. The supply of available seats on the flights is, as far as TAC agents are concerned, unlimited.

There is a separate flight auction for every combination of day and flight type (incoming or returning). The flight auctions are continuous one-sided auctions, and do not close until the end of the game instance. This means that whatever the current sell offer is, a buyer can place a bid that matches it and immediately transact for the item. In other words, the flights are sold at fixed prices that randomly fluctuate over time.

Hotel

There are two hotels in Boston: the Boston Grand Hotel and Le Fleabag Inn. The former is cleaner, more comfortable, and more convenient—all-around a nicer place to stay. There is no minimum bid for either type of hotel; however, each client is willing to pay at least $50 to upgrade to the Grand Hotel from the Fleabag. In turn, clients receive a higher utility from the Grand Hotel. Since clients need hotels only from the night they arrive through the night before they leave, no hotels are available (or needed) on the last possible day of travel (day 5).

Hotel auctions are standard English ascending, multi-unit auctions. The hotel owners (representing the competition) submit a sell bid to provide 16 rooms in each auction: {(-16, $0)}. By the rules of the auction, price quotes are issued immediately in response to new bids. The price quote is the ask price, calculated as the 16th highest price among all buy and sell bid units. For instance, if the offers {(-16, $0)}, {(6, $6), (2, $4)}, {(4, $8)} were active in a hotel auction, the ask price would be $0. For the set of offers {(-16, $0)}, {(6, $6), (2, $4)}, {(4, $8)}, {(7, $10)}, however, the ask price would be $6.

When agents submit new bids, they must adhere to a "beat the quote" rule. Let *ASK* be the current ask quote, that is, the 16th highest price. Any new bid *b* must satisfy the following conditions to be admitted to the auction:

- *b* must offer to buy at least one unit at a price of *ASK* + 1 or greater.
- If the agent's current bid b^0 would have resulted in a purchase of *q* units in the current state, then the new bid *b* must offer to buy at least *q* units at *ASK* + 1 or greater.

Agents may not withdraw bids from hotel auctions.

When a hotel auction clears, the 16 highest price buy units are

continued on p. 46

tickets. This information is randomly generated according to distributions specified as part of the game definition (and thereby made known to the contestants). The client preferences consist of ideal arrival and departure days and reservation values (maximum willingness to pay) for a hotel upgrade and for each type of entertainment.

The ideal arrival and departure days are generated for each client such that every legal pair of days (clients must stay at least one night) is equally likely. By the nature of this distribution, nights in the middle of the range tend to be in much greater demand—an important element of agent bidding strategies. The reservation values were all chosen from uniform distributions—$50 to $150 for hotel upgrades, and $0 to $200 for each entertainment ticket type.

Finally, we endow each agent with entertainment tickets, on average one of each kind. Specifically, for each combination of entertainment type (baseball, symphony, or theater) and day (1 through 4), agents receive zero tickets with probability 1/4, one ticket with probability 1/2, and two tickets with probability 1/4.

Utility Function

The utility score for an agent at the end of a game instance is simply the sum of utilities achieved for each client less its net expenditure in all auctions. The utility function for an individual client is a function of the travel package procured for that client relative to the client's preferences. If the travel package is not feasible, the client receives zero utility. A feasible package is one in which the client stays at least one night, stays in the same hotel for each night in town, and sees at most one form of entertainment per night.

The general form of the client utility function (in dollar units) is

$$1000 - \mathit{travelPenalty} + \mathit{hotelBonus} + \mathit{funBonus},$$

where

TAC Auction Types continued

continued from p. 45

matched, and the agents pay the ask price for the hotel rooms. For instance, assume the following bids were in a hotel auction at clearing time:

- Hotel seller: {(-16, $0)}
- Agent 1: {(8, $2)}
- Agent 2: {(6, $6), (2, $4)}
- Agent 3: {(4, $8)}
- Agent 4: {(7, $10)}

In this example, Agent 4 would win seven rooms, Agent 3 would win four rooms, Agent 2 would win five rooms, and Agent 1 would not win any rooms. The price of all rooms would be $6.

To prevent the agents from waiting until the last seconds of the game instance to bid, we trigger the clearing operation on a period of inactivity. After a random, and unspecified, amount of time with no bids admitted, the hotel auction clears and closes for the game. The actual probability distribution for the amount of inactivity before closing the auction (uniform from 30 seconds to 5 minutes) is not disclosed to the contestants. In the actual competition, however, this measure was not adequate to effect price formation in early stages of the hotel auctions. As recounted by the entrants, successful agents placed just enough incremental bids to keep the auctions open, saving their serious offers for the very end of the game.

Entertainment

Three entertainment events are offered: Boston Red Sox baseball, the Boston Symphony, and a Boston theater production. As with the hotels, a client cannot use an entertainment ticket on the day of departure. Unlike flights and hotels, there is no central seller for entertainment tickets. Rather, each agent is endowed with an initial set of tickets, which they can buy and sell to each other in continuous double auctions (CDAs). There is one entertainment CDA for each day/type combination, operating according to CDA standard rules. That is, any agent can buy or sell, and incoming bids are matched immediately with compatible standing bids. Price quotes, which are also continuously updated, represent a bid-ask spread, the bid quote being the highest standing buy bid price and the ask quote being the lowest standing sell bid price.

As an example, consider the following standing bids in an entertainment ticket auction: {(-1, $100)}, {(-4, $90), (-2, $50)}, {(-6, $60)}, {(1, $40), (3, $10)}, {(1, $20)}. The bid-ask spread (price quote) is [$40, $50]. The following examples show what happens as various new bids are submitted to the auction:

- A new bid of {(-1, $55)} does not match and is added to the list of standing bids.
- A new bid of {(3, $48)} is treated as above, but in this case the bid-ask spread is updated to [$48, $50].
- A new bid of {(6, $70)} matches two units at $50 each (the second in the list of standing bids), one unit at $55 (the first new bid above), and three units at $60. The bid-ask spread becomes [$48, $60].
- A new bid of {(-7, $15)} matches three units at $48, one unit at $40, and one unit at $20. Since this bid does not match completely, the remaining portion stands in the auction as {(-2, $15)}. The bid-ask spread becomes [$10, $15].

$$\begin{aligned} \mathit{travelPenalty} &= 100(|\mathit{arrivalDay} - \mathit{idealArrival}| + \\ &\qquad |\mathit{departureDay} - \mathit{idealDeparture}|) \\ \mathit{hotelBonus} &= \begin{cases} \mathit{hotelReservation} & \text{if Grand Hotel} \\ 0 & \text{otherwise} \end{cases} \\ \mathit{funBonus} &= \text{sum of reservation values for each type of entertainment in trip.} \end{aligned}$$

Note that no additional utility is given for seeing the same type of entertainment on multiple nights.

Final Scores and Client Package Allocation

At the end of the game, the travel agent holds a collection of plane tickets and hotel rooms bought from the designated sellers, and entertainment tickets bought and sold with other agents. Because the auction server has no information about trader inventory, it cannot prevent an agent from selling entertainment tickets it does not actually own. Instead, we assume there are effectively an infinite number of tickets available (say, from scalpers) at a price of $200 each. At the end of the game instance, we calculate scores under the assumption that agents cover any ticket deficits by buying from scalpers. Because the distribution of client valuations ensures that no entertainment ticket is worth more than the scalper price, a rational agent will try to avoid this situation. However, if an agent finds an overeager buyer, it may sell more than it has, hoping to make up the difference later and accepting the risk of a forced loss at the end of the game.

An agent's final task is to decide how to allocate its plane tickets, hotel rooms, and entertainment tickets to maximize total client utility. At the end of the game instance—when all the auctions close—the agent has four minutes to report this allocation to the server. The allocation specifies, for each client, an arrival and departure flight, a hotel type, and entertainment tickets.

TAC Auction Server

The TAC auction server is based on the Michigan

Internet AuctionBot,[9] and forms the core software infrastructure for the competition. The AuctionBot is a highly configurable auction server, operational since 1996, built to support research on e-commerce and multiagent negotiation. The first-generation architecture was simple and robust, designed to conduct a large number of simultaneous auctions with asynchronous and relatively sporadic bidder interactions conducted through a Web interface. Details of the AuctionBot are presented elsewhere, including a recent report focusing on the TAC performance requirements.[9,10]

Once we introduced an automated trading facility (that is, a software API), we found that the frequent interactions demanded by automated traders resulted in performance bottlenecks. We redesigned the control architecture to address these performance issues as well as to meet our original design goals.[11] Figure 1 shows a diagram of the auction server and its components.

Agent Programming Interface

TAC software agents interact with the AuctionBot via an agent programming interface (API),[12] which consists of a query-based communications protocol and basic client software. The API supports all agent/AuctionBot communications, including bid submission and queries for next game time, client preferences, auction IDs, bid state, price quotes, and transactions. Sixteen API commands are available to TAC agents.

An API call consists of an API command followed by a list of attribute-value pairs in CGI format. For instance, the call `submitbid?auctionid=347&bidstring="((2 30))"` specifies that the agent wishes to submit a bid to buy two units of the good in auction number 347 for no more than $30 per unit. The `bidstring` value contains a list of price/quantity points as in the discrete demand schedule described earlier. An agent can have at most one bid active in an auction at any given time; hence, if the new bid is admitted, it replaces any previous standing bid from the agent.

The AuctionBot might reply with `submitbid?bidid=1234&bidhash="((2 30))"&commandstatus=0`, indicating that the `submitbid` call was successful (`commandstatus=0`) and that the submitted bid's ID is 1234 (`bidid=1234`). The `bidhash` field helps avoid certain asynchrony problems, as we discuss in the following section.

To exchange the API message strings, agents maintain a TCP/IP socket connection with the AuctionBot throughout a game. TAC players were free to implement the string manipulation and socket communication in their favorite language, although we did provide low-level client software in C/C++, Java, and Mathematica to streamline development. This client software converts API strings to and from attribute-value symbol tables and handles the socket communications.

We also provided bidding agent classes in C++ to abstract away some of the API details. The bidding agent classes execute a loop that automatically queries for information on the auctions, next game, bid state, auction prices, and transactions, and assembles the information in convenient data structures. The classes also provide hooks for the agent designer to add bidding strategies and agent termination conditions.

Full details on the API and other aspects of the TAC infrastructure are available at http://tac.eecs.umich.edu/software.html.

Asynchrony

Asynchrony in various forms poses a challenge to designing effective agent strategies. The most obvious and fundamental cause of asynchrony is communication latency. The AuctionBot requires some time to process API calls, during which it asynchronously processes bid submissions, price quotes, and clear events. As a result, the information contained in any API response can be out-of-date by the time an agent receives it. Another source of asynchrony is the separation of bid reception and processing. The AuctionBot informs the agent upon submission if a basic error occurs, but decides whether to admit the bid in a distinct processing operation (that is, the bid may yet be rejected if it violates the auction rules or bid syntax).

Asynchrony in various forms poses a challenge to designing effective agent strategies.

This phenomenon could be controlled by explicit synchronization, for example, by forcing agents to take turns, as was done in the Santa Fe Institute double auction tournament,[7] and handling all AuctionBot events synchronously between turns. However, we rejected such an approach for several reasons:

- Turn-taking effectively reduces the system to the speed of the slowest agent.
- The internal asynchrony of the AuctionBot allows for more responsive API handling and more effective simultaneous usage of server resources.
- Asynchronous operations is a more realistic

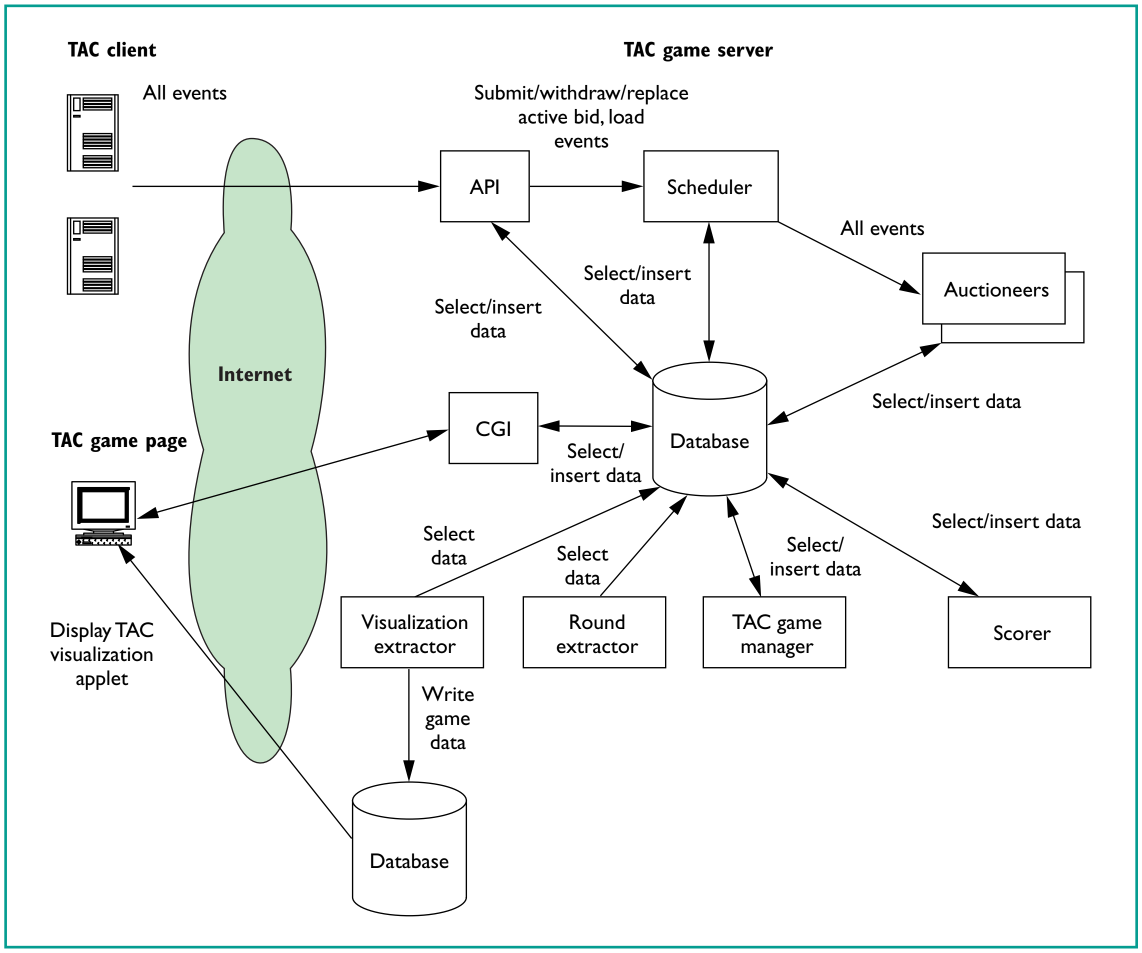

Figure 1. The TAC auction server and its components. The API, scheduler, auctioneers, and database perform core operations of the AuctionBot; other elements serve TAC-specific functions.

reflection of actual e-commerce. Indeed, some agents effectively adapted to varying asynchronous delays, a strategy we expect to be generally useful for e-commerce trading agents.

Due to the internal asynchrony of the AuctionBot, there is no strong relationship between the timing of events across different auctions. Yet the AuctionBot does enforce strict logical sequencing of events within an auction. That is, an unbounded real-time delay may pass between internal events, but when the event actually occurs, the AuctionBot performs the event using the correct state present at the logical time at which the event should have occurred.[13] For instance, an "immediate" transaction in a continuous double auction may be performed some time after a bid is admitted, but no other bids are allowed to "sneak in" between the time the bid is admitted and the transaction is recorded.

No single API call acquires the full (publicly available) state of an auction at a single instant. Rather, an agent has to make separate calls to get bid state, price quote, and transaction information. As a result, an agent's most recent API calls can have contradictory information. For instance, its latest report about the quantity it is tentatively "winning" may not jibe with its report of the going price. This example demonstrates how some characteristics of the trading interface not readily apparent at design time can be ferreted out with extensive use by multiple developers. Our experience with TAC led us to move the tentative win quantity information to the price quote API call. With TAC completed, we have been able to implement this and various other improvements to the trading interface.

Information about bids and transactions is more conceptually separate, and thus naturally maintained as distinct components of auction state. By querying for information on its bid, the agent can determine whether the AuctionBot admitted the bid, and whether the bid has fully or partially transacted. A separate API call yields direct information about the transactions themselves. As a result, the agent's knowledge about the bid status and remaining offer schedule may be inconsistent with its knowledge of transactions on the bid. Fortunately, the agent can infer much about the correct state from the combination of results and their associated time stamps.

It would be particularly undesirable if asynchrony caused an agent to unintentionally buy or sell goods. Consider the following sequence of events:

- An agent's current bid transacts in the auction.
- The agent submits a new bid to the auction before learning of the transaction.
- The new bid transacts in the auction, causing the agent to buy or sell more units than it had intended.

To ameliorate this problem, agents can use "replace active bid" semantics when replacing a standing bid. When replacing a bid, the agent includes a bid hash, which will match the bid hash of the agent's current standing bid if and only if the bid has not transacted since the agent last queried for information on the bid. The bid replacement will succeed only if the agent has an active bid in the auction with a matching bid hash; if not, then the agent did not have an accurate understanding of the state of its standing bid.

Game Phases

The *game manager* controls the scheduling, invocation, and operation of TAC game instances within the competition. Each game runs for 15 minutes (auction time), with a 10-minute interval between each game. Eight trading agents participate in a given instance, along with a predefined seller agent representing the competition's specified policy for hotel and flight auctions.

We divide a game instance run into three operational phases: pregame, in-game, and postgame.

Pregame Phase

The game manager provides facilities for users to schedule new game instances and sign up for future games. Each game starts with eight available slots, allocated on a first-come-first-served basis. Any slots not taken by game time are filled by dummy agents. Like popular Web-based recreational gaming systems, the objective is to support distributed organization of game instances reflecting the schedule preferences of the participants. This was especially useful during the development and practice periods. For tournament play, we prescheduled the games according to competition rules.

The dummy agents were provided to generate a representative distribution of bidding activity so that agents could practice in a somewhat realistic environment. In addition, the provided dummy agent code served to illustrate the agent programming facilities. We expressly did not program the dummy to follow an exemplary or even reasonable bidding strategy, to avoid biasing the entrants according to some idea of the TAC designers. Consequently, practice games involving dummies varied systematically from games comprising real agents.

API calls inform agents of their next scheduled game. Agents can thus run in a continuous loop of determining next game time, sleeping until that time, then awakening and initializing their state for the game instance. Game preference and endowment parameters are retrieved through the API, but are not available until the game actually starts.

In-Game Phase

To run a TAC game, the game manager selects the next game from its pending game queue. Before starting the game, it checks to make sure all auctions from the previous game were closed properly. Next, the game manager creates the auctions for the new game and runs them. If an error occurs, the game's status is set to NOTRUN, and the game scheduler returns to its main loop.

It would be undesirable if asynchrony caused an agent to unintentionally buy or sell goods.

Next, the game manager launches the TAC seller and any dummy agents participating in the instance. It also distributes game parameter information, so that participating agents can begin play.

The visualization extractor runs throughout the game, passing data to the game visualization tools. After 15 minutes of game time, the AuctionBot scheduler queue contains final clear events for all auctions that should be closed. Each event is popped from the AuctionBot scheduler queue and sent to the corresponding auction. When the auction receives the final clear event, it sends the scheduler a deactivated event informing it that the auction is exiting, and the auction exits. When the AuctionBot scheduler processes the deactivate event for an auction, it checks whether the auction has exited and then clears it from the process table.

Postgame Phase

Once the game is complete, the game data extractor generates text files describing all the relevant game events. These files serve as input for game visualization, and are archived for later retrieval through the Web interface.

After receiving the agents' reported allocations, the TAC game scorer confirms that each agent has acquired sufficient goods for the packages it assigned. If the agent has provided an invalid allocation, the scorer deletes assigned resources (without regard for optimality) until the allocation is feasible. In the absence of an allocation, the scorer produces an assignment based on (not very effective) greedy methods. Finally, it computes a score based on the utility function.

Players view the game results through a summary page showing the final utility scores of all agents in the game. Each agent's score is linked to an allocation page, which provides detailed information about the player's allocation, client preference profile, expenditures, and unused travel resources.

The TAC Tournament

The TAC finals were held 8 July 2000, at a workshop preceding the main ICMAS conference. The twelve finalists were selected from a pool of 20 contestants through a series of qualifying rounds held over an eight-day period. Selections were based on participants' average scores in the preliminary games.

On the day of the finals, we set up an ISDN network in the workshop room, and prepared 20 Ethernet connections for laptop computers. Most contestants brought laptops, from which they launched and monitored their agents running at their home institutions. Two Sun workstations monitored the TAC server, and a laptop projected real-time visualizations.

The games were played in round-robin style, with each agent participating in six games. The eight agents with the highest average scores progressed to the afternoon games. The final seven games were played with a constant profile of the remaining contestants. Throughout the day, contestants presented their agent designs in one part of the room, while computers continuously projected displays of the ongoing games on a side wall. This "two-ring circus" format balanced interest in the game outcomes with interest in strategy discussions. Most of the agents presented at the workshop are discussed in the companion article in this issue.[4] Boman provides some further observations on the event.[14]

We plan a follow-on trading agent event in conjunction with the Third ACM Conference on Electronic Commerce in October 2001. The consensus among participants was that we should retain the basic structure of the game, but modify the rules to eliminate some anomalies and unnecessary complexities. In revising the game, through a review process involving the TAC participant community, we will also aim to reduce the contributions of random elements on scores, and in other ways strengthen the connection between strategic effectiveness and outcome evaluations.

The TAC environment has been employed in teaching e-commerce and artificial intelligence, and we will continue to make it available for educational use. We are also optimistic about research progress in understanding the implications of autonomous trading agents on developments in e-commerce and automated marketplaces. We believe that exercises like this competition contribute to that progress by creating complex, realistic settings for dynamic programmed trade. We hope this experience has showcased the competence that can be achieved with current technologies, and has highlighted research issues bearing on the construction and understanding of the next generation of trading agents.

Acknowledgments

We are grateful to the TAC sponsors for making this event financially possible: Andersen Consulting, Ariba, IBM, and the North Carolina State University E-Commerce consortium. Work on the underlying infrastructure was supported in part by related projects funded by DARPA, the U.S. Air Force Office of Scientific Research, and the U.S. National Science Foundation. Christopher Healey provided interesting visualization software.

Members of the TAC Organizing Committee, Arne Andersson, Dave Cliff, Toru Ishida, Jeffrey Kephart, John Miller, Juan Antonio Rodriguez, Edmund Durfee (ex officio), and especially Fredrik Ygge, offered helpful suggestions on the competition design. Most of all, we thank the competition participants for their enthusiasm, persistence, and intellectual contributions to this event.

References

1. S. Coradeschi et al., "Overview of RoboCup-99," *AI Magazine*, vol. 21, no. 3, 2000, pp. 11-18.
2. P. Bonasso and T. Dean, "A Retrospective of the AAAI Robot Competitions," *AI Magazine*, vol. 18, no. 1, 1997, pp. 11-23.
3. D. Long et al., "The AIPS-98 Planning Competition," *AI Magazine*, vol. 21, no. 2, 2000, pp. 13-33.
4. A. Greenwald and P. Stone, "Autonomous Bidding Agents for the Trading Agent Competition," *IEEE Internet Computing*, vol. 5, no. 2, Mar./Apr. 2001, pp. 52-60.
5. P. Stone et al., "ATTac-2000: An Adaptive Autonomous Bidding Agent," to be published in *Proc. Fifth Int'l Conf. Autonomous Agents*, 2001.

6. M.P. Wellman and P.R. Wurman, "A Trading Agent Competition for the Research Community," *IJCAI-99 Workshop on Agent-Mediated Electronic Trading*, Aug. 1999.
7. J. Rust, J.H. Miller, and R. Palmer, "Characterizing Effective Trading Strategies: Insights from a Computerized Double Auction Tournament," *J. Economic Dynamics and Control*, vol. 18, 1994, pp. 61-96.
8. A. Eisenberg, "In Online Auctions of the Future, It'll Be Bot vs. Bot vs. Bot," *New York Times*, 17 Aug. 2000, p. D8.
9. P.R. Wurman, M.P. Wellman, and W.E. Walsh, "The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents," *Proc. Second Int'l Conf. on Autonomous Agents*, ACM Press, New York, May 1998, pp. 301-308.
10. K. O'Malley, "Auction Service for a Trading Agent Competition," *Dr. Dobb's J.*, to appear.
11. P.R. Wurman et al., "A Control Architecture for Flexible Internet Auction Servers," *IBM/IAC Workshop on Internet-Based Negotiation Technology*, Mar. 1999.
12. K. O'Malley and T. Kelly, "An API for Internet Auctions," *Dr. Dobb's J.*, Sept. 1998, pp. 70-74.
13. M.P. Wellman and P.R. Wurman, "Real-Time Issues for Internet Auctions," *Proc. IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, IEEE CS Press, Los Alamitos, Calif., 1998.
14. M. Boman, "Trading Agents," workshop report, *AgentLink News*, no. 6, Jan. 2001; also available online at http://www.agentlink.org/newsletter/6/.

Michael P. Wellman is an associate professor in computer science and engineering at the University of Michigan. He received a PhD in computer science from the Massachusetts Institute of Technology. For the past nine years, his research has focused on computational market mechanisms for distributed decision making and electronic commerce. He is a AAAI Councilor, executive editor of the *Journal of Artificial Intelligence Research*, and serves on the Steering Board of ACM SIGecom.

Peter R. Wurman is an assistant professor in computer science at North Carolina State University where he investigates electronic auctions and trading agents. He received his PhD in computer science from the University of Michigan in 1999. He is codirector of NCSU's E-commerce Initiative, and editor in chief of SIGecom Exchanges, the official newsletter of ACM SIGecom.

Kevin O'Malley is a systems research programmer at the University of Michigan Artificial Intelligence Laboratory. He specializes in network programming, object-oriented development, and software engineering. For the past four years he has been the software architect and lead developer of the Michigan Internet AuctionBot and the TAC software system. O'Malley also holds a Master's degree in music.

Roshan Bangera completed his master's degree in computer science and engineering at the University of Michigan in May 2000. He currently works for CommerceOne in New York.

Shou-De Lin will receive his master's degree in electrical engineering and computer science from the University of Michigan in May 2001. His current research focuses on resource allocation problems and trading agent competition postgame analysis.

Daniel M. Reeves is a PhD candidate in computer science and engineering at the University of Michigan. His research interests are in artificial intelligence for e-commerce, and his recent work involves automating the negotiation of declarative (rule-based) contracts. He is interested in various aspects of trading agents and considers Mathematica the programming language of choice for creating them.

William E. Walsh is a PhD candidate in computer science and engineering at the University of Michigan. His research focuses on automated negotiation technologies and market approaches to problems with complex dependencies, including supply chain formation, constraint satisfaction, and scheduling. He is a NASA/Jet Propulsion Laboratory graduate student researcher. If all goes as planned, he will defend his dissertation within a few months after this article is published.

Readers can contact the authors at the University of Michigan Artificial Intelligence Laboratory, 1101 Beal Ave., Ann Arbor, MI 48109-2110, USA, tac.support@umich.edu.

CLASSIFIED

Collective Communications

Dynamic early stage wireless solutions provider working with leaders like Sun and Palm has openings for professionals who love working on cutting-edge technology in an energetic, flexible environment. Research and development of high-visibility, leading-edge wireless software products; lucrative compensation packages.

Senior Software Developers and Software Architects: Experience in distributed systems, multi-threaded code, OOP, full software lifecycle, 3+ years Java. A plus includes operating system design and implementation experience.

E-mail: jobs@collective.com
Visit: www.collective.com