Rule-Based Specification of Auction Mechanisms

Kevin M. Lochner and Michael P. Wellman

University of Michigan
Artificial Intelligence Laboratory
1101 Beal Av
Ann Arbor, MI 48109-2110, USA
{klochner,wellman}@umich.edu

Abstract

Machine-readable specifications of auction mechanisms facilitate configurable implementation of computational markets, as well as standardization and formalization of the auction design space. We present an implemented rule-based scripting language for auctions, which provides constructs for specifying temporal control structure, while supporting orthogonal definition of mechanism policy parameters. Through a series of examples, we show how the language can capture much of the space of single-dimensional auctions, and can be extended to cover other novel designs.

1. Introduction

Technological and financial innovation adding to the complexity of commerce opportunities has placed new demands on allocating institutions. Standard market procedures may not adequately serve these complex trade situations, which call for increasingly sophisticated mechanisms, or *auctions*, for determining market outcomes. At the same time, the Internet medium coupled with advances in computational power has opened the door to previously infeasible auction designs. In consequence, recent years have seen a surge of academic interest in auction design [6, 9, 14], in conjunction with the increasing deployment of auctions (albeit slower than anticipated in the bubble years) in electronic marketplaces [17] and deregulated commodity markets [8]. These research efforts have led to numerous innovations in auction design, addressing such issues as incentive compatibility [10], computational efficiency [15], and timeliness of market operation [5].

As the relevant design space expands, there is increasing need for languages to express auction mechanisms [7, 12, 13, 21, 22]. Specifying an auction mechanism in a high-level, special-purpose language serves several important functions:

1. The specification can be executed to implement the auction mechanism.
2. High-level specifications support rapid deployment and testing of mechanisms. A series of mechanisms may be quickly deployed through modification and extension of existing mechanisms.
3. Publication of auction definitions promotes transparency, validation, and standardization of interfaces.
4. Precise definition of standard auction constructs facilitates communication among auction designers and analysts, and dissemination of design knowledge.
5. Machine-readable auction definitions can be interpreted by automated trading agents, who may adapt their behavior accordingly.
6. Formal mechanism descriptions can be manipulated computationally, enabling automated search and analysis of the design space [3, 11].

Our auction specification effort is motivated primarily by the implementation imperative, although we attempt to respect the other motivations, in hopes of ultimately accruing their benefits. Following the approach previously taken in the Michigan Internet AuctionBot [20], we emphasize configurability of auction service, thus attempt to build in the structures common across the wide auction mechanism space, yet provide sufficient latitude to accommodate novel designs.

Our approach builds on the parametrization of auction design space defined by Wurman et al. [21]. In attempting to extend this parametric specification to a wider variety of designs, we experienced a decreasing orthogonality as we introduced parameters. In other words, it became increasingly necessary to introduce parameters switching on and

off whole areas of the design space, thus diluting the benefit of previous specification effort. Parameters defining the auction *control structure*—the temporal pattern of auction events, seem particularly ill-suited to parametric description. In general, the timing of a particular action within an auction may depend on arbitrary features of the auction history. For specification of such functional dependence, control structures reminiscent of programming languages may be more effective than simple parameter settings.

For example, consider the *survival auction* [5], which proceeds in rounds with the lowest bidder eliminated at the end of each round, until a single bidder remains. In this case the set of eligible bidders must be modified over the duration of the auction. Capturing this auction with a parametrization would seem to require a dedicated parameter indicating the survival auction, with auxiliary parameters defining the length of rounds, bidding rules, etc. As we see below, rule-based definition of the survival auction leads to a more transparent encoding, where the repeated-round structure is reflected in the representation, and the process of eliminating bidders is apparent from the rule specification.

The remaining sections describe a simple and extensible rule-based language we have designed and implemented for the specification of auction mechanisms. The language combines parameter specification with rule-based invocation of auction behaviors, providing sufficient flexibility to capture a wide range of known and conceivable auctions. Following explanation of our auction specification framework in the next section, we present the details of our language excution environment and address some technical details for coordinating groups of related auctions. Several example scripts serve to demonstrate the current capabilities of the language.

2. Auction Specification Framework

The general class of *auctions* comprises all mediated mechanisms that determine market-based allocations (i.e., exchanges of goods and services for money) as a function of agent messages. These messages, or *bids*, are typically composed of *offers* specifying deals in which the agent is willing to engage. Although the form and content of bids, along with the auction's behavior given such bids, can vary widely among auction mechanisms, there are several common constructs we can define across the entire space [21].

Upon receiving a new or revised bid, the auction determines whether the bid is *admissible* given its current state. If so, the bid is admitted to the *order book*, a repository representing the current collection of active offers. At some point (depending on the auction rules, of course), the market *clears*, producing a set of exchanges matching compatible offers in the order book according to the auction's *clearing policy*. Along the way, the auction may send messages to participants providing information about auction state (often in highly summarized form), according to its *information revelation policy*. Since this information often—though not invariably—includes current *price quotes* (i.e., indications of what the hypothetical clearing prices would be in the current state), we refer to both the action and revealed information as a *quote*.

Following Wurman et al. [21], we maintain that the substantial differences among auction mechanisms can be characterized by their policies for the three major activities described above: processing bids, clearing, and revealing information. We further decompose the specification of these policies into

1. their functional implementation (i.e., the ***how***), and
2. their timing (the ***when***).

In our approach, the *how* is specified primarily through parameter settings, and the *when* through rule condition patterns.

We have implemented an auction engine capable of reading and implementing these auction definitions, or auction scripts. We call this auction system *AB3D*, in reference to the two previous implementation generations of our AuctionBot ("AB") [20]. The "3D" highlights our three-dimensional characterization of auction policy space, and also suggests the underlying motivation for our redevelopment: to support multidimensional market mechanisms.

3. The AB3D Scripting Language

An AB3D auction script comprises a sequence of statements, of four types:

1. *initialization* of auction parameters,
2. *rules* that trigger auction events and parameter changes,
3. *declarations* of user variables, and
4. *bid rules* defining additional bid admissibility requirements.

The language also supports the organization of statements into distinct phases or *rounds*, denoting named regimes of auction control.

3.1. Parameters

Static auction policies are characterized by predefined *parameters*, conditioning the criteria for admitting bids, matching offers, and summarizing state information in quotes. Extension of the auction system to implement a novel mechanism generally requires introducing new parameters, or new values for existing parameters,

along with new software modules selected by the corresponding parameter values. For example, one commonly adopted element of bidding policy is a "beat-the-quote" (btq) rule, which requires that any new or revised offer meet or exceed the standing offer (as represented by the quote), in some well-defined way (which might further depend on the bidding format). Wurman et al. [21] describe this policy element, as well as many others straightforwardly encoded as AB3D parameters.

In the AB3D scripting language, parameter values are initialized and revised through *assignment* statements, expressed using the *set* keyword. The statement

`set` *param expr*

dictates that the parameter *param* be (re)assigned the value of expression *expr*. For example, the statement

```
set bid_btq 1
```

activates the beat-the-quote requirement, thus instructing the auction that bids must be favorably compared with the price quote as prerequisite to admission to the order book.

Parameters may be implicitly defined through default values; others must be specified explicitly. As the parameters are not entirely orthogonal, some may be relevant or required conditional on the values of others. Similarly, AB3D may disallow certain value combinations, halting or behaving in an unspecified way when predefined constraints are violated.

Assignments may appear unconditionally as initializations or reassignments as part of a control structure, or conditionally as part of rules. The latter facility provides for qualitative modification of auction policy while the auction is active.

3.2. Rules

Whereas the static features of auction policy may be best characterized parametrically, such an approach is quite limited for specifying the dynamic control of auction events. For this purpose we employ a simple rule language, allowing that auction events be conditioned on arbitrary functions of auction state. These functions may also be parametrized, thus providing the benefits of both constructs.

An AB3D rule takes the form:

when—while **(***conditions***)** {*actions*}

In this rule, *conditions* is a conjunction of boolean-valued predicate expressions, written as a sequence of such expressions separated by `and` keywords, enclosed by parentheses. An individual predicate expression evaluates to *true* or *false* depending on the current auction state. If all are true, the rule is triggered, and *actions* are executed. Each action on the *actions* list (delimited by semicolons, enclosed in curly braces) corresponds to an auction activity, such as quoting or clearing, or an internal event such as assigning a parameter. The keyword *when* or *while* designates whether a rule is to be invoked only on becoming true (*when*), or continually until the condition no longer holds (*while*).

Multiple rules with the same actions essentially represent a disjunction of their corresponding conditions. Arbitrary boolean combinations can thus be expressed in this manner.

Condition expressions reference *state variables*—either predefined auction state variables or user-defined script variables. Auction state variables represent summary measures significant to auction operation, designated by auction developers for exposure to the auction script interpreter. Examples of state variables routinely provided by AB3D auctions include the current time (`time`), the last time a clear was executed (`lastClearTime`), and the number of bidders currently eligible to bid (`numBuyers` and `numSellers`). A typical condition expression using such variables would be to compare the current time with some other state variable. For example, an auction can specify a predefined duration (say, 500 seconds) with a rule of the form:

```
when (time ≥ auctionStartTime + 500)
  {close}
```

Though the previous rule may be slightly more intuitive than a parameterized representation, it could certainly be captured under parameterization without difficulty. Consider an auction that requires a clear to be performed whenever a bid is admitted (causing the state variable `validBid` to become *true*), but only before a predefined period of time:

```
when (validBid and
      time ≤ auctionStartTime + 500)
  {clear}
```

If this pattern of conditions is sufficiently common, then it too could be captured in a dedicated parameter. However, as we consider further state variables, the number of boolean combinations grows exponentially, as would the number of parameters required to capture the policy possibilities. The rule language enables this expressiveness without the associated blowup in primitives.

Periodic events are a common feature of auction mechanisms, as in the call market example provided in Section 6. In the AB3D scripting language, internal state variables referencing the last execution time of any given action provide a natural way to specify periodic events. For example, an auction that performs a clear every 100 seconds would include the following rule:

```
when (time ≥ lastClearTime + 100)
  {clear}
```

Assignment within a rule action is a powerful construct, enabling dynamic modification of auction behavior. For ex-

ample, the US Treasury auctions its "T-bills" in a uniform-price auction, after which winning bidders may trade the bills in a secondary market [1]. The script listed in Figure 1 defines such a two-phase mechanism, comprised of an ascending auction for the first 1000 seconds, followed by a continuous double auction (CDA) [4] for the second 1000.

```
defAuction twoPhase {
  set bid_btq 1
  set pricing_fn uniform
  set pricing_k 0
  when (time = auctionStartTime + 1000)
    {clear; quote;
     set pricing_fn chronological;
     set bid_btq 0}
  when (time ≥ auctionStartTime + 1000
        and validBid)
    {clear; quote}
  when (time = auctionStartTime + 2000)
    {close}
}
```

Figure 1. A two-phase auction: ascending followed by CDA aftermarket.

Our two-phase example employs several parameters controlling the clearing policy for a multiunit auction. The `pricing_fn` parameter identifies the criterion to be used for determining prices. A value of *chronological* indicates that price is determined according to the relative submission times of the matching bids comprising an exchange. Let p_0 be the price of the earlier bid, and p_1 the price of the later (by the fact that they match, we know the buy price is at least as great as the sell, but either could be earlier). The value $k \in [0, 1]$ of the `pricing_k` parameter dictates how the transaction price, p_*, is selected from the compatible range:

$$p_* = p_0 + k(p_1 - p_0).$$

For CDAs, a new bid transacts with a standing bid in the order book at the price specified by the standing bid. Thus, the CDA policy has chronological pricing with $k = 0$.

A *uniform* pricing function produces a single price governing a collection of simultaneous exhanges. In general (for a multiunit auction with divisible offers) there will be a range of possible clearing prices [19], delimited by the *bid* and *ask* quotes. Designating the bid by p_0 and the ask by p_1, the k-double auction [16] selects within this range according to the linear interpolation above. Note that we economize on parameters by reusing the interpolation parameter k for both the uniform (for which it was named) and chronological cases.

3.3. User-Defined Variables

User-defined variables add a degree of flexibility to the language, allowing for more general forms of auction control structures. For example, user-defined variables may be used as counters in looping constructs, with each iteration incrementing or decrementing the variable until a target value is achieved. Such a construct can be used to specify a predefined number of periods comprising the auction. The following set of declarations and rules would generate seven rounds of 60 seconds.

```
declare userTime auctionStartTime
declare counter 0
when (time ≥ userTime + 60)
  {set userTime time;
   set counter counter + 1;}
when (counter=7) {close}
```

By introducing and assigning user-defined variables (using the same `set` construct used for parameters), script writers can implement arbitrary computable functions.

Often an auction will have variables that are defined for each bidder, for example a user-specific credit limit. For this special case of user-defined variables, AB3D provides a bidder-indexed array construct, created (and optionally initialized to a value) with a statement of the form:

```
bidderAttribute ATTRIBUTENAME [val]
```

Bidder attributes can be referenced either by the bidder ID: `ATTRIBUTENAME(ID)`, or if the rule is associated with a bid submission by the keyword `bidder`: `ATTRIBUTENAME(bidder)`. For example, we can declare a credit limit and assign every bidder an initial value of 99 with the following statement: `bidderAttribute credit 99`.

3.4. Bid Rules

In the same way that we found rule-based methods necessary to reduce the complexity of parametrization in the case of auction control logic, we have found instances where it makes sense to allow rule-based definition of bid admissibility requirements. In addition to the parametrized bid rules provided by the AB3D scripting language, users may use the `bidRule` statement as a functional method of defining bidding requirements. For example an auction may allow only a single bid per user in a given round. To

achieve this, a script may include the bidderAttribute `hasbid` which is set to 1 on valid bid submission, along with the following bid rule: `bidRule(hasbid(bidder))`.

4. Composite Auctions

Often it is useful to define a group of loosely coupled auctions, which operate independently except for some specified points of interaction. For example, in a *simultaneous ascending auction* [8], a collection of related auctions is synchronized in time, yet each separately processes bids, issues quotes, and clears to produce transactions. In some versions, member auctions may further coordinate through *activity rules*, in which eligibility to bid in one auction may be conditioned on status in another.

To support such groupings, AB3D provides a general construct for specifying *composite auctions*, and facilities for coordinating their constituent *subauctions*. We define and name composite auctions using the keyword `superAuction` (in lieu of `defAuction`), and use the label `subAuction` to define an enclosed subauction.

4.1. Rounds

The `defRound` construct may be used to define commonly used sets of rules and parameters within an auction in a manner analogous to an auction definition. Once a round is defined within a script, a subauction can load the associated set of rules and parameters with the statement `set round ROUNDNAME`. Once the round parameter has been set, the subauction will admit bids only during an invoked round of the superauction, and will do so according to the rules defined by ROUNDNAME. A round is invoked by the superauction with the `round` action and terminated with `endRound`.

4.2. Synchronization

The superauction may require rules conditioned on predicates that must hold true across all subauctions. To achieve this we provide the higher level predicate `synchronized`, indicating that the predicate must hold true in all subauctions for the corresponding rule to be invoked. For example, the following rule would terminate a round on bidder inactivity of 50 seconds across all subauctions:

```
when(synchronized
     (time ≥ lastBidTime + 50)
  {endRound}
```

4.3. Shared Variables

Auctions often impose bidding eligibility conditions, for example credit limits or activity rules as mentioned above. In a composite auction, the eligibility conditions will typically depend on activity across all subauctions. To support the required information flow, AB3D provides a *shared variable* facility for composite auctions. To enforce credit limits, for example, subauctions must compare a bid offer amount to the current limit before admission, and if the bid is valid, debit the limit appropriately before bids from the same bidder are considered at other subauctions. The credit limit variable must thus be shared across subauctions.

In general, an AB3D rule may both depend on and modify a variable defined at the superauction scope. To ensure that no other subauctions test or modify the given variable while the rule is being processed, we provide a `lock` construct, which enforces serialization of rules dependent on a given variable across all subauctions. If a rule containing the predicate `lock(VAR)` is evaluated, all rules dependent on `VAR` will not be processed until the initial rule has been fully processed. For bid rules containing the lock construct, the lock will be invoked on all rules containing the validBid predicate. For example, the statement `bidRule(lock(bidderEligibility) and bidderEligibility(bidder) ≥ 1)` will lock the bidderEligibility variable when rules containing `validBid` are invoked, and will require positive bidderEligibility for a bid to be admitted. Such a bid rule combined with the rule

```
when(validBid)
  {set bidderEligibility(bidder)
       bidderEligibility(bidder) - 1;}
```

will lock, test, and decrement the bidderEligibility variable when a bid is admitted.

5. AB3D Operation

Figure 2 depicts the main functional components of AB3D, along with the information flows among them. The auction engine interprets the auction script, modifying the behavior of the message processor and order book according to specified parameters and rules. The message queue serves to synchronize bids and auction events, ensuring that bids are processed in the order received, and in the correct temporal relation to scheduled actions such as price quotes and clears. Information passes to agents through a data cache, designed to conserve communication bandwidth at the primary message processor by routinely pushing commonly needed data.

On launch of an auction script, the order book and message processor are initialized according to parameter set-

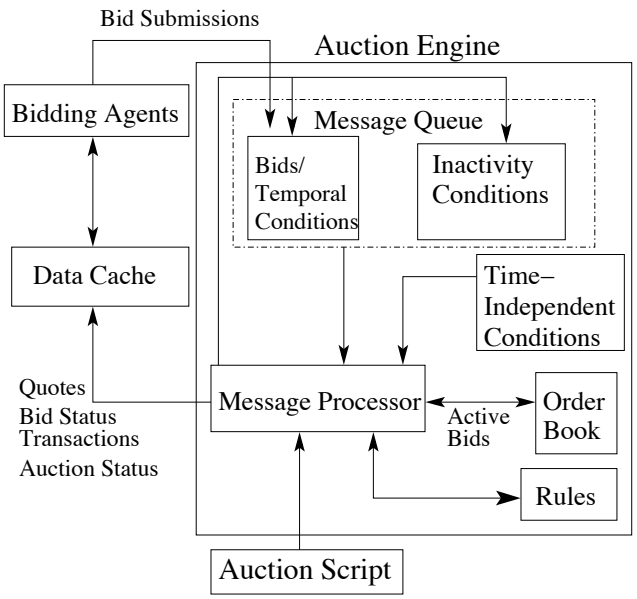

Figure 2. AB3D architecture.

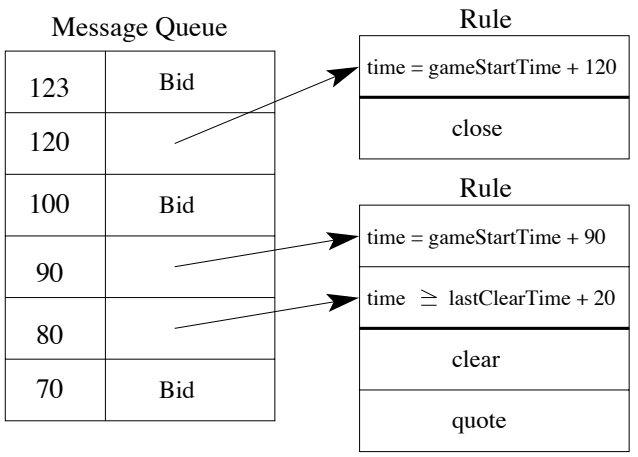

Figure 3. The message queue sequences bids and potential auction trigger events.

tings. The interpreter parses the rules, generating data structures employed by the condition matching process. To ensure timely triggering of rules when their conditions become satisfied, we provide special recognition procedures for three categories of conditions:

1. *temporal* conditions
2. those referencing *inactivity properties*,
3. *nontemporal* conditions.

Each requires its own methods based on the ways in which such conditions can become satisfied.

Conditions involving temporal predicates are organized in the same queue employed to process bids. On initial reading of the auction script, the interpreter calculates the time that each such condition will become true. A corresponding message object timestamped with this time is inserted into the queue, as illustrated in Figure 3. The message processor continually monitors the queue, processing the earliest-timestamped message in turn, with ties broken arbitrarily, as long as this timestamp precedes the absolute clock time. The *auction logical time* is the earlier of this clock time, and the timestamp of the in-process message, if any.

To process a bid message, the AB3D engine verifies that the bid is admissible according to current policies (specified in parameters and bid rules), and if so, maintains them in the active order book. Outcomes of bid processing (i.e., admittance or rejection notices) are transmitted to the bidder through the data cache.

To process a rule-condition message, the engine evaluates whether the associated condition is currently true. If so, it checks the status of other conditions of the rule (if any), and if all hold, executes the associated rule actions. Otherwise, for all time-dependent conditions of the rule that do not hold, it recalculates the earliest logical time at which each may become true, and inserts corresponding messages with the respective timestamps.

Conditions dependent on inactivity (e.g., bidder inactivity would be the length of time since the last admitted bid) are maintained in separate queues within the Message Queue, one for each inactivity property, with predicates ordered in ascending values of inactivity. A pointer in each queue is maintained to the condition with shortest inactivity that has not expired. The inactivity period of the next unexpired inactivity condition is added to the timestamp of the last activity time (e.g. the time of the last admitted bid), producing a scheduled processing time. The entire rule associated with an inactivity condition will be delivered to the Message Processor if this timestamp precedes the timstamp of the next message with absolute timestamp. For example, in Figure 3 the predicate `time ≥ lastClearTime+20` is scheduled to be processed at time 80, indicating that the last clear operation was performed at time 60.

To handle non-temporal conditions, the system maintains an index of referenced nontemporal state variables, and checks associated rule conditions whenever these are modified. At such time, the full set of conditions are evaluated, with execution of actions if warranted, similar to the procedure described for temporal state variables above.

6. Example Scripts

Call markets and continuous double auctions are common mechanisms to the financial world. A continuous dou-

ble auction clears any time that matching bids exist in the order book, and often does not include any bidding requirements. A CDA may therefore be defined by the following script:

```
defAuction CDA {
  set pricing_fn chronological
  set pricing_k 0
  when (validBid) {clear; quote}
}
```

A clear performed when no matching bids exist in the order book has no effect on the state of the order book, therefore the clear operation may be attempted whenever a successful bid is admitted.

A call market is a double auction with periodic uniform clearing. The following is the AB3D scripting language representation of this auction, assuming a clear period of 30 seconds and a clearing price at the midpoint between the bid and ask quotes:

```
defAuction callMarket {
  set pricing_fn uniform
  set pricing_k .5
  when (time = auctionStartTime + 30)
    {clear; quote}
  when (time = lastClearTime + 30)
    {clear; quote}
}
```

We now present an example of a more complex auction mechanism. Consider again the survival auction, proceeding in rounds, with the lowest bidder eliminated in each round until a single bidder remains. Two rules capture the dynamic behavior of the auction: one specifying the periodic elimination of bidders, a second specifying the condition for terminating the auction. Note that in the this example, `lowBuyer` is an exposed parameter pointing to the buyer with the weakest standing bid.

```
defAuction survival {
  set buyers allBidders
  set bid_btq 1
  set pricing_fn chronological
  set pricing_k 0
  declare roundTime time
  when (time ≥ roundTime + 600)
    {quote;
     set buyers buyers-lowBuyer;
     set roundTime time}
  when (numbuyers = 1) { close}
}
```

Alternatively, the round construct may be used to generate this behavior, where one bidder is eliminated at the end of each round until a single bidder remains.

To show the reader how rounds may be used in composite auctions, the following script defines such an auction intended to be a simplification of the FCC spectrum auctions. The superauction contains two subauctions, with eligibility rules restricting bidders to bid on only as many units as they were bidding on previously:

```
superAuction simultaneousAscending {
  bidderAttribute elig 2
  bidderAttribute nextElig 0
  defRound SARound {
    set pricing_fn uniform
    set pricing_k 0
    set bid_btq 1
    bidderAttribte valid 0
    bidRule(lock(elig) and
            elig(bidder)≥1)
    when(validBid and valid(bidder)=0)
      {set valid(bidder) 1;
       set elig(bidder) elig(bidder)-1;
       set nextElig(bidder)
           nextElig(bidder)+1}
  }
  subAuction auction1
    {set Round SARound}
  subAuction auction2
    {set Round SARound}
  when(synchronized(
    time ≥ lastBidTime + 100))
    {set elig nextElig;
     set nextElig 0;
     endRound}
  }
  declare roundnum 1
  while(roundnum < 4)
    {round;
     set roundnum roundnum + 1}
  when(roundnum = 4){
    {clear; close}
}
```

7. Conclusions

The increasing sophistication of auction mechanisms, coupled with further development in automated bidding agents, demands a structured method of auction representation. The AB3D scripting language provides a flexible medium for specifying a broad range of auction mechanisms. Its main innovation is rule-based invocation of auction events and policy revisions, supporting dynamically flexible auction behavior. By combining orthogonal parameters with rule-based control, we achieve many of the advantages of formal auction specification in a general and ex-

pressively convenient environment. We illustrate the use of our auction scripting language with a variety of simple examples, including standard mechanisms and innovative designs from the literature.

The scripting language is one important facility of our new configurable auction server, AB3D. Although the system as a whole remains a work in progress (e.g., we are currently working to extend it to support multiattribute auctions), we have implemented and used several mechanisms specified by AB3D scripts, for example the three auction types employed in the TAC Classic travel market [18], and another employed in a different resource-allocation game [2]. AB3D also supports specification of configurations of multiple auctions and the surrounding market environment (e.g., goods, agent endowments, and preferences), and thus provides a useful tool for our research on market mechanisms and agent strategies.

Acknowledgments

Our development of the auction scripting language was assisted by other members of the AB3D project, especially Kevin O'Malley, Daniel Reeves, and Shih-Fen Cheng. Some of the underlying ideas were conceived by the second author at TradingDynamics, Inc., with benefit of discussions with Yoav Shoham, Eithan Ephrati, and others. This work was supported in part by NSF grant IIS-0205435.

References

[1] S. Bikhchandani and C.-F. Huang. The economics of Treasury securities markets. *Journal of Economic Perspectives*, 7:117–134, 1993.

[2] S.-F. Cheng, M. P. Wellman, and D. Perry. Market-based resource allocation for information-collection scenarios. In *IJCAI-03 Workshop on Multiagent for Mass User Support*, Acapulco, 2003.

[3] D. Cliff. Explorations in evolutionary design of online auction market mechanisms. *Electronic Commerce Research and Applications*, 2:162–175, 2003.

[4] D. Friedman and J. Rust, editors. *The Double Auction Market*. Addison-Wesley, 1993.

[5] Y. Fujishima, D. McAdams, and Y. Shoham. Speeding up ascending-bid auctions. In *Sixteenth International Joint Conference on Artificial Intelligence*, pages 554–559, Stockholm, 1999.

[6] V. Krishna. *Auction Theory*. Academic Press, 2002.

[7] A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce: A European Perspective*, pages 19–33. Springer-Verlag, 2000.

[8] P. Milgrom. *Putting Auction Theory to Work*. Cambridge University Press, 2004.

[9] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[10] D. C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, 2001.

[11] S. Phelps, S. Parsons, P. McBurney, and E. Sklar. Coevolution of auction mechanisms and trading strategies: Towards a novel approach to microeconomic design. In *GECCO-02 Workshop on Evolutionary Computation in Multi-Agent Systems*, pages 65–72, 2002.

[12] D. M. Reeves, M. P. Wellman, and B. N. Grosof. Automated negotiation from declarative contract descriptions. *Computational Intelligence*, 18:482–500, 2002.

[13] J. A. Rodriguez-Aguilar, F. J. Martin, P. Garcia, and C. Sierra. Competitive scenarios for heterogeneous trading agents. In *Second International Conference on Autonomous Agents*, pages 293–300, Minneapolis, 1998.

[14] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[15] T. Sandholm and S. Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145:33–58, 2003.

[16] M. A. Satterthwaite and S. R. Williams. Bilateral trade with the sealed bid k-double auction: Existence and efficiency. *Journal of Economic Theory*, 48:107–133, 1989.

[17] M. P. Wellman. Online marketplaces. In M. Singh, editor, *Practical Handbook of Internet Computing*. CRC Press, 2004.

[18] M. P. Wellman, A. Greenwald, P. Stone, and P. R. Wurman. The 2001 trading agent competition. *Electronic Markets*, 13:4–12, 2003.

[19] P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.

[20] P. R. Wurman, M. P. Wellman, and W. E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second International Conference on Autonomous Agents*, pages 301–308, Minneapolis, 1998.

[21] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35:304–338, 2001.

[22] P. R. Wurman, M. P. Wellman, and W. E. Walsh. Specifying rules for electronic auctions. *AI Magazine*, 23(3):15–23, 2002.