

# Auctions without auctioneers: distributed auction protocols

Marc Esteva \* and Julian Padget \*\*

<sup>1</sup> Artificial Intelligence Research Institute, IIIA  
Spanish Council for Scientific Research, CSIC  
08193 Bellaterra, Barcelona, Spain.  
marc@iia.csic.es

<sup>2</sup> Department of Mathematical Sciences  
University of Bath, BATH BA2 7AY, UK  
jap@maths.bath.ac.uk

**Abstract.** It is quite natural for electronic institutions to follow the structure of their physical counterparts. However, this is not always appropriate or desirable in a virtual setting. We report on the prototyping of an alternative architecture for electronic auctions based around the concept of an *interagent* and building on the considerable body of work in the distributed algorithms literature to plot a path toward *resilient* trading frameworks. In particular, we have adapted the classical Leader Election algorithm for resolving bids in a generic auction scheme as well as identifying the factors which differentiate the physical auction protocols in such a way that new auction protocols can be plugged into the scheme by the specification of the relevant (sub-)processes. We have used the process algebra called the  $\pi$ -calculus to specify both the generic scheme and the specific protocols of first-price, second-price, Dutch and English. The bid resolution process has been prototyped in Pict and is now going to be integrated into the FishMarket electronic auction house.

**Keywords:** electronic commerce, auctions, distributed programming, process calculi.

---

\* This work has been partially supported by ESPRIT LTR 25500-COMRIS Co-Habited Mixed-Reality Information Spaces project and Marc Esteva enjoys the CIRIT doctoral scholarship FI/99-00012

\*\* Julian Padget's work has been partially supported by EPSRC grant GR/K27957 (<http://www.maths.bath.ac.uk/~jap/Denton>) and the CEC/HCM VIM project, contract CHRX-CT93-0401 (<http://www.maths.bath.ac.uk/~jap/VIM>)

## 1 Introduction

Electronic commerce has become more and more important with the growth of the Internet. In particular, auctioning has become one of most popular mechanisms of electronic trading, as we can see from the proliferation of on-line auctions on the Internet. Multi-agent systems appear to offer a convenient mechanism for automated trading, due mainly to the simplicity of their conventions for interaction when multi-party negotiations are involved. AI researchers have been interested in two areas: auction marketplaces and the trading agents' strategies and heuristics [5, 18, 20]. Apart from web-based trading, auctions are the most prevalent coordination mechanism for agent-mediated resource allocation problems such as energy management [20, 19], climate control [6], flow problems [6], computing resources [4], public monopolies [1] and many others [2].

From the point of view of multi-agent interactions in auction-based trading, the situation is deceptively simple. Trading within an auction house demands that buyers merely decide an appropriate price to bid, and that sellers essentially only have to choose the time to submit their goods.

The work related here is a continuation of the FishMarket (FM) project [17, 16]. The goal of FM is to develop means for the specification and design of electronic institutions, where by *electronic institution* we mean an organization [12] where the agents can interact following some protocols. Commonly, the institution is formed from a set of connected scenes and the agents can move from one to another when they satisfy some institution-defined conditions. The scenes are the basic elements of the institutions and it is there where the interactions between agents take place. Each scene has its own protocol that defines the interaction between the agents and the agents within a scene have to observe this protocol.

As an example of an electronic institution a group at IIIA have implemented the FishMarket [21]. The FM is an electronic auction house used as a test-bed for trading agents. The FM has six scenes, two for admissions (buyers and sellers), one for the staff, two for settlements and the auction room itself.

In this paper we focus on the auction room, and especially the process used for the resolution of bids. In all the versions of the FM to date, this process has been carried in a centralized manner. While that corresponds to the physical reality of auctions, it is not necessarily an appropriate model in a computing context due to two problems that are not common in physical situations (*pace* telephone bidders!): breakdown of processes or communications — so-called stopping failures — and intermittently faulty processes or communications, leading to unreliable messages — so-called Byzantine failures [7]. As a first step to addressing these problems we here present a distributed solution to which other techniques may later be added to handle resilience issues, which effectively does away with the auctioneer, thus removing a central single point of failure. In some sense the interagents — the market interfaces for the buyers —, as we shall see later, are really replications of the auctioneer, in common with the architecture proposed in [3].

In the current FM all the bids are submitted to an agent called the auctioneer who controls the whole auctioning process and determines the result of the round. From this point of view, the resolution of the bidding protocol is centralized. What we do in this

new version is distribute this process among the buyers' market interface agents (called interagents) — see Figure 1. Thus, the auctioneer sends the buyers (via the interagents) the information about the lot and then waits until one buyer interagent sends it the result of the round. During the intervening period the buyer interagents resolve the bids using a distributed protocol. Thus, the work of the auctioneer is reduced to:

- controlling which buyers participate in the auction,
- starting the rounds by sending the information on the lots
- waiting for the result of the round

We have two motivations: to have another way of applying auction protocols and to have a way to avoid the auctioneer becoming a bottleneck. With this new mechanism, the load of messages is distributed between all the buyers. The idea is that this algorithm can be applied to an existing auction house, providing an alternative way to deliver different auction protocols.

The basis of the distributed approach is the Leader Election algorithm [8]. It can be understood simply as an algorithm for choosing one processor in a network of many and the version we have adapted here derives from that used in token ring networks, where it is used to regenerate the token.

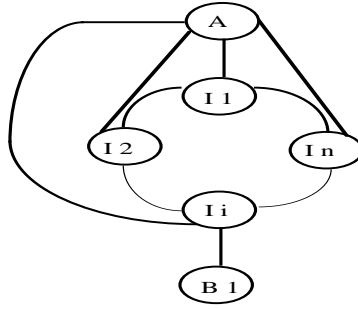
In the next section (2) we explain the new organization in the auction room in order to apply the algorithm. In section 3 we give a brief summary of  $\pi$ -calculus. In section 4 we outline how to apply the Leader Election algorithm in auction protocols. Finally (sections 5-9) we explain how to use this algorithm for a range of auction protocols and give their specification in  $\pi$ -calculus too.

## 2 Organization of the auction room

On first looking at the auction room we can see that we have two kind of agents, the auctioneer and the buyers. The auctioneer is an institutional agent who controls the correct running of the auctions. It controls when buyers enter and leave, starts the round and receives the result of it. Finally, it declares the end of the auctions when it has no further lots to auction.

We have added another kind of institutional agent, the so-called *interagent*, [9] an autonomous software agent which mediates interactions between a buyer and the agent society wherein it is situated. An interagent is connected to a buyer and abstracts it from communication problems. Thus, each buyer only has to communicate with its interagent so it can focus on its strategies for bidding. The use of an interagent also has advantages for the institutions, because the interagent also forces the buyer to follow the auction room protocol. For example, the interagent will prevent the buyer from making a bid between rounds. The interagents also implement the resolution of the bidding protocol. Previously, this task was done by the auctioneer, whereas now it is distributed among the interagents, which are linked in a ring in order to apply the particular version of Leader Election we have chosen [8].

An important point here, is that the change in the process of bidding resolution does not affect the buyers. It is an internal change of the system. This process is opaque to the buyers which cannot see the information passed between interagents in order to resolve



**Fig. 1.** Organization of the agents in the auction room

the bidding protocols. In other words, from the point of view of the buyers there is no difference between centralized bid resolution and the new scheme.

The buyers are the participants in the auctions. As we have said, each one is connected to one interagent in order to communicate with the other agents. Of course, we cannot specify the buyers here because each one is potentially unique. All we can define are the channels with which the buyer will be supplied in order to communicate with its interagent, the messages that it will receive and when it is allowed to submit a bid or leave the system. Therefore, in the auction room there is one auctioneer, a set of buyers and one interagent for each buyer (see Figure 1).

We will focus on the specification of the distributed resolution of the bidding protocol. We omit the specification of the auctioneer but note that its principal work now is to send the information about the lots at the start of each round.

### 3 The $\pi$ -calculus in brief

The main features of the  $\pi$ -calculus—and those necessary to read the remainder of this paper—are the means to read and write information over channels, the creation of channels, and parallel, alternative and sequential composition. Terms in the  $\pi$ -calculus are described as prefixes followed by terms, which is intentionally a recursive definition. Syntactic details are outlined below<sup>1</sup>.

#### 3.1 Summary of $\pi$ -calculus syntax and semantics

In order to keep this paper to a reasonable length, we cannot provide a full introduction to the  $\pi$ -calculus, limiting ourselves instead to this summary. For more information, the interested reader is referred to Pierce's excellent article [14] and subsequently to Milner [10] and the wider literature [11].

<sup>1</sup> A word of warning: this description should not be taken as definitive, since there are numerous interpretations which vary slightly in details of syntax, and sometimes of semantics. It does however represent  $\pi$ -calculus adequately for the purposes of the discussion in this paper.

- $x(y)$ : reads an object from channel  $x$  and associates it with the name  $y$ . This operation blocks until the writer is ready to transmit. The scope of  $y$  is limited to the process definition in which  $y$  occurs. Channel names, on the other hand may be local (see  $\nu$  below), parameters to process definitions (see below), or global.
- $\bar{x}(y)$ : writes the object named by  $y$  to the channel  $x$ . This operation blocks until the reader is ready to receive.
- $\nu x \dots$ : creates a new channel named  $x$ . The scope of  $x$  is limited to the  $\nu$  expression, but the channel may be passed over another channel for use by another process. For example, a common idiom is to create a channel using  $\nu$ , transmit it to another process and then wait for a reply on that channel:

$$\nu (x) \bar{y}(x, \text{question}) . x(\text{answer})$$

- $P \mid Q$ : the terms  $P$  and  $Q$  behave as if they are running in parallel. For example,  $\bar{x}(1) \mid \bar{y}(2)$  outputs 1 on channel  $x$  and 2 on channel  $y$  simultaneously.
- $P + Q$ : either one or the other (non-deterministic choice) of  $P$  and  $Q$  proceeds. Normally, the prefixes of  $P$  and  $Q$  are operations which could block, such as channel transactions, and this operation allows us to express the idea of waiting on several events and then proceeding to act upon one of them when it occurs. For example,  $x(a) + y(a)$  waits for input on channels  $x$  and  $y$ , associating the information in both cases with  $a$ . As soon as one branch of such an alternative succeeds, the others can be considered to have aborted (see discussion in section 3.2).
- $P . Q$ : the actions of term  $P$  precede those of term  $Q$ . For example  $x(y) . \bar{z}(y)$  reads  $y$  from  $x$  then writes  $y$  on  $z$ .

In addition, we include an ability to associate a term with a name — that is a definition — and furthermore, that in so doing a global channel is declared with that name, as in:  $P(x1, x2, x3) = \dots$ , which defines a process  $P$  taking a three-tuple. In practice this also means we have declared a channel  $P$  such that we may invoke the process  $P$  by writing a three-tuple to the channel named  $P$ . We will use this convention to obtain a form of parameterization, allowing us to pass processes as arguments (high-order processes) by passing the channel by which they are invoked. This syntactic convenience can be described primitively in the  $\pi$ -calculus but we omit these details here.

### 3.2 Events and choice

Of the many variants of the  $\pi$ -calculus, we chose as a starting point, the basic synchronous form as found in [10]. One of the essential properties of the kinds of markets we want to model is liveliness, which in practical terms means an event-based model. The non-deterministic choice (sum) operator has therefore been invaluable—although it also raises some interesting questions. To quote [10]:

The summation form  $\sum \pi_i . P_i$  represents a process able to take part in one—but only one—of several alternatives for communication. The choice is not made by the process; it can never commit to one alternative until it occurs, and this occurrence precludes the other alternatives.

When viewed as a mathematical description, for example, for the purpose of determining bisimilarity, there is no problem. However, when viewed as a program to run, there is an element of time and therefore sequence involved. Consider the process  $\overline{c_1}.P_1 + \overline{c_2}.P_2$ . If a message arrives on  $c_1$  just before one arrives on  $c_2$ , do we expect to become  $P_1$ , or do we expect a non-deterministic choice of  $P_1$  or  $P_2$ ? Certainly, we *can* become  $P_1$ , but most people (and the quote above can be interpreted to support this), would say we *should* become  $P_1$ . If not then the  $\pi$ -calculus would be a difficult tool indeed, requiring many synchronizations to enforce this natural behaviour, and these synchronizations would generally have no counterpart in a “real” program. In the following descriptions we have assumed that the natural interpretation is the case, this is, choices are determined as and when messages arrive on channels.

A further issue, of wanting to give priority to one channel over another, is addressed by the definition of the **test/0** process, which is much used later on. This process is used to check if there is information waiting to be read on a channel but without blocking the process attempting to read.

$$\begin{aligned} \text{test/0}(\text{event}, \text{then}, \text{else}) = \\ \nu (c1, c2, c3) \\ \overline{c3} \langle \rangle \\ | \text{event}(). c2(). \overline{c1} \langle \text{then} \rangle + c2(). \overline{c1} \langle \text{else} \rangle \\ | c3(). \overline{c2} \langle \rangle . c1(x) . \overline{x} \langle \rangle \end{aligned}$$

The function tries to read from the channel *event* and if it succeeds writes on channel *then*, otherwise writes on channel *else*. The version presented here does not read any data from the channel *event* but we assume that we have other versions that do and writes it to *then*. We will differentiate the number of arguments passed by adding arity to the name of the process (following a Prolog convention). Then the function **test/0** is the one that does not pass information, the process **test/1** is the one that passes one item and so on.

## 4 Leader Election

Leader election is a distributed algorithm used in some kinds of networks to elect a leader. For example, it is used in a token ring network when the token is lost and it is necessary to generate a new one. The algorithm assumes that all the nodes are identical, except in each having a unique identifier and they have to select one node to generate the token, but only one because there can only be one token in the ring.

There are different versions for solving the Leader Election problem and we have based our work on the LCR version [8]. This algorithm uses only unidirectional communication and does not rely on knowledge of the size of the ring. Other algorithms use

more knowledge (equals more constraints) to reduce the complexity of the algorithm, but do not change it in essence.

It is presumed that the unique identifiers support an ordering so that the leader will be the process with the largest identifier. First of all, each node sends a message with its identifier around the ring. When a process receives a message there are three possible actions:

1. if the identifier in the message is greater than its own, it passes the message on.
2. if the identifier in the message is less than its own, it discards the message.
3. if the identifier in the message is equal to its own, it declares itself the leader.

Thus, only the process with the greatest identifier will receive again its message and it will declare itself as a leader. We can see that all the other messages will be eliminated because at some point they will arrive at a process with a greater identifier.

The important point of the algorithm that we have to bear in mind is that it only uses local information in each process and all of them are identical except in their identifiers. The processes do not have global information.

The next step is to see how we can apply this strategy to the auction protocols. The first and obvious point is that we have to compare the bids submitted for the buyers. The winner will be the buyer with the greatest bid. When a process receives a message, it will have to compare the bid in the message with its bid. There is one aspect that makes processing bids trickier than straight leader election: all the identifiers were different, but it is quite possible there will be equal (highest) bids posted. Namely, there is a collision if there is more than one bid at the greatest price.

We have to define first when the messages are generated and what they should contain. When to generate is obvious, the interagents will send a new message when the buyer makes a bid. For the second point is not enough just to send the bid: we need to know who has generated the message because it can be that more than one buyer makes the same bid and whether there are more buyers that have made a bid at the same value.

Thus the messages will have two fields:

1. a list of the identifiers of the buyers that have bid at price *bid*,
2. the bid itself

From the list we can learn who has contributed to the message and if there is more than one buyer bidding at that price.

Now we have to analyze what happens when an interagent receives a message from its neighbour. When the bid in the message is different from its own, it acts as in the Leader Election algorithm: it passes the message on if the bid is greater and discards it if it is lower than its own.

The important point is what happens when it receives a message with a bid equal to its own. There are two possibilities:

1. *This is the message generated by itself* which implies that this interagent's buyer has made the greatest bid. However, this is not enough to elect itself as a winner because another buyer could have made a bid at the same value. To distinguish this case, it has to look at the list of identifiers in the first field of the message, and if

there is only its identifier, it can declare itself the winner, otherwise there has been a collision, in which case it will generate a collision message in order to inform the other interagents. We will explain later how to use collision messages.

2. *Or, it is a message generated by another interagent* which indicates that other buyer has made a bid at the same price and it *could* be a collision. This is not enough for declare a collision yet because another interagent, further round the ring, could have made a greater bid. It will only be a collision if this is the greatest bid. The problem is that the interagent has only local information and it only knows that it *could* be a collision. The interagent can only declare the collision when the message has made one complete round and it so it is sure that this is the greatest bid. All it can do is add its identifier to the list and pass on the message.

As we can see, no interagent eliminates a message with a bid equal to its own. Thus when there is a collision, this will be detected for all the interagents involved in it and each one of them will generate a collision message. Thus, after a collision, there will be one collision message for each buyer involved in the collision. In the version that we propose here, the interagents restart the round after a collision. Important remaining issues are how the collision messages are eliminated and how to ensure that each interagent receives only one collision message. The solution is that each collision message travels over the part of the ring from the interagent generating the message to the next interagent involved in the collision. Thus each interagent will receive one and only one collision message.

Another point is what should an interagent do when it knows that it has won a round. The answer is that it has to send a message around the ring in order to inform all the other interagents that the round is finished. An issue to note is *what* information should be passed on to the buyers. They could just be notified of the end of the round, but more likely they could be sent some information about the result, such as the price and/or the identity of the winner. Precise choices here depend on the conventions of the institution being modelled, but are not important otherwise to the discussion here. This message synchronizes all the interagents and subsequently the winner's interagent sends a message to the auctioneer to inform it of the result of the round and also that the round is finished.

Now we have described all the possible cases when an interagent receives a message. Hence, we can give a generic variant of the Leader Election for the resolution of bidding protocols (see Figure 2).

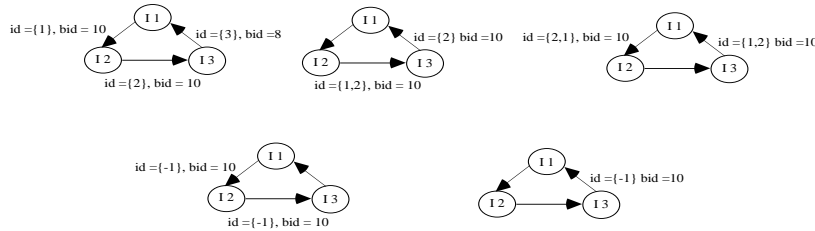
The last difficulty to address is what to do when no buyers make a bid. In this situation, no messages would be generated, so leading to deadlock, because each of the interagents will be waiting for messages. To avoid this, each interagent is required to generate a message for each round *unless* its buyer has not submitted a bid. These messages have a bid value of the negation of the identifier of the interagent. Thus if no one has submitted a bid, only the interagent with the lowest identifier will receive its message back. It will then detect that there have been no bids and it will notify the auctioneer that the lot is withdrawn.

Before explaining the duties of the interagent, we present an example in Figure 3. In this example there are three interagents, two of them which bid 10 and another which bids 8. This provokes a collision which is detected by both when the interagents re-

1. if the interagent receives an end of round message which it *did not* generate, it passes the message to the next interagent
2. if it receives an end of round message which it *did* generate, eliminate the message and send the result of the round to the auctioneer
3. if it receives a collision message in which it *was not* involved, it passes the message to the next interagent and restarts the round
4. if it receives a collision message in which it *was* involved, it eliminates the message and restarts the round
5. if it receives a message with a bid greater than its own, it passes the message to the next interagent
6. if it receives a message with a bid equal to its own and it *is not* its message it adds its identifier to the message and passes it to the next interagent
7. if it receives its own message back and it only contains its identifier in the first field, it is the winner of the round and it sends an end of round message
8. if it receives its own message back but there is more than one identifier in the first field, there is a collision and it generates a collision message
9. if it receives a message with a bid lower than its own, it eliminates the message.

**Fig. 2.** Specification of the generic bidding resolution protocol

ceived back their messages with more than one identifier. Then they generate a collision message, which has identifier  $-1$ . We can see that each interagent receives only one of them. The interagents which have participated in the collision wait until they receive another collision message. At that point, they eliminate the collision message and restart the round. The example finishes at that point but after that the interagents will restart the round.



**Fig. 3.** Example of collision resolution

In the next section we will give the specification of the interagent in each protocol. We will explain how to apply the algorithm outlined above to each one and the modifications to take account of their individual characteristics.

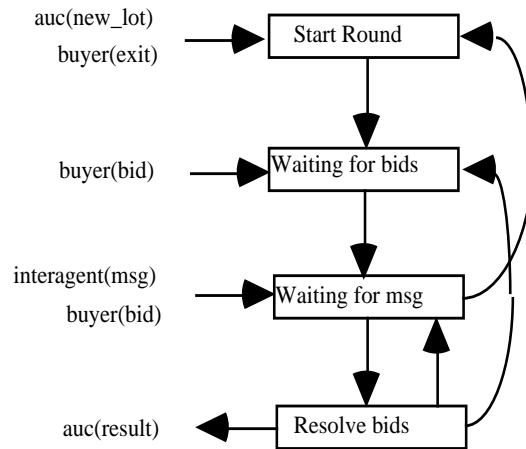
## 5 The Interagent

As we have said earlier, the interagent has two important functions, handling communication between buyers and the institution (external) and resolving the bidding protocols (internal).

The first function gives the buyer the communication infrastructure to enable participation in the auctions by passing to the buyer the information that it needs about the state of the auctions. For example, when a round starts, the characteristics of the lot offered, if he has won a round, etc.. Furthermore, it passes the buyer messages to the other institution agents. These are the bids of the buyer and when it wants to leave the system. It also checks that the messages of the buyer follow the protocol. For example it does not allow to the buyer to make a bid at the wrong time. The idea is to abstract buyer developers from communication problems, allowing them to concentrate on bidding strategies.

The second function which is independent of the buyers, consists of deciding who won a round or whether a lot is withdrawn, using the modified Leader Election protocol with some variations depending on the auction protocol. As we have said before the buyers are not allowed to see the information passed between interagents in order to resolve a round. The interagents have to be robust and have to incorporate security measures in order to protect them from malicious buyers. Otherwise, buyers could read the messages with the bids of the others buyers and then generate new bids out of sequence with help of additional information. This is a point to be borne in mind but here we focus just on the algorithm.

From the point of view of an interagent, a round is divided in four steps as we can see in Figure 4.



**Fig. 4.** The four steps of a bidding round

1. **Start round:** This step corresponds to the period between two rounds. In that period, new buyers are added to the auction and existing participants can leave. The step finishes when the interagent receives from the auctioneer the information of the next lot to be auctioned. This step is the same for each protocol.
2. **Waiting for bids:** This step can be seen as an initialization step. For each round it has a pre-determined time expressed in the lot information. The buyers have that time to make their first bid. Except in the English auction, that is the only moment that buyers are allowed to submit bids. When this step is finished we can be sure that each interagent has sent its neighbour a message. It is for that reason that we say that it is an initialization period.
3. **Waiting for next message:** In this step an interagent waits for a message from its source interagent (the previous one in the ring) or for a message from the buyer with a new bid. In the first case it passes to the next step in order to handle that message. If it receives a message from the buyer for a new bid it generates a new message and sends it to its destination interagent (the next one in the ring). In the protocols that we specify here, only in the English auction are the buyers allowed to submit bids at that moment. In the other three they can only submit bids at the second step and this step only consists in waiting for a message from the source interagent.
4. **Resolve bids:** This is the important point of the algorithm when an interagent uses the cases explained before with the last message received and its information to decide what it has to do. Subsequently, it returns to the previous step to handle the message.

As we can see, the first step is common for all the protocols and the others present some differences between each one. The interesting thing is that all of them follow the same pattern unless they have their own characteristics. We will focus on these three steps for each protocol.

The first step receives the information of the next lot from the auctioneer and calls the `WaitingBid` function of the corresponding protocol. This is when the buyers can leave the auction and new buyers can join in.

Before presenting the specification in  $\pi$ -calculus of each protocol we define the channels used for communication between one interagent and the others. These are:

- *in*: receipt of messages from its predecessor in the ring.
- *out*: transmission of messages to its successor in the ring.
- *b/int\_bid*: receipt of the bids from the buyer.
- *int/auc\_res*: transmission of the result of the round to the auctioneer.
- *int/b*: transmission of information to the buyer.
- *b/int\_exit*: receipt of message from buyer of desire to leave the auction room.

Apart from the channels, the interagent also keeps information in order to compare it with that in the incoming messages and the parameters of the actual lot in case there is a collision and it has to restart the round. Normally in the first case it keeps the last bid submitted by the buyer — but in the English auction more information is needed: we will explain this later.

First we give the specification of the protocols. We will begin with the complete specification of First-Price/Sealed-Bid, being the simplest. Then we will explain the changes, by reference to that for the Vickrey and Dutch protocols, because they are so similar to the first one. Finally we will explain in more detail the last and more complex English auction.

```

FP-WaitingBid(id, in, out, b/int_bid, int/auc_res, int/b, time) =
  ν (done, then, else)
   $\overline{int/b} \langle time \rangle$ 
  . FP-bid(b/int_bid, time, done)
    | done(bid)
    .  $\overline{out} \langle bid \rangle$ 
    . FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time)

FP-bid(id, b/int_bid, time, done) =
  ν (then, else, done2)
  delay(time, done2)
  | done2()
  . test/1(b/int_bid, then, else)
    | then(bid) .  $\overline{done} \langle bid \rangle$  + else(junk) .  $\overline{done} \langle -id \rangle$ 

FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time) =
  in(idi, n_bid)
  . FP-ResolveBids(id, bid, idi, n_bid, in, out, b/int_bid, int/auc_res, int/b,
    time)

```

## 6 First-Price/Sealed-Bid

The main characteristic of this kind of auctions is that you can divide it in two phases. There is a time for submission of bids and afterwards, analyzing the bids to choose a winner. For each lot there is given a time for the buyers to submit their bids and after that the interagents decide who is the winner. The winner will be the buyer who has submitted the greatest bid and this is the price that he will pay for the lot. The lots auctioned in this protocol are defined with one parameter which indicates the time that the buyers have to submit bids. The lots auctioned in this protocol are defined with one parameter which indicates the time that the buyers have to submit bids.

1. **Waiting for bids:** This step corresponds to the time that the buyers have for submitting bids. This process uses an auxiliary process called FP-bid to check if the buyer has submitted a bid in the time given. It will generate a message with the

value returned from this process and then passes to the next step of waiting for a message from its predecessor.

The FP-bid process first waits for the time specified using the `delay` process (definition not given here) which waits for the units of time that it receives as a parameter. After that it uses the function `test/1` in order to see if the buyer has made a bid. If it has, it returns the value of the bid, otherwise it returns the value `-id` which indicates that the buyer has not submitted a bid.

2. **Waiting for next message:** In this protocol this step only waits for one message from its predecessor in the ring because the buyers are not allowed to submit multiple bids. After reading from channel *in* it sends a message to the process `FP-ResolveBids` for further processing.
3. **Resolve bids:** This is the most complex and interesting function. It applies the algorithm explained before with its own bid and the last message received for the interagent. The algorithm here is exactly the same as explained above without changes.

## 7 Vickrey's auction

This protocol is very similar to the one before in the sense that it is also divided in two phases and with the same processes as the previous one. The sole difference is that the winner is the buyer who has submitted the highest bid *but* the price that he has to pay corresponds to the second highest bid. That implies adding a field to the messages to contain information about the second highest bid.

We do not give the specification because the only change from first-price is that the messages have one additional field. This field, corresponding to the second highest bid, is initialized to zero when an interagent generates a message. When an interagent passes a message on it has to compare this field with its bid and it updates it, if its own is higher. The only other change is when an interagent is the winner of the round. In that case it has to send the second bid to the auctioneer.

## 8 Dutch auction

The main characteristic of this protocol is that it is a descending price auction. The round starts with a high price descending until one buyer submits a bid. In real Dutch auctions it is the auctioneer who calls out the offers until one buyer bids. Here, each interagent sends independently the offers to its buyer<sup>2</sup>. While this confers several advantages in a distributed setting that cannot arise in the physical scenario, it does have the drawback that in contrast to a real auction, where when a buyer sees a offer he knows that no one has submitted a bid at a higher price, whereas here the buyer cannot be sure of this because each interagent is running independently from the others. This could change the bidding strategy of the buyers.

The parameters of this kind of auction are slightly different, being:

---

<sup>2</sup> This technique and its justification are presented in [13]

```

FP-ResolveBids(id, bid, idi, n_bid, in, out, b/int_bid, int/auc_res, int/b, time) =
{
  if car(idi) = -2
  {
    if n_bid = bid
    {
      if bid < 0
      {
         $\overline{int/auc\_res}$  ⟨0, 0⟩
      }
      otherwise
      {
         $\overline{int/auc\_res}$  ⟨id, bid⟩ |  $\overline{int/b}$  ⟨“Winner”⟩
      }
    }
    otherwise
    {
       $\overline{int/b}$  ⟨“End of round”⟩
      |  $\overline{out}$  ⟨-2, n_bid⟩
    }
  }
  elseif car(idi) = -1
  {
    if n_bid = bid
    {
       $\overline{int/b}$  ⟨“Collision”⟩
      | FP-WaitingBid(id, in, out, b/int_bid, int/auc_res, int/b, time)
    }
    otherwise
    {
       $\overline{int/b}$  ⟨“Collision”⟩
      |  $\overline{out}$  ⟨-1, n_bid⟩
      | FP-WaitingBid(id, in, out, b/int_bid, int/auc_res, int/b, time)
    }
  }
  otherwise
  {
    if n_bid = bid
    {
      if car(idi) = id
      {
        if  $|id_i| = 1$ 
        {
           $\overline{out}$  ⟨-2, bid⟩
        }
        otherwise
        {
           $\overline{out}$  ⟨-1, bid⟩
          | FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time)
        }
      }
      otherwise
      {
         $\overline{out}$  ⟨idi ∪ {id}, bid⟩
        | FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time)
      }
    }
    otherwise
    {
      if n_bid > bid
      {
         $\overline{out}$  ⟨idi, n_bid⟩
        | FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time)
      }
      otherwise
      {
        FP-WaitingMessage(id, bid, in, out, b/int_bid, int/auc_res, int/b, time)
      }
    }
  }
}

```

```

DA-bid(id, b/int_bid, actual_price, decrement, reserve_price, done) =
  ν (then, else, done2)
  int/bid ⟨actual_price⟩ . delay(1, done2)
  | done2()
  . test/0(b/int_bid, then, else)
  | then() . done ⟨bid⟩
  + else()
  . {
    if actual_price - decrement > reserve_price
    DA-bid(id, b/int_bid, actual_price - decrement, decrement, reserve_price,
    done)
    otherwise
    done ⟨-id⟩
  }

```

- *start\_price*: the starting price at which the interagents start sending offers to the buyers.
- *reserve\_price*: the minimum price at which the lot may be sold. If that price is reached the buyer loses the opportunity to for bid for it. If all the interagents reach this price without a bid being made, then the lot is withdrawn.
- *decrement*: the difference between successive offers.

The process that is different from First-Price/Sealed-Bid auction is FP-bid. All the other processes are identical except that the *time* parameter is replaced by the three parameters that capture the state of this protocol (given above).

The DA-bid process sends an offer to the buyer, then waits one unit of time and checks whether the buyer has submitted a bid. In that case it returns the actual price as the buyer bid, otherwise it repeats the process again. This function stops when the buyer submits a bid or the *reserve\_price* is reached.

## 9 English auction

This protocol is the most complex that we have specified and the one that presents the most differences with the others. Here bidding starts at a minimum price and the buyers submit increasing bids until all of them stop. Each buyer can submit as many bids as it wants before a winner is declared. In the real auctions the auctioneer says *going, going, gone* after a bid before declaring a winner. In order to model that here, before an interagent either declares itself the winner or detects a collision, its message has to make three rounds of the interagent ring.

This protocol starts with a descending bidding protocol as in the Dutch auction until one buyer submits a bid. After that the auction follows the pattern just described, with the bids going up.

In order to count the laps of the message with the greatest bid received, each interagent has to keep more information. It has to keep the last message that it has passed

EA-WaitingMessage(*id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price*) =

$$\left\{ \begin{array}{l} \text{if } l\_bid < 0 \vee counter = 2 \\ in(id_i, n\_bid) \\ \cdot EA\text{-ResolveBid}(id, bid, id_i, n\_bid, in, out, b/int\_bid, int/auc\_res, int/b, \\ \quad l\_id, l\_bid, count, start\_price, decrement, reserve\_price) \\ \text{otherwise} \\ in(id_i, n\_bid) \\ \cdot EA\text{-ResolveBid}(id, bid, id_i, n\_bid, in, out, b/int\_bid, int/auc\_res, int/b, \\ \quad l\_id, l\_bid, count, start\_price, decrement, reserve\_price) \\ + b/int\_bid(n\_bid) \\ \cdot \overline{out} \{id\}, n\_bid \\ \cdot EA\text{-WaitingMessage}(id, n\_bid, in, out, b/int\_bid, int/auc\_res, int/b, id, \\ \quad n\_bid, 0, start\_price, decrement, reserve\_price) \end{array} \right.$$

on because it is the one with the greatest bid so far, and the number of times that it has received it in order to count the laps.

There is another important point in which it differs from real auctions. There are two situations where the buyer is not allowed to submit bids but where they can be allowed later. The first one is when the buyer has not made a bid at the waiting for bids step. After that the buyer will be allowed to make bids if the interagent receives a bid from another one. The second situation is when the interagent receives a message for the third time. From the point of view of the interagent the round is over. Although there is one possibility it may not be: if another buyer submits a greater bid before it has received the message three times. In the real auctions this does not happen because the auctioneer declares the end of the rounds in a centralized way. When the auctioneer declares that a lot is withdrawn or there is a winner there is no possibility of continuing the round.

So, when an interagent receives a new message it has to compare it with the one that it has kept. If the bid is lower it discards the new message. If the bid in the new message is greater then it keeps it, it sets the counter to zero and it passes the message on.

We have to bear in mind that there can be more than one message in the ring with the same bid value because no interagent discards any message with a bid equal to its own. So, when an interagent receives a message with a bid equal to that in the message that it has kept it compares the identifiers to see if they are the same message. If not it passes the message on and waits for another one. If they are the same and the counter is less than two it passes the message on and increments the counter by one. If the counter is equal to two it checks if it is its own message. In that case it declares itself as a winner if there is only its identifier and the bid is greater than zero. It declares the lot withdrawn if the bid is lower than zero and a collision if there is more than one identifier in the message. If it is not its message then it passes the message on but its buyer will not be allowed to make more bids unless it receives a new message with a greater bid as we have explained above.

EA-ResolveBid( $id, bid, id_i, n\_bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ ) =

```

if  $car(id_i) = -2$ 
{
  if  $n\_bid = bid$ 
  {
    if  $bid < 0$ 
    {
       $\overline{int/auc\_res} \langle 0, 0 \rangle$ 
    }
    otherwise
    {
       $\overline{int/auc\_res} \langle id, bid \rangle \mid \overline{int/b}$  ("Winner")
    }
  }
  otherwise
  {
     $\overline{int/b}$  ("End of round")
    |  $\overline{out} \langle -2, n\_bid \rangle$ 
  }
}
elseif  $car(id_i) = -1$ 
{
  if  $n\_bid = bid$ 
  {
    EA-WaitBid( $id, in, out, b/int\_bid, int/auc\_res, int/b, start\_price, decrement, reserve\_price$ )
  }
  otherwise
  {
     $\overline{out} \langle -1, n\_bid \rangle$ 
    | EA-WaitBid( $id, in, out, b/int\_bid, int/auc\_res, int/b, start\_price, decrement, reserve\_price$ )
  }
}
elseif  $car(id_i) = l\_id \wedge n\_bid = l\_bid$ 
{
  if  $n\_bid > bid$ 
  {
     $\overline{out} \langle id_i, n\_bid \rangle$ 
    | EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count + 1, start\_price, decrement, reserve\_price$ )
  }

  elseif  $count = 2$ 
  {
    if  $|id_i| = 1$ 
    {
       $\overline{out} \langle -2, bid \rangle$ 
      | EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ )
    }
    otherwise
    {
       $\overline{out} \langle -1, bid \rangle$ 
      | EA-WaitingMessage( $id, in, out, bid, bm\_nbi2, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ )
    }
  }
  otherwise
  {
     $\overline{out} \langle id_i, n\_bid \rangle$ 
    | EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count + 1, start\_price, decrement, reserve\_price$ )
  }
}
elseif  $n\_bid = l\_bid \wedge count > 0$ 
{
   $\overline{out} \langle id_i, n\_bid \rangle$ 
  . EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ )
}
elseif  $n\_bid > bid$ 
{
   $\overline{out} \langle id_i, n\_bid \rangle$ 
  . EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, car(id_i), n\_bid, 0, start\_price, decrement, reserve\_price$ )
}
elseif  $n\_bid = bid$ 
{
   $\overline{out} \langle id_i \cup \{id\}, bid \rangle$ 
  . EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ )
}
otherwise
EA-WaitingMessage( $id, bid, in, out, b/int\_bid, int/auc\_res, int/b, l\_id, l\_bid, count, start\_price, decrement, reserve\_price$ )

```

One point here is that the identifiers in the list are always added at the end. So, in order to determine if two messages with the same bid are from the same buyer, it is enough to compare the first identifier of each one.

The last point to consider is that a buyer has always to submit a greater bid than the last that it received. Given this constraint, an interagent can only possibly receive a message that it has not generated with an equal bid to its own when its message is in the first lap around the ring. In this situation it has to add its identifier to the list in the first field of the message.

The first step is the same as in the Dutch auction, starting at one price and going down. Therefore, we will just specify the other two.

## 10 Related and future work

As we proposed at the beginning, we have presented a distributed method for the resolution of the classic bidding protocols. For that purpose we have distributed the task from the auctioneer to the interagents. Thus the load of messages is also distributed because in the centralized versions all the messages go from the auctioneer to each interagent or from each of them to the auctioneer.

With the use of interagents this very significant change can be done without affecting the buyers which can run without any knowledge of the way in which the bids are resolved. The interagents also define the communication infrastructure and thus the buyers can focus on bidding strategies.

The specification given has been satisfactorily implemented using Pict [15], a  $\pi$ -calculus interpreter, giving some confidence in the validity of the method, although without a directly executable specification, it does nothing to verify the specification.

We have given the specification of four protocols but we have defined the general steps in order to resolve bidding protocols in a distributed way. Thus it may simplify the specification of new auction protocols in a distributed manner. All that is necessary is the definition of the three steps for the new protocol.

Another important point is that the fact that in basing our algorithm on the Leader Election, we can use the theoretical results established for Leader Election. This includes the algorithmic improvements mentioned in the introduction and, more importantly, its combination with techniques for termination detection and handling of stopping and Byzantine failure, which is an important aspect of our future (implementation) work.

Our plans for the immediate future are an implementation in the framework of the FishMarket, so that we will have two ways of resolve the bidding protocols in our electronic auction house. The FM already uses interagents for communication with the buyers and with the other agents and so all have to do is modify them to handle bid resolution as outlined here. In consequence we can take benefit from all the other infrastructure of the FM in order to get our specification running easily.

## References

1. James Bushnell and Shmuel Oren. Two dimensional auctions for efficient franchising of public monopolies. Technical Report ERL-93-41, University of California, Berkeley, 1993.
2. S. Clearwater (ed.). *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific Press, 1995.
3. M. Franklin and M. Reiter. The Design and Implementation of a Secure Auction Service. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1996.
4. Ross A. Gagliano, Martin D. Fraser, and Mark E. Schaefer. Auction allocation of computing resources. *Communications of the ACM*, 38(6):88–102, June 1995.
5. Pere Garcia, Eduard Giménez, Lluís Godo, and Juan A. Rodríguez-Aguilar. Possibilistic-based design of bidding strategies in electronic auctions. In *The 13th biennial European Conference on Artificial Intelligence (ECAI-98)*, 1998.
6. B. A. Huberman and S. Clearwater. A multi-agent system for controlling building environments. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 171–176. AAAI Press, June 1995.
7. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
8. Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. ISBN 1-55860-348-4.
9. Francisco J. Martín, Enric Plaza, and Juan Antonio Rodríguez-Aguilar. An infrastructure for agent-based systems: An interagent approach. *International Journal of Intelligent Systems*, 1998.
10. Robin Milner. The Polyadic  $\pi$ -Calculus: a Tutorial. Preprint of Proceedings International Summer School on Logic and Algebra of Specification, 1991.
11. Uwe Nestmann. Calculi for mobile processes. available through <http://www.cs.auc.dk/mobility/>.
12. D. North. *Institutions, Institutional Change and Economics Performance*. Cambridge U. P., 1990.
13. J.A. Padget and R.J. Bradford. A  $\pi$ -calculus model of the spanish fishmarket. In *Proceedings of AMET'98*, volume 1571 of *Lecture Notes in Artificial Intelligence*, pages 166–188. Springer Verlag, 1999.
14. Benjamin C. Pierce. Foundational calculi for programming languages. In Allen B. Tucker, editor, *Handbook of Computer Science and Engineering*, chapter 139. CRC Press, 1996.
15. Benjamin C Pierce and David N Turner. Pict: A Programming Language Based on the Pi-Calculus. Technical Report 476, Indiana University, March 1997.
16. J.A. Rodríguez, F.J. Martín, P. Garcia, P. Noriega, and C. Sierra. Towards a formal specification of complex social structures in multi-agent systems. In Julian Padget, editor, *Proceedings of the workshop on Collaboration Between Human And Artificial Societies*, volume 1624 of *LNAI*, pages 289–305. Springer Verlag, 1999.
17. J.A. Rodríguez, P. Noriega, C. Sierra, and J.A. Padget. FM96.5 A Java-based Electronic Auction House. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology: PAAM'97*, 1997.
18. Hal R. Varian. Economic mechanism design for computerized agents. In *First USENIX Workshop on Electronic Commerce*, pages 13–21, New York, July 1995. USENIX.
19. Fredrik Ygge and Hans Akkermans. Power load management as a computational market. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, 1996.
20. Fredrik Ygge and Hans Akkermans. Making a case for multi-agent systems. In Magnus Boman and Walter Van de Velde, editors, *Advances in Case-Based Reasoning*, number 1237 in *Lecture Notes in Artificial Intelligence*, pages 156–176. Springer-Verlag, 1997.
21. The FishMarket Project. <http://www.iiia.csic.es/Projects/fishmarket>.