Assessment 4: Auctions

Maixent ASSI

28 April 2019

1. Description of the protocol combining the the Dutch and English auctions

In our abstraction, three (3) scenarios can be expected:

1. A ticket is sold resonably early enough, so that the English auction is entirely executed from end-to-end. The corresponding FIPA protocol is designed here.
2. A ticket is sold very close to the deadline. In this case, a dutch auction is immediately considered because of the time constraint. FIPA Dutch auctions protocol describes the setting for intelligent agents.
3. The ticket is sold starting with an English auction, but before the English auction completes, it is morphed into a Dutch auction to allow for the sale to quickly occur. Our protocol proposal covers this case.

We feel like the cases to focus on are when the time suggests a dutch auction(2) and when an english auction starts and swithes to a dutch auction(3). This latter case adds to scenario (1) above some continuous checks on whether the time lends itself to either an english or a dutch auction before the switch is made.

2. Description of the format and contents of all messages exchanged

We start with the message sent by the auctioneer before the ones sent by the bidder.

2.1 Messages sent by the auctioneer

We will not descripe the messages sent using ACL even though it is the communication standard of FIPA. Doing so will lengthen the description when in practice the tools we used gave us a high-level of abstraction. The auctioneer sends the following messages:

- Informs the bidders of the start of an auction. In ACL it would have been put this way **(inform :sender (agent-identifier :name-i) :receiver (set(agent-identifier :name-j)) :content “english auction started”)**. As we said to start we will describe them shortly instead of this length ACL-compliant form. From a practical standpoint, the auctioneer informed the bidders by **broadcasting** to them the start of the auction. The content of the message in our case was the type of the auction.

We get the form $.broadcast(tell, auction(AUCTIONTYPE))$ in AgentSpeak(L). It can be an english(AUCTIONTYPE=0) or a dutch auction(AUCTIONTYPE=1).

- Call for proposals. In AgentSpeak(L), this communication takes this form:

$.broadcast(tell, currentBid(CURRENTBID))$. The content of the message is current bid value the auctioneer is *shouting*. It corresponds to the **cfp** performative in FIPA ACL.

- Accepting or rejecting received proposals. We get the following AgentSpeak(L) syntax $.send(bidder_i, tell, accepted(BIDVALUE))$ or $.send(bidder_j, tell, rejected(BIDVALUE))$. We see that this time the **“.send”** function is used instead of **“.broadcast”** because it is directed to a specific agent. In ACL, it is done using the **accept-proposal/reject-proposal** performatives.
- Informing the participants who is the winner. Again, this is a typical broadcast in the form of $.broadcast(tell, winner(WINNER))$ sent to all the bidders. The information sent is the identity of the agent which won the auction.

- Request to the winner. This is a direct message to the winner, so no broadcast. Instead of the ACL **request** performative, this can be done in ".asl" with the "tell" performative because it acts on the belief. $.send(bidder_w, tell, request(PAYMENT))$ is the way it is done in AgentSpeak(L).

2.2 Messages sent by the bidder

This agent does not send a lot of messages.

The quite most exclusively used message is the AgentSpeak(L) version equivalent of FIPA-ACL propose. Which is again the tell performative.

The bidder sends the auctioneer $.send(Auctioneer, tell, placeBid(AUCTIONTYPE, BIDVALUE))$. The content of the messages are the auction type and the bid value.

3. Pseudocode for the auctioneer

```
auctionType = auction type
tttd = time to travel date
tttdts = time to travel date threshold
reservePrice = reserve price
priceIncrement = price increment
dutchInitCoef = initial coefficient to multiply the reserve price in case we start
with a dutch auction, set to 3*priceIncrement
winner = identity of the winner
winningBid = winning bid, initialised the same as the reserve price

StartOfAuctioneer(tttd, tttdts,auctiontype, reservePrice, priceIncrement,
dutchInitCoef, winner, winningBid):
  If tttd < tttdts:
    auctionType = 1 # dutch auction
    .broadcast(tell, auction(1)) # start of a dutch auction
    .broadcast(tell, currentBid(reservePrice*(1+dutchInitCoef))) # first call for proposals
    currentBid = reservePrice*(1+dutchInitCoef)
  else:
    auctionType = 0 # english auction
    .broadcast(tell, auction(1)) # start of an english auction
    .broadcast(tell, currentBid(reservePrice))
    currentBid = reservePrice

  AuctionOn <- True
  While AuctionOn is True:
    if auctionType = 0: # if it is an english auction
      Receive(Msg)
      if Msg is empty:
        AuctionOn <- False % Finish english auction
        if (winner, winningBid) != (null, null):
          .broadcast(tell, winner(WINNER));
        else:
          .print("There is no winner, no offer received.")
      else:
        remove "not-understood" messages from Msgs
        filter out messages sent with a past auctionType
        get placeBid(auctionType,BIDVALUE)[Source(Sender)] with the higest BIDVALUE
        winner  <- Sender
        winningBid <- BIDVALUE
        currentBid = currentBid * (1 + priceIncrement)
        if tttd-1 < tttdts:
          .broadcast(tell, auction(1))  # inform start of dutch auction
          .broadcast(tell, currentBid(currentBid*(1+dutchInitCoef)))
          currentBid = currentBid * (1 + dutchInitCoef)
          auctionType = 1
        else:
          .broadcast(tell, currentBid(currentBid*(1+priceIncrement)))
          currentBid = currentBid * (1 + priceIncrement)


    else: #if it is a dutch auction(auctionType = 1)
      Receive(Msg)
      if Msg is empty:
        if currentBid = reservePrice:
          AuctionOn <- False
          if (winner, winningBid) != (null, null):
            .broadcast(tell, winner(WINNER))
          else:
            .print("There is no winner, no offer received.")
        else if (currentBid * (1 - priceIncrement) <= reservePrice):
          .broadcast(tell, currentBid(reservePrice))
          currentBid = reservePrice
        else:
          .broadcast(tell, currentBid(currentBid * (1 - priceIncrement)))
          currentBid = currentBid * (1 - priceIncrement)

      else if number of Msg >=2:
        .broadcast(tell, currentBid(currentBid * (1 + 1/priceIncrement)))
        currentBid = currentBid * (1 + 1/priceIncrement)

      else: #exactly one bidder
        get placeBid(auctionType,BIDVALUE)[Source(Sender)]
        winner  <- Sender
        winningBid <- BIDVALUE
        AuctionOn <- False
        .broadcast(tell, winner(WINNER))

    tttd = tttd - 1
```

4. Pseudocode for the bidder

The auctioneer works with 2 “functions”. When it receives the auction type, the belief base is updated using one fonction, and when it receives a call for proposal, it is managed with another.

```
StartOfBidder(auctiontype, reservePrice, priceIncrement):
  While Auction is On:
    Receive(Msg)

    Case-of Msg[+auction(AUCTIONTYPE)]
      auctiontype <- AUCTIONTYPE

    Case-of Msg[+currentBid(CURRENTBID)[source(S)]]
      if (AUCTIONTYPE==0):
        if (CURRENTBID <=  reservePrice):
          if ( CURRENTBID*(1+ priceIncrement) <= reservePrice):
            NewCurrentBid = CURRENTBID*(1+priceIncrement)
          else:
            NewCurrentBid = (CURRENTBID + reservePrice)/2
            .send(Auctioneer, tell, placeBid(AUCTIONTYPE, NewCurrentBid))
      else:
        if (CURRENTBID <=  reservePrice):
          NewCurrentBid = CURRENTBID
          .send(Auctioneer, tell, placeBid(AUCTIONTYPE, NewCurrentBid))
```

Implementation

Our solution is extensively based on AgentSpeak(L) implementation in pyson through the **pyson** framework. This choice has been motivated in part by the fact that Python is the programming language we are most at ease with. Another reason for this choice is our non familiarity with asynchronous and distributed paradigms. So to summarise, 99% of the coding was done in AgentSpeak(L) with python just serving as the platform to run the agent. The technical choice we made allowed us to solely concentrate on agent behaviour design and not on any technical issue outside of the scope of the course.

The solution we present is based on two agents: an auctioneer agent and a bidder agent, both implement in ".asl" files.

In python a simple function serves to run one auction with one auctioneer and multiple bidders, each agent having their own parameters. In the main version of the python file, the parameters are generated randomly.

The ASL implementation of the bidder is fairly simple and can be included here:

```
+auction(N)[source(S)]
   :    true
   <-  if (N == 0) {
            -auction(1);
            +auction(0);
            .print("auction type: english");
        }
        else {
            -auction(0);
            +auction(1);
            .print("auction type: dutch");
        }.

+currentBid(CURRENTBID)[source(S)]
   :    true
   <-  ?reservePrice(RP);
        ?priceIncrement(PI);
    ?auction(AUCTIONTYPE);
    if (AUCTIONTYPE==0) {
        if (CURRENTBID <=  RP){
            if ( CURRENTBID*(1+PI) <= RP) {NCB = CURRENTBID*(1+PI);}
            else {NCB = (CURRENTBID + RP)/2;    }
            .print("bidding", NCB, "to english auction");
            .send(S, tell, placeBid(AUCTIONTYPE, NCB));
        }
    }
    else {
        ?reservePrice(RP);
        if (CURRENTBID <=  RP){NCB = CURRENTBID;
        .print("bidding", NCB, "to dutch auction");
        ?auction(AUCTIONTYPE);
        .send(S, tell, placeBid(AUCTIONTYPE, NCB))};
    }.
```

One can see that it is the exact translation of the bidder pseudocode. For the auctioneer, the implementation is longer and more fragmented. A **startAuction** plan announces the start of the english or dutch auction depending on parameters, then calls the **iterate** plan. The **iterate** plan identifies 5 cases:

- case 1: it is an english auction and we receive no bidders proposals, we end the auction, announcing a winner if there is.
- case 2: it is an english auction and we receive at least one bidder, we continue increasing the price until no proposal arrives.
- case 3: it is a dutch auction, there is no proposals and the current_bid is the reserve price. We end the auction
- case 4: it is a dutch auction, we receive no proposals, but the current_bid is higher than the reserve price. We continue and decrease the current bid in a subsequent round of call for proposals
- case 5: it is a dutch auction and we receive exactly on proposal. We end the auction broadcasting the identity of the winner
- case 6: it is a dutch auction and we receive more than 1 proposal. We go through another round of call for proposals, but we shout a high price and start a new price descent until we find only one proposal.

Instruction to set up, configure and run

The agent systems builds upon **pyson** framework. To set-up the environment, the following steps must be executed:

- Go to the **pyson** framework page and clone or download it.
- Unzip the folder if downloaded
- Access the downloaded folder with a terminal or via command-line
- Run this command: **python setup.py develop**
- Voilà. The pyson environment is ready to be used.

Test cases

3 folders have been created for each test case. To run each one, the user must use the terminal and go to the folder concerned and run **python auction.py**. The terminal will show the sequence of actions performed by the auctioneer and the bidders. Under is the execution of the first test case. However, for auctions with more bidders, the **auction.ipynb** notebook enables to use the custom function **runMassiveAuction(N)**.

```
#python auction.py

auctioneer round: 0 . Start of dutch auction
auctioneer calling for proposals starting at 1337.6399999999999
bidder_1 auction type: dutch
bidder_2 auction type: dutch
auctioneer start of round n_ 1
auctioneer 0 proposal was received.
auctioneer calling for proposals starting from 1083.4884
auctioneer start of round n_ 2
auctioneer 0 proposal was received.
auctioneer calling for proposals starting from 877.6256040000001
auctioneer start of round n_ 3
auctioneer 0 proposal was received.
auctioneer calling for proposals starting from 852
auctioneer There is no winner, no offer received.
```

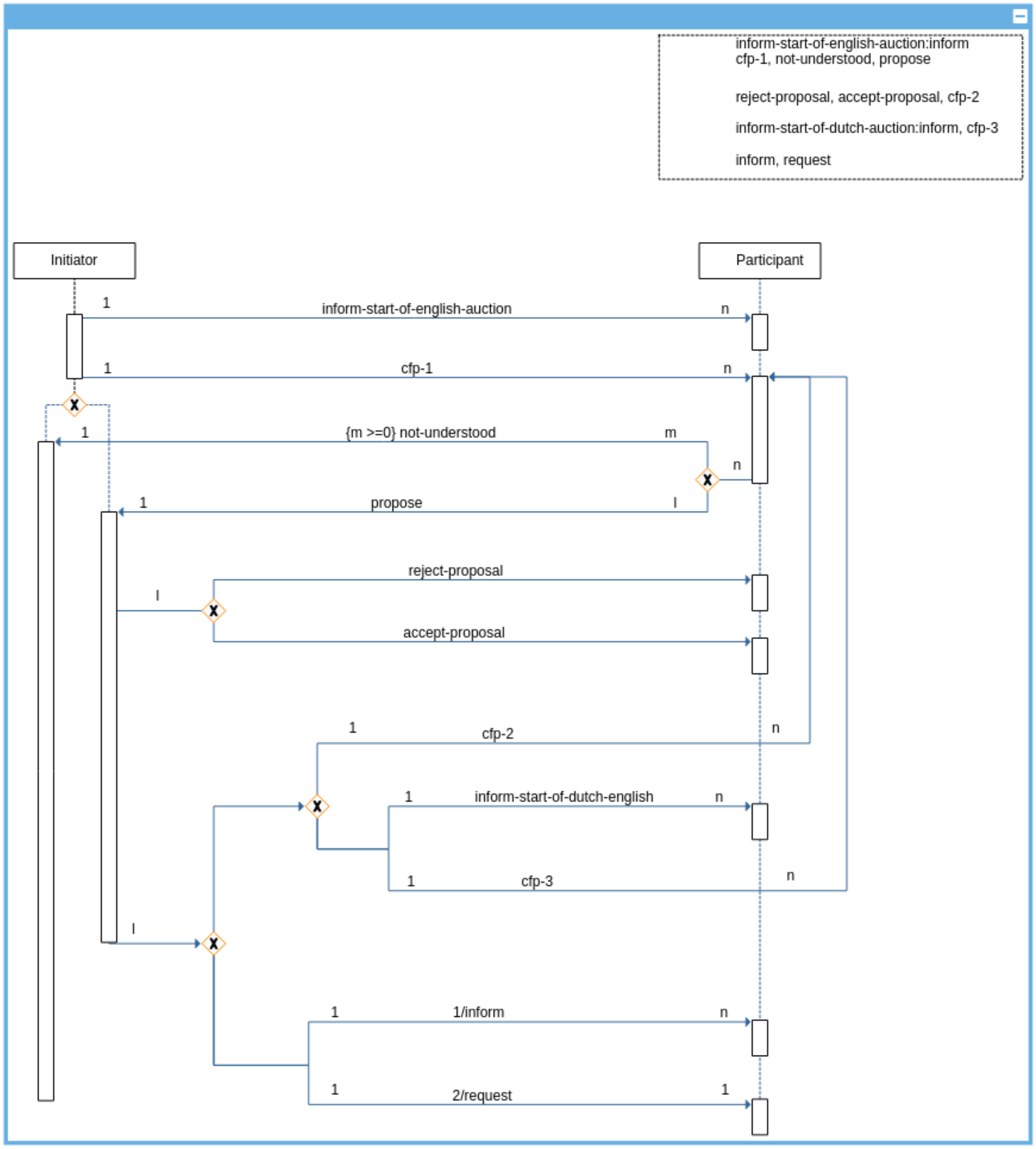
inform-start-of-english-auction:inform
cfp-1, not-understood, propose
reject-proposal, accept-proposal, cfp-2
inform-start-of-dutch-auction:inform, cfp-3
inform, request
Initiator
Participant
1
inform-start-of-english-auction
n
1
cfp-1
n
1
{m >=0} not-understood
m
n
1
propose
l
reject-proposal
l
accept-proposal
1
cfp-2
n
1
inform-start-of-dutch-english
n
1
cfp-3
n
l
1
1/inform
n
1
2/request
1

Figure 1: Dutch auction switching to English auction protocol