Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

DEVS AS A METHOD TO MODEL AND SIMULATE COMBINATORIAL DOUBLE AUCTIONS FOR E-PROCUREMENT

Juan De Antón[1], Cristina Ruiz-Martin[2], Félix Villafáñez[1], and David Poza[1]

[1] Department of Business Organization, INSISOC – University of Valladolid, Valladolid, SPAIN
[2] Department of Systems and Computer Engineering, Carleton University, Ottawa, CANADA

ABSTRACT

The surge in electronic procurement is fostering the proliferation of electronic marketplaces and advanced auctions as primary coordination mechanisms. Among these, combinatorial and double auctions are gaining traction in the procurement sector. However, prevalent implementations often assume participants to be perfectly rational, adhering to predefined behaviors within the auction model. These centralized models, while prevalent, fail to capture the intricate dynamics of real auction environments adequately. Consequently, there is a growing recognition of the necessity for decentralized models within an agent-based framework to simulate such auctions authentically. The contribution of this work is the application of the DEVS formalism to develop a decentralized model for a combinatorial iterative double auction to address the limitations of centralized implementations. The model is formally defined, and a case study is presented to verify it against its centralized version. This is the first step toward accommodating agents with varied behavioral patterns within auction simulations.

1 INTRODUCTION

The emergence of electronic platforms in the procurement sector has brought a new means of coordinating supply and demand. E-procurement systems are becoming popular in industrial sectors such as additive manufacturing, cloud computing or energy. Typically, these platforms employ auction-based mechanisms to coordinate the allocation of procurement requests from distributed customers to available resources from distributed suppliers.

Auctions have proved very useful for handling decentralized scheduling problems (Gorbanzadeh et al. 2000). In particular, auctions allowing combinatorial bids (i.e., combinatorial auctions) are gaining prominence as market mechanisms in the procurement sector (Palacios-Huerta et al. 2024). Most of the existing combinatorial auction designs for e-platforms assume a perfectly rational behavior of participants, which presumes that agents are perfect optimizers who always make correct bidding decisions to maximize their utilities. However, empirical studies on auctions have found that it is hardly realistic to expect all individuals to consistently act in a perfectly rational manner (Khalid et al. 2022; Jiang et al. 2013).

Combinatorial auction proposals designed under the perfect rationality premise are frequently modeled and implemented using mixed-integer and linear programming solvers. For the case of multi-round combinatorial auctions, specific algorithms are commonly developed to implement the iterative procedure. In these solutions, the behavior of the agents is predefined, predictable and assumes perfect rationality because the algorithm is implemented as an iterative and centralized one. Thus, the simulation results do not represent the real-world case; they are just the solution of an optimization problem solved in an iterative way. However, auction proposals with agents showing bounded rationality entail a higher level of complexity due to the unpredictable nature of participant behavior. As a result, there is a growing need for more flexible methodologies in the modeling and simulation of decentralized systems that can capture the intricacies of bounded rationality. In this scenario, the Discrete Event System Specification (DEVS)

formalism (Zeigler and Muzy 2018) can provide a helpful framework for modeling and simulating the behavior of such a decentralized market structure.

Most of the works proposing combinatorial auction mechanisms employ centralized models for the implementation of the auction. Only a few of the proposals employ agent-based modeling frameworks that enable decentralized simulations. However, these works either employ domain-specific frameworks that are not applicable to other problems, or they resort to methodologies that are neither backed by a robust formalism nor show a modular and incremental modeling protocol. In this paper, we address these limitations by using the DEVS formalism to develop a robust decentralized model for the simulation of the auction.

The present work builds upon prior research in De Antón et al. (2024), where we defined a centralized model of a combinatorial double auction for the allocation of additively manufactured production orders to additive-manufacturing (AM) providers. In that model, the resulting allocation from the auction was obtained by solving a binary integer programming problem in the Microsoft Visual Basic for Applications (VBA) environment, in which the involved agents (i.e., customers and providers) modified their bids following a perfectly rational pattern. Since we need to extend the auction formulation to reproduce a more realistic scenario by considering bounded rationality of agents, we need a more flexible simulation framework. This study aims to demonstrate the use of the DEVS formalism for developing a decentralized model of the combinatorial double auction defined in De Antón et al. (2024). We verify the developed *decentralized DEVS model* by comparing its behavior with that of the VBA model which we will refer to from now on as the *centralized VBA model*. The simulation results obtained from our DEVS model closely replicate those from the centralized VBA model, confirming the verification of our model. This work lays the groundwork for future research, paving the way for the development of a fully decentralized auction model within the DEVS framework that accommodates agents with bounded rationality and diverse behavioral patterns.

The rest of the paper is structured as follows. The related literature about auctions in electronic platforms and an introduction to the DEVS formalism are presented in Section 2. The DEVS model for the auction mechanism is detailed in Section 3. In Section 4, the implementation of the model in the Cadmium simulator is shown with a case study. Lastly, Section 5 draws the main conclusions and proposes further research.

2 BACKGROUND

2.1 Auctions in E-procurement Platforms

Auctions have become one of the preferred market mechanisms in online markets because they allow to effectively coordinate supply and demand while drastically reducing transaction costs (Abedrabboh and Al-Fagih 2023; Yang et al. 2021).

Different auction mechanisms have been designed to coordinate the market of AM subcomponents within an e-platform. The functioning of the auction mechanism considered in this work is as follows: first, each buyer will request a production order and will place a purchase offer (i.e., a bid) showing the amount of money that they are willing to pay to have that order produced; second, each seller will place a sell offer (i.e., an ask) for some combination of the buyers' orders showing the amount of money that they are willing to charge for the manufacturing of the whole bundle. After collecting bids and asks from buyers and sellers, respectively, the auctioneer will determine the temporary allocation of orders to sellers, thus ending the first round of the auction. The allocation maximizes the objective function for the current set of prices, for example, the social welfare. In moving to the next round, agents (either buyers or sellers) not winning their bids/asks will be allowed to update their prices so that they have a chance of winning the following round. This process will be repeated from one round to the next until the stopping criterion is met. The final allocation of orders to sellers will be obtained as a result.

Considering the above features, the auction mechanism is classified as a combinatorial iterative double auction according to auction theory (Abrache et al. 2004). However, this auction has three specific

characteristics: (i) biddable items do not preexist but are created from the buyers' demands; (ii) the combinations of items are set in the first round and cannot be changed in future rounds; (iii) sellers and buyers know nothing about other agents' bids and market situation. Besides, the auction will establish the minimum price steps that buyers and sellers must apply to update their bids/asks.

Combinatorial auctions determine the allocation of bundles of items to bidders by addressing the winner determination problem (WDP). This problem solves the allocation of bundles to the bidders that value them the most according to a specific optimization criterion, and it is generally formulated as a maximization problem of the social welfare, the buyers' utility or the sellers' revenue (De Vries and Vohra 2003). Since the WDP is NP-hard in its general formulation (Rothkopf et al. 1998), iterative procedures are employed to distribute the computation burden among agents (Parkes 2006).

In electronic procurement, combinatorial and double auctions are frequently used to coordinate the allocation of production orders to manufacturing resources. The main advantage of combinatorial auctions is that they allow participants to express preference over bundles of items as well as individual items. The features depicted by combinatorial and double auctions perfectly match the characteristics of the AM production environment (De Antón et al. 2024; Zehetner and Gansterer 2023).

From the modeling and simulation perspective, the main problem faced by auction mechanisms is that they are frequently developed within theoretical environments that do not fully reproduce the complexity of real contexts (Evans and Prokopenko 2023). Consequently, centralized models that assume a rational behavior are built to implement those auction mechanisms. Notwithstanding, according to empirical studies, human agents cannot be assumed to behave perfectly rationally, but they can show nonstandard behaviors (Podwol and Schneider 2016). These behavioral phenomena are generally referred to as bounded rationality (Shen and Su 2007), and systems exhibiting this property require decentralized models for their simulation.

Combinatorial auction mechanisms usually lack specialized simulation frameworks that allow a decentralized implementation. This situation is even worse in the case of combinatorial double auctions. Recent examples of combinatorial double auction proposals conducting centralized simulations can be found in Jiang et al. (2022) and Abedrabboh et al. (2023). Still, we can find some specific proposals for certain environments that execute a decentralized implementation, as in Umer et al. (2022), where they use the CloudAuction extension within the CloudSim implementation tool (Calheiros et al. 2024) for the auction simulation. However, this is a modeling tool specifically designed for cloud computing services, and yet there is no formalization of a combinatorial double auction system. In this context, the DEVS formalism provides the necessary framework to develop a formal decentralized model for our auction that can be easily extended to analyze more realistic scenarios. In this paper, we will show how this can be achieved by defining and implementing in DEVS the *centralized VBA model* presented in De Antón et al. (2024).

2.2 Discrete Event Systems Specification (DEVS) and Cadmium Simulation Tool

DEVS is a hierarchical and modular formalism designed for modeling discrete event systems, enabling the modular coupling of models to construct intricate systems from simpler ones (Zeigler and Muzy 2018). This approach, rooted in general systems theory, employs a rigorous methodology for representing models, breaking down complex systems into two types of models: atomic and coupled models. In this framework, the atomic models define system behavior, while the coupled models articulate the overall structure. Atomic and coupled models have the capability to link their inputs and outputs, facilitating the construction of complex models from simpler components.

The versatility of DEVS lies in its capacity to represent systems with finite possible states, where the transition to a new state after an event's arrival is contingent on the continuous time elapsed in the preceding state, thereby allowing for a comprehensive representation of systems governed by sequences of events. An interesting feature of DEVS is its bottom-up methodology, which maintains incremental complexity bounded and enables stage-wise verification, as each constructed coupled model can undergo independent testing. Also, a main advantage of the DEVS paradigm is the independence of model specification from the simulation mechanism, which facilitates reliability, correctness, and verification and validation of the model.

In this work, the DEVS model will be implemented in the Cadmium simulation environment (Cárdenas and Wainer 2022; Belloli et al. 2019). Cadmium is a C++17 header-only DEVS simulator that is easy to integrate into different projects, and it is freely available on the SimulationEverywhere GitHub repository (https://github.com/SimulationEverywhere/Cadmium_v2/wiki). In Cadmium, a class needs to be defined for each type of atomic model and coupled model. Each class can be instantiated into an object and integrated into larger coupled models. Cadmium applies the abstract hierarchical simulation algorithm presented in Belloli et al. (2019) for simulating DEVS models.

3 DEVS MODEL FOR AN AUCTION PROBLEM

In this section we will detail the structure and components of the DEVS model for a combinatorial double auction and the implementation procedure using the Cadmium engine.

3.1 DEVS Model

The Combinatorial Iterative Double Auction (CIDA) model presented in this section is a coupled model that is composed of three basic components: *Buyer* (with multiple instances), *Seller* (with multiple instances), and *Auctioneer* (one instance), as shown in Figure 1. *Buyer* and *Seller* are analogous coupled models representing a single buyer and a single seller in the auction, respectively. Both models can be disaggregated into two atomic components: *Buyer* is composed of *Filter* and *Bid*; *Seller* is composed of *Filter* and *Ask*. *Auctioneer* is an atomic model that plays the coordinating role of the auctioneer in an auction. In the instantiation phase, the number of instances of *Buyers* and *Sellers* to be generated will be adjusted according to the number of buyers and sellers participating in the auction. In this auction model, we have assumed the rational and self-interested behavior of all the participants (both buyers and sellers).

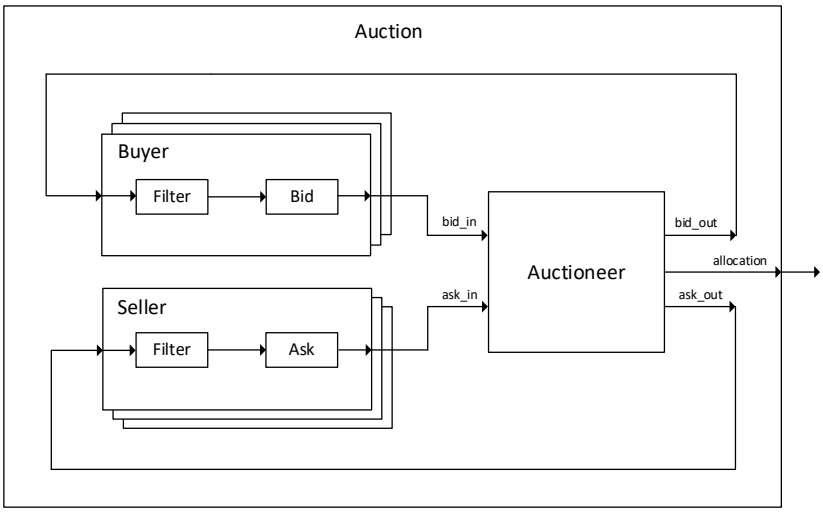

Figure 1: DEVS model of the auction.

The *Buyer* model reproduces the behavior of a buyer in an auction. Every buyer participating in an auction will have a maximum amount of money that they are willing to pay, called the reservation price. A rational and self-interested buyer will start bidding a quantity lower than their reservation price and, if they do not win the current auction round, they will increase their bid by the minimum amount required by the auction (i.e., the purchase price step). This purchase price update procedure will continue until the reservation price is reached, when the buyer will have no incentives to continue increasing the bid.

The behavior described above is modeled in the *Buyer* coupled model. In each round, the *Buyer* sends a message showing the purchase price offered for the item (in this case, the message goes to the *Auctioneer*

based on the connections). At some point, after the allocation of the current round is determined, it receives a message reporting whether its bid is winning.

Buyer is subdivided into two atomic models: *Filter* and *Bid*. Each buyer has a unique identifier represented as a parameter of both the *Filter* and the *Bid* atomic models. Every message received by the *Buyer* will contain an identifier to indicate the buyer to whom it is addressed. *Filter* is thus used to identify among all the messages received by the model, the one/s corresponding to its related *Buyer*. *Bid* is in charge of determining the initial purchase price for the bid and updating that price according to the feedback message received at the end of each round. Subsequently, we detail the formal definition of the two atomic models *Filter* and *Bid*.

Filter model only activates when it receives a message. Its function is to read the message received and evaluate if the identifier of the message matches its own identifier. If the test succeeds, it forwards the message through its output port (in this case, to *Bid* based on the connections) and passivates again; otherwise, it just passivates without sending anything.

Filter = < S, X, Y, δint, δext, λ, ta >

Parameters: ID_f
S = {s1 ∈ {active, passive}, sGotIt ∈ {True, False}}
Initialization: s1 = passive, sGotIt = False
X = {[Id, GotIt]; Id ∈ N, GotIt ∈ {True, False}}
Y = {[Id, GotIt]; Id ∈ N, GotIt ∈ {True, False}}
δint (s) = {if s1 = active, then s1 = passive}
δext (s,e,x) = {
    if ID_f = Id, then
        sGotIt = GotIt
        s1 = active }
λ (s1 = active) {send [ID_f, sGotIt]}
ta (s1 = active) = 1
ta (s1 = passive) = INFINITY

The Bid model represents a bid placed by a buyer over an item in the auction. The following parameters are given to each *Bid* in the definition of the instance: identifier (*ID_b*), reservation price (*RPr*), initial purchase price (*InitialPPr*), and purchase price step (*PPrStep*). Bid starts in active mode to submit the initial bid once the auction begins, then passivates until the next message arrives. After receiving a message with feedback about the result of the bid, three things can happen: (i) the buyer is winning their bid; (ii) the buyer is losing their bid, but the reservation price has already been reached; (iii) the buyer is losing their bid, and the reservation price has not yet been reached. In the first two cases, *Bid* does nothing (it stays passivated), whereas in the last case, *Bid* raises the price by an amount equal to the purchase price step.

Bid = < S, X, Y, δint, δext, λ, ta >

Parameters: ID_b, RPr, InitialPPr, PPrStep
S = {PurPr ∈ R0+, Notify ∈ {True, False}}
Initialization: PurPr = InitialPPr, Notify = True
X = {[Id, GotIt]; Id ∈ N, GotIt ∈ {True, False}}
Y = {[Id, PurPr]; Id ∈ N, PurPr ∈ R0+}
δint (s) = {if Notify = True, then Notify = False}
δext (s,e,x) {
    if GotIt = False and PurPr + PPrStep <= RPr, then
        PurPr += PPrStep
        Notify = True
}

λ (Notify = True) {send [ID_b, PurPr]}
ta (Notify = True) = 1
ta (Notify = False) = INFINITY

The *Seller* model is analogous to the *Buyer* model in terms of its architecture, but it represents the behavior of a seller. Every seller participating in an auction will have a minimum amount of money that they are willing to charge, called the production cost. A rational and self-interested seller will start asking for a quantity higher than their production cost, and if they do not win the current auction round, they will decrease their ask by the minimum amount required by the auction (i.e., the ask price step). This ask price update procedure will continue until the production cost is reached. To implement this behavior, *Seller* is subdivided into *Filter* and *Ask* atomic models. The *Filter* component has the same behavior as the *Filter* in *Buyer*; the *Ask* model is analogous to the *Bid* model, but, in this case, the price update procedure follows the seller's logic explained above (i.e., it is decreased instead of increased).

The *Auctioneer* model is the auction coordinator. At the beginning of the auction, it sets the length of the requesting period according to the *round_timer* parameter. During that period, the *Auctioneer* receives bids and asks (in this case, from the Buyer instances through the *bid_in* port and from the *Seller* instances through the *ask_in* port). The *Auctioneer* retrieves the bidding and asking information from these messages and determines the allocation of the current round (i.e., temporary allocation) by solving the WDP. Subsequently, the *Auctioneer* sends individual messages through the *bid_out* and *ask_out* ports informing each participant whether they are winning their corresponding bids/asks in the current round. In this case, the ports are connected to *Buyers* and *Sellers* models, respectively. Once the messages have been sent, the *Auctioneer* resets the requesting period to *round_timer*, and the next round begins. This procedure is repeated for several rounds until all the agents stop updating their prices or the maximum number of rounds set in the auction is reached. After a requesting period in which the *Auctioneer* receives no messages, the last temporary allocation will be announced as the final allocation, and the auction will be closed (i.e., the *Auctioneer* passivates). When this occurs, the *Auctioneer* sends a message through the allocation port to indicate that the auction is finished. The formal definition of *Auctioneer* is as follows:

Auctioneer = < S, X, Y, δint, δext, λ, ta >
Parameters: matrix (SxB) ∈ {0,1}x{0,1}, round_timer ∈ R0+
S = {time_to_calculate ∈ R0+, new_value ∈ {True, False}, send_out ∈ {True, False}, seller [1, …, S] ∈ {id ∈ N, price ∈ R0+, gotIt ∈ {True, False}}, buyer [1, …, B] ∈ {id ∈ N, price ∈ R0+, gotIt ∈ {True, False}}
Initialization: new_value = False, send_out = False
X = {bid_in: {[Id ∈ N, PurPr ∈ R0+]}; ask_in: {[Id ∈ N, AskPr ∈ R0+]}}
Y = {bid_out: {[Id ∈ N, GotIt ∈ {True, False}]}; ask_out: {[Id ∈ N, GotIt ∈ {True, False}]}; allocation: {True, False}}
δint (s) {
    if send_out = False {
        if new_value = False, then send_allocation = True
        else {
            solve WDP using seller & buyer state variables as defined in De Antón et al. (2024)
            send_out = True
            new_value = False }
    else {
        send_out = False
        time_to_calculate = round_timer}   }
δext (s,e,x) {
    for message in bid_in {
        if buyer in B, then update buyer.price = PurPr
        else add buyer to B }

```
    for message in ask_in {
        if seller in S, then update seller.price = AskPr
        else add seller to S  }
    new_value = True
    time_to_calculate -= e   }
λ (send_out = True) {
    for buyer in B {
        send [buyer.id, buyer.gotIt] }
    for seller in S {
        send [seller.id, seller.gotIt]  }
    if send_allocation = True, then send True}
ta (send_out = True) = 0
ta (send_out = False and send_allocation = True) = INFINITY
ta (send_out = False and send_allocation = False) = time_to_calculate
```

The matrix parameter stores a two-dimensional matrix with binary values with S rows and B columns showing the bundles of items included in each seller's asks (further detailed in Section 3.2). It will be used in the internal transition to solve the WDP. In our model, the WDP will select those bundles of items that maximize the difference between the prices paid by buyers and the prices received by sellers (i.e., the social welfare according to Xia et al. (2005)).

3.2 Model Implementation in Cadmium

Once the DEVS model of the auction has been formulated (i.e., our CIDA model), we implement the atomic and coupled models by using the class templates provided by Cadmium and generate the executable of the model (i.e., our CIDA simulator). When a DEVS model is simulated with Cadmium, a log file with the state transitions and the messages sent through the output ports is generated.

To run an instance of the auction, we first need to initialize the input values and the parameters. We have streamlined the initialization process by extracting input data from a text file. This eliminates the need to recompile the model for each simulation run, as updating the text file is now sufficient. Thus, the program will read from the text file (see Figure 2) the following input data: reservation prices and initial purchase prices from buyers, production costs and initial ask prices from sellers, and a binary matrix showing the bundles on which each seller is bidding. In the matrix, each row represents an ask from a seller, and each column represents an item from a buyer. One (1) indicates that the corresponding item is included in that seller's ask. From these data, the model will deduce the number of buyers and sellers participating in the current auction round so that it will instantiate the *Buyer* and *Seller* models accordingly.

Three additional parameters need to be initialized before starting the auction: the purchase price step, the ask price step, and the round-timer. The first two values are also read from the text file and represent the minimum amount that losing buyers should increase their purchase prices and the minimum amount that losing sellers should decrease their ask prices for the next round, respectively. The third parameter (i.e., the round-timer) is passed as an argument when the model is executed.

After reading the input data, the simulator creates all the corresponding model instances and initializes the values of their parameters. The simulation of the auction can now be run according to the behavior defined and explained in Section 3.1.

```
'Customers reservation price
78;177;113;105;85;105;102;50;60;149
'Customers initial purchase price
50;79;65;67;48;77;62;35;34;78
'Customers purchase price step
5
'Producers production Cost
59;161;95;91;142;51;216;397;145;66
'Producers first Ask Price
139;239;129;111;330;108;312;461;245;82
'Producers ask price step
10
'Items Combination
0;0;0;0;1;0;0;0;1;0
0;1;0;0;0;1;0;0;0;0
0;1;0;0;0;0;0;0;0;0
0;0;1;0;0;0;0;0;0;0
1;0;1;0;0;0;0;0;0;1
0;0;0;0;0;0;1;0;0;0
0;0;0;0;0;1;1;1;0;0
1;1;0;0;1;1;0;0;0;0
0;0;1;1;0;0;0;0;0;0
0;0;0;0;1;0;0;0;0;0
```

Figure 2: Input data for simulating an auction.

4 CASE STUDY AND MODEL VERIFICATION

In this section, we develop a case study to verify our CIDA DEVS model and simulator. We simulate the same instances of an auction using the previous *centralized VBA model* and the new *decentralized DEVS model* to verify that we obtain the same results. This will serve as verification of the DEVS model.

The scenario considered is composed of 10 sellers and 10 buyers participating in the auction. Each buyer will place one bid over one item, whereas each seller will place an ask over a bundle of items. The input data is displayed in Figure 2: initial and reservation values for each participant, the binary matrix showing the bundles submitted by each seller, and the parameters of the auction (purchase price step is set to 5; ask price step is set to 10). This text file will be read by both the VBA program and the CIDA simulator to simulate the model. Additionally, we need to provide the CIDA simulator with the round-timer parameter, which is set to 4 in this case. The VBA program does not require this parameter because it is a centralized optimization model with no time dynamics; each auction round is run as soon as the previous one is finished in an iterative procedure.

As previously mentioned, our CIDA simulator generates a log file with the state transitions and the messages sent through the output ports. Some excerpts from this log file are shown in the table displayed in Figure 3. In the *data* column, two types of data can be found: (i) the values of the state variables whenever the *port_name* value is empty; (ii) the message sent through the corresponding port whenever there is a value in *port_name*. In the upper part of the table, we can see how some of the atomic models are instantiated and initialized, with the auctioneer collecting all the bids and asks to determine the first temporary allocation (the data contains the id of the seller/buyer and the price they are willing to receive/pay). In the central part, we observe the feedback messages sent by the auctioneer and the new submissions sent by the participants accordingly. At the bottom of the table, we see the allocation obtained in the final round. The auctioneer output displays four elements (time remaining until the next assignment; whether or not new bids have been received in the current round; whether or not a new assignment should be sent to buyers and sellers; whether or not to notify that the auction has ended) and a list with the agent's ID, the buyer's bid / seller's ask, and a binary variable indicating whether the agent wins the auction (1) or not (0).

time	model_id	model_name	port_name	data
0	30	bid-1		{1,50}
0	31	filter_b-1		{0,0}
0	54	bid-2		{1,79}
0	55	filter_b-2		{0,0}
0	48	ask-1		{1,139}
0	49	filter_s-1		{0,0}
0	60	ask-2		{1,239}
0	61	filter_s-2		{0,0}
1	22	auctioneer		{3,1,0,0, {{1,139,0},{2,239,0},{3,129,0},{4,111,0},{5,330,0},{6,108,0},{7,312,0},{8,461,0},{9,245,0},{10,82,0}}, {{1,50,0}{2,79,0}{3,65,0}{4,67,0}{5,48,0}{6,77,0}{7,62,0}{8,35,0}{9,34,0}{10,78,0}}}
...				
4	22	auctioneer	bid_out	{1,0}
4	22	auctioneer	bid_out	{2,0}
4	22	auctioneer	ask_out	{1,0}
4	22	auctioneer	ask_out	{2,0}
6	30	bid-1	out	{1,55}
6	30	bid-1		{0,55}
6	48	ask-1	out	{1,129}
6	48	ask-1		{0,129}
8	22	auctioneer		{2,0,1,0, {{1,129,0},{2,229,0},{3,119,0},{4,101,0},{5,320,0},{6,98,0},{7,302,0},{8,451,0},{9,235,0},{10,72,0}}, {{1,55,0}{2,84,0}{3,70,0}{4,72,0}{5,53,0}{6,82,0}{7,67,0}{8,40,0}{9,39,0}{10,83,0}}}
...				
60	22	auctioneer		{4,0,0,1, {{1,109,1},{2,189,1},{3,99,0},{4,91,0},{5,240,1},{6,68,1},{7,222,0},{8,401,0},{9,145,0},{10,72,0}}, {{1,75,1}{2,104,1}{3,95,1}{4,102,0}{5,63,1}{6,102,1}{7,82,1}{8,50,0}{9,49,1}{10,123,1}}}

Figure 3: Excerpts from the output log file generated by the CIDA simulator.

We have parsed the allocation values obtained from the log file into a table to generate a more user-friendly visualization. As a result, we have built the table shown in Figure 4, where the allocation data of each auction round is summarized. The following data from each round is shown: the value of the objective function, in this case, the social welfare (Obj. F. column), purchase prices offered (Buyers columns), and ask prices requested (Sellers columns). The values highlighted in a darker color indicate the agents that are winning the corresponding auction round. We notice that, from the first successful allocation in round 3, the value of the objective function increases with each round.

		Buyers										Sellers									
Round	Obj. F.	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10
0	0	50	79	65	67	48	77	62	35	34	78	139	239	129	111	330	108	312	461	245	82
1	0	55	84	70	72	53	82	67	40	39	83	129	229	119	101	320	98	302	451	235	72
2	0	60	89	75	77	58	87	72	45	44	88	119	219	109	91	310	88	292	441	225	72
3	3	65	94	80	82	63	92	77	50	49	93	109	209	99	91	300	78	282	431	215	72
4	17	70	99	85	87	63	97	82	50	49	98	109	199	99	91	290	68	272	421	205	72
5	34	75	104	90	92	63	102	82	50	49	103	109	189	99	91	280	68	262	411	195	72
6	42	75	104	95	97	63	102	82	50	49	108	109	189	99	91	270	68	252	401	185	72
7	56	75	104	95	102	63	102	82	50	49	108	109	189	99	91	270	68	242	401	175	72
8	57	75	104	95	102	63	102	82	50	49	113	109	189	99	91	260	68	232	401	175	72
9	66	75	104	95	102	63	102	82	50	49	113	109	189	99	91	260	68	222	401	165	72
10	72	75	104	95	102	63	102	82	50	49	118	109	189	99	91	250	68	222	401	165	72
11	76	75	104	95	102	63	102	82	50	49	118	109	189	99	91	250	68	222	401	155	72
12	87	75	104	95	102	63	102	82	50	49	123	109	189	99	91	240	68	222	401	155	72

Figure 4: Temporary allocations at the end of each auction round for the CIDA model.

The values exposed in this table match the values obtained with the VBA centralized model. In the VBA program, the only output data that we obtain at the end of each round is the temporary allocation and the value of the objective function. According to the allocation results, prices are automatically updated,

and the solver is called again to solve the next round. A visualization of the simulation interface is shown in Figure 5, where the final allocation of the auction instance is displayed. It can be observed that the final allocation results coincide with those of the CIDA simulator displayed in Figure 3.

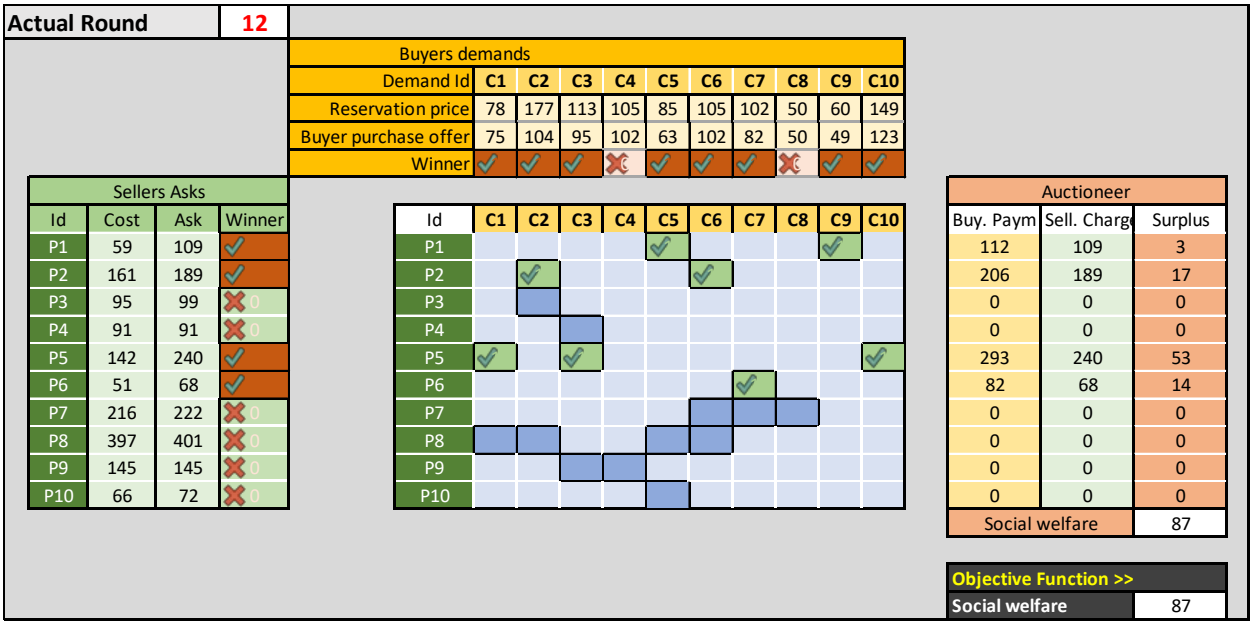

Figure 5: Visualization of the final allocation in the VBA simulator.

Besides the auction instance detailed in the case study, several other auction scenarios were run to compare the results obtained with both models. We varied the number of agents participating in the auction to prove that the results obtained with the DEVS model also matched the ones obtained with the VBA centralized model with different auction configurations.

The input data generation for all the auction instances run to test and verify the DEVS model has been derived from a preliminary tuning process aimed at designing feasible input values for each scenario. The initial and reservation prices are drawn at random from uniform distributions, while the binary matrix is randomly generated according to a density parameter.

As these tests were successful, we can verify the CIDA model proposed in this work. The configurations of the auction scenarios analyzed are shown in Table 1. Both price step parameters were set to 5 for all the scenarios, while the round-timer was set to 4 for the CIDA simulator. In the last row, we show the value of the objective function (social welfare) in the last round of each scenario. This value was the same in the VBA and the CIDA simulator for all the scenarios. We can see that an increase in the total number of participants generally leads to higher social welfare, as more trades are closed.

Table 1: Auction scenarios simulated and allocation results.

Auction parameters	**Purchase price step = Ask price step = 5; Round-timer = 4**						
Scenario	**I**	**II**	**III**	**IV**	**V**	**VI**	**VII**
Buyers	10	10	20	20	20	40	40
Sellers	10	20	10	20	40	20	40
Social welfare Final alloc.	209	206	316	384	525	506	576

5 CONCLUSIONS AND FURTHER RESEARCH

In this paper, a DEVS model for a combinatorial iterative double auction has been formulated and implemented in the Cadmium simulator. The formal definition presented in this work aims to address the scarcity of specialized simulation frameworks that enable decentralized implementation of combinatorial and double auction mechanisms. Besides, although initially designed with inspiration from the AM production context, the auction model can be easily adapted to simulate the dynamics of other e-procurement markets.

The DEVS formalism has allowed to develop a robust framework for our auction model that can be gradually extended to reproduce a more realistic auction system. This DEVS-based decentralized model has allowed to overcome some limitations of the previous VBA-based centralized model, such as the need to assume perfect rationality of agents and to define their price updating strategies, enabling a more flexible and realistic model within an agent-based framework. A case study where the output of the new DEVS model was tested against the output of the previous centralized VBA model has allowed verifying the CIDA model proposed.

As mentioned above, the constructed model allows simulating the interaction of agents with different behaviors. For example, we can simulate that manufacturers/customers have perfectly rational behaviors that lead them to make optimal decisions or simulate cognitive or emotional limitations that lead them to make decisions that are not as optimal from a rational point of view. The model also allows combining agents with different behaviors (rational/non-rational) and analyzing the outcome of their interactions.

Models of this nature can also be implemented using Agent-Based Modeling (ABM) languages. However, in these cases, the model definition and its implementation are typically mixed and tied to the language. Thus, there is no separation of concerns as it occurs in the DEVS formalism. Additionally, most ABM languages follow a time-step approach, and, in this case, given that agent decisions occur within well-defined time windows and not at predefined time steps, the DEVS formalism is more suited.

Further research could extend the model presented in this work to develop a more realistic auction scenario with agents not following standard behavior. New *Buyer* and *Seller* models with different behaviors that might be counterintuitive or, at least, not necessarily seek result optimization could be defined straightforwardly. Thanks to the modularity property of DEVS, these new buyer and seller models can be defined without modifying the *Auctioneer* model, thus allowing an incremental extension of the existing model. Also, some randomization could be introduced in the decision-making process of agents. A simulation of such auction scenarios can provide a better insight into the expected results in a real-world environment where agents can pursue unpredictable strategies.

ACKNOWLEDGEMENTS

The authors wish to acknowledge MCIN/AEI, Spanish Government, and /10.13039/501100011033/FEDER UE, European Union, for the partial support through the PID2022-137948OA-I00 Research Project and the grant FPU19/01304 received by the corresponding author.

REFERENCES

Abedrabboh, K. and L. Al-Fagih. 2023. “Applications of Mechanism Design in Market-based Demand-side Management: A Review”. *Renewable and Sustainable Energy Reviews* 171:113016.

Abedrabboh, K., A. Karaki, and L. Al-Fagih. 2023. “A Combinatorial Double Auction for Community Shared Distributed Energy Resources”. *IEEE Access* 11:28355–28369.

Abrache, J., T. G. Crainic, and M. Gendreau. 2004. “Design Issues for Combinatorial Auctions”. *4OR* 2(1):1-33.

Belloli, L., D. Vicino, C. Ruiz-Martin, and G. Wainer. 2019. “Building Devs Models with the Cadmium Tool”. In *2019 Winter Simulation Conference (WSC)*, 45–59 https://doi.org/10.1109/WSC40007.2019.9004917.

Calheiros, R. N., R. Ranjan, C. A. F. De Rose, and R. Buyya. 2009. “CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services”. *arXiv:0903.2525*.

Cárdenas, R. and G. Wainer. 2022. "Asymmetric Cell-DEVS Models with the Cadmium Simulator". *Simulation Modelling Practice and Theory* 121:102649.
De Antón, J., F. Villafáñez, D. Poza, and A. López-Paredes. 2024. "An Iterative Price-based Combinatorial Double Auction for Additive Manufacturing Markets". *Computers & Industrial Engineering*, to be published.
De Vries, S. and R. V. Vohra. 2003. "Combinatorial Auctions: A Survey". *INFORMS Journal on Computing* 15(3):284–309.
Evans, B. P. and M. Prokopenko. 2024. "Bounded Rationality for Relaxing Best Response and Mutual Consistency: The Quantal Hierarchy Model of Decision Making". *Theory and Decision* 96:71–111.
Gorbanzadeh, F., A. Asghar, and P. Kazem. 2012. "Hybrid Genetic Algorithms for Solving Winner Determination Problem in Combinatorial Double Auction in Grid". *International Journal of Artificial Intelligence* 1(2):54–62.
Jiang, X., Y. Sun, B. Liu, and W. Dou. 2022. "Combinatorial Double Auction for Resource Allocation with Differential Privacy in Edge Computing". *Computer Communications* 185:13–22.
Jiang, Z. Z., S. C. Fang, Z. P. Fan, and D. Wang. 2013. "Selecting Optimal Selling Format of a Product in B2C Online Auctions with Boundedly Rational Customers". *European Journal of Operational Research* 226(1):139–153.
Khalid, A., C.-H. Hsueh, T.-H. Wei, H. Iida, A. Levi, and S. Alkoby. 2022. "On the Reality of Signaling in Auctions". *Information* 13(11):549.
Palacios-Huerta, I., D. C. Parkes, and R. Steinberg. 2024. "Combinatorial Auctions in Practice". *Journal of Economic Literature* 62(2):517–53.
Parkes, D. C. 2006. "Iterative Combinatorial Auctions". In *Combinatorial Auctions*, edited by P. Cramton, Y. Shoham, and R. Steinberg, 41–78. Cambridge, MA: MIT Press.
Podwol, J. U. and H. S. Schneider. 2016. "Nonstandard Bidder Behavior in Real-world Auctions". *European Economic Review* 83:198–212.
Rothkopf, M. H., A. Pekeč, and R. M. Harstad. 1998. "Computationally Manageable Combinational Auctions". *Management Science* 44(8):1131–1147.
Shen, Z. J. M. and X. Su. 2007. "Customer Behavior Modeling in Revenue Management and Auctions: A Review and New Research Opportunities". *Production and Operations Management* 16(6):713–728.
Umer, A., B. Nazir, and Z. Ahmad. 2022. "Adaptive Market-oriented Combinatorial Double Auction Resource Allocation Model in Cloud Computing". *Journal of Supercomputing* 78(1):1244–1286.
Xia, M., J. Stallaert, and A. B. Whinston. 2005. "Solving the Combinatorial Double Auction Problem". *European Journal of Operational Research* 164(1):239–251.
Yang, H., R. Chen, and S. Kumara. 2021. "Stable Matching of Customers and Manufacturers for Sharing Economy of Additive Manufacturing". *Journal of Manufacturing Systems* 61:288–299.
Zehetner, D. and M. Gansterer. 2023. "Decentralised Collaborative Job Reassignments in Additive Manufacturing". *International Journal of Production Research* 62(14):5149–5167.
Zeigler, B. and A. Muzy. 2018. *Theory of Modeling and Simulation.* 3rd ed. London: Academic Press.

AUTHOR BIOGRAPHIES

JUAN DE ANTÓN is a Ph.D student in Industrial Engineering at the Department of Business Organisation and Market Research, University of Valladolid (UVa). He earned his B.Sc. and M.Sc. in Industrial Engineering in 2018 and in Project Management in 2019, from UVa. His research interests in the INSISOC Group include additive manufacturing, operations management and combinatorial auctions. His email address is juan.anton@uva.es.

CRISTINA RUIZ MARTIN is an Assistant Professor, Teaching Stream, at the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada). She received the DEVS Modeling and Simulation Ph.D. Dissertation Award from SCS (2019) and the Young Simulation Scientist Award from SCS (2020). She has been a member of the Board of Directors of SCS since July 2021. Her email address is cristinaruizmartin@sce.carleton.ca.

FÉLIX VILLAFÁÑEZ is an Associate Professor in the Department of Business Organisation and Market Research, University of Valladolid (UVa). He obtained his Ph.D. in 2014 on project scheduling optimization from UVa. His research interests in the INSISOC Group include agent-based modeling, combinatorial auctions, project management and project scheduling. His email address is felixantonio.villafanez@uva.es.

DAVID POZA is an Associate Professor in the Department of Business Organisation and Market Research, University of Valladolid (UVa). He received his Ph.D. in 2012 on applications of agent-based simulation to the emergence and diffusion of socio-economic norms from UVa. His research interests in the INSISOC Group include additive manufacturing, project management, project scheduling and combinatorial auctions. His email address is: david.poza@uva.es.