A Configurable Auction Framework for Open Agent Systems

Mao Weiliang Wang Xiaodong Sheng Huanye

Dept. of Computer S&E, Shanghai Jiao Tong University, Shanghai, China

Abstract. Auction mechanism is a good solution to the distributed resource allocation. However, in open agent system, it is impossible to fix the market configuration for resource allocation in advance because of the different participants and the variety of resources. It implies the necessity for configurability of auction mechanism.
This paper presents a configurable auction framework for resource allocation in the open agent system. Compared with other related work, the framework implemented here has its special features, including application-oriented, the more configurability flexibility, considerate auction abstraction, XML-based implementation and wider applicability.

1. Introduction

Auction has been proposed to be used to build price determination system as the core of market based resource allocation in various open agent based applications, just as done in the literatures [Clearwater1996, Mullen1996, Bredin1998, Cliff1998, Wellman1998, etc].
However, openness and dynamism of open agent system implies that it is impossible to fix the market configuration for resource allocation in advance in such type of system. On one hand, openness implies the variety of agent participants which are on behalf of different owners. Therefore, it is quite possible that different agents prefer different resource allocation mechanism. An example is about English Auction and Vickrey Auction. Theoretically, Vickrey auction has more advantages on the aspect of encouraging participant agents to truly reveal their valuation on the resource. However in Internet, English auction is the most popular type of auction and it is just because English Auction is more understandable. So, for the people who understand Vickrey mechanism, they may prefer to it instead of English Auction.
On the other hand, open and dynamic environment implies the variety of resource. Different resources often have different proper market-based mechanism. For instance, for the resource with multi-seller and multi-buyer, the CDA auction is fitting; for the multi-unit resource, the multi-unit auction is needed; for the information goods devaluating with time, Dutch auction is more proper that English auction. For the resource which needs to be allocated as soon as possible, sealed-bid types of auctions are more useful than open-cry types. Thus, in order to support different auction types corresponding to different types of resource, the market mechanism in the open agent system is required to be flexible enough.
Therefore, in order to provide sufficient flexibility, open agent system must support dynamic market configuration. The main work in this paper is just on a design and implementation of a configurable auction framework.

2. Abstracting Auction Model

This section will present auction model from two aspects:
- *Static* aspect: it deals with the components and their interrelation in an auction
-*Dynamic* aspect: it describes the process model of auction.

2.1. A Static Formulation of Auction

Definition 1 Participants P in an auction:

- Buyers set in an auction--$B= \{b_1, b_2, ...\}, |B|>0$
- Sellers set in an auction--$S= \{s_1, s_2, ... \}, |S|>0$
- Auctioneer in an auction—a, in an auction, auctioneer is traditionally single

Thus, participants P in an auction can be defined as a unification, i.e. $P=B \cup S \cup \{a\}$.

Definition 2 Activities A in an auction

- Bidding activity set in an auction--A_{bid}

Bidding action $a_{bid} \in A_{bid}$ is carried out by $b_i \in B$ and $s_j \in S$. For every a_{bid}, there is the parameter of bid information to be sent to the auctioneer a by which b_i and s_j indicates their willingness to participate in auction. Here, the parameter is generalized as *bidinfo*. Thus, $a_{bid}(bidinfo)$ denotes a full form of bidding action in auction.

- Call For Bids (CFB) activity set in an auction--A_{CFB}

Call-for-bids action $a_{CFB} \in A_{CFB}$ is carried out by auctioneer a. For every a_{CFB}, there is also the parameter of call-for-bids information to be sent to each b_i and s_i. Here, the parameter is generalized as *CFBinfo*. Thus, a_{CFB} *(CFBinfo)* denotes a full form of call-for-bids action in auction.

- Receiving bids activity set in an auction--A_{rbid}

Receiving bids action $a_{rbid} \in A_{rbid}$ is carried out by auctioneer a. For every a_{rbid}, there is the parameter of bids information *bidinfo* from b_i and s_i, which will be verified here according to the auction rules. a_{rbid} *(bidinfo)* is a full form of receiving bids action in auction.

- Activity set of clearing auction--A_{clear}

Clearing action $a_{clear} \in A_{clear}$ is carried out by auctioneer a to form auction result. Generally, in an auction, clearing action is done only one time, i.e. $|A_{clear}|=1$. In sequential multi-unit auction, there is $|A_{clear}|>=1$. Action a_{clear} needs all the *bidinfo* received from b_i and s_j, here which is generalized as *bidinfolist*. A_{clear} *(bidinfolist)* is a full form of clearing action in auction.

- Activity set of revealing intermediate information--A_{med}

Action revealing intermediate information $a_{med} \in A_{med}$ is carried out by auctioneer a to supply each b_i and s_j with some form of intermediate status information, typically in the form of hypothetical result, which is generalized as *quotesinfo*. $a_{med}(quotesinfo)$ is the full form.

- Activity set of revealing auction result information--A_{end}

Action revealing result information a_{end} is also performed by auctioneer a. Like action a_{clear}, it is in sequential multi-unit auction, there is $|A_{end}|>=1$. a_{end} has the parameter of result information which is denoted as *resultinfo*.

Thus, activities A in an auction can also be defined as unification, i.e. $A= A_{bid} \cup A_{CFB} \cup A_{rbid} \cup A_{clear} \cup A_{med} \cup A_{end}$. Specially, according to the performer of action, some subsets of A can be defined. If using *performer* (a_{act}) to denote the performer of action a_{act}, the

activities set of buyers in an auction is defined as A_B={ a_{act} | a_{act} $\in A_{bid}$ and *performer* (a_{act}) $\in B$}. The activities set of sellers is correspondingly to be defined as A_S={ a_{act} | a_{act} $\in A_{bid}$ and *performer* (a_{act}) $\in S$}. In an auction, there is $A_B \cup A_S = A_{bid}$. The activities set of auctioneer a is $A_a = A_{CFB} \cup A_{rbid} \cup A_{clear} \cup A_{med} \cup A_{end}$. So, also there is $A = A_B \cup A_S \cup A_a$. For every participant $pi \in P$, its activities set A_{pi}={ a_{act} | a_{act} $\in A$ and *performer*(a_{act})=p_i}

Definition 3 Auction rules R in an auction

Auction rules R regulates how every action in an auction is performed. Therefore, auction rules set corresponding to various activities could be defined. There are set R_{bid} for A_{bid} set R_{CFB} for A_{CFB}, set R_{rbid} for A_{rbid}, set R_{clear} for A_{clear}, set R_{med} for A_{med}, and set R_{end} for A_{end}. Therefore, auction rules R is also a form of unification, i.e. $R = R_{bid} \cup R_{CFB} \cup R_{rbid} \cup R_{clear} \cup R_{med} \cup R_{end}$. The rule set for buyers and sellers are R_B={ r_{act} | r_{act} $\in R_{bid}$ *and performer* (r_{act}) $\in B$ } and R_S= { r_{act} | r_{act} $\in R_{bid}$ *and performer* (r_{act}) $\in S$}. Rule set for auctioneer a is $R_a = R_{CFB} \cup R_{rbid} \cup R_{clear} \cup R_{med} \cup R_{end}$. For every participant $pi \in P$, its rule set R_{pi}={ r_{act} | r_{act} $\in R$ and *performer*(r_{act})=p_i}.

Definition 4 Goods G in an auction

Goods in an auction can be defined as a set, i.e. $G = \{g_1, g_2, ...\}$, $|G| >= 1$.

For each $g_i \in G$, g_i denotes goods set with type i. g_i may be indivisible or divisible. In the former case, there is the actual amount 1 of g_i. In the latter case, g_i may be discrete or continuous. In the computational environment, there is always the minimal indivisible unit. By assuming the minimal indivisible unit with value 1, *amount*(g_i) can be the uniform form denoting the amount of goods g_i, and there is *Amount*(g_i)>=1. For the $g_j \in G$, $g_j \neq g_j$, g_i and g_j may be substitutable with each other (multi-object but single-dimensional auction) or independent (Multi-dimensional auction) with each other.

Based on the above definitions of participants P, goods G, auction rules R and activities A, auction model Au can be defined:

Definition 5 auction static model:

Auction is defined as a tuple $Au = <P, G, R, A>$.

- For each participants $p_i \in P$, there is $p_i = <A_{pi}, R_{pi}>$
  For $p_i \in B$, there is $A_{pi} \subseteq A_B \subseteq A_{bid}$, $R_{pi} \subseteq R_B \subseteq R_{bid}$, and $\bigcup_{i=1}^{|B|} A_{bi} = A_B$, $\bigcup_{i=1}^{|B|} R_{bi} = R_B$.
  For $p_i = a$, there is $A_{pi} = A_a$, and $R_{pi} = R_a$
  For $p_i \in S$, there is $A_i \subseteq A_S \subseteq A_{bid}$, $R_i \subseteq R_S \subseteq R_{bid}$, and $\bigcup_{i=1}^{|B|} A_{si} = A_S$, $\bigcup_{i=1}^{|B|} R_{bi} = R_S$.
- In auction Au, each buyer $b_i \in B$ performs $a_{bid} \in A_{bi}$ by applying rule $r_{bid} \in R_{bi}$ to get part or all of goods G_{bi}={$g_{b(i,1)}$, $g_{b(i,2)}$ ... }, where $|G_{bi}| = |G|$ and for each $g_{b(i,j)} \in G_{bi}$ and $g_j \in G$, there is $g_{b(i,j)} \subseteq g_j$.
- Each seller $s_i \in S$ performs $a_{bid} \in A_{si}$ by applying rule $r_{bid} \in R_{si}$ to sell part or all of goods G_{si}={$g_{s(i,1)}$, $g_{s(i,2)}$... }, where $|G_{si}| = |G|$ and for each $g_{s(i,j)} \in G_{si}$ and $g_j \in G$, there is $g_{s(i,j)} \subseteq g_j$.
- Auctioneer a performs $a_{act} \in A_a$ by applying rule $r_{act} \in R_a$ to form the final goods allocation result for each b_i and s_i. The result will be a tuple $G_{result} = <G_{result1}, G_{result2}, ..., G_{resultN}>$, where $N = |P| - 1$ and $G_{resulti} = \{g_{resulti1}', g_{resulti2}', ...\}$ for $p_i \in P - \{a\}$.
  When $P_i = b_j \in B$, $G_{resulti}$ denotes the final goods assigned to buyer b_j at some price. When $P_i = b_k \in S$, $G_{resulti}$ denotes the final goods unassigned of seller s_k. Apparently, for any type of goods g_m, there is the balance, i.e.

$$\sum_{i=1}^{|P|-1} amount(g_{result(i,m)}) = \sum_{i=1}^{|S|} amount(g_{s(i,m)}).$$

– For the reason of $p_i=<A_{pi}, R_{pi}>$, essentially auction is also able to be denoted as the tuple $Au=<P, G>$. The difference is that tuple $Au=<P, G, R, A>$ has the emphasis on auction rules and actions.

2.2. Process Model of Auction

Definition 6 Atom action: in the auction static model, activities set A is defined, which consists of several main classes of actions shown as table 1. $a_{bid,}$, $a_{CFB,}$ $a_{rbid,}$ $a_{med,}$ $a_{clear,}$ and a_{end} are defined as the atom actions in the auction.

Participant p_i	**action**	**action description**
$b_i \in B$	a_{bid}	From b_i to a with parameter *bidinfo* for buying goods
$s_i \in S$	a_{bid}	From s_i to a with parameter *bidinfo* for selling goods
A	a_{CFB}	From a to each b_i or s_i with *CFBinfo* for calling for bids
	a_{rbid}	Receiving and verifying *bidinfo* from b_i and s_i
	a_{clear}	Computing result *resultinfo* with parameter *bidinfolist*
	a_{med}	From a to each b_i or s_i with *quoteinfo* to show intermediate information
	a_{end}	From a to each b_i or s_i with *resultinfo* to ending auction

Table 1 Main Actions in Auction Model

Definition 7 Action a_{act}:

1) Atom action a_{actm} is a a_{act}
2) Compound action a_{com} is also a a_{act}. Recursively, compound action a_{com} is an execution of a set of actions $\{a_{act1}, a_{act2}, ...\}$.

Definition 8 Sequential action a_{seq}: There are N actions a_{act1}, $a_{act2,...,}a_{actN}$ $(N>1)$ performed sequentially, when for any a_{acti} and a_{actj} $(N>=j>i>=1)$, a_{acti} is performed previously to a_{actj}. Sequential action can be denoted as the expression: $a_{seq} = \begin{pmatrix} a_{act1} \\ a_{act2} \\ \cdots \\ a_{actN} \end{pmatrix}$. When $N=1$, expression $[a_{act1}]$ is a special form of sequential action.

Definition 9 Parallel action a_{par}: There are N atom actions a_{act1}, $a_{act2,...,}a_{actN}$ $(N>1)$ performed parallelly, when for any a_{acti} and a_{actj} $(N>=i>=1, N>=j>=1, j\neq i)$, there is no determinate sequence, in other words both $\begin{pmatrix} a_{acti} \\ a_{actj} \end{pmatrix}$ and $\begin{pmatrix} a_{actj} \\ a_{acti} \end{pmatrix}$ are the possible cases.

Parallel action can be denoted as the expression: $[a_{act1}, a_{act2}, ..., a_{actN}]$. Similarly, when $N=1$, expression $[a_{act1}]$ is also a special form of parallel action.

Definition 10 Repeating action: For any atom action $a_{act,}$ if in the auction, a_{act} is performed repeatedly until some condition $cond_{rep}$ is satisfied, a_{act} is a repeating action, which is denoted as expression $(a_{act}, cond_{rep})^*$ with simplified form a_{act}^*. Thus a_{act}^* implies a_{act} will be performed at least one time.

Next, based on the above definitions, auction process model can be built, which formally must be a compound action.

Firstly, an auction process can be divided into three stages: Starting Stage, Interacting Stage and Ending Stage.

Starting Stage: in this stage, preparation activities will be done, including auction initiator p_1 sending goods information to auctioneer (including action a_{bid}), auctioneer a sending call for bids information (i.e. a_{CFB}).

Firstly, when $p_1 \in B$, there is $|B|=1$. Such case means the auction types with the single buyer side. In this case, $p_1=b_i$ performs $a_{atom}= a_{bid}$. Then a performs $a_{atom}= a_{CFB.}$ That is to say, there is a sequential action $\begin{pmatrix} a_{bid} \\ a_{CFB} \end{pmatrix}$.

Secondly, when $p_I \in S$, there is also $|S|=1$. Such case means the traditional auction types with the single seller side. In this case, $p_I=s_i$ performs $a_{atom}= a_{bid}$. Then a performs $a_{atom}= a_{CFB.}$ That is to say, there is also a sequential action $\begin{pmatrix} a_{bid} \\ a_{CFB} \end{pmatrix}$.

Thirdly, when $p_I=a$, in this stage, there is no action a_{bid} from any b_i or s_i. This case means the double-side type of auctions. There is the sequential/parallel action $[a_{CFB}]$.
If starting stage is considered separately, every action in this stage is performed only once.

Interacting Stage: This stage is the main body of auction process. In this stage, auctioneer a interacts with buyer $b_i \in B$ and seller $s_i \in S$ for receiving *bidinfo*s. Correspondingly, action execution in this stage is analyzed based on different cases of P_I.

When $p_I \in B$, $s_i \in S$ performs the action a_{bid} to a. After that, for the auctioneer a, there are actions a_{rbid} and a_{med}. It can be represented as a sequential action $\begin{pmatrix} a_{bid} \\ a_{CFB} \\ a_{med} \end{pmatrix}$.

When $p_I \in S$, $b_i \in B$ has the action a_{bid} to a. After that, the auctioneer a also performs the actions a_{rbid} and a_{med}. It can be represented as the same form of sequential action $\begin{pmatrix} a_{bid} \\ a_{CFB} \\ a_{med} \end{pmatrix}$.

When $p_I=a$, both $s_i \in S$ and $b_i \in B$ will perform the action a_{bid} parallelly. After that, auctioneer a performs the same actions as above. It can be represented as a sequential action embedding parallel action: $\begin{pmatrix} a_{bid} & a_{bid} \\ & a_{CFB} \\ & a_{med} \end{pmatrix}$

Next, repeating action can be further considered.

For each b_i and s_i, their action sets $A_{bi}=\{ a_{bid} \mid a_{bid} \in A_B$ and $performer(a_{bid})=b_i\}$ and $A_{si}=\{a_{bid} \mid a_{bid} \in A_S$ and $performer(a_{bid})=s_i\}$ have the properties:

1) When $p_I \in B$, there are $|A_{bi}|=1$ and $|A_{si}|>=1$. b_i performs one a_{bid} in the starting stage but no a_{bid} performed in interacting stage. s_i will perform at least one a_{bid} in interacting stage.
2) When $p_I=a$, there are $|A_{bi}|>=1$ and $|A_{si}|>=1$. Both bi and s_i will perform at least one a_{bid} in interacting stage.
3) When $p_I \in S$, there are $|A_{bi}|>=1$ and $|A_{si}|=1$. s_i performs one a_{bid} in the starting stage but no a_{bid} performed in interacting stage. b_i performs at least one a_{bid} in interacting stage.

That is to say, under the consideration of various auctions, in interacting stage, action a_{bid} can be performed by b_i or s_i once (e.g. in single-unit sealed-bid auction) or many times (i.e. in single-unit English auction).

For auctioneer a, according to the definition of action a_{rbid}, a_{rbid} is repeated along with action a_{bid}. Additionally, after auctioneer a performs action a_{med}, for example in open-cry auction, b_i or s_i will perform a_{bid} again, that is to say, interacting stage itself is repeated.

Ending Stage: This stage mainly involves auctioneer a computing the final goods allocation G_{result} by performing action a_{clear} and sending $G_{resulti}$ or whole G_{result} to every $p_i \in P-\{a\}$ by performing action a_{end}. In this stage, no matter what p_I is, the action execution in this stage is uniformed, i.e. a sequential action: $\begin{pmatrix} a_{clear} \\ a_{end} \end{pmatrix}$.

Auction dynamic model A_u': From the above analysis, it is known that each stage of an auction process is actually an action. That is to say, an auction process model itself can also be defined as a sequential action which is made of three stage actions with the form $Au' = \begin{pmatrix} a_{SS} \\ a_{IS} \\ a_{ES} \end{pmatrix}$.

However, in some auctions, such as multi-unit-multi-round auction, an auction will be repeated. That is to say, Au' should be $\begin{pmatrix} a_{SS} \\ a_{IS} \\ a_{ES} \end{pmatrix}^*$. Thus, overall auction process model can be shown as the following figure.

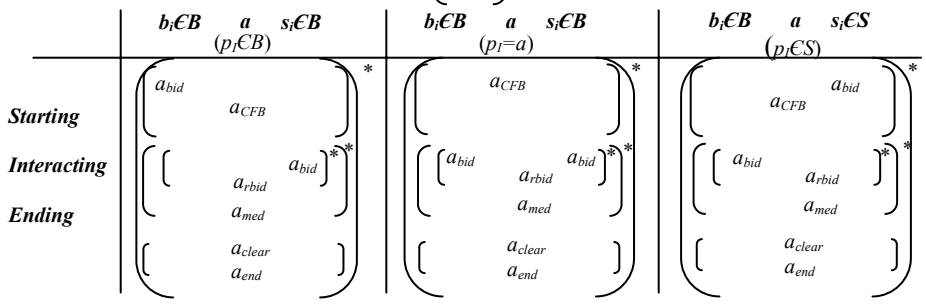


Figure 1 Auction Process Model

2.3. Configurable Items in Auction Model

The previous sections have formally described a full auction model from both static and dynamic aspects. Following that, the items and their various possible cases in the auction model can be shown in the table below:

Items		Possible Cases	Brief Description
P		$\|B\|>>1$ $\|S\|=1$	many buyers, single seller
		$\|B\|=1, \|S\|>>1$	single buyer, many sellers
		$\|B\|>>1, \|S\|>>1$	many buyers, many sellers
G	$\|G\|$	$\|G\|=1$ / $\|G\|>1$	Single object, multi-object
	g_i	$Amount(g_i)=1$/$amount(g_i)>1$,	Single unit, multi-unit
	g_i, g_j,	substitutable / independent	Single-dimension,multi-dimension
	g_i info	Described in some L_g	It can not be generalized.
A	$Bidinfo_{SS}$	((goods1number,[reservedprice], increment/decrement], goods1description,[…]), …)	All those parameters have so various forms that they can not be generalized into some determinate model. For example of $Bidinfo_{SS}$, not only there is unpredictable form of goods description, i.e. L_g, but also other new items are possible to be included when necessary, e.g. information about seller or buyer which may be related to reputation
	$Bidinfo_{IS}$	(([goodstype],[goodsnumber], price,[bidderinformation]) /(demandfunction/signal, [bidderinfomation])…)	
	$CFBinfo$	([bidinfo$_{SS}$],English/Dutch /Vickrey/ user-defined/…, [auctiontimesetting] ,[…])	
	$quoteinfo$	(latestclearingprice/latestgoodallocation/…, [CFBinfo])	
	$bidinfolist$	(bidinfo$_{IS}$, bidinfo$_{IS}$…)	
	$resultinfo$	((winner, winningprice, [goods type], [goods quantity])…)	
R	$cond_{bidSS}$	$Auctionstarting$	When performing a_{bidSS}
		English/Dutch/Vickrey/user-defined/…	Auction types for $bidinfo_{SS}$ form
		$TRUE$	Assign $bidinfo_{SS}$ in advance
	$cond_{bidIS}$	Receiving CFBinfo/quoteinfo	When performing a_{bidIS}
		English/Dutch/Vickrey/user-defined/…	Auction types for $bidinfo_{IS}$ form
		User-defined	User-specified condition for determining $bidinfo_{IS}$ value
	$cond_{CFB}$	Receivingbidinfo$_{SS}$/auctionstarting	When performing a_{CFB}
		English/Dutch/Vickrey/user-defined/…	Auction types for $bidinfo_{IS}$ form
		$TRUE$	Assign $CFBinfo$ in advance
	$cond_{rbid}$	(Receiving bidinfo$_{IS}$, English/Dutch /Vickrey/user-defined/…)	When performing a_{rbid} according to auction type.
		$TRUE$	$Bidinfo$ is from a_{bid}

		TRUE	*Bidinfo* is from a_{bid}
	$cond_{med}$	*nominal time set/newbid/nobid/ none*	cases for performing action a_{med}
		English/Dutch/Vickrey/user-defined/…	Auction types for *quoteinfo* form
		English/Dutch/Vickrey/user-defined/…	Auction types for *quoteinfo* computation policy, such as *Maximalprice/equilibriumprice/…*
	$cond_{clear}$	*(nominal time set/newbid/nobid/ randomtime/outersignal/ combination,English/Dutch/Vickrey/user-defined/…)*	Cases for performing action a_{clear}, auction type is for the form of *resultinfo*, and for clearing policy: *Maximalprice/equilibriumprice/ secondprice/user-defined/…* which determines *resultinfo* value.
		TRUE	*bidinfolist* is a list of $bidinfo_{IS}$
		TRUE	*Bidinfolist* is from $bidinfo_{IS}$
	$cond_{end}$	*nominal time set/newbid/nobid/randomtime/outersignal/ combination*	Cases for performing action a_{end}
		TRUE	*Resultinfo* is from a_{clear}
		TRUE	*Resultinfo* is from a_{clear}
	$cond_{rep0}$	*timespan/nogoods/assignedtimes /outersignal/combination*	Cases for stopping auction repeating
	$cond_{rep21}$	*timespan/assignedtimes /outersignal/statuspoint/combination*	Cases for stopping interacting stage repeating
	$cond_{rep2111}$	*timespan/nogoods/assignedtimes /outersignal/nobid/combination*	Cases for stopping action a_{bid} repeating

Table 2 Items in Auction Model

From the above table, it is not difficult to find that there are two types of configurable items, which includes the one that has enumerable cases. For example there are the configurable item participants *P*, the most items for Goods *G*, rules item $cond_{rep0}$, and so on. The other type of configurable item is that which has un-expectable cases. Actually, most of the items in the table are in this case, for example the various forms of action parameter.

3. An Open Auction Framework

Constructing a configurable framework is to shield that variety by providing a uniform interface to both auction developer and participants. That implies the configurability firstly has two layers, i.e. one for developer and the other for participants. Developer configurability involves the items in table 5.1, including configuring participants *P,* goods *G* and activities *A*. It also involves the part of rules *R* for basic process control of auction, i.e. the *cond1* and *cond2* of each rule r_{act} as well as conditions $cond_{rep0}$, $cond_{rep21}$, $cond_{rep2111}$ for compound actions. The user configurability mainly involves value determination of parameters of each action.

From other perspective, directly implied from the auction static and dynamic model, configurability should also include the static and dynamic aspects. For the developer, the former involves how to configure the components of auction, i.e. configuring participants *P,* goods *G* and activities *A*. Dynamic configurability means developer setting up conditions for rules to form the auction process, and also means how to determine the value of parameters including *quoteinfo, resultinfo, bidinfolist* where condition setting indicates the selection of computing policy, such as clearing policy. For the user, static configurability means the static configuration of various parameters' value, i.e. value assigned in advance including $bidinfo_{SS}$ and *CFBinfo*. And dynamic one indicates setting up the conditions, i.e. $cond_3$, for parameter value determination. This also implies the value is determined during the auction process. In our model, the related parameter is $bidinfo_{IS}$.

Consequently, the auction configurability can be abstracted as the following layered model:

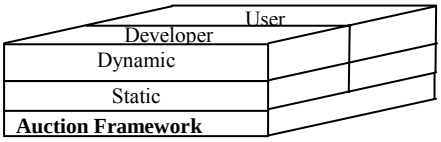


Figure 2 Auction Layered Model of Configurability

Thus, we can further give an open solution to auction configurability:

- Developer Configurability

The **static configurability** for auction developer includes configuring participants *P,* goods *G* and activities *A* in the table 2, which this paper calls auction feature information, they have enumerable cases. As for the good description language L_g, this paper assumes that every auction developer creates auction always for some special types of goods. Such decentralized solution has at least two advantages: firstly it makes describing various goods possible and endows framework more flexibility (e.g. new type of goods is not limited). Secondly, the goods language can be simplified because of the lack of generality (e.g. developer can directly simplify goods description if necessary).

As shown in table 2, the parameters' form of each action is not determinate. In this situation, framework also does not try to provide a uniformed form as each action parameter. On the contrary, in order to meet the requirement of different auction developer for different auction, the strategy allowing developer to define parameter form is taken. The work left to framework is just to provide interface to support such definition, for example maybe providing some description language and interpreter.

As said above, to developer, **dynamic configurability** implies setting up conditions for rules to form the auction process. According to the table 2, the cases for $cond_1$ are so complex (e.g. representing outer signal, random time, cases combination) that it is not enough or impossible for framework providing a fixed interface to cover them. Therefore, in order to enable developer flexibly making such configuration, providing rule description language is a more fitting solution. On the other hand, developer dynamic configurability also involves the condition setting for computing the value of parameters. For the parameter *quoteinfo*, condition mainly is a method, e.g. maximal price policy, equilibrium price, etc, for computing auction intermediate information. For the parameter *bidinfolist* which is from action a_{rbid}, the condition is for action a_{rbid} how verifying *bidinfo*. For the parameter *resultinfo* which is computed by action a_{clear}, the condition is also a method related to the clearing policy. The fact is that there are various auctions which have different *quoteinfo* and *resultinfo* computing policy and verifying policy for *bidinfolist*. More possible is that new types of auction with changed computing and verifying policy is ready to appear at all times especially when auction developer has their special purpose (e.g. auction developer is the seller, when he would like to add some more verify policy for goods auction). Therefore, auction framework should also allow auction developer to design and add their policies for parameter value determination. In other words, rule description language for auction developer should have such functionality.

- User Configurability

The **static configurability** means the static configuration of value of parameter $bidinfo_{SS}$, and *CFBinfo*. In this case, what user does is that assign the values according to the parameter form provided by auction developer.

The **dynamic configurability** requires the parameter value is determined during auction process. For the parameter $bidinfo_{IS}$, dynamic value implies bidder setting $bidinfo_{IS}$ in term of the current auction condition, i.e. $cond_3$ in the rule. Thus, in order to support such conditional bidding, framework should allow user to define their bidding strategies and embed them into the auction process described by developer, i.e. framework providing bidding strategies description interface which can be combined with rule description language for developer.

- Auction Framework Functionality Model

It can be found that openness of the framework is shown on both the developer and the user configurability. Framework gets more openness by encouraging the participation of developer and user into the framework. Based on the above solution to configurable auction framework, functionality of framework can be concluded into the following figure:

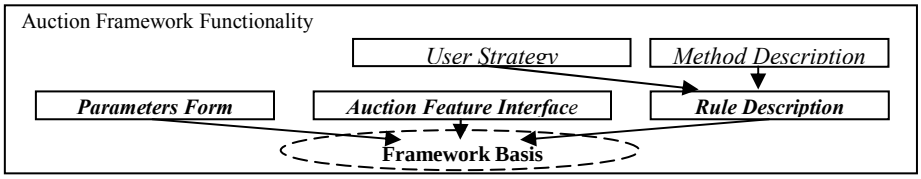


Figure 3 Auction Framework Functionality Model

For developer configurability, auction framework provides auction feature interface for developer configuring auction feature information, supports developer creating parameter form, and supports developer creating auction process by configuring rules. At the same time, in order to support the dynamic value determination, framework supports developer's method description which is embedded in the auction process.

For the user configurability, auction framework supports user specifying the strategies for dynamically assigning parameter value, which is also embedded in the auction process.

- Framework Basis

In the above auction framework functionality model, all the parts associated with developer and user configurability are the interfaces that auction framework provides for auction developer and user. For a truly applicable auction framework, a basic implementation framework to support those interfaces' implementation and integration is needed, that is the framework basis in figure 3. For instances, framework basis will have the necessary interpreter for rule description, the interpreter for user strategy description, methods for visiting parameter form definition and auction feature definition, and so on.

4. Implementation

4.1. Auction Feature Interface

Under the consideration of advantages of XML technology on expressivity, XML is used as the meta-language to form auction feature interface by embedding the above keywords as markups. Shown as below, the figure is the structure of auction feature interface and an example setting for auction with multiple heterogeneous objects, single-unit, single seller, multiple buyers and with developer specified auction type name *MultiDimEnglish*.

```
<?xml version=""1.0>
<start>
  <Auction_Feature_Information>
    <auction_type>value</auction_type>
    <buyer_number>value</buyer_number>
    <seller_number>value</seller_number>
    <goods_type_number>value</goods_type_number>
    <goods_number>value</goods_number>
    <goods_relation>value</goods_relation>
  </Auction_Feature_Information></start>
```

```
<?xml version=""1.0>
<start>
  <Auction_Feature_Information>
    <auction_type>MultiDimEnglish</>
    <buyer_number>many</>
    <seller_number>one</>
    <goods_type_number>many</ >
    <goods_number>one</>
    <goods_relation>independent</ >
  </Auction_Feature_Information></start>
```

Figure 4 XML-Based Auction Feature Interface for Developer

4.2. Parameter Form

Four basic points are considered to support auction developer defining parameter form. Firstly, framework should provide easy-to-use interface. Secondly, from the table 2, there are some determinate items, such as *goodsname, goodsnumber* in $bidinfo_{SS}$. It is needed to define those items in advance in auction framework. Thirdly, those parameters are not independent with each other. Some parameters' form can be included in some other parameters' form i.e. $bidinfo_{IS}$ in *bidinfolist*, *CFBinfo* in *quoteinfo*, $Bidinfo_{SS}$ in *CFBinfo*. Auction framework should support the relation description. Fourthly, parameter form is often repeated. Parameter *bidinfolist* is a repeated form of the form of $bidinfo_{IS}$; $bidinfo_{SS}$ and $bidinfo_{IS}$ will do when there are multiple types of goods; *resultinfo* and *quoteinfo* may be repeated for multiple winners and multiple goods. Under the above consideration, XML is also used to implement the interface as shown in figure 5(a). Figure 5 (b) shows an example of parameter form definition for single-object, single-unit English Auction.

```
<Parameter_Form>
  <bidinfo_SS>
    <good_item>
      <good_name></><good_number></>
      <good_description></>
      <developer_defined></>
      <repeat> developer_defined</></>
    <repeat>good_item</></>
  <bidinfo_IS>
    <developer_defined></>
    <repeat> developer_defined</></>
  <CFBinfo>
    <developer_defined></>
    <repeat> developer_defined</></>
  <quoteinfo>
    <developer_defined></>
    <repeat> developer_defined</></>
  <bidinfolist>
    <param_form>bidinfo_IS</>
    <repeat> param_form </></>
  <resultinfo>
    <result_item>
      <winner></><winning_price></>
      <developer_defined></>
      <repeat> developer_defined</></>
    <repeat>result_item</></></>
```

(a) Interface Description

```
<Parameter_Form>
  <bidinfo_SS>
    <good_item>
    <good_name/><good_number/>
    <good_description/></></ >
  <bidinfo_IS>
    <bidder_name/>
    <price/></>
  <CFBinfo>
    <param_form>bidinfo_SS</></>
  <quoteinfo>
    <maximal_price/ >
    <param_form>CFBinfo</ ></>
  <bidinfolist>
    <param_form>bidinfo_IS</>
    <repeat>param_form </></>
  <resultinfo>
    <result_item>
      <winner/>
      <winning_price/></></></></>
```

(b) Example

Figure 5 XML-based Parameter Form Definition Interface for Developer

4.3. Rule Description

Rule Description and AIML

Auction model has described that there are three types of participants in auction, i.e. *buyer, seller* and *auctioneer*. Those participants will apply their rules to perform actions and eventually form auction process. Therefore, auction actually can be seen as a process where participants coordinate with each other. Auction itself is a multi-agent system where the constituent agents are those participants. So, AIML[Mao2001], an XML-based agent markup language are used here.

In order to simplify the implementation of rule descriptions, when applying AIML, participants are further divided—auctioneer is divided into monitor, mediator, and bids receiver. Besides, registrar agent is introduced for the real implementation.

The sketch of rules is shown in the table below:

Participant	Rule set
Monitor	1) **(start)** ==>readFormOfCBFinfo, readValueOfCBFinfo,[**monitorInitAuction]**, **[actions]** S:a_{CFB}(CFBinfo) 2) R:Mediator.$S_{bidlistinfo}$ and **condition$_{end}$** ==>**a_{clear}($S_{bidlistinfo}$.bidinfolist)**, **[actions]**, S:a_{end}(resultinfo) 3) R:Mediator.$S_{bidlistinfo}$ and **condition$_{con}$** ==>**a_{clear}($S_{bidlistinfo}$.bidinfolist)**, **[actions]**, S:a_{con}(resultinfo)
Mediator	1) R:Monitor.a_{CFB}==> saveCFBinfo, [**mediatorInitAuction], [actions],** S:$a_{startCFB}$(Monitor.a_{CFB}.CFBinfo) 2) R:Monitor.a_{con} ==> **updateCFBinfo, mediatorResetAuction, [actions],** S:a_{conCFB}(CFBinfo, Monitor.a_{con}.resultinfo) 3) R:Monitor.a_{end} ==>**mediatorFinalizing,** S:a_{end}(Monitor.a_{end}.resultinfo) 4) R: Bids_Receiver.$S_{Bidlistinfo}$ and **!condition$_{Bidlistinfo}$** ==> **saveBidlistinfo**, readFormOfQuoteinfo, **setQuoteinfo, [actions],** S:a_{med}(quoteinfo) 5) **condition$_{Bidlistinfo}$**==>S:$S_{Bidlistinfo}$(Bids_Receiver.$S_{Bidlistinfo}$.bidlistinfo)
Bids Receiver	1) R:Mediator.$a_{startCFB}$ ==>saveCFBinfo, **[actions]**, S:AskAbout 2) R:Mediator.a_{conCFB} ==>saveResultinfo, **[actions]**, S:AskAbout 3) R:mediator.a_{med} ==> saveQuoteinfo, **[actions]**, S:AskAbout 4) R:Registrar.reply ==>saveBidderinfo,**[actions],** S:a_{CFB}((Mediator.a_{CFB}.CFBinfo)|(CFBinfo, mediator.a_{med}.Quoteinfo)|(CFBinfo, mediator.a_{conCFB}.resultinfo)) 5) R:Registrar.reply and condition$_{end}$ ==> saveBidderinfo, **bidsReceiverFinalizing**, **[actions],** S:a_{end}(mediator.a_{end}.resultinfo) 6) R: mediator.a_{end} ==> [**actions],** S:AskAbout 7) **condition$_{Bidlistinfo}$** ==>**[actions],** S:$S_{Bidlistinfo}$ (bidlistinfo) 8) [R:bidder.a_{bid} ==>verifyAndSaveBidinfo,**[actions]** S:bidResponse(verifyingresult)]
Registrar	1) R:Bids_receiver.AskAbout==>getBidderinfo,**[actions]**,S:Reply(bidderinfo) 2) R:Bids_receiver.a_{end} ==>registrarFinalizing, **[actions]**, S:Virtual Message. 3) R:bidder.register ==>saveBidderinfo, **[actions]**, S:Virtual Message
Bidder	1) R:bids_receiver.a_{CFB} ==> ***executeUserStrategy***, saveCFBinfo|(saveCFBinfo, saveQuoteinfo)| (saveCFBinfo, averesultinfo), **[actions]**,S:a_{bid}(bidinfo) 2) **(Start)** ==>formBidderinfo, **[actions]**, S:register(bidderinfo) 3) R:Bids_receiver.a_{end}==>**bidderFinalizing**, **[actions],** S:Virtual Message

Table 3 Sketch of Rules in AIML

1) "R:XXX" denotes receiving message "XXX". "S:XXX" denotes sending message "XXX". "XXX" denotes an action. [actions] denotes the possible actions may be there.
2) "XXX" with normal font format indicates such item is defined in auction framework; "**XXX"** with bold font format indicates such item is defined by auction developer. "***XXX"*** with bold and underlined font format indicates such item is defined by auction bidders. In the latter two cases, auction framework provides interface.
3) "XXX1|XXX2|…" denotes the possible items.

Figure 7 shows the basic auction process implemented in AIML.

4.4. Method Description

By taking advantages of AIML on supporting calling methods in outer Java class, auction framework directly allows auction developer to design and embed outer methods for complex computation. Definitely, in AIML scripts, auction framework provides method interface for determining *quoteinfo* value and *rsultinfo* value and for verifying *bidinfo* value. The following gives an interface for method of verifying *bidinfo* of *bids receiver* in table 3.

```
<!--[R:bidder.a_bid ==>verifyAndSaveBidinfo,[actions] S:bidResponse(verifyingresult)]-->
<message MsgLabel="BidResponse" OwnerName="BidsReceiverAgent" MsgName="BidResponse" Priority=""
MsgType="O" ReceiverName="">
  <para Name="%SET%" Type="%PROCEDURE%">
   <paraee><set> <setee Type="USERVAR" VarName="verifyingresult"></setee>
     <setee><method ClassName="AuctionFrame" MethodName="verifyingBidinfo">
       <para Name="quoteinfo" Type="STRING">
         <paraee Type="USERVAR" VarName="quoteinfo"></paraee></para>
       <para Name="bidinfo" Type="STRING">
         <paraee Type="MSGVAR" MsgOwner="*" MsgName="Bid" ParaName="bidinfo">
              <indexee>-1</indexee>
         </paraee></para></method></setee></set></paraee> </para>…</message>
```

Figure 6 Interface in AIML Script for the Method of Verifying *bidinfo*

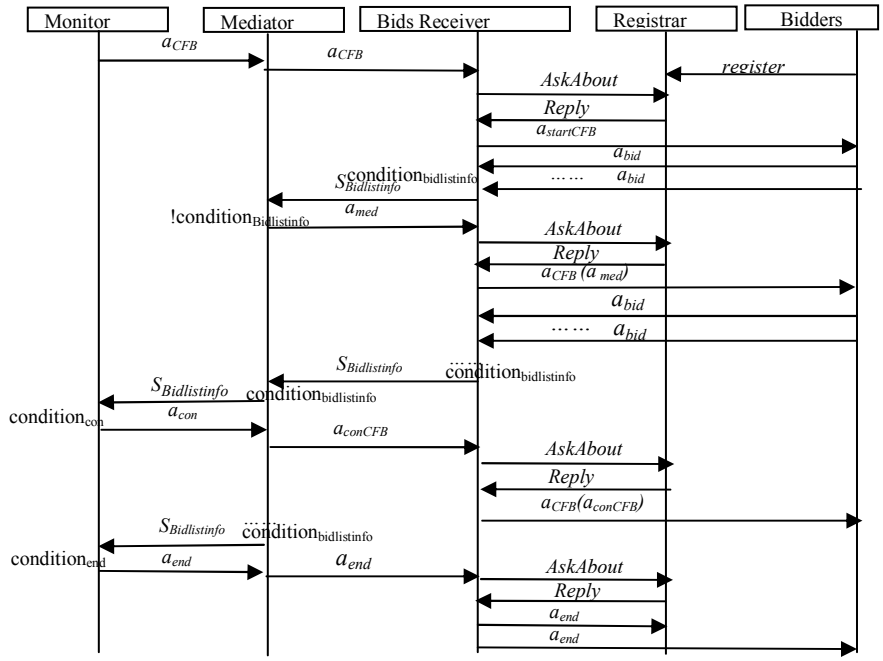


Figure 7 Messages Coordination among Auction Participants

4.5. User Strategy Description

Rule Description Language for User Strategy Description

Auction framework supports user (i.e. the bidders) to specify the strategies for dynamically determining *bidinfo* value. In order for that, auction framework also

applies rule as the basic idea. It provides a flexible XML-based rule description language. From the perspective of functionality of rule definition, such language can be seen as a subset of AIML (no sub-role support, no inherence mechanism support, and no concepts of message sending and receiving). At the same time, this language also has the same syntax for the description of each type of element.

```
<START>
<variables></><outerclass></>
<RULES>
   <rule rulelabel="rule1" Priority=""><para></><condition></></>
            …
</RULES></START>
```

Figure 8 An XML-based Rule Description Language

Interface with AIML

Bidders specify strategies in advance for dynamically determining *bidinfo* value. The action for value determination is performed in auction process by *bidders* in terms of the interaction rules in AIML. In the AIML script about *bidder* rules, auction framework implements the interface through an outer method *setBidinfo* call, which is embedded in the rule 1) of *bidder* in table 3. Method *setBidinfo* which is located in the java class *CAuctionFrame* will be responsible for executing user strategy and return the corresponding *bidinfo* value. The AIML script is shown below.

<!--R:bids_receiver.a_{CFB} ==> executeUserStrategy, saveCFBinfo|(saveCFBinfo, saveQuoteinfo)| (saveCFBinfo, saveresultinfo), [actions],S:a_{bid}(bidinfo)-->

```
<message MsgLabel="Bid" OwnerName="BidderAgent" MsgName="Bid" MsgType="S" Priority="" ReceiverName="BidsReceiverAgent;">
            …
  <para Name="%SET%" Type="%PROCEDURE%">
   <paraee><set>
     <setee Type="USERVAR" VarName="bidinfo"></setee>
     <setee><method ClassName="CAuctionFrame" MethodName="setBidinfo">
          <para Name="bidscriptURL" Type="STRING">
               <paraee Type="USERVAR" VarName="bidscriptURL"></paraee></para>
          <para Name="CFBinfo" Type="STRING">
               <paraee Type="USERVAR" VarName="CFBinfo"></paraee></para>
            <para Name="quoteinfo" Type="STRING">
               <paraee Type="USERVAR" VarName="quoteinfo"></paraee></para>
            <para Name="resultinfo" Type="STRING">
               <paraee Type="USERVAR" VarName="resultinfo"></paraee></para></method></setee>
   </set></paraee>
 </para> …</message>
```

Figure 9 Interface in AIML Script for Combining User Strategy

4.6. Framework Basis

Framework basis mainly includes three types of components. They are interpreter, agent body, and basic method library.

Interpreter: Interpreter is mainly implemented for agent interpreting rule scripts. The first one is for AIML scripts and the second one is for user strategy scripts. For the AIML interpreter, it is responsible for matching the receiving message and acquiring the corresponding sending message and doing corresponding actions. For the user script interpreter, it is responsible for observing the current values of *CFBinfo*, *quoteinfo* and *resultinfo* and computing the corresponding *bidinfo* value. They are both implemented in Java classes by making use of IBM XML parser XML4J.

Agent body: Agent body is just the implementation of basic agent framework specially designed for AIML IBM Aglets [aglets].

Basic Method Library: Auction framework provides basic method library mainly from two direct purposes. One purpose is to provide some basic methods which can be directly used in AIML scripts. Those methods include reading auction feature information, reading parameters forms, reading $Bidinfo_{SS}$, *CFBinfo*'s values which are assigned before auction starting. The other purpose of such method library is to provide the interface with developer-defined methods, at the same time provide a fixed interface with AIML script.

4.7. Auction Architecture from the Framework

The figure below shows the auction architecture implemented in auction framework: Auction implemented in auction framework has three layers. The lowest layer is the system layer, which means the systematic environment that auction runs. The second layer is the framework layer. This layer denotes the framework basis of auction framework. The third layer is the developer layer. This layer includes all the configurations auction developer makes. This layer with system layer and framework layer denotes some type of runnable auction. The top layer is user layer, where auction initiator initiates a concrete auction and bidders specify their bidding strategies that bidder agents use in the auction.

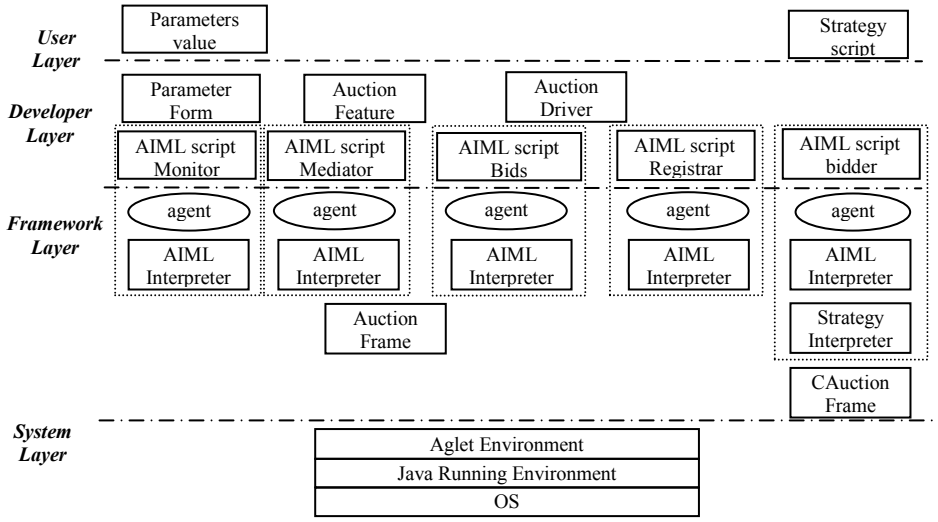


Figure 10 Auction Architecture Implemented in Auction Framework

5. Conclusions

It is from the perspective of software engineering that this paper makes research on constructing a configurable auction framework for resource allocation in the open agent system. Compared with the work in the literature [Mullen1996, Wurman1998, Wurman1999, Wurman2000, etc], the paper's work has different features.

- Application-Oriented Feature

The work in this paper is to try to construct a really directly applicable auction framework. Although, in the [Mullen1996], an auction specification language has also

been created for describing auction, the main purpose of that language is not for creating real auction implementation. Rather, it describes parameters of auction for better organizing the software implementation. Another purpose is that agent can refer to auction descriptions to create particular auctions or to inquire about their characteristics. As for the work in the [Wurman2000], its main purpose is to captures the essential similarities and differences of many auction mechanisms in a more descriptive and useful form. Essentially, it is the further exploration of auction specification language in [Mullen1996].

-Flexibility of Configurability

The configurability of auction framework in this paper has more flexibility. AuctionBot in the [Wurman1998] is an application of the works in [Mullen1996]. In AuctionBot, users can use AuctionBot directly to select and create their own auction for selling goods. In this point, the work in this paper seems to have more similarity to AuctionBot. However, they present auction configurability on different layers. AuctionBot's configurability is on the layer of user selecting and using existing auction types. Meanwhile, this paper's auction framework has the configurability on the lower layer of user defining auction types. Therefore, this point has determined that this paper's auction framework has more flexible configurability than AuctionBot. Even some classic auction types can be implemented in advance in the form of *drivers,* thus the auction framework can easily provide the same layer of configurability as AuctionBot.

The flexibility of configurability of the auction framework in this paper also presents at the consideration of configurability implementation. As shown in this paper, there is a layered two-dimension configurability model. It provides configurability for both common user and auction developer. In particular, it provides a rule-based user-strategy description language for bidders. By that, conditional bidding of bidder can be realized. In AuctionBot, such function is not supported.

-Considerate Auction Abstraction

With the analysis-oriented design purposes, both the work in the [Mullen1996, Wurman2000] is only about abstracting and parameterizing the common and different characteristics from various auctions. The goal of their work is to retrieve a descriptive form by that the main characteristics of various auctions can be described in the formal way. They do not care how those parameters are used to form the corresponding auction. In other words, they do not abstract and describe the auction process. With the application-oriented purpose, the work on auction abstraction in this paper not only involves the common and different characteristics in various auctions, but also has more emphasis on auction process. As shown in the paper, the auction process model is built. In more detail, the rule for performing each atomic action and compound actions constituting the auction process model is analyzed and formulated.

-Wide Applicability

The flexibility of configurability of this paper's auction framework implies it has more applicability. Not like AuctionBot, it is merely for users using the existing auction types to create online auction. This paper's auction framework is designed for resource allocation as an independent module. It can be directly used to create some auction, such as SBSP auction. It can be used to integrate various auctions in the forms of *drivers*, for auction user selecting and using, just like AuctionBot. It can also be used to create some more special but still configurable auction framework. For

example, set $\boldsymbol{condition_{end}}$ in the rule 2) of monitor agent as ***auction_numer<=****maximal_number,* the auction framework will be a configurable but auction repeating times limited one. Even, the rule set for auction framework can be changed in order to be embedded into application.

-XML-based Implementation

As shown in the section 4, except the framework basis, all the functional parts of auction framework are implemented on the basis of XML approach. The advantages, i.e. simplicity, openness, extensibility, interoperability, SGML community support, and WWW-friendly nature, of XML approach will be favorable of the auction framework. For example, the XML's symmetrical syntax structure (Simplicity) helps make auction configuration directly readable to auction users; Wealth of tools supports will further enhance the understandability of auction configurability; the WWW-friendly nature of XML not only enable the auction configuration to be shared by WWW, but also makes auction framework easy to be used in WWW application.

References

[Clearwater1996] Scott Clearwater, Ed. "Market-Based Control: A Paradigm for Distributed Resource A llocation". World Scientific, 1996.

[Mullen1996] T Mullen, MP Wellman, "Market-based Negotiation for Digital Library Services", In proceedings of the second USERNIX Workshop on Electronic Commerce, Oaklang, CA, November., 1996

[Cliff1998] Dave Cliff, Janet Bruten : "Simple Bargaining Agents for Decentralized Market-Based Control", Technical Report, HPL-98-17, Hewlett Packard laboratories, Bristol 1998

[Bredin1998]Jonathan Bredin and David Kotz and Daniela Rus: "Market-based Resource Control for Mobile Agents". In Proceedings of the Second International Conference on Autonomous Agents, pages 197-204, May, 1998. ACM Press.

[Wellman1998]Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. Technical report, University of Michigan, July 1998. http://citeseer.nj.nec.com/article/wellman98auction.html

[Mao2001] Mao Weiliang, Sheng Huanye: "An XML-Based Language for Coordination Protocol Description in Multi-Agent System", In proceedings of The Fourteenth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, LNAI 2070, Springer, 2001

[Wurman2000] Peter R. Wurman, Michael P. Wellman, William E. Walsh "A Parameterization of the Auction Design", Journal of Economic Literature, 2000

[Wurman1998]P.R. Wurman, M.P. Wellman, and W.E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Second Int'l Conf. on Autonomous Agents, May 1998.

[Wurman1999]Peter R. Wurman. Market Structure and Multidimensional Auction Design for Computational Economies. PhD thesis, University of Michigan, 1999.

[Aglets] http://www.trl.ibm.co.jp/aglets