SOCS

A COMPUTATIONAL LOGIC MODEL FOR THE DESCRIPTION, ANALYSIS AND VERIFICATION OF GLOBAL AND OPEN SOCIETIES OF HETEROGENEOUS COMPUTEES

IST-2001-32530

Experiments on Combinatorial Auctions

Project number:	IST-2001-32530
Project acronym:	SOCS
Document type:	D (deliverable)
Document distribution:	I (internal to SOCS and PO)
CEC Document number:	IST32530/UNIBO/250/D/I/a1
File name:	5250-a1[].pdf
Editor:	M. Milano
Contributing partners:	UNIBO, DIFERRARA, CITY, ICSTM
Contributing workpackages:	WP2,WP4,WP6
Estimated person months:	
Date of completion:	10 March 2005
Date of delivery to the EC:	18 March 2005
Number of pages:	31

ABSTRACT
This paper contains the description of experiments done so far in the combinatorial auction scenario. We describe the computee side and the society side. In this context we introduce also *external objects* providing services to the computees themselves, and in particular implementing a complex optimization algorithm. Experimental results are given for a traditional combinatorial auction and for two variants of it: a double auction and an auction plus the successive payment with Netbill.

Experiments on Combinatorial Auctions

M. Alberti, F Chesani, U. Endriss, M. Gavanelli, A. Guerri, E. Lamma, W. Lu, P. Mello, M. Milano, K. Stathis, F. Toni, P. Torroni

UNIBO, DIFERRARA, CITY, ICSTM

ABSTRACT
This paper contains the description of experiments done so far in the combinatorial auction scenario. We describe the computee side and the society side. In this context we introduce also *external objects* providing services to the computees themselves, and in particular implementing a complex optimization algorithm. Experimental results are given for a traditional combinatorial auction and for two variants of it: a double auction and an auction plus the successive payment with Netbill.

Contents

1 Introduction

In the vision of computational infrastructures available globally and able to provide services of different kinds (e.g., for communication, co-operation and mobility, resource usage, security policies and mechanisms, etc.,), a central issue is that of resource and task allocation. To this end, particularly suited tools are market-based negotiation protocols and theories, such as those related to auctions and the increasingly popular combinatorial auction mechanism. Combinatorial auctions are a good candidate to address issues of service programmability, resource distribution and management, scalability, and distribution transparency, all central topics in the global computing perspective.

Software agent technology seems to be an attractive paradigm to support auction applications, because the introduction of software agents acting on behalf of end-users could reduce the effort required to complete auction activities. Agents are intrinsically autonomous and can be easily personalized to embody end-user preferences. In addition, they are adaptive in order to cope with changing operating conditions and evolving user requirements [13]. Electronic auctions have already demonstrated the potential of software agents [6], where agents are required to perform auction activities on behalf of human users.

1.1 Motivation

Consider, for example, the situation of an Internet auction where a user asks an agent to bid on its behalf. Assuming that the user might have a way to specify an upper limit for the bids, and possibly a bidding strategy, an agent that could play in the auction according to the rules would save a lot of time to the user, mainly because Internet auctions can last for a long period of time. Even if the auction can last only for a short period, for instance in applications that allow the provision of location-independent services [20], an agent might start a reverse auction on the fly [31], so that to receive the right services for a user.

However, the very idea of having agents acting on behalf of people raises all sorts of issues for auction applications. In this context, this work is particularly concerned with the following ones:

- the provision of a generic framework that deals with auction applications;
- a model of agency where an agent must be able to report back to the user the outcome of its actions, so that the agent's actions are transparent to the user;
- a model of the interaction that takes into consideration the rules that govern the auction activities at hand;
- an implementation framework that will integrate the models and the framework in a single computing platform.

1.2 SOCS Approach

Within the SOCS project we see the support of auctions using agents as an opportunity for experimenting with the technology and models produced by the project. This kind of experimentation treats auctions as societies of agents whose interactions can be verified on demand. More specifically, we first propose the computational logic approach in SOCS as the generic framework for building auctions. We then treat agents as computees whose reasoning capabilities are developed according to the KGP model. The KGP model is proposed here as the logical model that will allow agents to report their actions back to the user, thus achieving, as argued in [32], the required transparency. We then view the rules that govern the interactions amongst computees in an auction as social rules that can be specified and verified using the social model of the project, which will allow the different auction protocols to be tested for compliance.

To show how to apply the framework in detail, we experiment with different auction protocols implemented using the PROSOCS platform. This platform is intended as the implementation framework, which has also been extended with the use of objects (i.e. entities that cannot be naturally seen as agents) so that to provide with a general implementation framework for agent-based auctions. Based on this extension of PROSOCS, we then show how the class of auctions, known as *combinatorial auctions*, can be specified in PROSOCS, by allowing a constraint solver to be treated as an object that computees can interact with in order to solve optimisation problems.

1.3 Preliminaries on Auctions as Societies of Computees

In an auction we have two classes of participants: the *auctioneer* and *bidders*. There is a particular kind of auction, called combinatorial exchange where there is no auctioneer, but an administrator of the exchange (which does not buy nor sell anything) and bidders that either buy or sell. Participants can be humans or computees. While in the past, bidders were only humans, recently Internet auction servers have been built and allow software agents to participate in the auction on behalf of end-users [35]. Some of these auction servers even have a built-in support for mobile agents [28]. As the rise of the Internet and electronic commerce continues, dynamic automated markets will be an increasingly important domain for computees and software agents.

Depending on the kind of auction, the auctioneer either sells goods/services or buys a set of goods or services. Bidders have the goal to obtain/sell their goods/services under convenient conditions as far as price is concerned. The auctioneer has the goal to sell/obtain a set of goods/services maximizing the profit (or minimizing the cost) at the minimum risk. In a combinatorial exchange, the goal of the administrator is to select bids so as to maximize the surplus.

The interaction mechanisms can be different in accordance with the role of the computee in the auction. For example, in a *single unit reverse auction*, we have a customer that wants to buy a set of goods/services at the minimum cost. The customer is self interested and does not collaborate nor negotiate with suppliers which in turn are again self interested and, in a sense, compete with one another. The information exchanged and their form change depending on the type of auction we are modelling. In general we have the auctioneer proposal (set of goods and services to be sold or bought) with constraints (temporal and minimum or maximum prices) the auctioneer wants to be respected by the participants.

A bid in an auction is defined by the participant's name, the name of the items to be purchased, possibly the number of identical indistinguishable items, temporal windows, and the value of the bid. The auctioneer answer contains the set of winning bids. In addition, auctions can also be used among cooperative computees. For example, two suppliers can cooperate and put together their resources to obtain better prices and more appealing bids. As another example, if the constraints (max. price and time constraints) imposed by the customer are too strict, after a failure of the bid evaluation process, the customer can start negotiating with suppliers to obtain information on the relaxation of the costumer's constraints.

We treat auctions as societies. Each time an auction is proposed, a society is created. The auctioneer and all participants are the members of the society. The social setting is formal, with formal rules, constraints and goals which should be respected by all participants.

1.4 Structure of the report

The rest of the document is organized as follows: we first introduce some motivation for running this experimentation, then we present the problem faced and provide two models for the combinatorial problem at hand. Then, we describe our approach to the problem and in particular from the computee side, the model of the auctioneer (properly extended to cope with external objects) and the model of bidders, and from the society side the protocols used to perform the tests. Experiments are then described on all components involved in the scenario. Related work conclude the paper.

2 Experimentation with combinatorial auctions

This document reports on some experimentation on combinatorial auctions by means of PROSOCS. In this context, both auctioneers and bidders are seen as computees (computational-logic agents), and, each time an auction is set-up, a corresponding society is created, with members the auctioneer and bidder computees, and with formal rules, constraints and goals which should be taken into account by all society members.

We have chosen Combinatorial Auctions as a scenario for experimentation since it is a real world example in the field of e-commerce, that promises to test and exploit various aspects and features of PROSOCS, as well as guide us through the extension of PROSOCS to incorporate a number of additional features that we believe important for real-life applications.

In particular, this scenario requires for the computee to have skills for

- High-level communication
- Negotiation
- Policy definition
- Planning towards the achievement of goals
- Reactivity to changes in the environment

which we foresee to be mostly achievable by the Planning and Reactivity capability, integrated as foreseen by the KGP model and as implemented within the SOCSiC component of PROSOCS. The scenario also requires to be able to check conformance of the computees involved in it to communication protocols governing the interaction in the auction, which can be done by means of the SOCS-SI component of PROSOCS. Thus, this scenario will allow to test the expressive power of social integrity constraints that can be dealt with by SOCS-SI. Finally, the scenario shows the need and benefits of the integration between computees and society within PROSOCS.

In addition to exploiting existing features of PROSOCS, the Combinatorial Auctions scenario promises to test the extensibility of PROSOCS to deal with aspects that are not currently dealt with by the platform (nor by the abstract models behind it), but that are envisaged as possible extensions of (the models and the) platform. These aspects include

- Optimization, to solve the winner determination problem by the auctioneers and to design the bidding strategies of the bidders, and
- Learning, e.g. to discriminate intelligently amongst different ways to reach solutions to the winner determination problem
- Conformance enforcement (by the society over computees) to make sure that the auctions is carried out according to the rules

We foresee that some of these additional features may be obtained via the integration within PROSOCS of some non-logical though declarative elements other than computees, that in PROSOCS are referred to as *external objects*.

In the experiment, we exploit a commercial constraint solver, namely the ILOG solver [15], for efficiently solving combinatorial optimization problems. We have implemented in ILOG a module called *Auction solver* [11] embedding a portfolio of algorithms and an automatic algorithm selection strategy, described in [12]. In particular, in combinatorial auctions, bidders can bid on a combination of items, and the auctioneer has to solve the winner determination problem. The problem of determining the set of winning bids is NP-hard and its efficient solution is the result of a set of constraint based technologies, namely constraint and integer programming combined.

In this document, we propose a first solution for integrating the ILOG solver within a society of computees. We consider different aspects:

- problem description;
- computees involved in the experiment;
- how the auction solver implemented in ILOG solver can be interfaced with a computee representing the auctioneer;
- different protocols defined through social integrity constraints;
- experimental results.

We will show later that the Auction Solver will be wrapped as a PROSOCS object providing an auctioneer the winner determination computation in a combinatorial auction. Therefore, external objects provide additional facilities to computees.

3 Problem description

In combinatorial auctions, bidders can bid on combination of items. In this context, we have a major problem: the Winner Determination Problem (WDP). In the WDP the auctioneer has to find the set of winning bids at a minimum cost or maximum revenue. The winner determination problem is NP-hard. We have different variants of combinatorial auction. We have different kinds of auctions.

- The **single unit auction**, where the auctioneer wants to sell a set M of goods/tasks maximizing the profit. Goods are distinguishable. Each bidder j posts a bid B_j where a set S_j of goods/tasks $S \subseteq M$ is proposed to be bought at the price p_j, i.e., $B_j = (S_j, p_j)$.

- The **multi single unit auction** where the auctioneer wants to sell a set M of goods/tasks and for each good/task a number of indistinguishable units, maximizing the profit. Each bidder posts a bid $B_j = (S_j, \Lambda_j)$ where S_j is the set of types of items and $\Lambda_j = (\lambda_{j1}, \dots, \lambda_{jm}), p_j)$ where each λ_{ji} is the number of units of the goods/tasks i in the j-th bid he/she wants to buy. The price p_j refers to the whole set of goods.

- The **single unit reverse auction** is a single unit auction where the auctioneer wants to buy and bidders are suppliers.

- The **multi unit reverse auction**, is a multi unit auction where the auctioneer wants to buy and bidders are suppliers.

- The **exchanges** are particular auctions where no auctioneer is present, but only an administrator of the exchange that does not buy nor sell anything. Bidders can buy and sell items/services. Each bidder posts a bid $B_j = (S_j, \Lambda_j)$ where S_j is the set of types of items and $\Lambda_j = (\lambda_{j1}, \dots, \lambda_{jm}), p_j)$ where each λ_{ji} is the number of units of the goods/tasks i in the j-th bid. Positive λ_j represent buying while negative λ_j represent selling. p_j is the price and has the same sign.

In this paper, we consider **single unit reverse auctions** but other kind of combinatorial auctions can be faced in a similar way.

We now describe the case study used to test our approach: combinatorial auctions. In combinatorial auctions bidders can bid on combination of items. In this context, we have two major combinatorial optimization problems. The Bid Evaluation Problem (BEP) and the Winner Determination Problem (WDP). In the WDP, that is NP-hard, the auctioneer has to find the set of winning bids at a minimum cost or maximum revenue. In the WDP the optimization part of the problem is predominant, so an Integer Programming (IP) approach represents the technique of choice.

In the BEP, beside a winner determination problem, time windows and temporal constraints are stated among bids. We mainly consider an auction where the auctioneer buys a set of items (in our case tasks or services) which are sequenced by temporal precedence constraints and are associated to temporal windows and durations. When temporal constraints are added to the problem, the feasibility part of the problem can become predominant and so a Constraint Programming (CP) approach can take advantage of it and become the best technology. Depending on the instance structure, that is depending on which part is predominant between the feasibility one and optimization one, a CP or an IP approach can supply the best results.

In the following, we describe the 2 models we developed to solve the BEP, one based on IP and one on CP. We used ILOG to implement and solve the models, in particular ILOG CPLEX 8.1 to solve the IP model and ILOG Solver 5.3.

3.1 Bid Evaluation Problem: IP model

Each bidder j ($j = 1 \dots n$) posts one or more bids. A bid is represented as $B_j = (S_j, Est_j, Lst_j, D_j, p_j)$ where a set $S_j \subseteq M$ of services i ($i = 1 \dots m$) is proposed to be sold at the price p_j. Est_j and Lst_j are lists of earliest and latest starting time of the services in S_j

and D_j their duration. A precedence constraint between two tasks t_p and t_s is represented as $t_p \prec t_s$.

We introduce a decision variable x_j for each bid j, that takes the value 1 if the bid B_j is a winning one, 0 otherwise. We also introduce a variable $Start_{ij}$ for each item i taken from bid j. These variables range on the temporal windows $[Est_{ij}, Lst_{ij}]$ for item i taken from bid j.

The objective function is $min \sum_{j=1}^{n} p_j x_j$, that minimizes the total cost.

In the BEP we have two kinds of constraints: covering constraints and precedence constraints. Covering constraints have the form $\sum_{j|i \in S_j} x_j = 1$, and state all items should be covered and each item should be covered by exactly one bid.

To represent precedence constraints we consider each pair of items t_p and t_s such that $t_p \prec t_s$, and we find all pairs of bids b_p and b_s containing that items; if S_{b_p} and S_{b_s} have an empty intersection we compute $Est_{b_p t_p} + D_{b_p t_p} - Lst_{b_s t_s}$, where $D_{b_p t_p}$ is the duration of t_p in bid b_p. In case the result is positive, that is, domains of $Start_{t_p b_p}$ and $Start_{t_s b_s}$ do not contain any pair of values that could satisfy the precedence relation, we introduce the constraint $x_{b_p} + x_{b_s} \leq 1$, which prevents both bids from appearing in the same solution; otherwise, if the result is zero or negative, we introduce the constraint $Start_{t_p b_p} + D_{t_p b_p} - Start_{t_s b_s} + M(x_{b_p} + x_{b_s}) \leq 2M$, where M is a large number. The term $M(x_{b_p} + x_{b_s})$ makes the constraint satisfied in the case where either $x_{b_p} = 0$ or $x_{b_s} = 0$.

3.2 Bid Evaluation Problem: CP model

Auctions can be easily modelled in Constraint Programming. We recall that $B_j = (S_j, Est_j, Lst_j, D_j, p_j)$.

We introduce four sets of domain variables: X, that is an array of m variables representing the items to be bought. Each variable X_i ranges on a domain containing the bids mentioning item i. $Cost$, that is an array of n variables representing the cost of the bid in the solution. Each variable $Cost_j$ can assume only values 0, if bid j is a losing bid, and p_j, if it is a winning one. $Duration$ and $Start$, that are arrays of m variables. $Duration_i$ ranges on the union of all duration D_{ij} for item i taken from all bids j mentioning i. $Start_i$ ranges on the union of all temporal windows $[Est_{ij}, Lst_{ij}]$ for item i taken from all bids j mentioning i.

The objective function is $min \sum_{j=1}^{n} Cost_j$.

Each time a variable X_i is assigned to a value j, that is item i is assigned to bid j, we propagate the fact that all other items in S_j should be assigned to the same bid, setting to j variables X_k, $\forall k \in S_j$. Similarly, we set variables $Start_k$ and $Duration_k$, $\forall k \in S_j$ to values proposed by bid B_j. We also set to p_j the variable $Cost_j$.

To consider precedence among tasks, we introduce the constraint $Start_p + Duration_p \leq Start_s$ for each couple of tasks t_p and t_s such that $t_p \prec t_s$.

Finally, we use $Distribute(X, J, 0, |S|)$, that can trigger an effective propagation. It is a variant of the global cardinality constraint whose semantics can be explained as follows: if the bid B_j is chosen as winning, the number of variables X_i that take the value j is exactly the cardinality of the set S_j. Otherwise, if the bid B_j is not chosen as winning, that number is 0. Parameters in $Distribute(X, J, 0, |S|)$ have the following meaning: X is the array of variables representing items to be sold, J is an array containing numbers tidily from 1 to n, where n is the number of bids, and $|S|$ is an array where each element $|S_j|$ is the cardinality of the set of items contained in the bid j. This constraint holds iff the number of occurrences of each value $j \in J$ assigned to X is exactly either 0 or $|S_j|$.

4 Computees Involved in the Experiment

Computees involved in the experiment include an *auctioneer* proposing a set of items to be bought and a set of *bidders*, possibly embedding different bidding strategies. In this setting, the auctioneer has to interact with the *auction solver* to solve the combinatorial problem defined by the bids received so as to maximise revenue. In this section, we describe the relevant aspects of our experiment: how the auctioneer computee has been programmed in PROSOCS, how the integration of the external object representing the auction solver has been achieved, and how we have represented bidding computees.

4.1 Modelling the Auctioneer in PROSOCS

The main reasoning problem in this scenario, as far as the computee representing the auctioneer is concerned, is the problem of winner determination in a combinatorial auction. This problem has been outsourced to an external module implementing a highly efficient algorithm developed specifically for this purpose, namely the ILOG auction solver (see Section 4.3). In the sequel we describe how we have programmed a computee in PROSOCS to carry out the remaining tasks, such as opening an auction, forwarding bids to the auction solver, and distributing the result received from the solver to the bidders. Most of these tasks have been mapped to the computee's *reactivity* capability; in one case we also use *planning.*

Opening an auction. Opening an auction is done by sending a corresponding message to other computees. This message will be invoked by the planning capability in response to a corresponding goal. This is achieved by including a clause such as the following into KB_{plan}:

```
initiates(Action,T,do_auction(ID,Bidders,Items)) :-
  me(Name),
  Action = tell(Name,Bidders,openauction(Items,30,40),ID).
```

That is, we assume that an auction lasts until time 30 units and that the deadline for announcing the result of the auction is another 10 time units later. These values have been hard-wired here, but they could also be included into the specification of the goal and then passed on to the corresponding action.

We assume that `me/1` has been implemented (as a fact), specifying the name of the auctioneer computee. Furthermore, we assume that communicative actions are executable for this computee (by programming `executable/1` accordingly [1]).

Collecting bids. The next task for an auctioneer is to check the bids that it receives for validity and to collect those that are considered valid for processing after the end of the auction. For a bid to be valid, (i) it has to reach the auctioneer in time, (ii) it has to come from a bidder that has actually been invited to participate in the auction, and (iii) it has to specify a set of items that is a subset of the items put on auction.

To collect the valid bids, we make use of another external object (besides the object implementing the auction solver). This so-called `notepad` object can be used to collect a list of items (identified by the same identifier); it will store them and we can later retrieve the entire list by, again, specifying the appropriate identifier. This approach has turned out to be considerably simpler than programming the collection of of past observations (of valid bids) directly in KB_{react}. Besides, it also showcases another application of the integration of objects into

PROSOCS. The following rule implements both the checking of a bid for validity according to the three criteria given above and the forwarding of valid bids to the `notepad` for collection:

```
[ observed(Bidder,tell(Bidder,Auctioneer,bid(Price,Items),ID,_),T),
  executed(tell(Auctioneer,AllBidders,openauction(AllItems,T1,T2),ID),T0),
  T0 #< T,
  T #=< T1,
  member(Bidder,AllBidders),
  subset(Items,AllItems)
] implies [
  assume_happens_after_once(do(notepad, makeNote(ID,bid(Bidder,Price,Items))),T)
].
```

This reactive rule uses the following auxiliary predicates to test for membership in a set and to to test for being a subset of a given set, respectively (where sets are represented as lists, using the notation familiar from Prolog):

```
member(Elem,[Elem|_]). member(Elem,[_|Tail]) :- member(Elem,Tail).

subset([],List). subset([Head|Tail],List) :- member(Head,List),
subset(Tail,List).
```

Note that these predicates are not being evaluated by the Prolog interpreter directly; they form part of KB_{react} and accordingly they are being evaluated by CIFF.

The predicate used on the righthand side of the reactive rule for collecting incoming bids has been implemented as follows:

```
assume_happens_after_once(Action,T) :-
  assume_happens_after(Action,T),
  not(already_executed(Action,T)).

assume_happens_after_once(Action,T) :-
  already_executed(Action,T).
```

It uses two auxiliary predicates:

```
assume_happens_after(Action,Limit) :-
  assume_happens(Action,T),
  Limit #< T.

already_executed(Action,T) :-
  executed(Action,T0).
```

That is, making a predicate of the form `assume_happens_after_once(Action,T)` true requires (i) `assume_happens(Action,T1)` to be true for some `T1` in the future of `T` and (ii) KB_0 not to contain a fact according to which the same `Action` has already been executed in the past. Note how the implementation of `assume_happens_after_once/2` explicitly enumerates the two possible cases: either the action has not yet been executed (in which case it should be carried out in the future), or it is known to have been executed already. By making the second case explicit, we can use classical negation (rather than negation by failure), which is the negation operator implemented by the CIFF proof procedure underlying the reactivity capability [9].

For simplicity, we assume that any physical actions (of the form `do(_)`) are executable by the computee (by programming `executable/1` accordingly).

Closing an auction. The final task of an auctioneer is to close an auction once the specified time has passed and to communicate the result to the bidders (at least to those that have submitted a valid bid). This has been programmed, again, by providing appropriate rules for KB_{react}. The following rule "fires" as soon as the specified end time `T1` has passed and queries the `notepad` object for the list of collected bids (provided the deadline `T2` for announcing the result has not yet passed):

```
[ executed(tell(Auctioneer,AllBidders,openauction(AllItems,T1,T2),ID),T0),
  time_now(T),
  T1 #< T,
  T #=< T2
] implies [
  assume_happens_after_once(do(read_notes(ID)),T)
].
```

Then there is a simple rule that, upon receiving the list from the `notepad` (by means of a sensing action), forwards this list to the auction solver (omitted here). The result of the auction received from the auction solver is again picked up by means of a sensing action. It specifies the list of winners as well as the list of losers. The following reactive rule causes a message to be sent to each computee given in the list of winners (there is a similar rule for informing the losers; omitted here):

```
[ me(Auctioneer),
  observed(auction_solver,see(auction_solver,solution(ID,WinBids,LoseBids)),T),
  member((Bidder,Price,Items),WinBids)
] implies [
  assume_happens_after_once(tell(Auctioneer,Bidder,
                                      answer(win,Bidder,Items,Price),ID),T)
].
```

We should stress that the implementation of KB_{react} for an auctioneer computee is independent from the exact syntax used to specify bids (i.e. it could, for instance, be used for combinatorial auctions that either do or do not specify time windows within bids, and for both reverse auctions and normal auctions).

4.2 External Objects Used by the Auctioneer

To allow the reuse of functionality from other components that are not necessarily computees, PROSOCS has been extended with the notion of *objects*. An object is an entity that has no reasoning capability in the sense of the KGP model. Conceptually, objects are simply a way in which we can introduce in a PROSOCS environment entities that computees can interact with physical actions. In other words, objects are simply parts of the environment that allow physical interaction in a PROSOCS application. Objects can also be acting as wrappers to *external objects*, if necessary. An external object is any software component with an API (Applications programmer's Interface). Such external object can be included in the PROSOCS environment using the PROSOCS objects facility. A detailed discussion on the incorporation of objects in PROSOCS is discussed in [19].

The combinatorial auctions experiment can be seen as an example of showing how to instantiate objects in a practical PROSOCS application. As an example of a simple object we have made generally available a *notepad* object to allow agents to make notes about dialogues

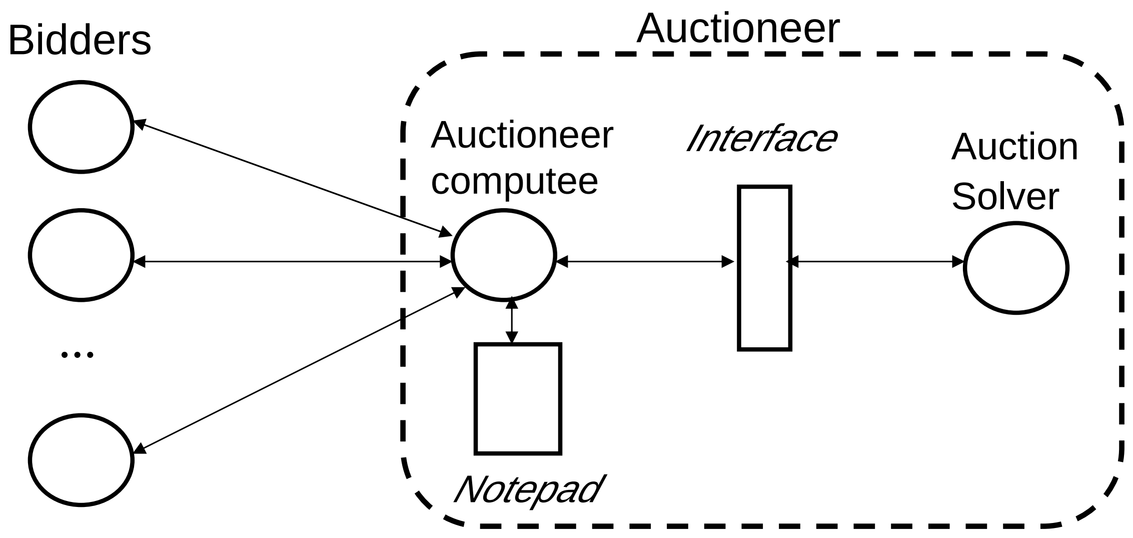

Figure 1: Computees and object involved in a Combinatorial Auctions

they engage in, including accessing these notes on demand. In the context of combinatorial auctions, we instantiate such an object to allow the auctioneer to make notes on bids made by the bidders and access these notes when required. Similarly, as an example of an external object we have introduced an *auction solver* object that allows an auctioneer to use the optimisation techniques and constraint capabilities of the ILOG solver whose functionality can be reused across applications to provide solutions to sets of bids. In this way we exemplify the full potential of the PROSOCS platform for practical applications.

4.3 Interface between the Auctioneer an the Auction Solver

In the experiments we carried out, the Auction solver implemented in ILOG is conceived as an external object while the auctioneer is a computee. Their interaction is not monitored by the society, therefore at society level they are indeed considered as a unique entity as shown by the dashed circle in Figure 1.

The computee representing the auctioneer provides the auction solver a winner determination problem instance and receives its optimal solution. ILOG solver has been wrapped into Java, so as to define a simple interface for providing a problem instance to ILOG solver and receive the solution of the problem. The interface is reported in figure 2.

The JavaDoc can be found at
`http://www-lia.deis.unibo.it/Staff/AlessioGuerri/JavaDoc/index.html`
In this case the auctioneer should collect the data of the instance and send them to the auction solver solver (with the addBid method). The auctioneer at the end of the auctions asks the auction solver to compute the solution which is returned as an array of boolean values representing the result for each submitted bid (possibly within a limited time t, possibly using algorithm a).

`AuctionSolver(java.lang.String[] t,java.lang.String[] p, java.lang.String URL)`
Is the constructor where t is the list of tasks, p the list of possible precedences between tasks and URL is the URL of the licence server

`boolean addBids(Bid[] b)`
Adds a set of Bids to the auction.

`Bid[] getAllBids()`
Returns an array, eventually of dimension 0 if no bids have been placed

`boolean[] getSolution()`
Starts the search for the solution and returns the solution found

`boolean[] getSolution(int t)`
Starts the search for the solution and returns the solution found within a max time t.

`boolean[] getSolution(java.lang.String a, int t)`
Starts the search for the solution and returns the solution found within a max time t and choosing the algorithm a

`boolean verifySolution(Bid[] b, boolean[] sol)`
Verifies the optimality of a solution.

`public Bid(java.lang.String name, int price, java.lang.String[] item, int[] eStart, int[] lStart, int[] duration)`
Creates a new Bid Parameters: name is the symbolic name of the bid, price the price proposed, item the items in the bid, eStart is the early start times for the execution of the corresponding items, lStart is the late start times for the execution of the corresponding items and duration the durations of the execution of the corresponding items

Figure 2: Excerpt of the JavaDoc

4.4 Modelling Bidders in PROSOCS

For most of our experiments we have implemented concrete hard-wired bids for specific sets of items that are on auction using simple reactive rules. Alternatively, experiments can be carried out by using the *speak* tool available in the PROSOCS demonstrator, which allows a user to type in messages (in this case bids) directly.

Here is a simple example. The following reactive rule causes a computee to reply to any `openauction` message that offers a set of items including `van_gogh` by offerening an amount of 1000 for that item, provided it is in fact amongst the computees that have been invited to bid:

```
[
observed(Auctioneer,tell(Auctioneer,Bidders,openauction(Items,T1,T2),ID,_),T),
  me(Name),
  member(Name,Bidders),
  member(van_gogh,Items)
] implies [
  assume_happens_after_once(tell(Name,Auctioneer,bid(1000,[van_gogh]),ID),T)
].
```

In principle, it would also be possible to implement more sophisticated *bidding strategies* (in the true sense of the word) for combinatorial auctions. The present experiments are mostly aimed at testing whether PROSOCS can provide the basic communication needs for running combinatorial auctions. Another possible approach of generating interesting problem instances would be to use the *Combinatorial Auction Test Suite (CATS)* [18] to automatically generate instances of combinatorial auction problems that could be adopted by bidding computees to test the overall architecture more extensively.

5 Society involved in the experiment: the protocol

By using the syntax of social integrity constraints we write now three different protocols that can be defined under the hypothesis that the society should check the interactions among computees but not the interaction between the auction solver in ILOG (the external object) and a computee. We also make an additional hypothesis: that the society trusts the auction solver and therefore the auctioneer computee. This means that the society does not check (in its protocol) wether the solution provided by the auctioneer is indeed the optimal one and is consistent with all the problem constraints described in the model in sections 3.1 and 3.2. In all the experiments we performed, social integrity constraints are used to check that the auction protocol is respected, but they are not used to check that the result provided by the auctioneer is indeed the optimal (and obviously feasible) solution of the winner determination problem it received.

In the following, we present three protocols describing variants of a combinatorial auction

- single auction: this scenario models a traditional auction where a set of bidders post their bids on combinations of items and the auctioneer decides which bids win.
- double auction: the second experiment implements two auctions. Indeed, when an auction is opened and the bidders have access to the list of items the auctioneers posts, some of them can decide of opening a second auction to collect more items and perform more appealing bids.

- an auction plus the buying/selling step: this experiment involves an auction and the next step where items that are promised in the auction are sold and bought at the corresponding price. We used the Netbill protocol for this step.

5.1 Single combinatorial auction

The protocol is the following. Each time a bidding event happens, the auctioneer should have sent an *openauction* event (to all bidders).

$$\begin{array}{l}\mathbf{H}(tell(S,R,bid(ItemList,P),A_{number}),T_{bid})\\ \rightarrow \mathbf{E}(tell(R,S,openauction(Items,T_{end},T_{deadline}),A_{number}),T_{open}),\\ T_{open}<T_{bid}\wedge T_{bid}\leq T_{end}\end{array} \tag{1}$$

This ic_S imposes that a bidder cannot start placing bids if an auctioneer has not opened an auction.

Incorrect bids always loose; e.g., a bid for items not for sale must loose. Indeed, the answer loose refers also to not acceptable bids.

$$\begin{array}{l}\mathbf{H}(tell(A,B,openauction(Items,T_{end},T_{deadline}),A_{number}),T_1)\wedge\\ \mathbf{H}(tell(B,A,bid(ItemBids,P),A_{number}),\_)\wedge\\ not\ included(ItemBid,Items)\\ \rightarrow \mathbf{E}(tell(A,Bidder,answer(lose,Bidder,ItemBids,P),A_{number}),\_)\end{array} \tag{2}$$

$included([],\_).$
$included([H|T],L):-member(H,L),included(T,L).$

The auctioneer should answer to each bid. The answer should be sent after the auction is closed within the deadline $T_{deadline}$.

$$\begin{array}{l}\mathbf{H}(tell(S,R,bid(ItemList,P),A_{number}),T_{bid})\wedge\\ \mathbf{H}(tell(R,S,openauction(Items,T_{end},T_{deadline}),A_{number}),T_{open})\rightarrow\\ \quad\mathbf{E}(tell(R,S,answer(X,S,ItemList,P),A_{number}),T_{answer}),\\ \quad\quad T_{answer}\geq T_{end}\wedge T_{answer}\leq T_{deadline},X::[win,lose]\end{array} \tag{3}$$

Another rule is the following: it is not possible that a bidder receives for the same auction on the same bid two conflicting answers. Therefore, each bidder either wins or looses, but not both:

$$\begin{array}{l}\mathbf{H}(tell(R,S,answer(loose,S,ItemList,P),A_{number}),\_)\rightarrow\\ \quad\mathbf{NE}(tell(R,S,answer(win,S,ItemList,P),A_{number}),\_)\end{array} \tag{4}$$

$$\begin{array}{l}\mathbf{H}(tell(R,S,answer(win,S,ItemList,P),A_{number}),\_)\rightarrow\\ \quad\mathbf{NE}(tell(R,S,answer(loose,S,ItemList,P),A_{number}),\_)\end{array} \tag{5}$$

5.2 Double combinatorial auction

This protocol is intended for those cases where a bidder opens an auction for buying items he needs for posting a more appealing bid. For describing this protocol, we still have constraints (1) to (5). In addition, there are specific integrity constraints for this kind of auction: the first tells that is the first openauction has happened and one bidder has opened another auctions referring to a set of items involved in the first auction, then the second should be closed and winning bids decided before the first closes.

$$\begin{aligned}&\mathbf{H}(tell(A, B, openauction(Items1, T_{end1}, T_{deadline1}), A1), T_{open1})\wedge\\&\mathbf{H}(tell(B, C, openauction(Items2, T_{end2}, T_{deadline2}), A2), T_{open2})\wedge\\&intersect(Items1, Items2) \wedge T_{deadline2} >= T_{end1} \rightarrow\\&false.\end{aligned} \tag{6}$$

The second constraint tells that one cannot bid for the same item in two auctions.

$$\begin{aligned}&\mathbf{H}(tell(B, A1, bid(ItemList1, P1), A_{number1}), T_{bid1})\wedge\\&\mathbf{H}(tell(B, A2, bid(ItemList2, P2), A_{number2}), T_{bid2})\wedge\\&A_{number1}! = A_{number2} \wedge intersect(ItemList1, ItemList2) \rightarrow\\&false.\end{aligned} \tag{7}$$

5.3 Combinatorial auction with Netbill

This protocol extends the tradition auction protocol with the payment at the end. In this case, the auction integrity constraint (1) to (5) remain the same. We should add the following.

After the bid has been declared a winning bid, the bidder should deliver the list of items involved in the bid.

$$\begin{aligned}&\mathbf{H}(tell(B, A, bid(ItemList, P), A_{number}), T_{bid})\wedge\\&\mathbf{H}(tell(A, B, answer(win, B, ItemList, P), A_{number}), T1) \wedge T_{bid} < T1 \rightarrow\\&\mathbf{E}(tell(B, A, deliver(ItemList, P), A_{number}), T3) \wedge T3 > T1.\end{aligned} \tag{8}$$

Clearly, a bidder should not deliver any item if his/her bid was determined as losing.

$$\begin{aligned}&\mathbf{H}(tell(B, A, deliver(ItemList, Price), A_{number}), T3) \rightarrow\\&\mathbf{E}(tell(A, B, answer(win, B, ItemList, P), A_{number}), T1) \wedge T1 < T3.\end{aligned} \tag{9}$$

After an *ItemList* has been delivered, it should be paid. The *EPOsign* is a receipt of payment.

$$\begin{aligned}&\mathbf{H}(tell(B, A, deliver(ItemList, Price), A_{number}), T3) \rightarrow\\&\mathbf{E}(tell(A, B, payment(ItemList, Price, EPOSign), A_{number}), T4) \wedge T4 > T3.\end{aligned} \tag{10}$$

This constraint is the backward version of the previous one. It is important to have it also, since no payment should be done if the items have not been delivered.

$$\begin{aligned}&\mathbf{H}(tell(A, B, payment(ItemList, Price, EPOSign), A_{number}), T4) \rightarrow\\&\mathbf{E}(tell(B, A, deliver(ItemList, Price), A_{number}), T3) \wedge T3 < T4.\end{aligned} \tag{11}$$

If A should pay B, B should sign the payment receipt, and send it to a netbill server computee with the access key. For this purpose, we use the NetBill protocol: we assume to have a netbill server in charge of checking bank accounts and guaranteeing the ability to pay.

$$\begin{aligned}&\mathbf{H}(tell(A, B, payment(ItemList, Price, EPOSign), A_{number}), T4) \rightarrow\\&\mathbf{E}(tell(B, netbill, endorsedEpo(ItemList, Price, A, EPOSign, Key), A_{number}), T5) \wedge T5 > T4.\end{aligned} \tag{12}$$

This constraint is the backward version of the previous one.

$$\mathbf{H}(tell(B, netbill, endorsedEpo(ItemList, Price, A, EPOSign, Key), A_{number}), T5) \rightarrow \\ \mathbf{E}(tell(A, B, payment(ItemList, Price, EPOSign), A_{number}), T4) \wedge T4 < T5. \tag{13}$$

This integrity constraint states that if a bidder asks the netbill server to certify the transaction, the netbill server performs the transaction and sends back the result to the bidder.

$$\mathbf{H}(tell(B, netbill, endorsedEpo(ItemList, Price, A, EPOSign, Key), A_{number}), T5) \rightarrow \\ \mathbf{E}(tell(netbill, B, signedResult1(ItemList, Price, A, Result, Key), A_{number}), T6) \wedge T6 > T5. \tag{14}$$

This constraint is the backward version of the previous one.

$$\mathbf{H}(tell(netbill, B, signedResult1(ItemList, Price, A, Result, Key), A_{number}), T6) \rightarrow \\ \mathbf{E}(tell(B, netbill, endorsedEpo(ItemList, Price, A, EPOSign, Key), A_{number}), T5) \wedge T5 < T6. \tag{15}$$

Finally, when the netbill server informs on the bank transaction state, the bidder sends the auctioneer the same message and the access key to use the items.

$$\mathbf{H}(tell(netbill, B, signedResult1(ItemList, Price, A, Result, Key), A_{number}), T6) \rightarrow \\ \mathbf{E}(tell(B, A, signedResult2(ItemList, Price, Result, Key), A_{number}), T7) \wedge T7 > T6. \tag{16}$$

This constraint is again the backward version of the previous one.

$$\mathbf{H}(tell(B, A, signedResult2(ItemList, Price, Result, Key), A_{number}), T7) \rightarrow \\ \mathbf{E}(tell(netbill, B, signedResult1(ItemList, Price, A, Result, Key), A_{number}), T6) \wedge T7 > T6. \tag{17}$$

6 Human users and SOCS-SI: extending SOCS-SI with e-mail support

Before starting the description of the experiments done, we introduce now an extension to the SOCS-SI that enables us to enable human users to exchange messages in the society and participate, say to a combinatorial auction.

A new plug-in, `MailRecorder`, has been added to SOCS-SI, in order to cope with e-mail messages. Until now in fact SOCS-SI was equipped to interface with the Medium (the default communication layer in PROSOCS), with the JADE platform, with a file system, and with a user prompt. Hence a new component, able to properly interact with the e-mail, has been developed and tested.

The main advantage of considering the e-mail system as an agent communication layer is that whether it can be easily used by software agents, it can be used also by human users without much effort. Other communication layers, like the Prosocs Medium or the one supported by JADE, require a specific software interface in order to allow human users to send and receive messages with agents. The e-mail system instead doesn't require any additional software except for a Mail User Agent (known also as mail clients). The e-mail system is widely used. Therefore, it can be considered as a good mean for exchanging messages in a society.

Of course the e-mail system introduces some disadvantages that make it not suitable in a society populated by software agents only. For example, the delay between the action of "sending" and "receiving" a message can be not tolerable, unless specific delays are introduced in the protocol definition. Also the intrinsic non-robustness of the system is not acceptable.

However, the e-mail system represents a valid solution to easily develop and test societies and protocols where human users act as agents.

6.1 E-mail messages structure

Human users usually do not structure strictly their e-mails messages. However, in order to let SOCS-SI understand correctly these messages, some rules have been established on how to compose the mails. The actual implementation of `MailRecorder` recognizes and accepts only messages whose format respects the following rules:

1. Each e-mail address is recognized as a different user. A user that communicates with two different e-mails addresses is seen as two different users.
2. The "from" identifies always the sender of the message.
3. Intended receivers are specified always by using only the "to" field.
4. Intended societies are specified by using only the "cc" field.
5. The field "bcc" is simply ignored, but it could be used for hiding the control to the user.
6. The content of the message is expressed through the "subject" field. In particular, the subject should always be like:

   ```
   <dialogId> performative perfContent1,perfContent2,...
   ```

   where the dialog identifier is always the first token of the subject, enclosed by angular brackets. Then follows the performative of the message, separated by a blank space. Finally the parameters related to the specified performative: each parameter is separated from the others by commas.
7. The "content" field of the mail messages is ignored. This means that users can introduce text, or anyting else, into their messages. The rules above constraint only the fields *from*, *to*, *cc*, and *subject*.

6.2 Configuring the MailRecorder properly

The basic assumption for using the e-mail as communication layer is that each agent participating the society should have a mail box and a valid mail address. This is true also for SOCS-SI. Although in the experiments we have conducted SOCS-SI didn't send any message to other users, a mail box was needed for receiving messages.

To allow the `MailRecorder` to properly work it is then necessary to provide the following information:

i) A mail account where the messages can be retrieved.

ii) A correspondent password

iii) The name of the pop server and the corespondent port number

iv) The name of a smtp server and the correspondent port number. This information is not really necessary, but in order to allow future experiments about the publication of the social expectations it has been introduced here.

All these data must be properly specified int the configuration file `society.config`, which contains all the configuration information for SOCS-SI.

7 Combinatorial Auctions Experiments

We performed different types of experiments in this scenario. First, we have tested the Auction Solver implemented in ILOG to test its efficiency for increasing number of bidders. Then, we experimented the proof with the three protocols defined in section 5. We experimented the society alone with an email system to test if the proof could be used also for enabling humans to perform an auction. Finally, we performed the first integrated experiments where computees and society interact through the medium. This last experiment is matter of current work and it will be enriched in the near future.

All the experiments have been performed on a Pentium 4, 2.4 GHz, 512 MB.

7.1 Experiments on the Auction Solver

In the Combinatorial Auction scenario we have to cope with a complex combinatorial optimization problem: the Winner Determination Problem. Having an efficient, scalable and flexible tool that solves this problem is crucial for the efficiency of the overall system.

In this experiment, we exploit an Auction Solver implemented in ILOG solver [15] suitably wrapped in to Java. The Auction Solver is so efficient that it gives the possibility of scaling the auction dimension, and test the performances of the SOCS social infrastructure. Results are reported in Figure 3. We can see that auctions with 1000 bidders can be solved in less then 2 seconds. In the graph, we report the number of bids and the mean of task per bid. For each kind of problem, we report results on ten runs. The results refer to the time for finding the optimal solution plus the proof of optimality.

7.2 Experiments on Computees

We have also tested the scalability of the overall system, namely a computee representing the auctioneer, several computees representing bidders, and the Auction Solver using ILOG (but without a society observing the interaction). We have done this by varying the number of bidders involved in an auction and by varying the number of bids submitted by each bidder. As the main computational burden lies with the auctioneer, these experiments test, in particular, the scalability of the computee implementing the auctioneer.

These experiments have been carried out on two different machines, one was a desktop PC (2 GHz Pentium IV CPU, 512 MB of RAM, running Windows XP Professional), and the other was a laptop (Pentium III, 512 MB of RAM, running Windows 2000 Professional). The latter was running the auctioneer computee and the notepad object; the former was running the bidder computees and the Auction Solver.

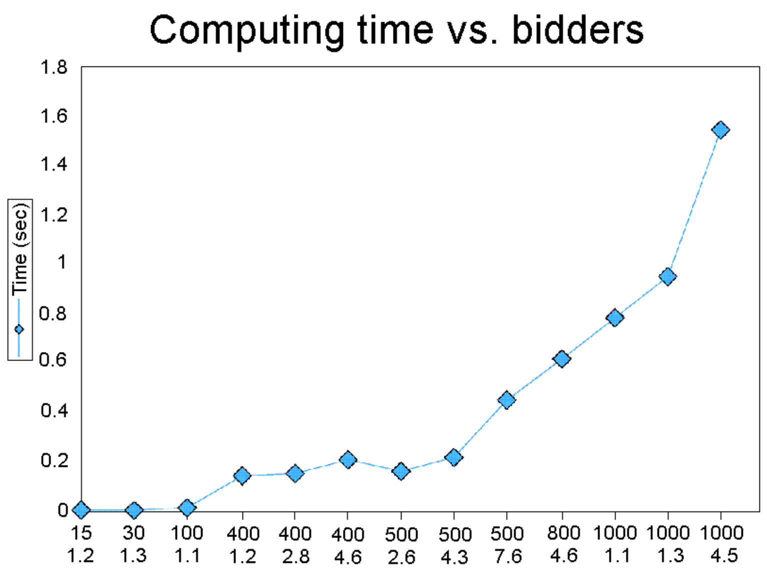

Figure 3: Test of the Auction Solver performance

A run of the experiment starts by the auctioneer sending an open-auction message to all bidders. Once a bidder has received such a message, it reacts to it by sending back several bids. Once a bid is received by the auctioneer, this bid is recorded using the notepad object. Once the deadline for bidding has passed, the auctioneers collects the recorded bids from the notepad and sends them to the Auction Solver. Finally, after having received the answer back from the Auction Solver, the auctioneer sends out either a win- or a lose-message for each of the bids received earlier.

The overall runtimes for 9 different settings (with 2–4 bidders and 2–4 bids per bidder concerning 6 items on auction) are shown in Figure 4. In the case of 2 bidders and 2 bids per bidder, for instance, the auctioneer executes 1 open-auction action, 5 notepad access actions, 1 Auction Solver access action, and 4 win/lose actions, and it receives 4 messages from the bidders and 1 message each form the notepad and the Auction Solver. The overall experiment takes an average of 36 seconds

While our experiments show that the system copes well with smaller problem instances, we have also observed clear performance problems as the overall number of bids increases. This is not surprising given the prototypical character of our platform and the generality of our approach.

7.3 Experiments on the Proof Procedure (society alone)

The experiments on society alone fall into two different groups: experiments devoted to test the performances of the proof procedure, and experiments devoted to test if the proof can work in a real setting, with delayed messages.

For the first purpose, we have arranged a first set of experiments where the proof procedure performances are tested for an increasing number of bidders. Bidders in fact send messages

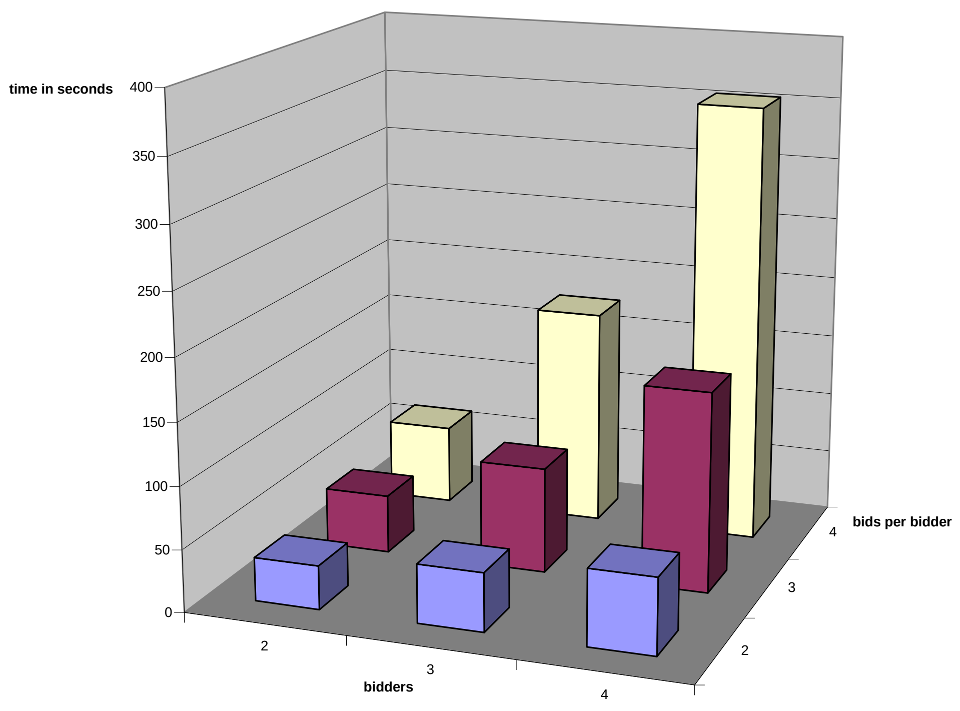

Figure 4: Results on computees performances

which should be controlled by the society and the more the number of bidders the more the number of messages to control. We have tested the three protocols described in subsections 5.1, 5.2 and 5.3.

Concerning the single auction, we can see that the tests provide an idea on how the proof scales for an increasing number of bidders and consequently of messages. We can see that results are good, since the prototype we implemented works well up to 50 bidders answering in half a minute.

As far as the double auction is concerned, it is intuitive that the number of exchanged messages is almost doubled with respect to a traditional auction. In fact, for the same number of bidders, two auctions are indeed started. We can see from the figures 7 (where the conformance test succeeds) and 8 (where the conformance test fails) that the proof scales perfectly, being the time for testing conformance almost doubled w.r.t. a single auction.

As far as the auction plus the Netbill protocol, results are very good. In fact, the time for checking conformance is similar to that of the traditional auction. We can see from the figures 9 (where the conformance test succeeds) and 10 (where the conformance test fails) that the proof scales perfectly.

As we can see from Figure 3 the Auction Solver implemented in ILOG is far more efficient since it scales up to 1000 bids within 2 seconds. Slower performances are achieved by the conformance checking of the society protocol (around 30 seconds for checking messages exchanged in auctions with 50 bidders). However, even if the components have different performances, from the testing, we can conclude that both components can be used in a real combinatorial auction

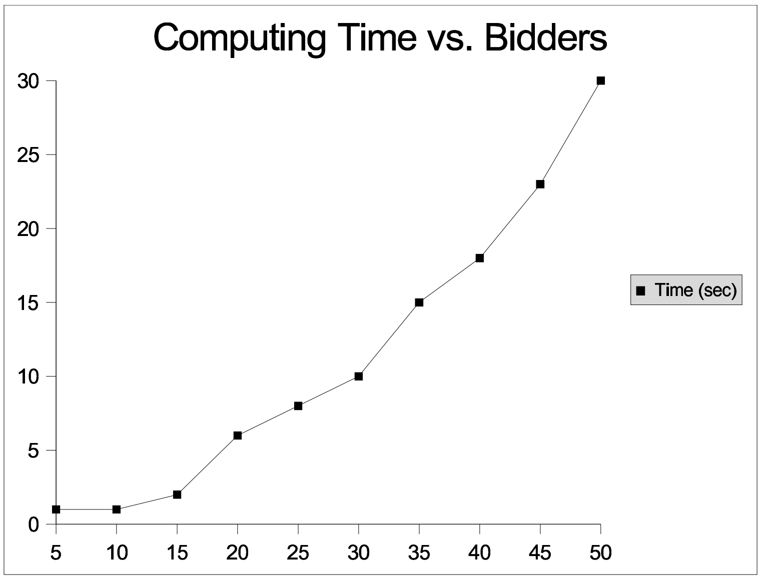

Figure 5: Proof performance on a traditional auction (conformant protocol)

scenario since time limits required for answering are much larger than sum of the answer times of the components.

7.3.1 Tests with the e-mail support

Two runs of the experiments have been executed, with the following configuration:

- The Auctioneer was located in Bologna.
- Two bidders were located in Bologna while the remaining two were in Ferrara.
- The Netbill server was located in Ferrara.
- The SOCS-SI was running on a machine physically located in Bologna.

Both the runs were concluded in around 20 minutes, but with different results. In the first run the protocol was *violated*, while in second SOCS-SI returned as answer the success of $\mathcal{S}$CIFF, hence the interaction between the human users complied with the protocol.

The first run violated the protocol since one of the winning bidders tried (by mistake) to cash a payment check he wasn't allowed to. The NetBill server executed the payment operation, but the SOCS-SI revealed the violation of some expectations, thus invalidating all the interaction.

The test has been considered as a positive demonstration of the use of SOCS-SI within human societies. The result of the first experiment in particular has shown two different aspects of the use of the e-mail and SOCS-SI with humans:

i) SOCS-SI is a useful tool for detecting possible users misbehavior (malicious or unintentional). The first run of this experiment in fact was automatically declared as not compliant with the protocol.

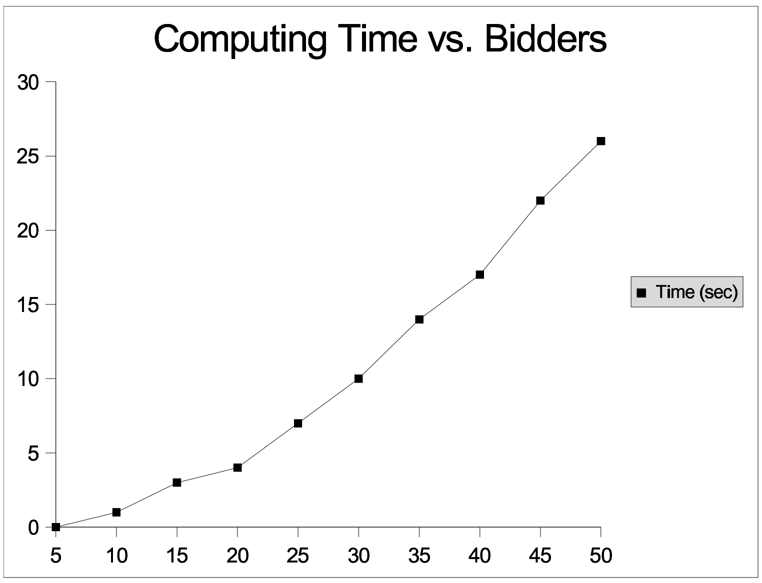

Figure 6: Proof performance on a traditional auction (non conformant protocol)

ii) The misbehavior detected by SOCS-SI was unintentional, and was due by a cut&paste mistake of a bidder. Such mistakes are quite common between humans, and happen quite often. A real combinatorial auction system based on e-mails should take care of the complexity of the messages exchanged: higher the complexity, in fact, higher is the probability of a mistake by a user. Pre-formatted e-mails, or tools that address the problem of composing the e-mail with right content could greatly reduce the unintentional violations of the protocol.

All these data must be properly specified int the configuration file `society.config`, which contains all the configuration information for SOCS-SI.

7.4 Integrated Experiment

After testing the single components separately, we have also tested the scalability of the overall system by putting together computees exchanging messages as done in section 7.2 with the society infrastructure checking for computees communication conformance. The experiments have been done on the traditional combinatorial auction since the aim of this test is to check the real integrability of the components and their interactions.

We have had two meetings for integrating the society with the computees managing the auction. We first solved some problems arisen in merging different components, then solved some problems coping with deadlines and we finally ended up with a stable system. We have performed some preliminary test and the results are good, even if auction instance run are small-size ones. The results are very similar to those presented in figure 4, since the society checking for conformance the communication protocol introduces a negligible overhead for small auction instance on the overall computation, as can be noted in figure 5 and figure 6.

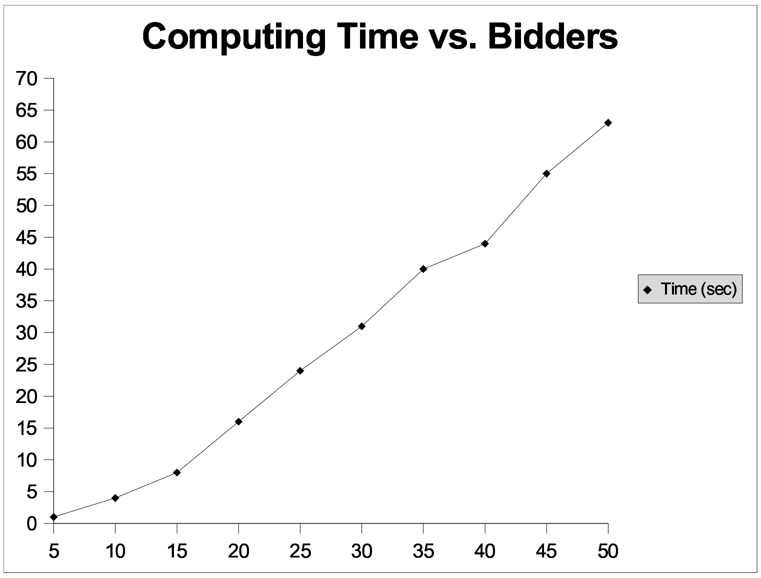

Figure 7: Proof performance on a double auction (conformant protocol)

The society is able to check the non conformance to deadline at runtime, while other kind of failures are detected when the history is closed.

We plan to have an extensive test session on the 15th-16th of March in Bologna. One integrated example will be presented at the review meeting in Edinburgh in April.

8 Related Work

Combinatorial Auctions are increasingly studied since they have the advantage that bidders can bid on combinations of items. This leads to more efficient allocation than traditional auctions where the bidders valuations are only additive. The drawback is that evaluating bids and determining the winning bids is a NP-hard problem. However, there are different systems and methods to solve a combinatorial auction in an efficient way. The methods used to solve the problem exploit

- dynamic programming techniques [24]
- approximate methods that look for a reasonably good allocation of bids [10, 26]
- integer programming techniques [7, 21]
- search algorithms [29]

An aspect which plays an important role in auctions in general is negotiation. Negotiation is one of the main research areas in distributed systems. The area of negotiation is broad and applies to different scenarios, such as for instance multi-agent systems [16, 23]. In most cases agents need to negotiate because they operate in environments with limited resource availability.

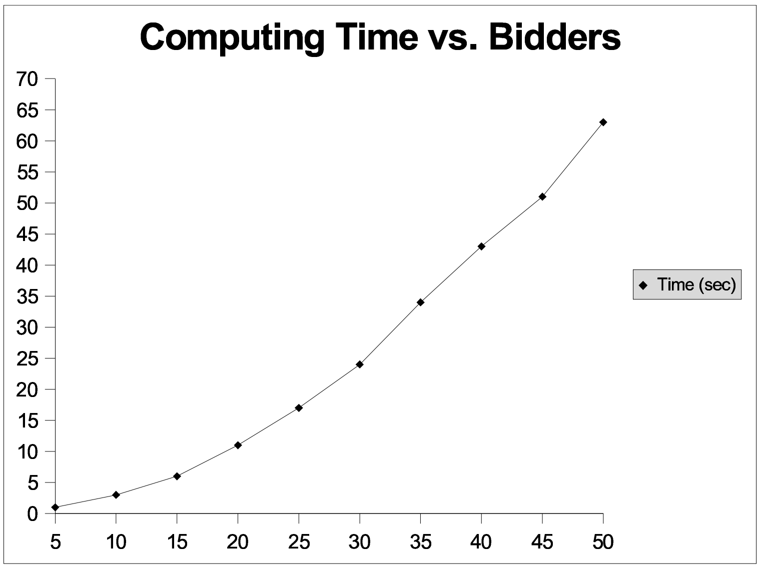

Figure 8: Proof performance on a double auction (non conformant protocol)

For example, one-to-many negotiation is used for auctions, where auctioneers and bidders reach an agreement on the cost of the items on sale. One-to-one negotiation is used, for instance, for task reallocation [27] and for resource reallocation [22], where the limited resources may be time, the computational resources of agents, or physical resources needed to carry out some tasks. Persuasion also plays a role when the agent goals are conflicting and the resources in the system are not enough to carry out all agent plans, therefore agents try to modify each other goals or intentions.

An interesting example of system similar to ours is ISLANDER [30], a tool to specify protocols in a system ruled by electronic institutions that has been applied to a Dutch auction (and other scenarios). Their formalism is multi-levelled: agents have *roles*, agents playing a role are constrained to follow *protocols* when they belong to a *scene*; agents can move from a scene to another by means of *transitions.* As in various works, protocols are defined by means of transition graphs, in a finite state machine. Our definition of protocols is wider than finite state machines, and leaves more freedom degrees to the agents. In our model, an event could be *expected to happen*, *expected not to happen* or have no expectations upon, thus there can be three possible states, while in finite state machines there are only two states. Moreover, they apply the model to the Dutch auction, while we focus on combinatorial auctions, that are more expressive, as widely documented in the literature, but also makes the solving problem NP-hard. For this reason, a general purpose proof-procedure that checks the compliance to the protocol could be inefficient. We proposed a specialized solver and integrated it in our system.

Many argumentative frameworks are proposed in this respect [2, 17]. They often try to embrace very hard problems and result in very good descriptive models for them. In general, an execution model for most argumentation based systems is not straightforward to obtain, and it is also difficult to give properties and forecast the behavior of an implemented system, if any. Recent approaches to negotiation are based on dialogue, as one of the most flexible interaction

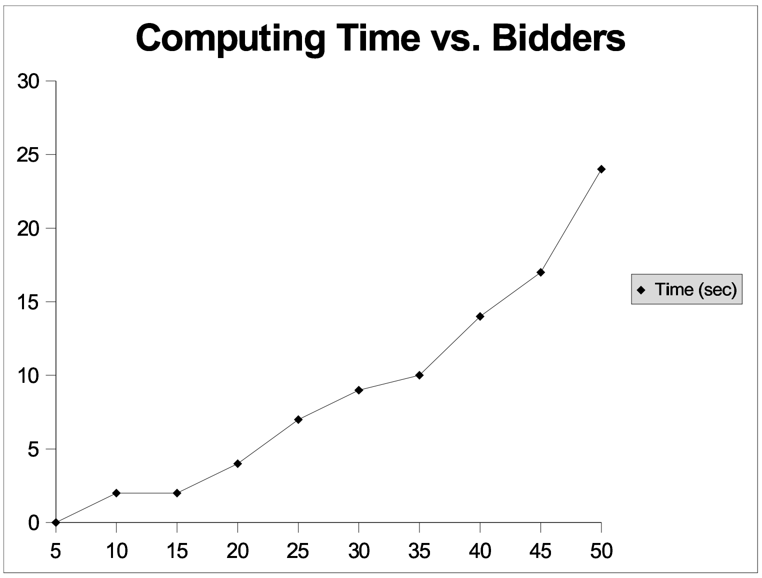

Figure 9: Proof performance on an auction plus Netbill (conformant protocol)

patterns in multi-agent systems, being something between completely fixed protocols and totally free conversations [8]. Among them, the work of [25] aims at combining in a negotiation framework both a logic-based formal approach and an execution model to be used in order to implement the system.

Learning has been extensively applied to (bilateral) auctions and call market scenario, where there exist two agents, a buyer and a seller: this scenario is very similar to the two-players game one. Two basic approaches have been traditionally identified in the literature:

1. Reinforcement learning: reinforcement models update some unobserved propensity, reinforcement level, or attraction, to what a chosen strategy actually earned;

2. Belief learning: belief models form beliefs based on some weighted average of previous observations of what other computees (players, agents, etc.) have done.

Extensive work about online learning (e.g., based upon reinforcement learning when the model of the other agent is not available) has been done in two-player setting. Also, on-line learning in single state multi-agent systems have been considered, for instance, in [5]. Explicit recursive modeling of other agents in multi-agent settings in a particular exchange market was proposed by Vidal and Durfee [33]. Relevant work about the application of learning techniques to auction scenario is contained in [14], and [4]. In particular, in [14] the authors have modeled a dynamic multi-agent system consisting of more than two agents where learning is applied in order to learn just an action (the next worth to be done), or a policy /strategy. Various degrees of learning have also considered (as done by Vidal and Durfee [33]), depending on the ignorance or awareness of the behaviour/status/policy of the other agents. The learning methodology introduced in [14] has been experimented in double actions, where both buyers and sellers submit bids. Camerer and Ho [3] proposed the Experience-Weighted Attraction

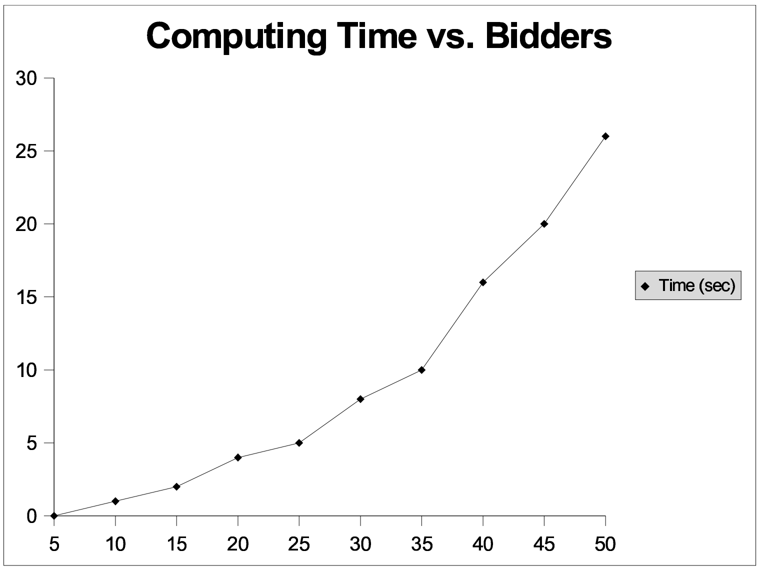

Figure 10: Proof performance on an auction plus Netbill (non conformant protocol)

(EWA) learning which hybridizes the main features of reinforcement and belief learning. EWA has been extended later in [4] to the scenario of bilateral call markets, where a single buyer and a seller are each privately informed about their own value and cost, and the probability distributions of values and costs are commonly known. These scenaria are named bilateral call markets because they are two-agent examples of general call markets, where usually many traders submit demand and supply schedules, and the market is "called" at the price where supply meets demand. Finally, it is worth to mention the work of [34], which is of interest for SOCS even if it does not properly fits the auction scenario: the authors consider, in fact, scenaria where there is no centralized control, but the learning algorithm of each agent (based upon reinforcement) is such that the collective behavior of the agent system maximises a provided global utility function. Therefore, the (overall) dynamics is governed by the collective effects of the individual agents each modifying their behavior via their (local) learning algorithm.

References

[1] M. Alberti, A. Bracciali, F. Chesani, U. Endriss, M. Gavanelli, W. Lu, K. Stathis, and P. Torroni. SOCS prototype. Technical report, SOCS Consortium, 2003. Deliverable D9.

[2] L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence, Berlin, Germany (ECAI 2000)*. IOS Press, Aug. 2000.

[3] C. Camerer and T. Ho. Experienced-weighted attraction learning in normal form games. *Econometrica*, 67(4):827–874, 1999.

[4] C. F. Camerer, D. Hsia, and T. Ho. Learning in bilateral call markets. Technical report, Caltech Working Paper, 2000.

[5] D. Carmel and S. Markovitch. Opponent modeling in multi-agent systems. In *Proceedings of the IJCAI-95 Workshop Adaption and Learning in Multi-Agent Systems*, number 1042 in Lecture Notes in Computer Science, pages 40–52. Springer-Verlag, 1996.

[6] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 75–90, London, Apr. 1996.

[7] J. Collins and M. Gini. An integer programming formulation of the bid evaluation problem for coordinated tasks. In B. Dietrich and R. V. Vohra, editors, *Mathematics of the Internet: E-Auction and Markets*, volume 127 of *IMA Volumes in Mathematics and its Applications*, pages 59–74. Springer-Verlag, New York, 2001.

[8] F. Dignum, B. Dunin-Keplicz, and R. Verbrugge. Dialogue in team formation. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, number 1916 in Lecture Notes in Computer Science, pages 264–280. Springer-Verlag, 2000.

[9] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure: Definition and soundness results. Technical Report 2004/2, Department of Computing, Imperial College London, May 2004.

[10] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden (IJCAI-99)*, volume 1, pages 548–553. Morgan Kaufmann Publishers, 1999.

[11] A. Guerri and M. Milano. Exploring CP-IP based techniques for the bid evaluation in combinatorial auctions. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 863–867. Springer-Verlag, 2003.

[12] A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the Sixteenth European Conference on Artificial Intelligence, Valencia, Spain (ECAI 2004)*, pages 475–479. IOS Press, Aug. 2004.

[13] R. Guttman, A. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 13(2):143–147, 1998.

[14] J. Hu and M. Wellman. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.

[15] ILOG S.A., France. *ILOG Solver*, 5.0 edition, 2003.

[16] N. R. Jennings. Automated haggling: Building artificial negotiators (invited talk). In *AISB'01 Convention, York, UK*, March 2001. Electronically available slides, `http://www.ecs.soton.ac.uk/~nrj/download-files/negotiation.pdf`.

[17] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation; a logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[18] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, 2000.

[19] W. Lu and K. Stathis. Incorporating objects in PROSOCS artificial worlds. Technical report, SOCS Consortium, 2004. IST32530/CITY/015/DN/I/b2.

[20] P. J. McCann and G.-C. Roman. Modeling mobile ip in mobile unity. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(2), Apr. 1999.

[21] N. Nisan. Bidding and allocation in combinatorial auctions. pages 1–12. ACM Press, 2000.

[22] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

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

[24] M. Rothkopf, A. Pekec, and R.M.Harstad. Computationally manageable combinational auctions. *Management Science*, 44(8):1131–1147, 1998.

[25] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Intelligent Agents VIII: 8th International Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 405–421. Springer-Verlag, 2002.

[26] Y. Sakurai, M. Yokoo, and K. Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00)*, pages 30–37, 2000.

[27] T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents.* Computer science, University of Massachusetts at Amherst, September 1996.

[28] T. Sandholm. eMediator: a next generation electronic commerce server. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000)*, 2000.

[29] T. Sandholm. Algorithm for optimal winner determination in combinatorial auction. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[30] C. Sierra and P. Noriega. Agent-mediated interaction. From auctions to negotiation and argumentation. In M. d'Inverno, M. Luck, M. Fisher, and C. Preist, editors, *Foundations and Applications of Multi-Agent Systems, UKMAS Workshop 1996-2000, Selected Papers*, volume 2403 of *Lecture Notes in Computer Science*, pages 27–48. Springer-Verlag, 2002.

[31] K. Stathis. Location-aware SOCS: The Leaving San Vincenzo scenario. Technical Report IST32530/CITY/002/IN/PP/a1, SOCS consortium, 2002.

[32] K. Stathis and F. Toni. Ambient Intelligence using KGP Agents. In *Proceedings of the 2nd European Symposium for Ambient Intelligence*, Lecture Notes in Artificial Intelligence, Eindhoven, Nov. 2004. Springer-Verlag.

[33] J. Vidal and E. Durfee. Agents learning about agents: A framework and analysis. In *Working Notes of the AAAI-97 workshop on Multiagent Learning*, pages 71–76, 1997.

[34] D. Wolpert, K. Wheeler, and K. Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, Washington*, 1999.

[35] P. Wurman, M. Wellman, and W. Walsh. The michigan internet auctionbot: A configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, 1998.