Expressing Interaction in Combinatorial Auction through Social Integrity Constraints [⋆]

Marco Alberti[1], Federico Chesani[2], Marco Gavanelli[1], Alessio Guerri[2], Evelina Lamma[1], Paola Mello[2], and Paolo Torroni[2]

[1] Dip. di Ingegneria - Università di Ferrara - Via Saragat, 1 - 44100 Ferrara, Italy.
{malberti|m gavanelli|elamma}@ing.unife.it
[2] DEIS - Università di Bologna - Viale Risorgimento, 2 - 40136 Bologna, Italy.
{fchesani|aguerri|pmello|ptorroni}@deis.unibo.it

Abstract. Combinatorial Auctions are an attractive application of intelligent agents; their applications are countless and are shown to provide good revenues. On the other hand, one of the issues they raise is the computational complexity of the solving process (the Winner Determination Problem, WDP), that delayed their practical use. Recently, efficient solvers have been applied to the WDP, so the framework starts to be viable.
A second issue, common to many other agent systems, is *trust*: in order for an agent system to be used, the users must *trust* both their representative and the other agents inhabiting the society: malicious agents must be found, and their violations discovered. The *SOCS* project addresses such issues, and provided a language, the *social integrity constraints*, for defining the allowed interaction moves, together with a proof procedure able to detect violations.
In this paper we show how to write a protocol for the combinatorial auctions by using social integrity constraints. In the devised protocol, the auctioneer interacts with an external solver for the winner determination problem.

1 Introduction

The software agent technology seems an attractive paradigm to support auction applications [1]: agents acting on behalf of end-users could reduce the effort required to complete auction activities. Agents are intrinsically autonomous and can be easily personalised to embody end-user preferences. In addition, they are adaptive and capable of learning from both past experience and their environment, in

[⋆] The authors wish to thank Michela Milano for her precious help. This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 project in the context of the Global Computing initiative of the FET (Future Emerging Technology) initiative.

order to cope with changing operating conditions and evolving user requirements [2].

In an auction we have two roles: the *auctioneer* and the *bidders*. While in the past bidders were only humans, recent Internet auction servers [3] allow software agents to participate in the auction on behalf of end-users, and some of them even have a built-in support for mobile agents [4]. As the rise of the Internet and electronic commerce continues, dynamic automated markets will be an increasingly important domain for agents.

Depending on the kind of auction, the auctioneer either sells a set of goods trying to maximise the profit, or buys a set of goods minimising the cost. Bidders have the goal to obtain (respectively, sell) the goods under convenient price conditions.

Of course, one of the issues in e-commerce and, in particular, in electronic auctions, is *trust*: in order for the system to be used at all, each user must trust its representative agent in the auction. The agent must be well specified, and a correspondence between specification and implementation must be formally proven. Also, even if the agents are compliant to their specifications, the compliance to the social rules and protocols must be provable, in order to avoid, or, at least, detect malicious behaviours.

A typical answer to such issues is to model-check the agents with respect to both their specifications and requirements coming from the society. However, this is not always possible in open environments: agents could join the society at all times and their specifications could be unavailable to the society. Thus, the correct behaviour of agents can be checked only from the external in an open environment: by monitoring the communicative actions of the agents.

The *SOCS* project [5] addresses these issues by providing formal definitions both for the agents, that are based on Computational Logics, and are thus called *Computees*, and for the society in an open environment. Being logic-based, the computees are more *trustable*, as their specification and implementation almost coincide.

In this paper, we focus on the societal aspects, and on the compliance of the computees (or, in general, agents) to protocols and social rules. These can be easily expressed in a logic language, the *Social Integrity Constraints* (ic_S) that are an extension of the integrity constraints widely used in Abductive Logic Programming, and, in particular, extend those of the IFF proof procedure [6].

We implemented an abductive proof-procedure, called $\mathcal{S}$CIFF (extending the IFF [6]), that is able to check the compliance to protocols and social rules given a history of communicative actions. Besides a posteriori check of compliance, $\mathcal{S}$CIFF also accepts dynamically incoming events, so it can check compliance during the evolution of the societal interaction, and raise violations as soon as possible. $\mathcal{S}$CIFF extends the IFF in a number of directions: it provides a richer syntax, it caters for interactive event assimilation, it supports fulfillment check and violation detection, and it embodies CLP-like constraints [7] in the ic_S. $\mathcal{S}$CIFF is sound [8] with respect to the declarative semantics of the society model, in its abductive interpretation.

The $\mathcal{S}$CIFF has been implemented and integrated into a Java-Prolog-CHR based tool, named SOCS-SI (*SOCS* Social Infrastructure [9]). This implementation can be used to verify that agents comply to ic_S-based specifications. The intended use of SOCS-SI is in combination with agent platforms, such as PROSOCS [10], for on-the-fly verification of compliance to protocols.

2 ***SOCS*** social model

The society knowledge is specified declaratively, and is mainly composed of two parts: a *static* part, defining the society organisational and "normative" elements, and a *dynamic* part, describing the "socially relevant" events, that have so far occurred.

The static knowledge in a society $\mathcal{S}$ is given by:

- a *Social Organization Knowledge Base (SOKB)*: a logic program;
- a set $\mathcal{IC}_S$ of *Social Integrity Constraints* (ic_S): implications that can relate elements in the dynamic part, CLP constraints and predicates defined in the SOKB.

We assume that the societal infrastructure is time by time aware of social events that dynamically happen in the environment (*happened* events). The "normative elements" are encoded in the ic_S. Based on the available history of events, and on the ic_S-based specification, the society can define what the "expected social events" are, i.e., what events are expected (*not*) to happen. The expected events, called *social expectations*, reflect the "ideal" behaviour of the agents.

The society knowledge evolves, as new events happen. The evolving part of the society knowledge is called *Social Environment Knowledge Base*, ($SEKB$), and consists of:

- *Happened events*: ground atoms $\mathbf{H}(\mathit{Event}[, \mathit{Time}])$;
- *Expectations*: events that should (but might not) happen (atoms $\mathbf{E}(\mathit{Event}[, \mathit{Time}])$), and events that should not (but might indeed) happen (atoms indicated with functor $\mathbf{EN}(\mathit{Event}[, \mathit{Time}])$). Explicit negation ($\neg$) can be applied to expectations.

In our context, "happened" events are all the observable events which are relevant to the society. The collection of such events is the history **HAP**.

While **H** atoms are always ground, the arguments of expectations can contain variables. Intuitively, if an $\mathbf{E}(X)$ atom is in the set of generated expectations ($\mathbf{E}(X) \in \mathbf{EXP}$), "**E**" indicates a wish about an event $\mathbf{H}(Y) \in \mathbf{HAP}$ which unifies with it: X/Y. One such event will be enough to fulfill the expectation: thus, variables in an **E** atom are always existentially quantified.

For instance, in an auction context the atom:

$$\mathbf{E}(\mathit{tell}(\mathit{Auctioneer}, \mathit{Bidders}, \mathit{openauction}(\mathit{Item}, \mathit{Dialogue})), T_{open})$$

stands for an expectation about a communicative act *tell* made by a computee ($\mathit{Auctioneer}$), addressed to a (group of) computees ($\mathit{Bidders}$), with subject $\mathit{openauction}(\mathit{Item}, \mathit{Dialogue})$, at a time T_{open}.

3 The Combinatorial Auctions scenario

There exist different kinds of auctions. In this paper, we consider *single unit reverse auctions.*

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

The Winner Determination Problem in combinatorial auctions is NP-hard [11], so it cannot be addressed naively, but we need to exploit smart solving techniques. We address the problem by exploiting a commercial constraint solver, *ILOG* solver [12], for efficiently solving a combinatorial optimization problem. We have implemented in *ILOG* a module called *Auction solver* [13] embedding an algorithms portfolio and an automatic algorithm selection strategy of constraint based technologies, namely constraint and integer programming combined [14].

So, one of the actors is the Auction Solver, that can be integrated in a number of different ways. We first give the general auction protocol in terms of ic_S in Section 3.1, then we propose one such scenario, and outline some alternative solutions that will be investigated in the future.

3.1 Auction Protocol

To start with, we can use ic_S to check that the auction protocol is indeed respected, without caring for the NP-hard aspects of the protocol. I.e., we will not check that the result provided by the auctioneer is indeed the optimal solution of the WDP.

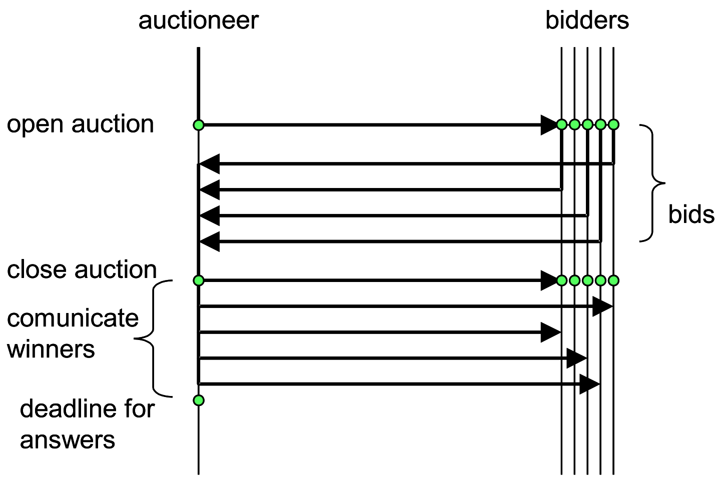


Fig. 1. Auction Protocol

At this level, the auction protocol is the one depicted in Figure 1. Each time a bidding event happens, the auctioneer should have sent an *openauction* event:

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(B, A, \mathit{bid}(\mathit{ItemList}, P), \mathit{Dialogue}), T_{\mathit{bid}}) \rightarrow \\ \mathbf{E}(\mathit{tell}(A, B, \mathit{openauction}(\mathit{Items}, T_{\mathit{end}}, T_{\mathit{deadline}}), \mathit{Dialogue}), T_{\mathit{open}}) \wedge \\ T_{\mathit{open}} < T_{\mathit{bid}} \wedge T_{\mathit{bid}} \leq T_{\mathit{end}}\end{array} \tag{1}$$

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

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(A, \mathit{Bidder}, \mathit{openauction}(\mathit{Items}, T_{\mathit{end}}, T_{\mathit{deadline}}), \mathit{Dialogue}), T_1) \wedge \\ \mathbf{H}(\mathit{tell}(\mathit{Bidder}, A, \mathit{bid}(\mathit{ItemBids}, P), \mathit{Dialogue}), \_) \wedge \\ \mathit{not\ included}(\mathit{ItemBid}, \mathit{Items}) \\ \rightarrow \mathbf{E}(\mathit{tell}(A, \mathit{Bidder}, \mathit{answer}(\mathit{lose}, \mathit{Bidder}, \mathit{ItemBids}, P), \mathit{Dialogue}), \_) \\ \\ \mathit{included}([], \_). \\ \mathit{included}([H|T], L) : -\mathit{member}(H, L), \mathit{included}(T, L).\end{array} \tag{2}$$

The auction should also be closed at time T_{end}

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(A, B, \mathit{openauction}(\mathit{Items}, T_{\mathit{end}}, T_{\mathit{deadline}}), \mathit{Dialogue}), T_{\mathit{open}}) \\ \rightarrow \mathbf{E}(\mathit{tell}(A, B, \mathit{closeauction}, \mathit{Dialogue}), T_{\mathit{end}})\end{array} \tag{3}$$

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

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(S, A, \mathit{bid}(\mathit{ItemList}, P), \mathit{Dialogue}), T_{\mathit{bid}}) \wedge \\ \mathbf{H}(\mathit{tell}(A, \mathit{Bidders}, \mathit{openauction}(\mathit{Items}, T_{\mathit{end}}, T_{\mathit{deadline}}), \mathit{Dialogue}), T_{\mathit{open}}) \\ \rightarrow \quad \mathbf{E}(\mathit{tell}(A, S, \mathit{answer}(X, S, \mathit{ItemList}, P), \mathit{Dialogue}), T_{\mathit{answer}}) \wedge \\ \qquad T_{\mathit{answer}} > T_{\mathit{end}} \wedge T_{\mathit{answer}} < T_{\mathit{deadline}} \wedge X :: [\mathit{win}, \mathit{lose}]\end{array} \tag{4}$$

A bidder should not receive for the same auction on the same bid two conflicting answers:

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(R, S, \mathit{answer}(\mathit{lose}, S, \mathit{ItemList}, P), \mathit{Dialogue}), \_) \rightarrow \\ \qquad \mathbf{EN}(\mathit{tell}(R, S, \mathit{answer}(\mathit{win}, S, \mathit{ItemList}, P), \mathit{Dialogue}), \_)\end{array} \tag{5}$$

$$\begin{array}{l}\mathbf{H}(\mathit{tell}(R, S, \mathit{answer}(\mathit{win}, S, \mathit{ItemList}, P), \mathit{Dialogue}), \_) \rightarrow \\ \qquad \mathbf{EN}(\mathit{tell}(R, S, \mathit{answer}(\mathit{lose}, S, \mathit{ItemList}, P), \mathit{Dialogue}), \_)\end{array} \tag{6}$$

Two different winning bids cannot contain the same item:

$$\begin{aligned}
&\mathbf{H}(tell(A, Bidder_1, answer(win, Bidder_1, ItemList, P), Dialogue), \_)\\
&\wedge\mathbf{H}(tell(Bidder_2, A, bid(ItemList', P), Dialogue), T_{bid})\\
&\wedge Bidder_1 \neq Bidder_2\\
&\wedge intersect(ItemList, ItemList') \rightarrow\\
&\mathbf{EN}(tell(A, Bidder_2, answer(win, Bidder_2, ItemList', P'), Dialogue), \_)
\end{aligned} \tag{7}$$

Example. Suppose that a bidder tries to force the auctioneer to buy an item that was not requested in the *openauction*; e.g., the history could be:

$$\mathbf{H}(tell(auct, bidder_1, openauction([pc, monitor, mouse], 10, 20), 1), 1)$$
$$\mathbf{H}(tell(bidder_1, auct, bid([keyboard], 10), 1), 3)$$

In this case, the protocol defines the correct behaviour of the auctioneer: it raises the expectation

$$\mathbf{E}(tell(auct, bidder_1, answer(lose, bidder_1, [keyboard], 10), 1, \_).$$

If the auctioneer does not give a matching reply, the proof procedure will raise a violation.

3.2 Implementation with the Auction Solver

In this scenario, the Auction solver is conceived as a passive 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 2.

The auctioneer provides the auction solver a WDP instance and receives its optimal solution.

In this case the auctioneer should collect the data of the instance and send them to the auction solver.

We have some preliminary results for this scenario, shown in Figure 3, where the computation times of the $\mathcal{S}$CIFF proof procedure and of the Auction Solver are shown.

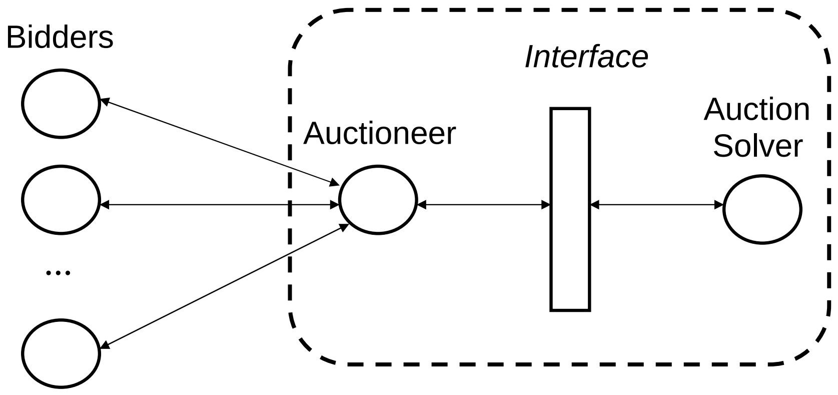
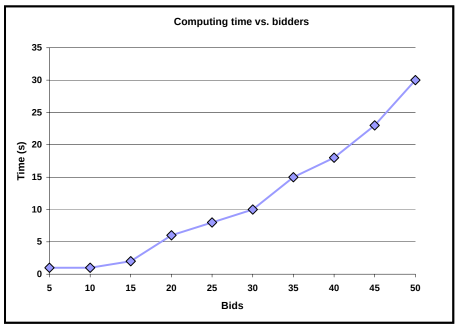
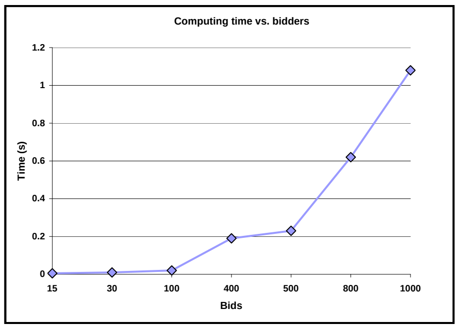


Fig. 2. Computees and object involved in a Combinatorial Auctions



(*a*) Time taken by the $\mathcal{S}$CIFF proof procedure to verify compliance



(*b*) Time taken by the Auction Solver to solve the WDP

Fig. 3.

4 Other possible solutions

There are obviously other solutions for this problem, that will be taken into account in future work. In particular, we can, at society level, monitor the communications between the auctioneer and the auction solver.

Since the auctioneer is autonomous, it might decide to take a sub-optimal decision, for various reasons: e.g., the auctioneer might have been bribed by some bidders to make them win. Since the WDP is hard, we assume that only a specific solver can efficiently find the optimum: it is unlikely that the auctioneer can find it within the given deadlines without relying on the auction solver. It could be

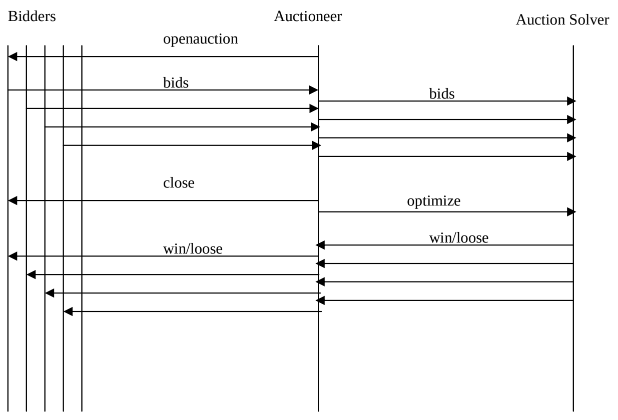


Fig. 4. Auction protocol when the society checks the communications between the auctioneer and the auction solver

sensible to have, in the society, the auction solver as a *trusted object*: its replies are supposed to be correct. If the society does not trust the auctioneer but only the passive object *auction solver*, we have various possibilities; we show two of them.

In the first, the auctioneer is separated from the auction solver, and the society checks (besides the compliance to the protocol given in Section 3.1), that the auctioneer sends indeed to the auction solver the same bids it receives (not a fake problem in which, e.g., some bids were excluded to make some bidder win) and that the replies of the auctioneer are indeed the same computed by the solver (Figure 4):

$$\begin{array}{r}\mathbf{H}(tell(Bidder_1, A, bid(ItemList', P), Dialogue), \_) \rightarrow \\ \mathbf{E}(ask(A, ILOG, bid(ItemList', P), Dialogue), \_)\end{array} \tag{8}$$

$$\begin{array}{r}\mathbf{H}(say(ILOG, A, answer(X, S, ItemList, P), Dialogue), \_) \rightarrow \\ \mathbf{E}(tell(A, S, answer(X, S, ItemList, P), Dialogue), \_)\end{array} \tag{9}$$

In the second solution, we do not want the society to force the auctioneer to use a given object, or algorithm, to solve the WDP, but we leave it free to choose autonomously how to solve the problem. It might use our solver, or it might do it in other ways. In such a case, however, the auctioneer must face an external check: a controller can check that the provided solution is indeed correct. In order to check

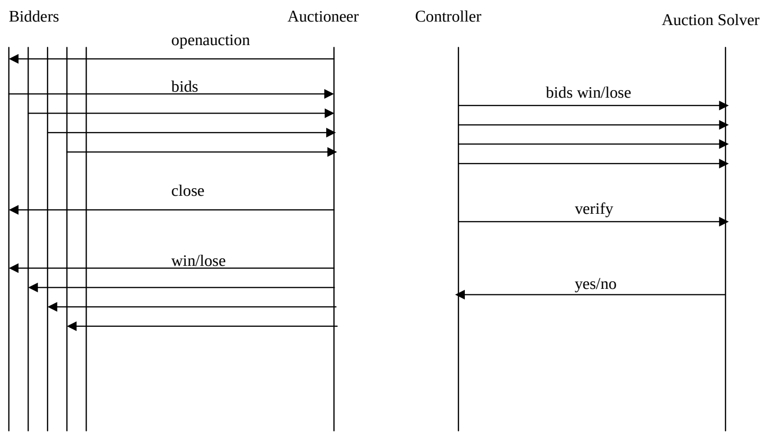


Fig. 5. Auction protocol: the society controls the auctioneer via a controller interacting with the auction solver

the validity of a solution, a computee takes the role of *controller*, and uses the trusted object *auction solver* in order to prove to the society the correctness of the solution (Figure 5).

5 Conclusions

Combinatorial auctions are recently starting to withdraw from the set of *practically unusable* applications as more efficient solvers are being produced for the winner determination problem. One of their natural applications involve intelligent agents as both bidders and auctioneers, but this raises the problem of humans trusting their representatives, and the other agents in the society.

Through the tools provided by the *SOCS* project, we give means for the user to specify the fair and trusty behaviour, and a proof procedure for detecting the unworthy and fallacious one. We defined the combinatorial auctions protocol through social integrity constraints, also exploiting an efficient solver for the winner determination problem.

References

1. Chavez, A., Maes, P., Kasbah: An agent marketplace for buying and selling goods. In: Proceedings of the 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. (1996)

2. Guttman, R., Moukas, A., Maes, P.: Agent-mediated electronic commerce: A survey. Knowledge Engineering Review **13(2)** (1998) 143–147
3. Wurman, P., Wellman, M., Walsh, W.: 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)
4. Sandholm, T.: eMediator: a next generation electronic commerce server. In: Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000). (2000)
5. : (Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees) `http://lia.deis.unibo.it/Research/SOCS/`.
6. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
7. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
8. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of interaction protocols: a computational logic approach based on abduction. Technical Report CS-2003-03, Dipartimento di Ingegneria di Ferrara, Ferrara, Italy (2003) Available at `http://www.ing.unife.it/aree_ricerca/informazione/cs/technical_reports`.
9. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based tool. In Trappl, R., ed.: Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4), Vienna, Austria, Austrian Society for Cybernetic Studies (2004) 570–575
10. Stathis, K., Kakas, A.C., Lu, W., Demetriou, N., Endriss, U., Bracciali, A.: PROSOCS: a platform for programming software agents in computational logic. In Trappl, R., ed.: Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4), Vienna, Austria, Austrian Society for Cybernetic Studies (2004) 523–528
11. Rothkopf, M., Pekec, A., R.M.Harstad: Computationally manageable combinatorial auctions. Management Science **44** (1998) 1131–1147
12. ILOG S.A. France: ILOG Solver. 5.0 edn. (2003)
13. Guerri, A., Milano, M.: Exploring cp-ip based techniques for the bid evaluation in combinatorial auctions. In: LNCS - Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP2003, Springer Verlag (2003)
14. Guerri, A., Milano, M.: Learning techniques for automatic algorithm portfolio selection. In Lopez de Mantaras, R., Saitta, L., eds.: Proceedings of the 16th European Conference on Artificial Intelligence, IOS Press (2004) to appear.