# The KGP Model of Agency for Decision Making in e-Negotiation

K. Stathis[1] and F. Toni[2]

[1] School of Informatics, City University London,
kostas@soi.city.ac.uk.
[2] Department of Computing, Imperial College London,
ft@doc.ic.ac.uk.

**Abstract.** We investigate the suitabilility of the KGP (**K**nowledge, **G**oals, **P**lan) model of agency for autonomous decision making in dynamically changing environments. In particular, we illustrate how this model supports the decision making process of an agent at different levels, while the agents generates goals, plans for these goals, and selects actions to achieve the goals that it has planned for. We also exemplify the approach by illustrating how the model and a prototype implementation in the PROSOCS platform can be adopted to support e-negotiation, using a particular kind of internet auctions as a case study.

## 1 Introduction

If the ultimate goal of e-negotiation research is to provide a solid scientific foundation for the design of e-negotiation systems, we need to lay the groundwork for achieving effective principles for building and analysing such systems. Within the EU-funded project SOCS, we have developed a model for autonomous decision making agents defined via computational logic which is used to define the internal organisation, reasoning and mutual interactions of agents [12, 4]. These agents are *autonomous*, *intelligent* entities capable of operating in environments which are *open* and highly *dynamic*. The model is called $KGP$ since agents' internal state consists of a knowledge base ($K$), from which they reason, goals ($G$) that they need to achieve, and plans ($P$) for their goals, consisting of actions that may be physical, sensing or communicative. Agents pursue their goals while being alert to the environment and adapt their goals and plan to any changes that they perceive.

The KGP model incorporates and integrates various kinds of decision making, concerning in particular the goals adopted by the agents, the plans adopted to achieve these goals, the actions selected for execution at any particular time, the goals selected for plan introduction at any particular time, as well as various high-level decisions, for example when to be interrupted to observe changes in the environment, when to execute actions or introduce plans, etc. These decisions are taken while taking into account the evolution of the environment. Most of these decisions need to be taken in a timely manner.

As the rise of the internet and electronic commerce continues, dynamic automated markets and negotiation will be an increasingly important domain for agents. In general, agents may negotiate to obtain resources that they are lacking but that are necessary to carry out their plans. In this setting, negotiation is used to solve problems of resource re-allocation and sharing.

The paper is organised as follows. In section 2 we summarise the main features of the KGP model and we describe the prototype implementation of KGP agents in the PROSOCS platform [23]. In section 3 we outline the main idea underlying using KGP agents for e-negotiation in general, and auctions in particular. Section 4 concludes.

## 2 KGP agents: recap

### 2.1 KGP model

Here we briefly summarise the KGP model for agents. Formal details can be found in [12, 4]. This model relies upon

- an *internal (or mental) state*,
- a set of *reasoning capabilities*, in particular supporting planning, temporal reasoning, reactivity and goal decision,
- a *Sensing capability*, linking the agent to its environment, by allowing it to observe whether properties hold or do not hold, and that other agents execute actions,
- a set of *transition rules*, defining how the state of the agent changes, and defined in terms of the above capabilities,
- a set of *selection functions*, to provide appropriate inputs to the transitions,
- a *cycle theory*, for deciding which transitions should be applied when, and defined using the selection functions.

**Internal state.** This is a tuple $\langle KB, Goals, Plan, TCS \rangle$, where:

- $KB$ is the knowledge base of the agent, and describes what the agent knows (or believes) of itself and of the environment. $KB$ consists of various modules supporting the different reasoning capabilities of agents, including
  - $KB_{plan}$, for Planning,
  - $KB_{react}$, for Reactivity, and
  - $KB_0$, for holding the (dynamic) knowledge of the agent about the external world in which it is situated.
- *Goals* is the set of properties that the agent wants to achieve, each one explicitly time-stamped by a time variable. Goals may also be equipped with a temporal constraint (belonging to $TCS$) bounding the time variable and defining when the goals are expected to hold.
- *Plan* is a set of actions scheduled in order to satisfy goals. Each is explicitly time-stamped by a time variable and possibly equipped with a temporal constraint, similarly to *Goals*, but defining when the action should be executed. Actions are partially ordered, via their temporal constraints. Each action is also equipped with the preconditions for its successful execution.

– $TCS$ is a set of constraint atoms (referred to as *temporal constraints*) in some given underlying constraint language with respect to some structure equipped with a notion of *Constraint Satisfaction*. We assume that the constraint predicates include $<, \leq, >, \leq, =, \neq$. These constraints bind the time of goals in $Goals$ and actions in $Plan$. For example, they may specify a time window over which the time of an action can be instantiated, at execution time.

Goals and actions are uniquely identified by their associated time variable, which is implicitly existentially quantified within the overall state.

To aid revision and partial planning, $Goals$ and $Plan$ form a *tree* [3]. The tree is given implicitly by associating with each goal and action its parent. *Top-level* goals and actions are children of the root of the tree, which is chosen from outside the language of actions and goals in the state.

**Reasoning capabilities.** These include:

– Planning, which generates a partial plan for any given set of goals, if one exists in the overall state. These plans consist of (temporally constrained) sub-goals and actions.
– Reactivity, which reacts to perceived changes in the environment, by replacing (some) goals in $Goals$ and actions in $Plan$ with (possibly temporally constrained) goals and actions.
– Goal Decision, which revises the top-most level goals of the agent, adapting the agent's state to changes in its own preferences and in the environment.
– Temporal Reasoning, which reasons about the evolving environment, and makes predictions about properties (fluents) holding in the environment, based on the partial information the agent acquires.

Planning, Reactivity and Temporal reasoning are modelled via Abductive Logic programming (ALP). Goal Decision is modelled via Logic Programming with Priorities (LPP).

**Transitions.** The state of an agent evolves by applying transition rules, which employ capabilities and the constraint satisfaction. The transitions are:

– *Goal Introduction (GI)*, changing the top-level $Goals$, and using the Goal Decision capability.
– *Plan Introduction (PI)*, changing $Goals$ and $Plan$, and using the Planning capability.

[3] In the full model we actually have two trees, the first containing *non-reactive* goals and actions, the second containing *reactive* goals and actions. All the top-level non-reactive goals are either assigned to the agent by its designer at birth, or they are determined by the Goal Decision capability, via the GI transition (see below). All the top-level reactive goals and actions are determined by the Reactivity capability, via the RE transition (see below). Here for simplicity we overlook the distinction amongst the two trees.

- *Reactivity (RE)*, changing *Goals* and *Plan*, and using the Reactivity capability.
- *Sensing Introduction (SI)*, changing *Plan* by introducing new sensing actions for checking the preconditions of actions already in *Plan*, and using the Sensing capability.
- *Passive Observation Introduction (POI)*, changing $KB_0$ by introducing unsolicited information coming from the environment, and using the Sensing capability.
- *Active Observation Introduction (AOI)*, changing $KB_0$ by introducing the outcome of (actively sought) sensing actions, and using the Sensing capability.
- *Action Execution (AE)*, executing all types of actions, and thus changing $KB_0$.
- *State Revision (SR)*, revising *Goals* and *Plan*, and using the Temporal Reasoning capability and Constraint Satisfaction.

The effect of transitions is dependent on the concrete time of their application.

**Selection functions.** Inputs to (some of the) transitions is given via selection functions, taking the current state $S$ and time $\tau$ as input:

- *action selection function*, returning the set of actions in to be executed by AE;
- *goal selection function*, returning the set of goals to be planned for by PI;
- *fluent selection function*, returning the set of properties to be sensed by AOI;
- *precondition selection function*, returning the set of preconditions of actions for which sensing actions are to be introduced by SI.

**Cycle Theories** The behaviour of agents results from the application of transitions in sequences, repeatedly changing the state of the agent. These sequences are not fixed a priori, as in conventional agent architectures, but are determined dynamically by reasoning with declarative cycle theories, giving a form of flexible control.

The role of the cycle theory is to dynamically control the sequence of the internal transitions that the agent applies in its "life". It regulates these "narratives of transitions" according to certain requirements that the designer of the agent would like to impose on the operation of the agent, but still allowing the possibility that any (or a number of) sequences of transitions can actually apply in the "life" of an agent. Thus, reasoning with cycle theories to decide, at any stage, the next transition can be seen as a sophisticated form or decision making, taking into account the time at which the decision is taken, the decision taken earlier on, the current state of the agent, and, in particular, any constraints on the time of goals and actions, including the ordering imposed upon them, the current top-level goals, any plans already decided for them and their sub-goals,

and any changes in the environment. By means of reasoning with cycle theories, decisions making also depends upon agents' "personalities" or "behavioural profiles".

Cycle theories are given in the framework of LPP, also used for the Goal Decision capability. More details on cycle theories and how they can provide control for agents can be found in [13].

### 2.2 PROSOCS: Implementing KGP agents

To realise the KGP model we have developed PROSOCS [23], a platform which allows us to deploy and test the functionality of KGP agents via the SOCSiC (standing for SOCS individual Computee) component of PROSOCS. Deployment of KGP agents using SOCSiC is based on an *agent template* whose design [22] builds upon previous work in multi–agent systems, in particular, the *head/body* metaphor described by [25] and [8], and the *mind/body* architecture introduced by [3] and more recently used by [9].

In the mind part of a PROSOCS agent, the ALP-based components of the KGP model are implemented in CIFF [6, 7] and the LPP-based components of the KGP model are implemented in GORGIAS [1]. Overall, we build the mind using SICStus Prolog [20] and the bidirectional Java-Prolog interface Jasper it provides; Jasper is used by the body to exchange information with the mind.

To implement the body of the agent we use Java *on top* of the Peer-to-Peer JXTA Project [26]. JXTA is suitable for the low-level functionality of a PROSOCS agent, such as interaction with the environment, and is provided in the form of an API (Application Programming Interface). By importing this API when we instantiate specific PROSOCS agents, we enable such agents to discover bodies of other PROSOCS agents (using JXTA's peer discovery protocols facilities for dynamic discovery in a GC network) as well as communicate with other agents (using JXTA's facilities for message transport and structuring via a pipe binding and resolver protocols).

To allow the reuse of functionality from other components that are not necessarily agents, 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 agents 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 [15].

## 3 E-Negotiation

Negotiation has become an important research area in distributed systems [18] and has recently become an important interaction mechanism for multi-agent systems [11]. Agents need to negotiate because they often need to operate in environments with limited resource availability. For example, one-to-many negotiation is used for auctions, where participants reach an agreement on the cost of the items on sale. One-to-one negotiation is used, for instance, for task reallocation [19] and for resource reallocation [17], where the limited resources may be time, the computational resources of agents, or physical resources needed to carry out some tasks. In this section we demonstrate how the decision making ability of KGP agents can support one-to-many negotiation of the type one finds in auctions.

### 3.1 E-Negotiation in Auctions

An auction describes a process where two or more parties negotiate on the values of goods until an acceptable agreement is reached for the exchange of these goods. The value of goods on which the negotiation is based on is usually price (but any other attribute of the goods e.g. quality) can also be used.

Typically, auctions have two types of participants: *auctioneer* and (two or more) *bidders*. Bidders aim at buying (or selling, depending on the application) goods from one another. To do so, they place bids on these items. The auctioneer decides the winning bid(s), which determine the allocation of the goods amongst the bidders.

Different auction types exists, regulating differently the interaction mechanism amongst auctioneer and bidders and their decision making mechanism. E.g., in *single unit reverse auctions*, the objective of bidders is to obtain the goods at the lowest price, and the objective of the auctioneer in taking this decision is to maximise its profit (or the profit of the parties the auctioneer is representing). In this kind of auctions, bidders and auctioneer are self-interested and do not collaborate nor negotiate with one another.

The information exchanged during auctions and its format depend on the type of auction. In general, we have the auctioneer proposal, namely the set of items (goods or services) to be sold, with constraints, which may be temporal or setting minimum/maximum prices.

A bid in an auction is defined by the bidder's name, the name of the items to be purchased, any temporal constraints, and the value of the bid, possibly defined by constraints. The auctioneer answer specify the winning bids.

Auctions can also be used among cooperative agents. 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 (maximum price and temporal 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 customer's constraints.

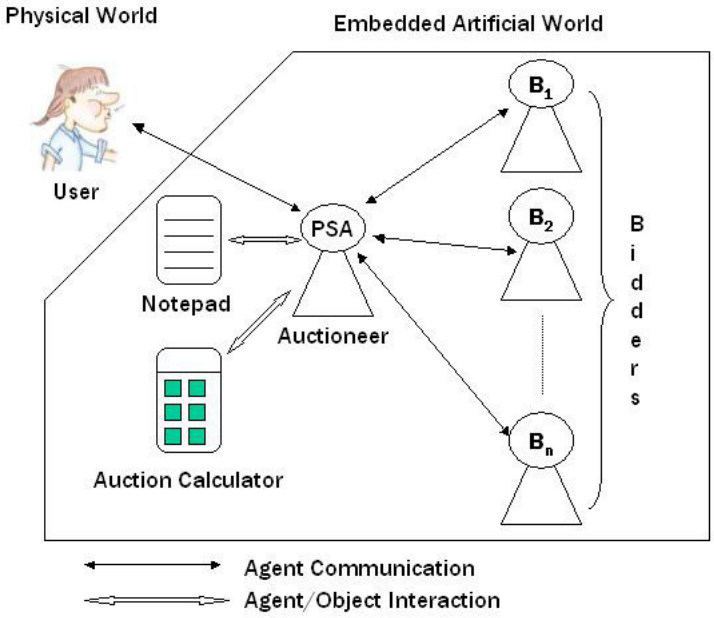


**Fig. 1.** A combinatorial auction organised as a set of agents and objects in PROSOCS.

In conventional auctions, auctioneer and bidders are human. In electronic (e.g. internet-based) auctions, software agents (as well as humans) can participate in auctions on behalf of end-users [27]. We present below an example of distributed e-negotiation where people and software agents interact to exchange goods via a network of computers.

We have experimented with the development of a combinatorial auction [16] in order to experiment and test [2], amongst other things, with the PROSOCS platform and distributed decision making using the KGP model. In this context we have experimented with an ambient intelligence application [24] (whose scenario is described in [21]) where people are represented in electronic environments by *personal service agents* (PSAs)[5]. The specific scenario illustrates how a PSA facilitates a person's travel by creating a combinatorial auction to meet that person's travel requirements for reaching an airport. Fig. 1 shows the PSA initiates a reverse auction where travel agents bid to meet the person's requirements.

More specifically, the $PSA$ becomes the auctioneer of a reverse auction where a set of real or artificial travel agents ($B_1$, $B_2$,...,$B_n$) are bidding to try matching the price required by the user. The $PSA$ creates a PROSOCS object, called the

*Notepad*, which allows the $PSA$ to make notes of the bids offered by the bidders, and giving an example of how an agent can annotate the external environment to its advantage. The PSA also uses an *auction calculator* to solve the constraints of the bids. This object shows how to incorporate the ILOG solver [10] as an object into our system, without having to treat it as an agent (which arguably it is not). Both the notepad and the auction calculator involve the auctioneer agent to interact using the ideas of objects, described in the previous section.

One important advantage of using PROSOCS objects is that, in principle, a PROSOCS agent could use different auction calculators, possibly using different constraint and optimisation techniques, to decide on different aspects of an auction (e.g. winner determination). Moreover, these objects could again, in principle, run on different nodes of a computer network, and thus making the overall approach more flexible and more scalable.

### 3.2 Modelling the Auctioneer

*Opening an auction.* An `Auctioneer` starts an `auction` after selecting the communicative act `openauction` to the `Bidders`. This act will be initiated by the planning capability in response to producing a plan for a goal. We represent this in $KB_{plan}$ by `initiates/3` rules of the form:

```
initiates(Act, T, auction(ID, Duration, Deadline, Bidders, Items)):-
  self(Auctioneer),
  Act=tell(Auctioneer, Bidders, openauction(Items, Duration, Deadline), ID).
```

The representation of the above rule assumes amongst other that the description of an `Act` initiating an auction contains the auction's `Duration` (i.e. a fixed number of time units for which the auction should last) as well as the auction's `Deadline` (i.e. the additional time required for the auctioneer to announce the auction's result after the auction's duration has expired).

*Collecting bids.* After opening an auction the 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. Valid bids should (i) reach the auctioneer in time, (ii) come from a bidder that has actually been invited to participate in the auction, and (iii) 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,Bidders,openauction(Items,T1,T2),ID),T0),
  self(Auctioneer),
  T0 #< T,
  T #=< T1,
  member(Bidder,Bidders),
  subset(Items, Items),
  NextAct=do(notepad, makeNote(ID,bid(Bidder,Price,Items)))
] implies
[
  assume_happens_after_once(NextAct,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). For simplicity, we will assume that any physical actions (of the form `do(_)`) are executable by the agent (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),
  self(Auctioneer),
  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 agent given in the list of winners (there is a similar rule for informing the losers; omitted here):

```
[
  self(Auctioneer),
```

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

We should stress that the implementation of $KB_{react}$ for an auctioneer agent 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).

### 3.3 Modelling Bidders

For most of our experiments we have represented concrete bids for specific sets of items that are on auction using simple reactive rules. The following reactive rule causes an agent to reply to any `openauction` act that offers a set of items including `central_station` by offering an a price of 5 monetary units for that item, and provided it is amongst the agents that have been invited to bid:

```
[
  observed(Auctioneer,Observation,T),
  Observation=tell(Auctioneer,Bidders,openauction(Items,T1,T2),ID,_),
  self(Bidder),
  member(Bidder,Bidders),
  member(central_station,Items),
  NextAct=tell(Name,Auctioneer,bid(5,[central_station]),ID)
] implies
[
  assume_happens_after_once(NextAct,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 a KGP agent can decide using the cycle theory, its plans, and goals, together with its reactivity rules. We have also tried to test these decisions in PROSOCS and provide the basic communication needs for running combinatorial auctions. However, other possible approaches for generating interesting problem instances would be to use the *Combinatorial Auction Test Suite (CATS)* [14] to automatically generate instances of combinatorial auction problems that could be adopted by bidding agents to test the overall architecture more extensively. The details of such an extension, however, is beyond the scope of this work.

## 4 Conclusions

In this paper we have summarised the KGP model of agency [12, 4] and its implementation [23]. The model incorporates and integrates a number of decision

making components, one that assists the agent to generate goals, one to allow the agent to plan for these goals, and one that allows the agent to select actions to achieve the goals that it has planned for. We have exemplified the approach by illustrating how the model and a prototype implementation in the PROSOCS platform can be adopted to support e-negotiation, using a combinatorial auction as a case study.

**Acknowledgements**

This work was funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the GC proactive initiative.

## References

1. *Gorgias User Guide:* `http://www.cs.ucy.ac.cy/~nkd/gorgias/`, 2003.
2. M. Alberti, F. Chesani, U. Endriss, M. Gavanelli, A. Guerri, E. Lamma, P. Mello, M. Milano, K. Stathis, and P. Torroni. Experiments on combinatorial auctions. Discussion Note IST32530/UNIBO/250/DN/I/a1, SOCS Consortium, June 2004.
3. J. Bell. A Planning Theory of Practical Rationality. In *Proceedings of AAAI'95 Fall Symposium on Rational Agency*, pages 1–4. AAAI Press, 1995.
4. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency for global computing: Computational model and prototype implementation. In *Global Computing 2004 Workshop*, page 342. Springer Verlag LNCS 3267, 2005.
5. K. S. E. Mamdani, J. Pitt. Connected Communities from the standpoint of Multi-agent Systems. *New Generation Computing*, 17(4):381–393, 1999.
6. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In *Proceedings JELIA04*. To appear.
7. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Abductive logic programming with CIFF: implementation and applications. In *Proceedings CILC2004, Convegno Italiano di Logica Computazionale*, 2004.
8. H. Haugeneder, D. Steiner, and F. McCabe. IMAGINE: A framework for building multi-agent systems. In S. M. Deen, editor, *Proceedings of the 1994 International Working Conference on Cooperating Knowledge Based Systems (CKBS-94)*, pages 31–64, DAKE Centre, University of Keele, UK, 1994.
9. Z. Huang, A. Eliens, , and P. de Bra. An Architecture for Web Agents. In *Proceedings of EUROMEDIA'01*. SCS, 2001.
10. The ILOG Solver. http://www.ilog.com/products/solver/. Visited 28/06/04.
11. N. R. Jennings. Automated haggling: Building artificial negotiators (invited talk). In *AISB'01 Convention, York, UK*, 2001.
12. A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proceedings ECAI2004*, 2004. To appear.
13. A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In *Proc. CLIMA V*, 2004.
14. K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76, New York, NY, USA, 2000. ACM Press.

15. W. Lu and K. Stathis. Incorporating Objects in PROSOCS Artificial Worlds. IST32530/CITY/015/DN/I/a1, June 2004.
16. N. Nisan. Bidding and allocation in combinatorial auctions. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 1–12, New York, NY, USA, 2000. ACM Press.
17. 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.
18. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994.
19. T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. Computer science, University of Massachusetts at Amherst, September 1996.
20. SICStus Prolog user manual, release 3.8.4, 2000. Swedish Institute of Computer Science.
21. K. Stathis. Location-aware SOCS: The Leaving San Vincenzo scenario. Technical Report IST32530/CITY/002/IN/PP/a1, SOCS consortium, 2002.
22. K. Stathis, C. Child, W. Lu, and G. K. Lekeas. Agents and Environments. Technical report, SOCS Consortium, 2002. IST32530/CITY/005/DN/I/a1.
23. K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In J. Müller and P. Petta, editors, *Proceedings of From Agent Theory to Agent Implementation (AT2AI-4 – EMCSR'2004 Session M)*, pages 523–528, Vienna, Austria, 2004.
24. K. Stathis and F. Toni. Ambient Intelligence using KGP Agents. In *Proceedings of the 2nd European Symposium for Ambient Intelligence*, pages 351–362, Eindhoven, November 2004. LNCS, Springer-Verlang.
25. D. E. Steiner, H. Haugeneder, and D. Mahling. Collaboration of knowledge bases via knowledge based collaboration. In S. M. Deen, editor, *CKBS-90 — Proceedings of the International Working Conference on Cooperating Knowledge Based Systems*, pages 113–133. Springer Verlag, 1991.
26. B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J. C. Hugly, and E. Pouyoul. Project JXTA-C: Enabling a web of things. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*, pages 282–287. IEEE Press, 2003.
27. 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.