An Agent Architecture for Concurrent Bilateral Negotiations

Bedour Alrayes and Kostas Stathis

Computer Science, Royal Holloway, University of London, Egham Hill,
Egham TW20 0EX, UK
{B.M.Alrayes,Kostas.Stathis}@cs.rhul.ac.uk,
URL: http://www.cs.rhul.ac.uk

Abstract. We present an architecture that makes use of symbolic decision-making to support agents participating in concurrent bilateral negotiations. The architecture is a revised version of previous work with the KGP model [23, 12], which we specialise with knowledge about the agent's self, the negotiation opponents and the environment. Our work combines the specification of domain-independent decision-making with a new protocol for concurrent negotiation that revisits the well-known *alternating offers* protocol [22]. We show how the decision-making can be specialised to represent the agent's strategies, utilities and preferences using a Prolog-like meta-program. The work prepares the ground for supporting decision-making in concurrent bilateral negotiations that is more lightweight than previous work and contributes towards a fully developed model of the architecture.

Key words: Negotiation architectures, Interaction Protocol, e-Markets

1 Introduction

Electronic marketplaces (e-marketplaces or e-markets) are typically construed as inter-organizational information systems that allow participating buyers and sellers to exchange information about prices and product offerings. The organization operating an e-marketplace is normally referred to as an intermediary, which may be a market participant - a buyer or seller, an independent third party, or a multi-firm consortium [6].

E-marketplaces provide an electronic, or online, method to facilitate transactions between buyers and sellers, and will typically support all of the steps in the entire order fulfilment process. In recent years, we have witnessed a huge number of successful e-marketplaces on the Web, such as E-Bay, Amazon, or Gumtree. E-markets of this kind have become popular because they support mechanisms such as advertising, buying/selling, and paying for goods online, thus providing an efficient and convenient way to perform commercial activities on the Web.

However, one problem with the existing e-marketplaces is that a market participant must repeatedly be online in order to follow the progress of an activity such as buying a product, especially if such a product requires bargaining. In

addition, the duration of some market mechanisms (e.g. auctions) can be long, so e-marketplace activities can often become tedious and tiring for a participant. What is worse, buyers and sellers come and go, they have different goals and preferences. Thus pursuing best deals in an e-marketplace requires from participants to engage in multiple, possibly conflicting, activities at the same time, sometimes in different e-marketplaces.

To deal with the dynamic and complex environment that e-marketplaces give rise to, we follow a multi-agent systems approach [24] whereby agents negotiate bilaterally with other agents on behalf of their users. The idea here is that agents can satisfy user preferences by isolating emotions and feelings when they negotiate [11] as well as save users time. In this context, the issue becomes how to build models that support such agents to engage in multiple and dynamic market environments concurrently, take accountable decisions in these environments, and produce results that are approved by the users they represent.

We revisit previous work with the KGP model [23] applied to decision making [12] to equip it for decision-making in multiple and concurrent negotiations as required by an agent's participation in multiple markets. We represent markets explicitly via the introduction of an environment model, we profile other participating agents by introducing an opponents' model, and we also keep a representation of self [5]; none of these models were available in [23]. The focus of this paper is to integrate the models of the environment, the opponents and self in a revised agent-architecture to aid the symbolic representation of decision making in negotiations. Our longer term goal involves how to support an agent wishing to acquire a product decide first in which market to buy it and then what offers to make to a seller at different stages of the negotiation.

The paper is organised as follows. Section 2 is a brief overview of negotiation and decision-making. In Section 3 we present our agent architecture. Section 4 is a discussion of the protocol. The skeleton of decision making is discussed in Section 5. We conclude with Section 6.

2 Negotiation and Decision Making

Negotiation is the process of agents communicating with one another in order to reach agreements or solve conflicts on matters of common interest. As in [10], the negotiation process is like bargaining by which a joint decision is made by two parties. The parties first verbalise contradictory demands and then move towards agreements. Negotiation of this kind has recently gained a lot of attention in multi-agent systems, especially in e-commerce [23] but also in other application areas too, for instance, service selection and composition [7], virtual organisations [18], supply chain [13, 14, 15], service-oriented [19] and cloud computing [9].

To enable multiple agents participate in the negotiation process we assume a negotiation protocol that is often construed as the rules of encounter [21] between the participants. Such a protocol states who can say what, to whom and at what time. Given a protocol, an agent strategy then defines the model

that the individual participants apply to act in line with the protocol in order to achieve their negotiation objectives. However, in our setting it is not possible to pre-compute an optimal negotiation strategy at design time, as in classical mechanism design. Rather the agents need to adopt a heuristic and satisfying approach for their strategy.

As argued in [10], when the assumptions of classical mechanism design do not apply, we need to seek to develop a distributed approach where solutions are sought when agents do not know the other player's preferences for negotiation outcomes, their reservation values, or any other resource constraints. Also, agents are computationally bounded in both time and resources. Moreover, agents may engage in a multi-criteria decision problem whose interactions follow the rules of an alternating sequential protocol in which the agents take turns to make offers and counter offers [22]. This protocol terminates when the agents agree or when one of them withdraws from the negotiation. In the most complex form, the negotiation is conducted concurrently via a number of bilateral negotiations rather than sequentially [10]. Here the agent may be required to apply different strategies for different opponents in order to achieve its goals with the advantage of overcoming slowness of human negotiation.

Unlike [10], however, we want to develop a heuristic approach that combines efficient reasoning techniques and approximate decision models to develop negotiating agents. We look at an agent architecture that interprets the negotiation process as consisting of three main stages: initial, middle and final. In the initial phase, planning and preparation for the negotiation is taking place. The goal of negotiation is set by the agent's user in the initial phase and consists of specifying the required product and its properties, such as price range and deadline. In the middle phase, the actual negotiation takes place, which includes deciding the agent's offers, evaluating the opponent offers, modelling the environment and/or opponents' behaviour changes, and measuring the performance of the negotiating position. In this phase, a number of questions arise: what changes should the buyer take into consideration in order to maximise its negotiation outcome? How it will behave in the light of these changes? In the final phase, the negotiation agreement is implemented.

3 Agent Architecture

To specify our problem, a negotiating agent architecture must be presented. In brief, there are two types of architecture. The first is an agent architecture that provides a clear demonstration of the agent structure without declaring how the agent behaves during negotiation because the agents are autonomous in nature [26]. The second is an architecture that combines both agent structure and behaviour [5]. The agent's behaviour addresses the process of deciding what actions to perform at each stage.

In Figure 1, we develop an agent architecture with four components: Physical Capabilities, Domain Knowledge, Current State, and Cognitive Capabilities. The following is a brief description for each component:

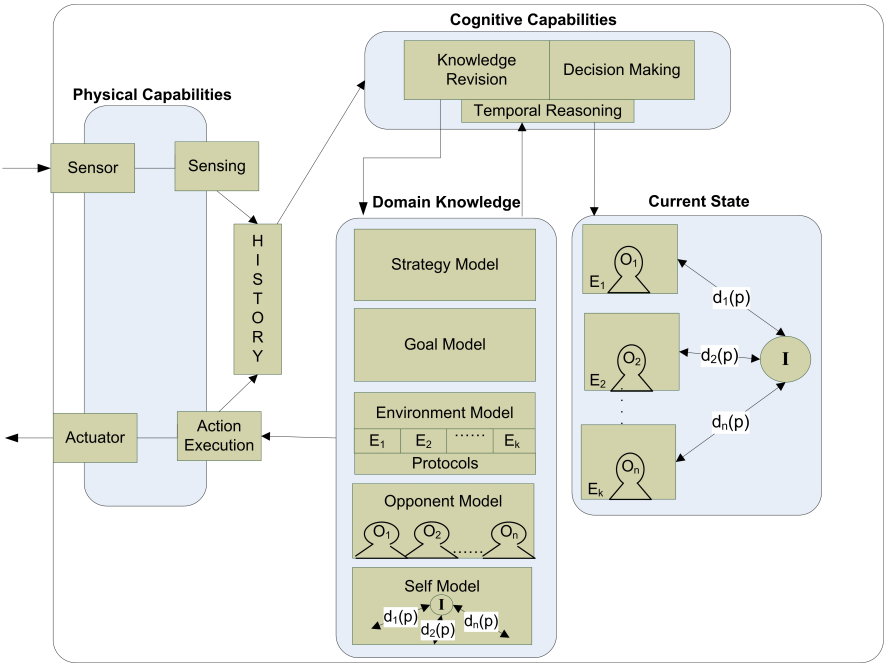

Fig. 1. Architecture of negotiating agent I that is interacting concurrently in different sub-environments $E_1, E_2, \ldots, E_k$ with opponents $O_1, O_2, \ldots O_n$.

The *Domain Knowledge* represents generic and dynamically changing knowledge about the negotiating application at hand. It consists of five subcomponents: a *Strategy Model* detailing how the agent selects actions in the application domain, a *Goal Model* outlining which goals the agent can achieve and how they can be achieved using the strategies, an *Environment Model* representing knowledge about classes of different types of environments and the protocols they require, an *Opponent Model* detailing classes of different opponents and a *Self Model* representing information about the agent itself and its preferences.

The *Current State* contains instances of the on-going negotiations in terms of the participants for each negotiation, the protocol used for each negotiation, and the progress made in these so far.

The *Physical Capabilities* situate the agent in an environment by connecting the agent's sensors and effectors to its internal state. It consists of two subcomponents: a *Sensing capability* that captures environment information via the sensors and an *Action Execution* capability that performs the actions in the environment via the actuators. Both capabilities operate on the *History* of the agent, another component that holds the experience of the agent represented as a series of events that have happened, either in the form of observations (events from the Sensing capability) or actions (events from the Action Execution capability).

The *Cognitive Capabilities* allow the agent to reason about the negotiation and take decisions. This component consists of three subcomponents: *Deci-*

sion Making is responsible for evaluating the *Current State*, and uses *Domain Knowledge* to decide what to do next, *Knowledge Revision* updates the *Domain Knowledge* component either through simple revisions or through learning over time, and *Temporal Reasoning* supporting changes in the *Domain Knowledge* due to events happening in the *History* of the agent.

Finally, the *Control* component details how the capabilities are being invoked and under which circumstances, represented by the outer container and the arrows in Figure 1. The typical control cycle is an instance of the more general framework described in [16]. It involves the agent sensing an event from the environment, updates its history, which in turn causes a revision of the knowledge and the current state, then the agent makes a decision to act, and the action is executed in the environment and recorded in the history.

4 Concurrent Alternating Offers Protocol

We study a practical concurrent negotiation setting where a buyer agent engages in multiple bilateral negotiations in order to acquire a good. In doing so we experimented with the alternating offers protocol [22] due to its simplicity and wide use. As shown in Figure 2, in this protocol a buyer and a seller take turns to negotiate with possible actions: *offer, accept* and *reject*. The negotiation terminates either because the agreement is reached (buyer/seller accept) or the negotiation has terminated (buyer/seller reach their deadlines).

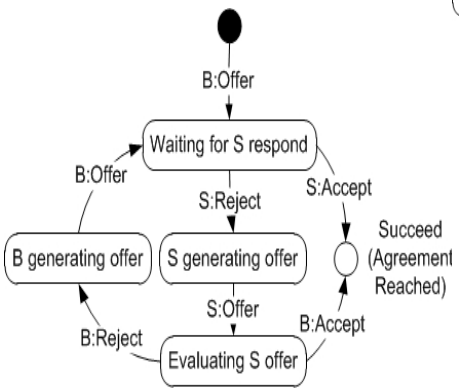

Fig. 2. Alternating Offer State Diagram

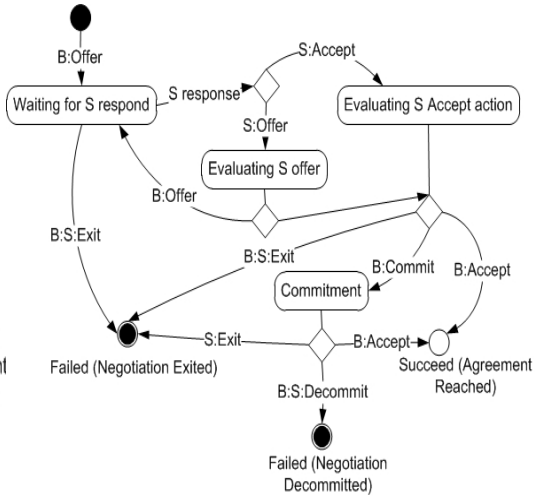

Fig. 3. Concurrent Alternating Offers State Diagram

However, the alternating offers protocol is not sufficient to handle the complexity of concurrent negotiation due its limited actions. In Figure 3, we extended

the protocol allowing for: *offer, accept, commit, de-commit* and *exit*. Commit provides the buyer with opportunity to hold a preferred offer for a certain time until they find a better one, reflecting what happens in practice. If an agent (buyer/seller) finds a better agreement then it can de-commit from committed offer(s). If an agent reaches its deadline, it will accept one committed offer, thus an agreement is reached. The agent may maintain a number of committed offers at the same time. However, the de-commitment process is subject to a penalty that asserts fairness by avoiding unnecessary de-commitments.

There have been two previous attempts to extend the alternating offers protocol. In [2], confirm action is presented, which comes after an accept; a buyer cannot de-commit from confirmed offers. In [25], a concurrent protocol is proposed but does not distinguish between the commitment and agreement actions (both are represented by confirm). In our protocol, we have commit and accept instead. In addition, de-commitment in [25] happens after the agreement has been made, which can be inefficient in real time applications. In contrast, the de-commitment in our protocol occurs before the agreement is finalized.

5 The Skeleton of Decision Making

To test our negotiation protocol we study the development of agent strategies with emphasis on the strategy of a buyer agent in a multiple market setting. Existing literature (e.g. [3, 20, 25]) provides answers only on when to offer or accept and what to offer, since the protocol used does not allow for concurrency. In addition, offers are computed using an opponent's model but ignore both the environmental and self models [20, 25], while some work assumes complete and certain information about the negotiation environment and the opponents [20]. In most approaches the agent strategy is an isolated component without clear illustration of how the model is designed and how different component interact with each other [3, 20, 25]. Consequently, the produced offers are inadequate for our purposes. To overcome these shortcomings, in Section 5.1 we developed a heuristic negotiation strategy and demonstrate an example run in Section 5.2.

5.1 Representation Framework

We represent the state of an agent during negotiation as a temporal logic program. We specify the rules of such a program in Prolog, which we assume the reader is familiar with. Prolog uses the convention that a constant is an identifier starting with a lower-case letter, while a variable is an identifier starting with a upper-case letter. A constant stands for a specific entity, and different constants stand for different entities. On the other hand, a variable can stand for any entity, and different variables can stand for the same entity. The predicate name must be a constant, while each argument can either be a constant or a variable. As Prolog program can be used to answer queries, or to achieve goals, the prompt "?-" denotes a query whose truth will be judged by the program.

The temporal part of the logic is represented using the Event Calculus (EC) [17], a logical language for reasoning about actions and their effects. We will use a dialect of the EC based on temporal variables known as multi-valued fluents due to the fact that these variables can take different values at different times. At a time T a fluent is represented in the form F=V to denote that it holds at T that F has value V [1]. For instance, to query that the initial price of specific labtop, say `laptop12`, costs `1200` at the current time `15`, we write:

```
?- holds_at(initial_price(labtop12) = 1200, 15).
```

The state of the agent will generally contain many other fluents to represent information about negotiation threads, information about other agents and knowledge about the environment.

From a representation of state as the one above, we develop a strategy as an action selection policy whose specification is defined by Prolog rules of the form:

```
select(Thread, Action, T):- Conditions [T].
```

Such rules state that given a negotiation `Thread` that the agent is engaged in at time `T`, the `Action` is selected if the `Conditions` hold at `T`. For example, the rule below shows how the agent exits a negotiation thread if the agent's deadline for the `Thread` has passed:

```
select(Thread, exit, T):-
     holds_at(self_deadline(Thread, Td)=true, T), T > Td.
```

Our model allows the agent to make decisions using utilities over actions. The top-level decision process finds the most promising `Thread` in the current time `T`, generates all the `Options` that the agent is capable of performing at this time, evaluates the options and returns the one with the highest utility:

```
decide(in(Thread, Action), T):-
    promising(Thread, T),
    generate(Thread, Options, T),
    evaluate(Thread, Options, Evaluation, T),
    return(Thread, Evaluation, Action, T).
```

The way we measure that a `Thread` is promising at time `T` is by measuring the eagerness of the user for obtaining a negotiation item and how well the opponent has behaved during negotiation. Lack of space does not allow us to explain in detail how these concepts are formulated, however, we plan to elaborate them in our future work. We discuss next how we generate all the options within the most promising `Thread` as shown below:

```
generate(Thread, Options, T):-
    findall(Option, select(Thread, Option, T), Options).
```

`findall/3` is a Prolog primitive that finds every `Option` that can be selected by the agent in the `Thread` and storing it in a list of `Options` to be returned.

Once generated, the `Options` are evaluated by the program:

```
evaluate(_, [], [], _).
evaluate(Thread, [Option|Options], [(Option,Util)|Pairs], T):-
          utility(Thread, Option, Util, T),
          evaluate(Thread, Options, Pairs, T).
```

The first clause defines the termination conditions of the evaluation. The underscore sign '_' indicates that we do not care about a variable. The second clause of the evaluation defines the main procedure that computes for each `Option` a utility `Util`. The result returned is a list of pairs `(Option, Util)`. A utility `Util` is obtained via a dynamic domain-specific function evaluated at run-time. For example, we calculate the utility of an offer act as:

```
utility(Thread, offer(Price), Utility, T):-
    holds_at(product(Thread, Product)=true, T),
    holds_at(initial_price(Thread, Product)=IP, T),
    holds_at(reservation_price(Thread, Product)=RP, T),
    Utility is (RP-Prics)/(RP-IP).
```

The preference process of the agent then returns the option with highest utility, once it orders the options in increasing order:

```
return(Thread, Evaluation, Action, T):-
    order(Evaluation, Ordered),
    choice(Thread, Ordered, Action, T).

choice(_, [(Option,_)|_], Option, _).
```

Other definitions for `choice/4` are possible, to deal with options having the same utility, but discussion on this issue is beyond the scope of the paper.

Table 1. Observations, deliberations and decisions of *buyer* during a negotiation.

Time	Observation	Deliberation	Decision
t1	$seller_1$ & $seller_2$.	Make offer to all visible sellers.	Offer(700)
t2	$seller_1$ offers 950.		
t3	$seller_2$ offers 920.		
t4		Active threads: $seller_1$, $seller_2$. Options for selected thread $seller_1$: [offer(750), exit, commit, accept]. Utilities for $seller_1$: [(offer(750), 0.8), (exit,0.5), (commit,0.6),(accept,0.1)]. Highest Utility Action: offer(750).	Offer(750) to $seller_1$.

5.2 Example Run

We have prototyped a small agent market in the GOLEM platform [8], which acts as a proof-of-concept of our architecture. GOLEM supports experimenting

with Prolog agents like ours and endows them with additional features such as agent communication and interoperability. We have focused on how a user specifies an item to be bought by a *buyer* agent. Table 1 illustrates how the *buyer* agent interacts with *sellers* in the market in order to purchase a laptop. The scenario is presented from a buyer's perspective and assumes the user has specified that (a) the laptop's price to be in the range [700, 900] and (b) the maximum negotiation duration to be 5 minutes. At time t1, *buyer* is negotiating concurrently with $seller_1$ and $seller_2$. At time t2, $seller_1$ sends a counteroffer to *buyer* while at time t3, $seller_2$ sends back an offer to *buyer*. In t4, *buyer* will follow the skeleton strategy in section 5.1 to select the best action.

6 Conclusion and Future Work

We have outlined the components of an agent architecture to support the deployment of negotiating agents participating in negotiations of multiple electronic markets. The architecture is influenced from previous work with the KGP model [23] revisited here to contain an explicit representation of environment, opponent, and self models for the purposes of negotiation. The newly proposed model also provides a light version of the cognitive capabilities of the agent to aid decision making in multiple concurrent negotiations for which we have provided a protocol. We have also described the skeleton of the decision making capability and we have shown how it links to utilities and preferences.

Future work involves developing the heuristic negotiation strategy that involves the environment, opponent and self models to decide the market to negotiate in, what, how, and when to offer, and whether to accept or exit. Finally, we will also develop a test bed [4] to compare our agent's performance in negotiation with other negotiating agents (e.g. [3, 20, 25])).

References

1. A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. In *ACM Trans. Comput. Logic*, 10(1):1-42, 2009.
2. B. An, N. Gatti, and V. Lesser. Extending alternating-offers bargaining in one-to-many and many-to-many settings. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 423–426, Washington, DC, USA, 2009.
3. B. An, V. Lesser, and K. M. Sim. Strategic agents for multi-resource negotiation. *Journal of Autonomous Agents and Multi-Agent Systems*, 2011.
4. S. Bromuri and K.Stathis. Distributed agent environments in the Ambient Event Calculus. *DEBS 2009*, ACM. NY, USA, 2009.
5. R. Ashri, I. Rahwan, and M. Luck. Architectures for negotiating agents. In *Proceedings of the 3rd Central and Eastern European conference on Multi-agent systems*, CEEMAS'03, pages 136–146, Berlin, Heidelberg, 2003.
6. J. Y. Bakos. A strategic analysis of electronic marketplaces. *MIS Quarterly*, 15(3):pp. 295–310, 1991.

7. S. Bromuri, V. Urovi, M. Morge, K. Stathis, and F. Toni. A multi-agent system for service discovery, selection and negotiation. In *AAMAS 2009*, 1395–1396, 2009.
8. S. Bromuri and K. Stathis. Situating Cognitive Agents in GOLEM. *EEMMAS'07*, 115-134, Springer, 2007.
9. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.
10. P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142:205–237, 2002.
11. R. Fisher and W. L. Ury. *Getting to Yes: Negotiating Agreement Without Giving In.* Penguin (Non-Classics), 2nd edition edition, Dec. 1991.
12. J. Forth, K. Stathis, and F. Toni. Decision making with a KGP agent system. *Journal of Decision Systems*, 15(2-3):241–266, 2006.
13. J.E. Hernández, J. Mula, R. Poler, and A.C. Lyons. Collaborative Planning in Multi-tier Supply Chains Supported by a Negotiation-Based Mechanism and Multi-agent System. *Group Decision and Negotiation*, 1-35, 2013.
14. J.E. Hernández, A.C. Lyons, J. Mula, R. Poler, and H.S. Ismail. Supporting the collaborative decision-making process in an automotive supply chain with a multi-agent system. *Production Planning & Control*, 2013.
15. J.E. Hernández, A.C. Lyons, R. Poler, J. Mula, and R. Goncalves. A reference architecture for the collaborative planning modelling process in multi-tier supply chain networks: a Zachman-based approach. *Production Planning & Control*, 2013.
16. A. Kakas. P. Mancarella, F. Sadri, K. Stathis and F. Toni. Declarative Agent Control. *CLIMA 2004*, pages 96–110, Springer, Lecture Notes in Computer Science, vol 3487, 2005.
17. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4:67–95, January 1986.
18. J. McGinnis, K. Stathis, and F. Toni. A formal framework of virtual organisations as agent societies. In *FAVO*, pages 1–14, 2009.
19. M. Morge, J. McGinnis, S. Bromuri, F. Toni, P. Mancarella, K. Stathis. Toward a modular architecture of argumentative agents to compose services. In Proc. of *EUMAS*, 2007.
20. T. Nguyen and N. R. Jennings. Coordinating multiple concurrent negotiations. In *3rd International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1064–1071, 2004.
21. J. S. Rosenschein and G. Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers.* MIT Press, Cambridge, MA, USA, 1994.
22. A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):pp. 97–109.
23. K. Stathis and F. Toni. The KGP model of agency for decision making in e-negotiation. In *Joint-Workshop on Decision Support Systems, Experimental Economics E-Participation*, June 2005.
24. P. West, D. Ariely, S. Bellman, E. Bradlow, J. Huber, E. Johnson, B. Kahn, J. Little, and D. Schkade. Agents to the rescue? *Marketing Letters*, 10(3):285–300, 1999.
25. C. R. Williams, V. Robu, E. H. Gerding, and N. R. Jennings. Negotiating concurrently with unknown opponents in complex, real-time domains. In *20th European Conference on Artificial Intelligence*, volume 242, pages 834–839, August 2012.
26. M. Witkowski, K. Stathis. A Dialectic Architecture for Computational Autonomy. *Agents and Computational Autonomy 2003.* pp 261-274. Springer, 2003.