An adaptive agent architecture for automated negotiation

Yong Yuan， Yong-quan Liang
College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao, 266510, China
elisen66@yahoo.com.cn

Abstract. Adaptability is regarded as an essential capability for negotiating agent to deal with dynamic environments. In current literatures, however, little attention has been paid on improving this capability. This paper presents a new agent architecture to support adaptive negotiation, and discusses the mechanisms of protocol parsing and strategy generation in detail. An experiment is carried out on JADE platform to test the feasibility of this architecture, and the results validate that agents can adaptively understand protocols and respond with optimal strategies.

1 Introduction

With the amount of online commercial transactions increasing at a spectacular rate, automated negotiation has become a key technique in developing intelligent and flexible agent-mediated e-commerce (AMEC) systems [1]. Broadly speaking, current research concentrates mainly on the autonomy and intelligence of negotiating agents, aiming at empowering agents with capability of solving particular negotiating scenarios with predefined protocols and specifically tailored strategies. In such open and dynamic environments as Internet, however, adaptability is also essential. An adaptive agent should have such capacities as (1) dealing with the interoperability problem caused by heterogeneous knowledge sources; (2) understanding various user-defined protocols and adapting to their dynamic changes; (3) generating optimal strategies for arbitrary protocols without human intervention. Currently, the first capacity can be obtained from the mapping of domain ontologies [2]. However, little attention has so far been paid on the latter two problems.

The absence of adaptability hampers the real application of AMEC paradigm to a large extent. This can be embodied from two facts: Firstly, negotiation protocols are usually hard-coded implicitly in agents' code. As a result, agents can understand only predefined protocols and any modification to the protocol implies that agents must be taken offline to be reprogrammed. Secondly, it is hard for agents to respond an arbitrary protocol with optimal strategies. Two strategy-reasoning approaches are currently in use. One is deliberative agents' complex reasoning based on the logical argumentation and the other is reactive agents' direct response according to the past experience or predefined heuristic functions. Each has its drawbacks. The former is difficult to be realized for its high complexity while the latter usually generates sub-optimal strategies. Therefore, in order to improve adaptability, these two problems must be solved.

In this paper, an adaptive architecture of negotiating agents is put forward and tested on JADE platform. Our solution borrows some ideas from semantic web and co-evolutionary computation. In this architecture, negotiation protocol is annotated with semantic in terms of an explicit and shared protocol ontology so that it can be understandable to negotiating agent and separated from the agent kernel; Meanwhile, co-evolution is used to help the agent to generate optimal strategies adaptively.

The remainder of this paper is organized as follows: Section 2 introduces briefly the protocol ontology and co-evolution; Section 3 describes the agent architecture. Section 4 and 5 discuss the working principles of the protocol parser and the strategy generator respectively. A preliminary experiment is presented in section 6 based on JADE platform. Finally, section 7 concludes.

2 Theoretical Underpinnings

2.1 Protocol Ontology

Negotiation protocol is a set of rules that govern the interaction. To ensure an adaptive negotiation, a protocol should be: (1) Public. Unlike strategies are usually private, protocol must be public and explicit; (2) Sharable. Protocol should provide a common understanding to all agents; (3) Flexible. Protocol should be capable of being defined and modified dynamically without agents' code reprogrammed.

The emergence of ontology technique provides a perfect solution to satisfy above requirements. Ontology is a hot topic in semantic web and can be defined as a formal and explicit specification of a shared conceptualization [3]. In fact, each negotiating agent has its private ontology of negotiation protocol [4]. Thus their understandings of a same protocol may differ due to the structural or semantic heterogeneities between these ontologies. In this case, a common ontology is necessary, to which a mapping should be established to the private ontologies to ensure a common understanding. In dynamic scenarios, various concrete protocols can be instantialized and modified in terms of the common ontology. In this way, a negotiation protocol can be considered as a plug-in and consequently separated from the agent kernel.

2.2 Co-evolution

Co-evolution is a class of multi-population evolutionary algorithms dedicated to solve the dynamic self-adaptability problems in complex systems. According to the interspecies relationship, it can be classified into cooperative and competitive ones, which respectively imitate symbiosis and parasitism [5]. Currently, competitive co-evolution is often used to search for the optimal agent behaviors in strategic contexts. Its basic idea is to encode the strategy spaces as co-evolving strategy populations one for each competing agent. The co-evolution process starts with randomly generated populations and updates these strategies in successive iterations. Genetic operations are limited within the respective populations while the fitness evaluation lies on the direct interactions among strategies of different populations. Thus, these populations compete with and adapt to each other to form an evolutionary "arm race", and converge respectively. The optimal or approximate optimal strategy profiles are combinations of the best strategy individuals in the last generation.

3 Negotiating Agent Architecture

Technically speaking, a behavior-driven architecture is used in our design of the agent kernel. As is shown in figure 1, a negotiation task will be encapsulated into a composite behavior based on a finite state machine (FSM). A FSM behavior consists of a finite number of states and transition rules, and each state recursively contains a sub-behavior. Other functional components (represented by rectangular blocks) of the agent kernel coordinate to facilitate the generating, updating and scheduling of this FSM behavior.

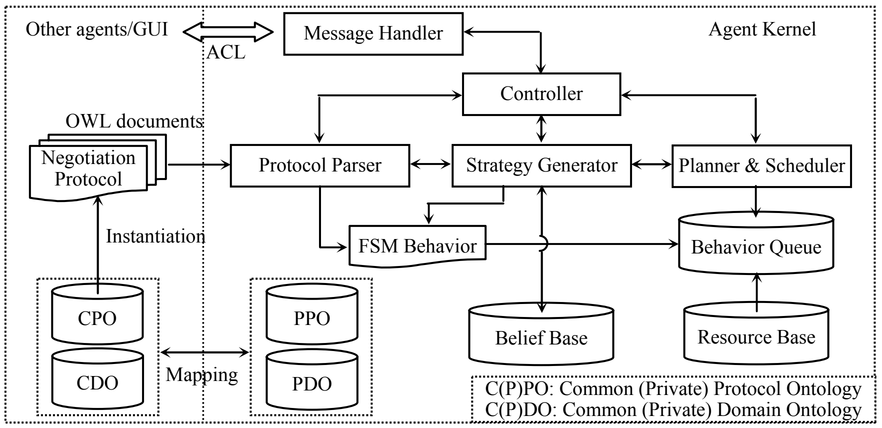


Fig.1. The architecture of an adaptive negotiating agent

As the functionalities of the above components are explicit, we hereby focus our emphasis on their coordination, which will be illustrated with a bilateral negotiation scenario. Typically, the negotiation process is as follows:

Step 1. As is shown in the left of figure 1, the initiating agent publishes the URLs of the CPO and CDO, defines a concrete negotiation protocol through instantiation from the CPO, and starts up a negotiation thread waiting for responders.

Step 2. Responding agent activates its controller and joins the negotiation thread. Based on lexical and semantic similarities, it establishes a mapping from its private ontologies to the common ones. Then negotiation protocol is sent to protocol parser.

Step 3. In terms of the CPO, the protocol parser parses the input protocol as a FSM behavior in which the sub-behavior of each state is empty. The strategy space of each agent is also generated and sent to the strategy generator.

Step 4. Strategy generator encodes the input strategy spaces as some co-evolving populations, and extracts current belief (especially of opponents) from the belief base as parameters of the co-evolution. Each population searches for one agent's strategy space and converges to its optimal strategies. Using a particular criterion, the planner will select one strategy for execution from these generated strategies. This strategy is used to fill or update the sub-behavior in each state of the FSM behavior. A belief revision operation will be performed to update the belief base if necessary.

Step 5. The FSM behavior is appended to the behavior queue, which is managed by the scheduler with a particular behavior-scheduling algorithm. Once the FSM behavior is executed, negotiation process begins.

Step 6. During negotiation, step 4 and step 5 are performed repeatedly until a final state of the FSM behavior is reached. In case the negotiation protocol changes, the protocol parser will be invoked once again to generate a new FSM behavior and provide the strategy generator with new strategy spaces.

4 The Protocol Parsing Mechanism

This section will discuss the working principle of the protocol parser, which enables agents to understand various protocols dynamically.

As stated above, the main goal of the protocol parser is to translate the protocol from an ontological format to an executable behavioral format. As a useful tool in modeling complex protocols, a FSM is used as a bridge to facilitate the translation. Figure 2 presents the flow of the overall parsing process, which includes three stages.

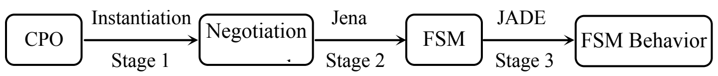


Fig. 2. The flow of the protocol parsing process

The first stage is the instantiation of the negotiation protocol from CPO. This is a precondition of the parsing process, and can be regarded as the semantic annotation of the protocol. In this stage, CPO plays an importance role. As a shared view of the commonalities across different protocols, CPO must capture the common concepts and their relationships. The instantiated protocol can be built from the instances of these concepts, and will be stored in OWL documents.

In the second stage, the OWL protocol will be parsed into a FSM. This can be done with the help of Jena. Jena is a Java framework for building semantic web applications and provides a java API for OWL manipulation [6]. Using Jena model interface, the protocol parser can explore the document structure of the OWL protocol, and extract instances of the CPO classes. In this way, the instances in the input OWL document can be considered as building blocks, from which a FSM can be assembled. In addition, the strategy space of each agent will also be generated in this stage simply by aggregating all possible actions in each move.

The last stage is to encode the FSM to an executable behavior. Fortunately, this can be done by the agent platform itself. The advantages of FSM lie not only in its rich expressive power, but also in the efficient supports from a variety of agent platforms. For instance, JADE [7], an open-source and fully FIPA compliant agent platform, provides a useful class jade.core.behaviours.FSMBehaviour which can translate from FSM to the corresponding FSM behavior conveniently. It is worth noting that in each state of the resulting FSM behavior, sub-behavior will be empty since only possible actions are specified in FSM. Which action is actually performed will be determined by the strategy generator.

5 The Strategy Generation Mechanism

This section presents the rationale of the co-evolutionary algorithm (CEA) used by strategy generator, which enables agents to generate optimal strategies adaptively.

As stated above, the strategy generator aims at specifying a determinate action for each move of an agent to maximize its payoff. This can be achieved using CEA. Essentially speaking, the co-evolution can be considered as an iterative searching and learning process in strategy spaces. The strategy generator will maintain several populations co-evolving with coupled fitness. In each generation, new strategies arise to defeat old ones, and this creates new challenges for the opponent populations. Consequently, fitter strategies must be found in the opponent populations to adapt to the challenges. This process will lead to an evolutionary “arm race” and eventually converge to the optimal strategies.

```
Input: An agentID id; Strategy spaces of N agents; Current belief; Instruction from the Planner;
Output: An optimal strategy of agent$_{id}$.
Begin
  Use the current belief to initialize the parameters of the co-evolution and set generation = 0;
  For i = 1 to N do Begin // Step 1: strategy encoding
    Encode agent$_i$'s strategy space into a population P$_i$ containing p$_i$ random strategies;
  End;
  While (generation <= maximum generation) do // Iterative co-evolution
    For i =1 to N do Begin // Step 2: fitness evaluation
      For j =1 to p$_i$ do Begin
        Select the j'th strategy s$_j$ from P$_i$; //Host strategy
        Repeat //Negotiate with all parasite strategies
          Select an untested strategy from each opponent population; //Parasite strategies
          Calculate the payoff of s$_j$ after negotiating with these strategies;
        Until (all opponent strategies are tested).
        Fitness of s$_j$ = the average payoff obtained against all the opponent strategies;
      End;
    End;
    For i=1 to N do Begin // Step 3: genetic operation
      Perform the selection operation on P$_i$ based on the fitness;
      For j=1 to p$_i$ do Perform the crossover and mutation operation on strategy s$_j$.
    End;
    Set generation = generation+1;
  End;
  Select an optimal strategy from the individuals in P$_{id}$ under the Planner's instruction and output it.
end
```

Fig. 3. Pseudo code of the co-evolutionary algorithm

The pseudo code of the detailed CEA is shown in figure 3. It consists of three main steps: strategy encoding, fitness evaluation and genetic operation. Specially, we will clarify the realization of the former two steps.

Strategy encoding is a mapping process from agent's strategy space to CEA's code space. Generally speaking, each input strategy space from the protocol parser will be encoded as a strategy population, in which a strategy individual is encoded as a chromosome. An action in each move of this strategy will be encoded as a gene on the chromosome, and all possible actions on one move constitute alleles.

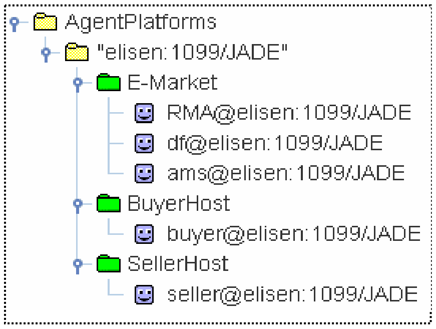
Fitness evaluation of CEA differs from that of canonical evolutionary algorithm. Similarly as in [8], we use term "host" to refer to the population where the currently evaluated strategy is resided, and "parasites" to opponent populations. Each strategy of the host population competes against all the parasite strategies and its fitness will be the average payoff obtained against these parasite strategies. In this way, each population takes turns to be host and all strategies can be evaluated.

6 Experiment and Results

We implemented this adaptive agent architecture by integrating a protocol parser and a strategy generator with a JADE agent. In this section, we will carry out an experiment to validate that it can understand a simple protocol to which it has no prior knowledge, and respond with an optimal strategy.

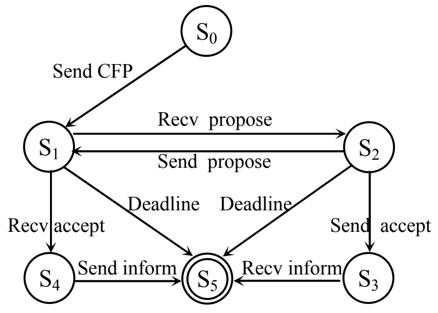
6.1 Experiment Scenario

The experiment scenario involves two agents negotiating over a single issue. As is shown in figure 4(a), we initialized JADE with three containers: A main container serves as the E-market and two attached containers as the buyer host and seller host. The buyer and seller agents, denoted as ag_B and ag_S respectively, reside on their own host container initially and will migrate to the E-market later on. For simplicity, assume they are homogeneous and each has complete information about its opponent. In this case, ontology mapping and belief revision will not be taken into account.



```
AgentPlatforms
  "elisen:1099/JADE"
    E-Market
      RMA@elisen:1099/JADE
      df@elisen:1099/JADE
      ams@elisen:1099/JADE
    BuyerHost
      buyer@elisen:1099/JADE
    SellerHost
      seller@elisen:1099/JADE
```

(a) Screen capture of the JADE GUI



(b) The alternating-offer protocol of ag_B

Fig. 4. The JADE set-up and the negotiation protocol

A finite-horizon version of the alternating-offer protocol depicted in [9] is used in our experiment. It has a unique sub-game perfect equilibrium (SPE), which can be used to check the optimality of strategies. In this protocol, agents offer alternately in discrete rounds $t = 1, 2, ..., T$ until reaching the deadline or agreement. Agents' time preferences are expressed by discount factors, denoted as δ_B and δ_S respectively. Without loss of generality, assume ag_B makes the first offer with a CFP message. The states and transitions of ag_B are depicted in figure 4(b), and those of ag_S can be obtained simply by exchanging the Send and Recv (abbr. of Receive) predicates.

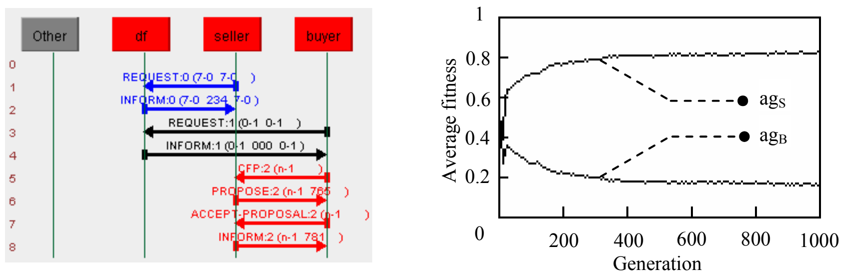
6.2 Experiment Results

Let us first present the FSM behavior generated by the protocol parser. As stated above, this protocol must be semantically annotated based on CPO to be understood by agents. Due to space constraints, the class hierarchy of CPO will be omitted here. However, we think the concepts and instances below are sufficient to characterize this simple protocol: 1) six instances of class state from S0 to S5; 2) nine instances of class transition, each corresponding to an arrow in figure 4(b); 3) nine instances of class action, and each action triggers a transition; 4) seven instances of class message, each attached to a Send or Recv predicate.

```
registerFirstState(new S0Behavior(), S0); registerDefaultTransition(S0,S1);
registerState(new S1Behavior(), S1); registerTransition(S1,S2,1); registerTransition(S1,S4,2);
                                    registerTransition(S1,S5,3); // 1,2,3 are return values of S1Behavior();
registerState(new S2Behavior(), S2); registerTransition(S2,S1,1); registerTransition(S2,S3,2);
                                    registerTransition(S2,S5,3); // 1 2,3 are return values of S2Behavior();
registerState(new S3Behavior(), S3); registerDefaultTransition(S3,S5);
registerState(new S4Behavior(), S4); registerDefaultTransition(S4,S5);
registerLastState(new S5Behavior(), S5);
```

Fig. 5. The code of the generated FSM behavior

Figure 5 presents the FSM behavior generated from parsing these instances. We can see that each piece of code registers an instance of class state or transition within agent. The instances of class action and message are encapsulated in a sub-behavior of each state, whose return values will be used to decide the next state of a transition



```
REQUEST:0 (7-0  7-0   )
INFORM:0 (7-0  234  7-0 )
REQUEST:1 (0-1  0-1   )
INFORM:1 (0-1  000  0-1 )
CFP:2 (n-1       )
PROPOSE:2 (n-1  765   )
ACCEPT-PROPOSAL:2 (n-1       )
INFORM:2 (n-1  781   )
```

(a) Screen capture of the sniffer agent (b) Average fitness of strategy populations

Fig. 6. Negotiation results

Next we will check the optimality of strategies resulting from the co-evolution in the strategy generator. Figure 6 shows the negotiation results in case the deadline $T = 3$ and discount factors $\delta_B = \delta_S = 0.8$. Similarly as in [9], if we assume the total negotiation surplus is equal to unity, the SPE in this case will be "Negotiation ends in the first round, and ag_S gets payoff 0.84 while ag_B gets the rest 0.16".

Figure 6(a) depicts the messages on the E-market container captured by a JADE sniffer agent. Obviously, negotiation will begin with the fifth CFP message. ag_B accepts the first offer of ag_S, and negotiation ends immediately. Figure 6(b) shows the average fitness of each agent's strategy population during the co-evolution. Here average fitness can be used to predict and interpret agent's payoff, and they converge stably to 0.84 and 0.16 respectively. These results are consistent with the SPE, which validates that optimal strategies have been obtained from the co-evolution.

7 Conclusions

This paper presents a new agent architecture capable of negotiating adaptively. Such techniques as ontology and co-evolution are employed to empower agents with capabilities of understanding ever-changing protocols and responding with optimal strategies adaptively. An experiment carried out on JADE platform indicates that this architecture is feasible and efficient. In the future work, we plan to experiment with more complex protocols and develop a practical AMEC prototype system.

8 References

1. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra and M. Wooldridge, “Automated Negotiation: Prospects, Methods and Challenges”, *International Journal of Group Decision and Negotiation*, 10(2), 199-215, 2001.
2. N. Silva, P. Maio and J. Rocha, “An Approach to Ontology Mapping Negotiation”, In: Proceedings of the Third International Conference on Knowledge Capture Workshop on Integrating Ontologies, Banff, Canada, 2005.
3. G. Antoniou and F. van Harmelen, *A Semantic Web Primer*, MIT Press, 2004.
4. V. Tamma, M. Wooldridge and I. Dickinson, “An Ontology for Automated Negotiation”, In: Proceedings of the Workshop on Ontologies in Agent Systems, Bologna, Italy, 2002.
5. J. Paredis, “Co-evolutionary Computation”, *Artificial Life*, 2(4), 355-375, 1995.
6. Jena-A Semantic Web Framework for Java. URL: http://jena.sourceforge.net/.
7. JADE-Java Agent Development Framework. URL: http://jade.tilab.com.
8. C. D. Rosin and R. K. Belew, “New Methods for Competitive Co-evolution”, *Evolutionary Computation*, 5(1), 1-29, 1997.
9. A. Rubinstein, “Perfect Equilibrium in a Bargaining Model”, *Econometrica*, 50(1), 97-110, 1982.