Agent-Oriented Software Engineering for Successful TAC Participation

Clemens Fritschi, Klaus Dorer
living systems AG
Humboldtstr. 11
D-78166 Donaueschingen, Germany
+4977189870
{clemens.fritschi, klaus.dorer}@living-systems.de

Tracking Number: 325

ABSTRACT

Simplified modeling of complex domains is one of the many advantages of multi-agent systems that is abundantly mentioned. In this paper we describe our approach for an efficient design and implementation of multi-agent systems using agent oriented methodologies and tools. We demonstrate the strength of this approach taking the example of the TAC domain. The trading agent competition (TAC) domain is a challenging e-marketplace domain for autonomous auction agents. A competition with 19 participants from 9 countries was held for the second time in October 2001 in Tampa, Florida. The development process turned out to be not only very effective with respect to time but also with respect to success with the living agents team finishing as the highest scoring team in the competition.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents, Multiagent systems.

General Terms

Design, Economics.

Keywords

Agent-based software engineering, bidding and bargaining agents, electronic commerce, methodologies and tools, adaptive profit optimization

1. INTRODUCTION

With the increasing number of agent systems and approaches to achieve intelligent autonomous behavior the need for benchmarking domains in which different agent systems can be compared to each other became more and more evident. One early example of such a domain is the RoboCup domain, in which teams of autonomous agents compete in simulated soccer since 1997 (www.robocup.org).

An especially interesting domain for business agents is the trading agent competition (TAC). It was introduced in 2000 by a team led by Michael Wellman and Peter Wurman to provide a complex e-marketplace domain where agents bid in online simultaneous auctions for complimentary and substitutable goods [Wellman et. al., 2001]. At EC'01, for a second time the competition was organized in which 19 teams of autonomous agents from 9 countries competed. Running through a qualification and seeding round, 16 teams were admitted to the semi-finals and 8 teams qualified for the finals.

In this paper we describe the process of efficiently designing and creating the highest scoring agent team living agents[1] and the strategy used by the agents to achieve this success. We give a detailed description of the application of an agent-oriented software engineering methodology and give some insight into the tools that supported us in this.

The remainder of this paper is organized as follows. Section 2 briefly describes the TAC domain. Section 3 presents the living agents team with descriptions of the agent-oriented development process and the strategy used. Section 4 gives some results of the development process and the TAC competition. Finally, section 5 concludes and presents suggestions for future research.

2. THE TAC DOMAIN

In the TAC domain, autonomous travel agents have to organize a trip for (simulated) clients by purchasing flight, hotel and entertainment tickets in simultaneous auctions. In each instance of auction rounds 8 teams of agents compete in organizing an optimal trip for 8 clients each. The trip includes an inflight ticket to Tampa, up to 4 nights stay in a hotel, some optional entertainment tickets and the flight back. The preferences of the clients are randomly assigned by the auction server at the beginning of an auction round. They include the desired days of stay, how much the client is willing to pay additionally for the luxury hotel and what entertainment the client prefers. The auction servers, located at the University of Michigan, maintain the current auction states, execute bids of the agents and answers requests on price quotes. The tickets are sold in 28 auctions of 3 different types.

[1] For technical reasons our team's name was spelled livingagents during the competition.

- **Hotel**: There are 2 types of hotels the clients can stay: The Tampa Towers (luxury) and the Shoreline Shanty (economy). Clients stay between 1 and 4 nights and have individual preferences for luxury or economy rooms. Each hotel has 16 rooms and each night must be purchased in a separate auction. The auction is a 16^{th} –price English auction, where the highest 16 bids get a hotel room at the price of the 16^{th} highest bid. Hotel rooms may not be resold and bids may not be withdrawn in hotel auctions. Hotel auctions end at an unspecified time.
- **Flight**: There are 8 auctions for flights: 4 for inflights on days 1 to 4 and 4 for outflights on days 2 to 5. The supply of tickets is unlimited, so sending a bid at the ask price or above will always result in a ticket. As opposed to last year, the ask price for a flight was not approximately constant over the time of an auction, but rose continuously especially at the end of an auction. This gave an advantage for early decisions on flights at the risk of not succeeding in buying all desired hotel rooms.
- **Entertainment tickets**: There are 3 types of entertainment tickets available on each of the first 4 days: Tickets for a museum, an amusement park and alligator wrestling. The participating teams initially receive a random endowment of tickets. The tickets are then bought and sold in continuous double auctions, meaning that whenever a bid is placed at another team's sell price the deal is performed.

The goal is to maximize customer satisfaction on one hand and to minimize one's own expenditure on the other hand. Both are taken into account when calculating the final score for a team serving 8 clients:

$$score = \sum clientUtility_i - costForTickets, \text{ where}$$

$$clientUtility = 1000 - travelPenalty + hotelBonus + funBonus$$

where a travelPenalty is assigned for each day the client can not stay, but desired or additionally has to stay. The hotelBonus is assigned if the client stays in the luxury hotel and the funBonus is assigned for each desired entertainment ticket available.

The challenge is to bid in 28 simultaneous auctions for an optimal allocation of tickets, where the results of auctions depend on each other. A cheap flight ticket, for example, bought early in the flight auction could turn out to be counter productive if later in the auction hotel prices explode for the first night of stay. A flight on the following day could have revealed a higher overall utility.

More detailed descriptions of the TAC domain and auction mechanisms can be found in [Wellman et. al., 2001; Stone et. al., 2001; Greenwald & Stone, 2001] or at the TAC homepage at `http://auction2.eecs.umich.edu`.

3. LIVING AGENTS DESIGN

The living agents team is based on the living agents runtime system (LARS) of the living systems AG. LARS-agents operate in various domains ranging from research domains such as TAC or RoboCup to various operating business platforms in the areas of adaptive profit optimisation for the logistics and finance industry. While the agent technology itself provides the foundation for such a broad applicability, the efficient application of agents to various, complex domains can be considerably improved with the use of agent oriented software engineering methodologies and tools.

Therefore, we first explain the process of agent development using the example of our TAC team creation. In the second part of this chapter we give the details on the domain specific strategy incorporated into the agents.

3.1 Agent Development

To be able to deal with increasingly complex domains, high-level abstractions, object oriented analysis and design for instance, have been introduced to model and implement complex systems. Specialized methods have been developed to document and support those software engineering methods. It can be argued [Jennings, & Wooldridge, 2001; Weiß, 2001] that agent oriented software engineering is a further step in advancing the abstraction level to model again more complex domains. New methods have been developed to support the analysis and design of agent-based systems such as KGR [Kinny, Georgeff, & Rao, 1996], Agent-UML [Odell, Parunak, & Bauer, 2000] or Gaia [Wooldridge, Jennings & Kinny, 2000].

The Gaia methodology defines 2 models for the analysis of agent systems and 3 models for the design. In the analysis phase, one has to define a roles model with roles that will later be occupied by agents as well as an interactions model that specifies the interplay of the roles. In the design-phase, an agent model is derived from the roles model defining the agent types and instances of agents. A services model defines the services that are provided by the agents. Finally an acquaintance model defines the communication links between the agents [Wooldridge, Jennings & Kinny, 2000].

The living markets development suite (LMDS), the development environment used to design our TAC agent team, builds on a methodology comparable to Gaia. It supports the design-phase of an agent system by allowing an *agent scenario* to be defined, which is roughly a compact representation of the 3 models defined in Gaia for the design phase. To illustrate, we describe the complete design process of the living agents team for TAC. The steps, with explanations hereafter, are:

1. Defining the agent types
2. Defining the agents' interactions
3. Defining the perceptions, actions and services

3.1.1 Defining the agent types

As a first step, the agent types have to be identified, which allow the easiest and most natural modeling of the domain. An agent type consists of a name, a verbal description, the decision engine type, the responsibilities, and permissions of the agent instances.

The decision engine type specifies how business logic of this agent type will be represented and processed. Specifically it determines, whether the agent will be proactive or reactive. The LMDS currently supports a reactive workflow engine, and a proactive extended behavior network engine [Dorer, 1999]. Since little proactive behavior was needed all agents used the workflow engine. Within the workflow engine, all responsibilities of an agent type directly translate into a workflow. The way of defining a workflow will be explained in section 3.1.3. Since the TAC

scenario is a closed and relatively small agent system we did not need to implement a permissions model.

In our example we created 6 agent types (see Fig. 1):

- TACManager (1): Responsible for starting and stopping the other agents when a new round of auctions started. In the qualification rounds, auctions continuously took place over a time span of 1 week which made it highly desirable that the system was completely autonomous during the whole time and not only during one round of auctions.
- TACClient (8): Responsible for the calculation of the best combination of tickets for the client it represents.
- TACDataGrabber (5): Responsible for providing current auction information from the auction server.
- TACAuctioneer (4): Responsible for bidding in the auctions according to the suggestion of the TACClients.
- TACEntertainmentAuctioneer (1): As an extension of the simple TACAuctioneer type, this agent type was introduced to deal with the specifics of the continuous double auctions. Therefore it was responsible for bidding in entertainment ticket auctions and regularly observe the current auction prices for opportunities to buy or sell tickets.
- TACResultGrabber (1): Responsible to grab information of previous auction rounds and generate statistics to be used in later auctions.

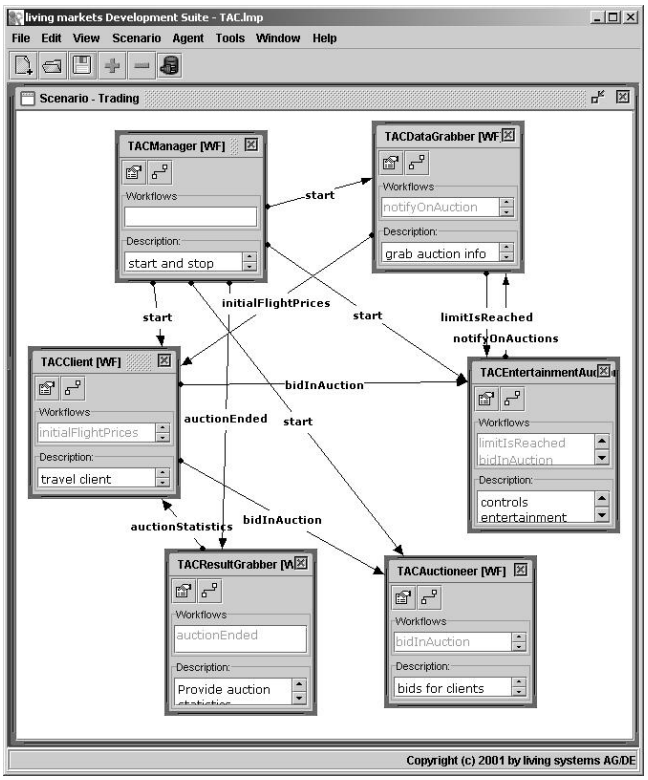

Figure 1. TAC agent scenario.

3.1.2 Defining the agents' interactions

Apart from the agent type definitions an agent scenario also contains the interactions between the agent types. As opposed to the acquaintance model of Gaia, which only models the interactions itself without specifying any messages, an agent scenario directly specifies the messages that are exchanged between the agent types. While the agent scenario contains more detailed information of the interaction between agent types, the acquaintance model of Gaia will be easier to read in cases where the same agent types share multiple messages since it contains only a single link between 2 agent types.

An interaction is defined by a sender agent, a receiver agent and a corresponding message name. At the receiver side such an interaction directly creates a (yet empty) service the agent is providing. An interaction does not define, when these messages are sent. This is part of the services description, which is described next.

3.1.3 Defining the perceptions, actions and services

As a final step, the services of each agent type have to be defined. The definition contains the name of the service, the inputs and outputs as well as the content of the service. Pre- and post-conditions, which are part of the Gaia services model, are only specified in the proactive extended behavior network engine. A service is triggered whenever the agent receives a message with the name of the service. The type of the service is already defined by the decision engine of the agent type, which is in this case a workflow engine. To be able to define the workflows of each service, or any other decision logic, the perceptions and actions of the agent have to be defined first. This is the only step, where Java-programming is necessary.

A perception is a method that receives the agent's current context containing the inputs of the service and returns true (1.0) or false (0.0) if the perception is true or false[2]. A perception is not allowed to change the context of the agent. An action is a similar method, but is allowed to change the context and environment of the agent. Actions and perceptions are organized as beans and can be used within any decision engine. The decision at which level of detail to draw the cut between low level actions and perceptions and high level strategy is up to the designer.

The definition of a service is illustrated by the 'start' service of the TACDataGrabber, which is triggered by the TACManager at the beginning of a new round of auctions (see Fig. 2). The purpose of this service is to notify the TACEntertainmentAuctioneer (TAE) if any of the auctions it is interested in has reached a specified upper limit, in case of selling, or a lower limit, in case of buying interest. For the service to be defined, 6 perceptions and 3 actions have been specified:

Perceptions:

- moreAuctionsResponsibleFor: true, if there are auctions remaining that the agent is responsible for.
- auctionEnded: true, if the auction in the context has ended.
- existingNotifications: true, if further buy or sell requests of the TAE are available for this auction.

[2] The extended behavior network engine also supports continuous truth values for continuous domains.

- notifyMeOnPriceAboveLimit: true, if the TAE wants to sell a ticket.
- actualPriceAboveLimit: true, if the price limit specified by TAE is reached.
- actualPriceBelowLimit

Actions:

- getAuctionPriceFromServer: loads the current information of the auction in the context from the TAC server.
- send: sends a message to another agent. The input parameters of this action are the name of the service, the receiver and the content of the message.
- deleteNotifyEntry: removes the notification request of the TAE from the list of requests

The content of the service is specified by the workflow, which is a tree of simple control elements (while, if), perceptions and actions (see Fig. 2).

As with most design methodologies, there is no need to strictly adhere to the order of steps described above. In practice the steps usually are repeated, e.g. to extend the agent system by a new agent type as was done with the TACResultGrabber which was only introduced after the qualification rounds.

The agent scenario is then saved into a separate XML file for each agent type. This XML strings can directly be sent to the LARS agent platform which will start the specified number of agents for each agent type and configure them with the specified decision engine and workflows.

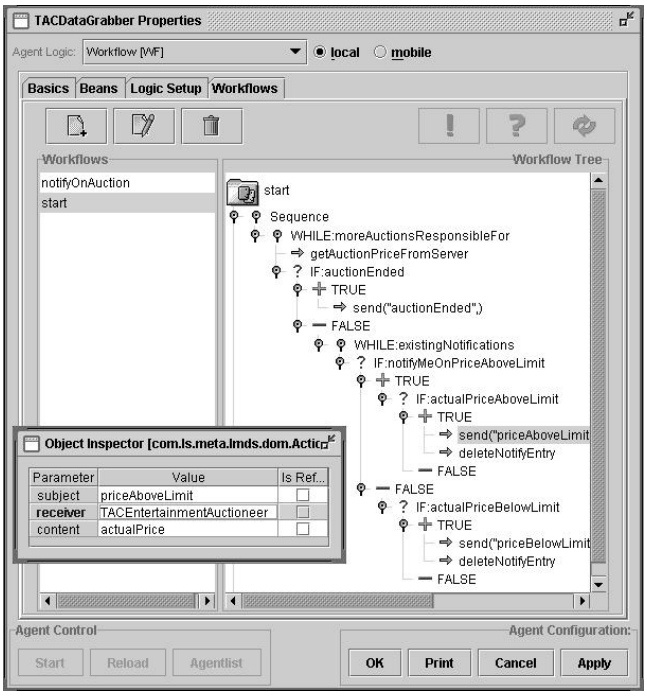

Figure 2. Workflow defining the 'start' service of the TACDataGrabber and input parameter description for the 'send' action used within the workflow.

3.2 Strategy

In this section we describe the overall strategy of our TAC agent scenario. It was guided by the following assumptions, which have been reinforced during the qualification round and seeding round of TAC:

1. The steadily increasing flight prices favor early decisions for flight tickets.
2. Especially the good performing teams are following a strategy to maximize their own utility. They are not trying to take the risk to reduce other team's utility, which is anyhow difficult in a 16th price auction for the hotels.

These 2 points, derived from the change in auction rules, lead to a completely different strategy than was used last year by the highest scoring team ATTac2000. Last year, ATTac tried to delay most of the purchases to the very end of an auction round [Stone et. al., 2001]. Our strategy did the very opposite and took most of its decisions at the very beginning of an auction round. Only decisions to buy or sell entertainment tickets have been taken during the auction round. This saved considerable amounts of money for flight tickets. While this strategy was less effective in the qualification rounds, where weaker teams drove hotel prices up, and would have been most likely completely unsuccessful in last year's competition, assumption 2 assured a winning success in the finals (see Tab. 1).

Hotel	**Day**	**Seeding Round** (n = 759)	**Final** (n = 24)	**Significance of difference**
Tampa Towers (luxury)	1	75	66	0.34
	2	167	120	0.007
	3	174	86	<0.001
	4	81	61	0.02
Shoreline Shanty (economy)	1	22	12	0.31
	2	109	77	0.061
	3	104	74	0.031
	4	28	13	<0.001

Table 1. Comparison of the average hotel prices in the seeding round and in the final round of the competition.

The strategy in more detail:

1. The TACManager is waiting for an auction round to start. When it is started it gets the general auction information and the client preferences from the server. It then sends the 'start' message to all other agents, containing this information.
2. The TACDataGrabbers responsible for inflight and outflight auctions request current prices of the flights and send them to the TACClient agents.
3. The 8 TACClient agents are calculating their best journey based on the preferences of the client they represent, the current flight prices and the average hotel prices from auctions of previous rounds (see below). Then they send the information for the desired tickets and upper limits to the TACAutioneers and to the TACEntertainmentAuctioneer.

4. The flight and hotel TACAuctioneers are bidding for the needs of the clients.
5. The TACEntertainmentAuctioneer informs the TACDataGrabber responsible for entertainment auctions about the auctions it wants to sell or buy in. If it is informed about an auction it wants to buy in that is cheaper than its desired strike price, it is placing a bid just above the price. If an auction it wants to sell in is above the desired price, it is placing a bid at the desired price. After 7 minutes the TACEntertainmentAuctioneer places all the remaining bids for entertainment tickets at the desired price. This was not done at the beginning of an auction round to exploit cheap starting auctions. The desired price we used was the average price for entertainment tickets from previous qualification rounds, which was of $80 used for the finals.

The approximately optimal allocation of tickets in point 3 are calculated by calculating the benefits of all permutations of possible stays and taking the combination with the highest benefit:

```
for (arrivalDay = firstDay to forthDay)
   for(departureDay = arrivalDay+1 to fifthDay)
      for (both types of hotels)
         benefit = 1000
            - inflightPrice
            - outflightPrice
            - sum(averageHotelPrices)
                  – penalty
                  + tampaHotelBonus
                  + entertainmentBenefit
```

In this calculation, the `inflightPrice`, `outflightPrice`, `penalty` and `tampaHotelBonus` are known. The `averageHotelPrice` was estimated based on the values from earlier qualification rounds. As can be seen from Table 1, the average hotel prices differed considerably from the seeding round to the finals. Thus a better strategy would have taken previous auction prices of the same round into account. Although our agents were able to do this from a technical point of view, we did not use this optimisation, because we were not sure if it complies with the TAC rules. The `entertainmentBenefit` was also estimated based on the assumption that we will get all desired entertainment tickets at the average price of the previous rounds.

4. RESULTS

This section provides results of the TAC competition and information about the development process of the living agents team.

4.1 Agent Oriented Software Engineering

It is in general hard to get qualitative or even quantitative meaningful data to support claims that agent oriented software engineering (AOSE) is easier, faster or able to handle more complex problems than other software engineering approaches. Nevertheless, we decided to give the few quantitative information we have for the reader's own interpretation.

At the time we decided to participate in TAC, no Java interface to the TAC auction server was available. Hence approximately 3 weeks were necessary to implement the low-level communication interface to the server.

The agent scenario design used for the qualification rounds started 7 days before the first TAC qualification round on 12th of September. An earlier prototype used for testing the low-level interface to the server was completely thrown away. During the qualification round, the main efforts were directed to improve the robustness of the software with respect to own bugs and reaction to server shutdowns to assure smooth participation in overnight runs.

In the remaining 4 weeks to the finals we introduced the TACResultGrabber to automatically calculate the average hotel and entertainment ticket prices of previous auctions. We also introduced the more sophisticated strategy of the TACEntertainmentAuctioneer explained in section 3.2.

4.2 TAC

The trading agent competition was held in 4 rounds: a qualification round, a seeding round to determine the grouping for the semi-finals and a final round conducting more than 1800 rounds of auctions with a sum of above 50000 auctions.

Round	Teams	Rounds	Date
Qualification	28*	992	10.9. – 17.9.
Seeding	19	775	24.9. – 5.10.
Semi-final	16	12	14.10.2001
Final	8	24	14.10.2001

Table 2. Overview of the TAC schedule.

* The number is higher than the number of participants since 2 benchmark teams (ATTac-2000 and a dummy) also entered the competition and some participants entered more than one team.

During the qualification round, the living agents team suffered from instabilities on the client and sometimes server side leading to series of 0 score games over night. The qualification was finished at 11th place with a score of 2639. The seeding round showed improvements with living agents at 4th place with 3012 points, but still had 17 out of 305 games with 0 score.

The finals were conducted in 24 rounds of auctions with all 8 finalists potentially bidding in the same auctions. The average scores and the standard deviations of the 8 finalist teams are displayed in Table 3. For the finals, the technical problems have been solved, avoiding any 0 score games and having a lowest score game of 2332 points.

The results are hardly comparable to last year's results, since regulations have changed from TAC00. The scores indicate, however, that the performance of the best 8 teams was much closer this year. While last year's 8th team scored only 34% of ATTac's points, last year's winner, this year the 8th team's score was 78% of the top score.

To also document the excitement during the finals, Figure 3 gives the average scores of the final over time.

5. CONCLUSIONS AND FUTURE WORK

During our TAC participation we experienced that the use of an agent-oriented software engineering methodology and especially the use of tools supporting it eases and speeds up the creation of a multi-agent system. We chose the TAC domain as a test domain for these tools because of its complex setting of simultaneous auctions in an adversarial domain of multiple agent teams.

Rank	Team	Avg. Score	Std. Dev.	Institution
1	livingagents	3670	622	living systems AG, GER
2	ATTac	3622	692	AT&T Labs – Research, USA
3	whitebear	3513	700	Cornell University, USA
4	Urlaub01	3421	698	Pennsylvania State University, USA
5	Retsina	3352	668	Carnegie Mellon University, USA
6	SouthamptonTAC	3254	1467	University of Southampton, UK
7	CaiserSose	3074	656	University of Essex, UK
8	TacsMan	2859	1054	Stanford University, USA

Table 3. Final scores of the eight finalists after 24 rounds of auctions in the TAC finals.

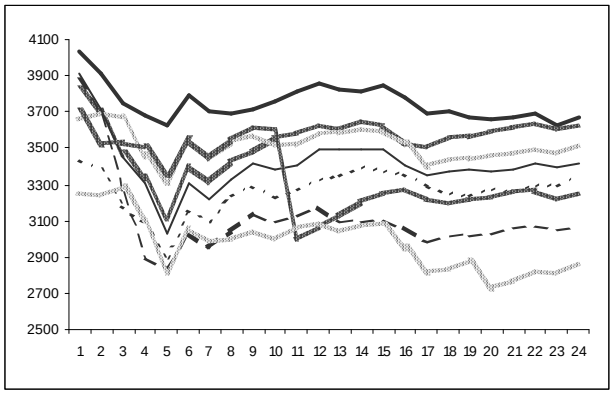

Fig 3. Average scores during the final over time.

The lines can be mapped to team names with the help of Table 3.

One of the most interesting things we learned from TAC-2001 is the interdependence of strategies. A fairly straightforward strategy, which is vulnerable to simple strategies of opponent agents, can turn out to work excellent among and surpass more sophisticated strategies[3]. The latter themselves have been more successful against the simple strategies.

Knowing this, our research efforts will focus on opponent modelling to automatically detect and predict the strategy of opponents and adapt our own strategy according to this.

The agent-oriented software development tools used during our TAC participation have been recently developed and are now integrated into the production process. They will be used within 'real' business projects of living systems.

6. ACKNOWLEDGMENTS

Thanks to Michael Wellman and his team for providing and maintaining the TAC servers and for their immediate and helpful responses to our questions. Thanks to Torsten Eymann and Peter Stone for valuable feedback to earlier versions of this paper. Special thanks to Sudha Raver Veetil and Bhupendra Bhavsar for their committed help in the early implementation phase.

[3] We assume here that the strategies of last year's teams at least maintained the degree of sophistication.

7. REFERENCES

[1] Dorer, K. Extended Behavior Networks for Continuous Domains using Situation-Dependent Motivations. *Proceedings of the 16th International Joint Conference of Artificial Intelligence* (IJCAI), pages 1233-1238, Morgan Kaufmann, Cambridge MA, 1999.

[2] Greenwald, A., and Stone, P. Autonomous Bidding Agents in the Trading Agent Competition. *IEEE Internet Computing*, pages 52-60, March/April, 2001.

[3] Jennings, N. R., and Wooldridge, M. Agent-Oriented Software Engineering. In: J. Bradshaw (ed.), *Handbook of Agent Technology*, AAAI/MIT Press, 2001.

[4] Kinny, D., Georgeff, M., and Rao, A. A Methodology and Modelling Technique for Systems of BDI Agents. Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. Springer: Berlin, 1996.

[5] Odell, J., Parunak, H. V. D., and Bauer, B. Extending UML for agents. *2nd International Workshop on Agent-Oriented Information Systems* (AOIS), 2000.

[6] Stone, P., Littman, L., Singh, S., and Kearns, M. ATTac-2000: An Adaptive Autonomous Bidding Agent. *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.

[7] Weiß, G. Agentenorientiertes Software Engineering. *Informatik Spektrum* 24 (2), pages 98-101, 2001.

[8] Wellman, M. P., Wurman, P. R., O'Malley, K., Bangera, R., Lin, S., Reeves, D., and Walsh, W. E. Designing the Market Game for a Trading Agent Competition. *IEEE Internet Computing*, pages 43-51, March/April, 2001.

[9] Wooldridge, M., Jennings, N. R., and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems* 3 (3), pages 285-312, 2000.