TOWARDS A KNOWLEDGE-BASED APPROACH FOR AUTONOMOUSE TRADING AGENT

Alireza Moradkhani
National Academy of Science of Armenia
alireza.moradkhani@gmail.com

Tadevos Baghdasaryan
National Academy of Science of Armenia
tadevos.baghdasaryan@gmail.com

ABSTRACT

The trading agent competition for supply chain management (TAC SCM) is an annual competition. An agent is a software program that acts autonomously among other agents. An agent's goal is to maximize profit while participating in a supply chain scenario. In this article we present a knowledge-based approach describing the accumulation and optimization of a knowledge base for supply chain management. A framework for designing strategies for supply chain management trading agent is implemented. The strategy plan is described in simple high level language as a sequence of statements called strategy scripting language. The agent's knowledge base and strategy builder tool is represented in this paper. Game flowchart describing game and knowledge base interactions is represented. In addition we discuss the SOARS Beer Game which is a modification of the original Beer Game by Northwester University. We suggest the application of our proposed strategy design platform to this supply chain game. We have chosen TAC SCM game as a simulation platform for applying and proving our methodology. In addition we show that it can be employed by other various simulation platforms.

INTRODUCTION

The Trading Agent Competition for Supply Chain Management (TAC SCM) (Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2007) has been an active competition since 2003. It is a simulation of certain features of supply chain process with all activities occurring in an artificial economic environment. It was designed as an experiential tool to encourage understanding the relationships between and interactions of participants in this field. The simulation requires participants (agents) to operate in multiple markets (customer and supplier) that have inherited interdependencies as well as incomplete information (Moradkhani A. 2011). Individual agents make decisions and compete in two interconnected markets, that of customers (sales potential) and that of suppliers (necessary for the producing products wanted by customers) market.

The use of multi-agent systems technology is proposed to handle the complex and distributed nature of supply chain in an effective way. Through many researches it has been confirmed that intelligent agent technology is the appropriate technology for addressing supply chain management issue (Moradkhani A. 2011).

A game tree approach is suggested for modeling agent-customer and agent-supplier parts in TAC SCM domain (Baghdasaryan T., Danielyan E., Pogossian E. 2006).

In this paper we describe a knowledge based approach for supply chain management problematic in a multi-agent environment. We also present the knowledge base itself and its suggested accumulation and optimization mechanisms. In addition the authors present a framework for strategy design for Trading Agent Competition for Supply Chain Management (TAC SCM) which consists of a strategy builder tool and a strategy scripting language as explained in (Baghdasaryan T. 2007). Finally we suggest a solution to SOARS beer game strategy design issue (Manabu Ichikawa, Yuhsuke Koyama and Deguchi, Hiroshi 2008).

TAC SCM GAME

A TAC SCM game (Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2007) consists of a number of days where six personal computer assembly agents compete for customer orders and for procurement of a variety of components. Each day, customers issue requests for quotes and select from quotes submitted by the agents, based on delivery dates and prices. The agents are limited by the capacity of their assembly lines and have to procure components from a set of eight suppliers. Four types of components are required to build a PC: CPUs, Motherboards, Memory, and Disk drives. This is shown in figure 1.

Each component type is available in multiple versions.

Customer demand comes in the form of requests for quotes for different types of PCs, each requiring a different combination of components.

A game begins when an agent connects to a game server. The server simulates the suppliers and customers, and provides banking, production and warehousing services to the participant agents. The game continues 220 simulated days. At the end of a game, the agent with the highest sum of money in the bank is declared the winner.

Four basic decisions that an agent must make during each simulated day:

1. *Procurement*: To decide what parts to purchase, from whom and when to have them delivered.
2. *Production*: To schedule its manufacturing facility.
3. Sales: To decide which customer RFQs to bid to what prices.

Fulfillment: To ship completed orders to customers.

To succeed, agents will have to demonstrate their ability to react to variations in customer demand and availability of supplies, as well as adjust to the strategies adopted by other competing agents. They must be able to sense and model the environment and predict their own impact on the environment.

SOARS BEER GAME

Beer Game developed by Northwester University is used by teachers as a practice tool to simulate supply chain scenarios (MIT Forum has also a web-based version) (Manabu Ichikawa, Yuhsuke Koyama and Deguchi, Hiroshi 2008). It is an agent-based game with four decision making agents that compete in a supply chain scenario. These agents must decide how many beers to order, supply beers and so on.

The *SOARS* (Spot Oriented Agent Role Simulator) is a new type agent-based simulation language which is designed to facilitate social simulation modeling (www.soars.jp).

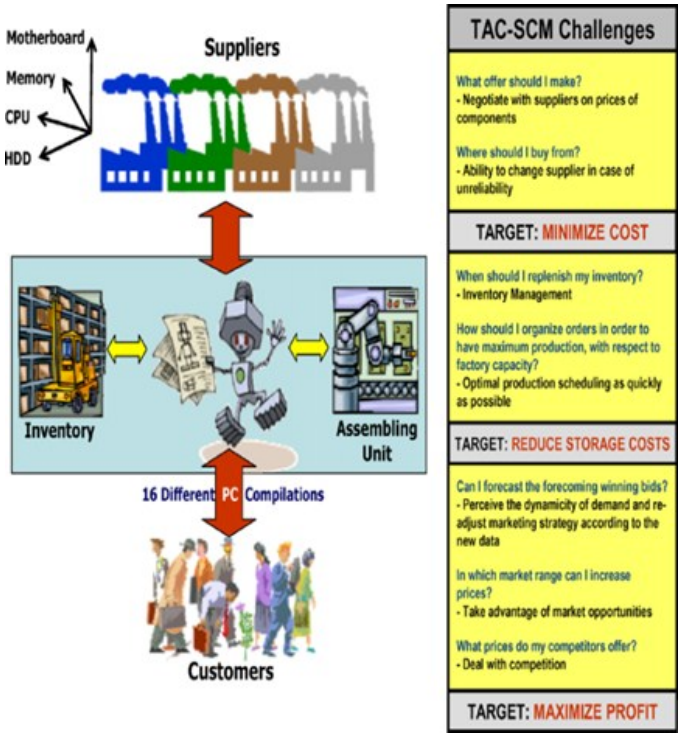
SOARS Beer Game proposed by (Manabu Ichikawa, Yuhsuke Koyama and Deguchi, Hiroshi 2008) is a all-in-one scientific tool for making social game simulations. It contains an animation tool for viewing results and a graph tool for graphical presentation of the results. It is using simulation language SOAR in order to create strategies. Its main disadvantage is that in order to design strategies the user must know the SOARS simulation language.

UNDERLYING METHODOLOGY

In this section we discuss briefly the methodology behind our approach and its application towards TAC SCM game.

A *game* represents all possible allowed actions that a

Figure 1
TAC SCM game



player can perform in each and every situation. We define a *situation* as a set of environmental parameters. Each *action* modifies the value of one or several environmental parameters that will further alter the situation itself (Baghdasaryan T., Danielyan E., Pogossian E. 2006).

In oligopoly competitions several companies compete for some success criteria (max cumulative profit, max return on investment, etc.). In our model of oligopoly competition (TAC SCM) six agents compete for maximal profit.

A *game tree* is a tree structure that models the game where nodes represent situations that are created by actions and contra actions made by the agent and its competitors (Baghdasaryan T., Danielyan E., Pogossian E. 2006). Every path starting from the root situation and ending at a goal situation represents a *strategy*.

We define a class of problems where the space of hypothesis of solutions can be specified by reproducible game trees (SSRGT) (Pogossian E. 2005). It has following requirements:

a) Interacting actors (competitors or players)
b) Identified types of actions
c) Specified moments of time
d) Specified types of situations
e) Defined benefits for each of the actors
f) Reduction of the game tree from any given position (positional game)

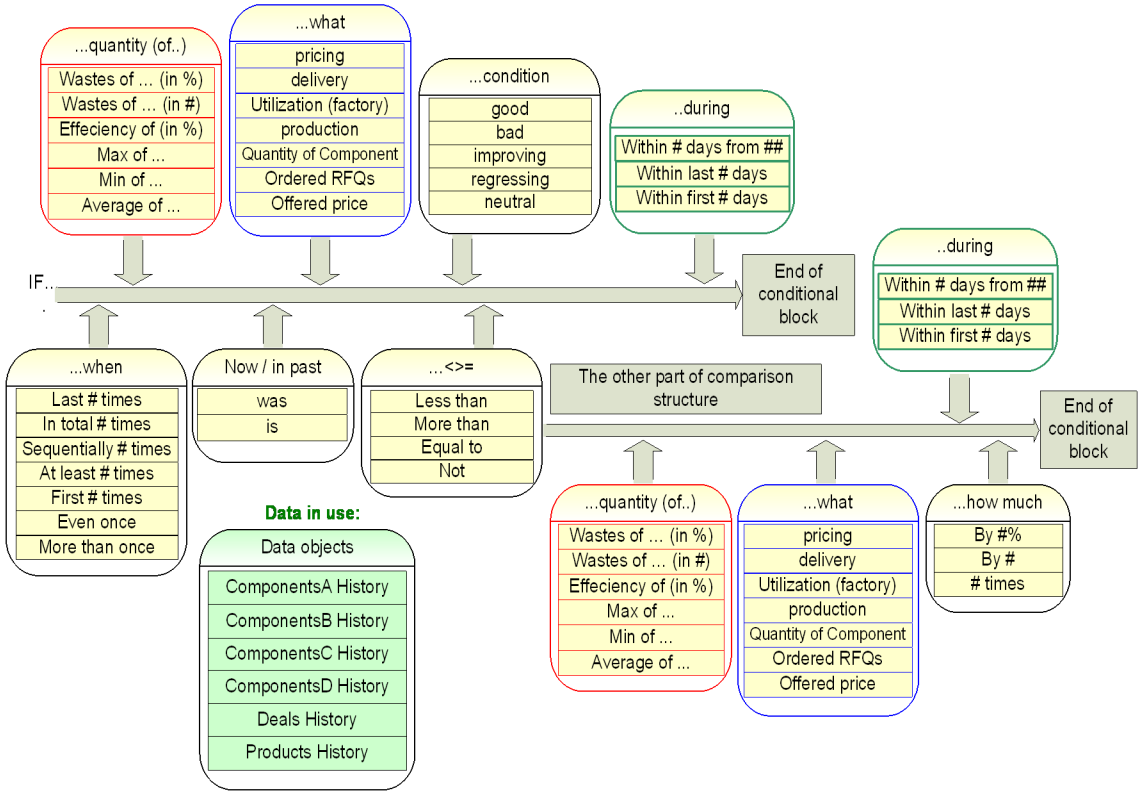
Management Optimal Strategy Provision (MOSP) seeks to provide the most optimal strategy for chosen game with respect to defined criteria (Pogossian E. 2001). By *most optimal strategy* we mean a composition of paths in our game tree that ends to goal situations with the lowest cost.

The TAC SCM game tree have extremely huge branching caused by the combination of high number of environmental parameters. Thus all exhaustive search methods based on enumeration of possible actions become ineffective. To solve this problem we employ the Common Planning and Dynamic Testing (CPDT) algorithm (Pogossian E., Baghdasaryan T. 2003). This algorithm reduces the strategy search space by using expert knowledge in combination with dynamic testing of strategies.

FRAMEWORK FOR STRATEGY DESIGN

We present *Strategy plan builder tool* as a framework for strategy development. The purpose of this tool is to assist strategy designers with the different aspects of their work. It is a graphical instrument aimed to facilitate strategy plan development (Baghdasaryan T. 2007). Strategy plans are specified by users through a graphical schema. The user mainly selects an event and a set of actions among predefined data objects and their corresponding operators

Figure 2
The scheme describes a conditional statement construction on "Customer RFQs are received" event



as shown in figure 2 and 3. This schema is further parsed using the tool to generate strategy plans in scripting language format which are readable for the agent (Baghdasaryan T., Grigoryan A., Naghashyan Z. 2009). Strategy plan is a collection of statements that specifies how the agent would act and make decisions. *Strategy scripting language* is a high level language with great similarity to the natural language.

TRIGGERS

Triggers are incoming events that cause different rules to be invoked. We have identified following triggers in TAC SCM game:

Recieve_RFQ
Recieve_Order
Recieve_Offer
Send_Order
Send_Offer
Send_RFQ

DATA OBJECTS

For each event there is a list of predefined data objects. *Objects* are defined as a set of data structures that encapsulates required data.
For example – "Deal history" object contains data about previous deals made by the agent. "Production history", "Customer orders history", "Supplier orders history", "Profit history", "Delivery history", "Factory utilization" and "CPU Component history" are other examples.

FUNCTIONS

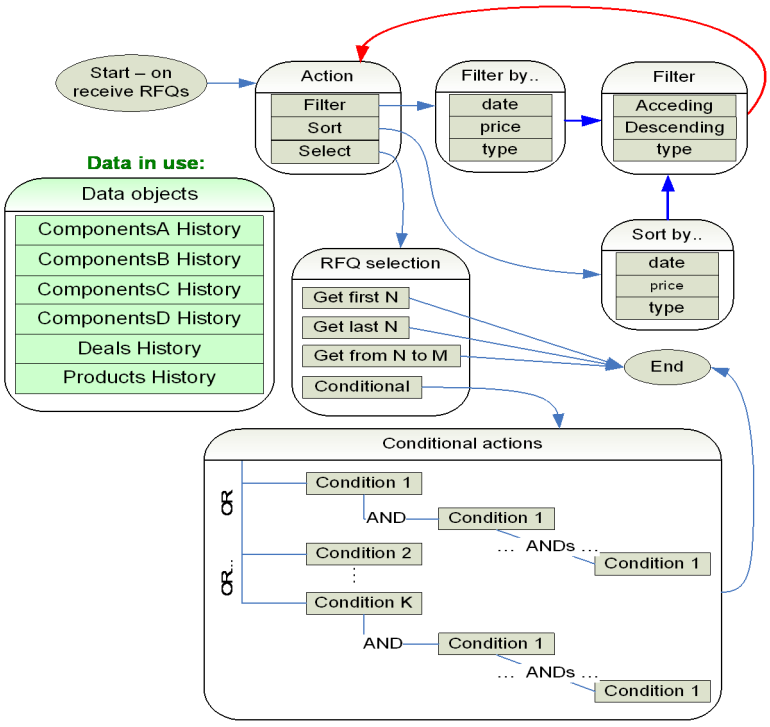
Actions in strategy plans are modeled to be represented as *functions* in scripting language definition. They are predefined for each particular event. Examples of functions are "keep count of ...", "order from supplier ...", "do like ..." and "do nothing".

OPERATORS

The strategy builder tool employs following operators:

Period processing: terms like "today", "yesterday", "before", "… days ago", "… days before", "ever", "during", "when", etc.
Comparative: terms like "better than", "like (equal)", "more than", "faster than", etc.
Evaluative: terms like "good", "better", "fast", "slow", "efficiency", etc.
Calculative: terms like "more", "less", "count of", etc.
Process flow control: cyclic operators with implicit number of loops.
Critical values: terms like "max of …", "average of …", "min of …", etc.

Figure 3
Shows a strategy construction on "Customer RFQs are received" event



STRATEGY SCRIPTING LANGUAGE EXAMPLES

We consider three different examples of strategy plans in scripting language:

Example one:
If 'efficiency_of' 'deal' 'yesterday' (was) 'better than' 'M days ago' then 'keep_count_of_CPU_component' 'today' 'like' 'count_of_CPU_component' 'yesterday'.

This strategy is based on comparing the efficiency of two deals, the one made yesterday with another one made M days ago. The data object "*deal history*" contains the efficiency value for previous deals. In case of true the second part is executed. The data object "*CPU component history*" contains the quantity of CPU components.

Example two:
If 'efficiency_of' 'supplier_order_of _CPU_Supplier_A' 'during last T days' (was) 'going worse' then 'order_from_CPU_Supplier_B'

The second strategy plan compares the efficiency of CPU orders from supplier A against supplier B. In this example data object "*Supplier orders history*" I used to gain data on passed CPU orders from different suppliers.

Example three:
'do filtering by price'; 'do filtering by date'; 'do sort by due date, acceding'; [if 'last_time' 'max_of' 'factory utilization' (was) 'more_than' 'average_of' 'factory_utilization' 'within K days_before_day M' then 'select more requests_than ' 'last_time']

This strategy is defined for "*Customer RFQs are received*" event. It filters and sorts them, then it uses "*Factory utilization history*" object to extract data for calculating the number of requests. Figure 2 and 3 shows strategy design using this tool.

STRATEGY DESIGN FOR SOARS

In the SOARS Beer Game, it is required that the user is familiar with SOARS simulation language to be able to design strategies (Manabu Ichikawa, Yuhsuke Koyama and Deguchi, Hiroshi 2008).

We suggest using aforementioned described strategy design framework which makes the user capable of designing strategies without any programing or simulation language knowledge. The user can create strategies graphically by using strategy plan builder tool as described. The strategies are further parsed to generate strategies in strategy scripting language and saved in text files. Hence it is necessary to modify the agent to use produced text file as input.

KNOWLEDGE BASE

Knowledge is represented as a set of rules in form of horn clauses (IF <antecedent> THEN <consequent>). These rules might be inserted in the knowledge base before the game start as game specific rules or during the game as a consequence of learning process.

Rules are invoked at different situations in order to assist the agent in its decision process. Knowledge base is accessed on every incoming event during the game and corresponding rule is invoked.

The agent program utilizes the knowledge base to facilitate the search process as follows:

- Consult the knowledge base to find the generic knowledge that subsume or closely match the current situation in the game tree.
- Identify which of the knowledge listed for those generic situations in the knowledge base appear to be important for this particular situation.
- For each of these search cases identify which of the knowledge described in the knowledge base seem best suited for this particular case.

There are two different classes of knowledge in the knowledge base, the expert knowledge and the common knowledge.

Expert knowledge

This is the type of knowledge that is deduced during the simulation. Expert knowledge is vital for our search process and is stored in the knowledge base in tables and accessed for insertion, accumulation and update.

We use knowledge base technology in combination with CPDT search algorithm to make it possible to search the giant tree that we get during the simulation.

In order to decrease the search time we have to limit the search space. To do this we need to have the corresponding knowledge component or its subsume in the knowledge base.

In case of knowledge component exists:

- If yes; we terminate the search and replace it with the knowledge component that exists in the THEN part of this component.
- If no; we continue the search. At the end of the game we will deduce a new knowledge component and insert it into the base.

When multiple alternative knowledge components exist for addressing a particular search case in the knowledge base, we utilize the grading system (grade of applicability). Every knowledge component has a grading associated with it. It is a variable that contains the number of cases where obtained result is same as the expected one. The rule with the highest grade will be chosen to be applied.

The results R = {P_1, P_2 ..., P_n} and R' = {P_1, P_2 ..., P_n} are assumed to match if the corresponding P_i parameters differs from each other within the ΔP_i range of acceptable values defined for P_i. The number and quality of rules increases with the agent's game experience.

We describe expert knowledge accumulation mechanism by some examples:

When five or less days remain to the end of the game there is no profit to buy any parts since there is no time to assemble and sell the products.

Example1:

We had a case where we did buy products on four days to the end of the game and since they were abandoned in the repository we did not have any profit on them. Meanwhile we did not find any knowledge component that described the current situation therefore we added following knowledge element to the base:

IF left_days < 4 THEN Reject_RFQ.

Example2:

We received a RFQ on five days to the end of the game. We looked into the knowledge base and again did not find any knowledge element that is applicable for this situation.

Thus we continue simulating ahead and find out that we will not have any profit on this RFQ. Thus we will normally add following rule into the base:

IF left_days < 5 THEN Reject_RFQ.

But before adding this new element we find subsume element to that rule which is:

IF left_days < 4 THEN Reject_RFQ.

Hence we update already existing element resulting:

IF left_days < 5 THEN Reject_RFQ.

Having this knowledge element in the base, when we get a RFQ we will check the day and if it is less than five we will terminate the search.

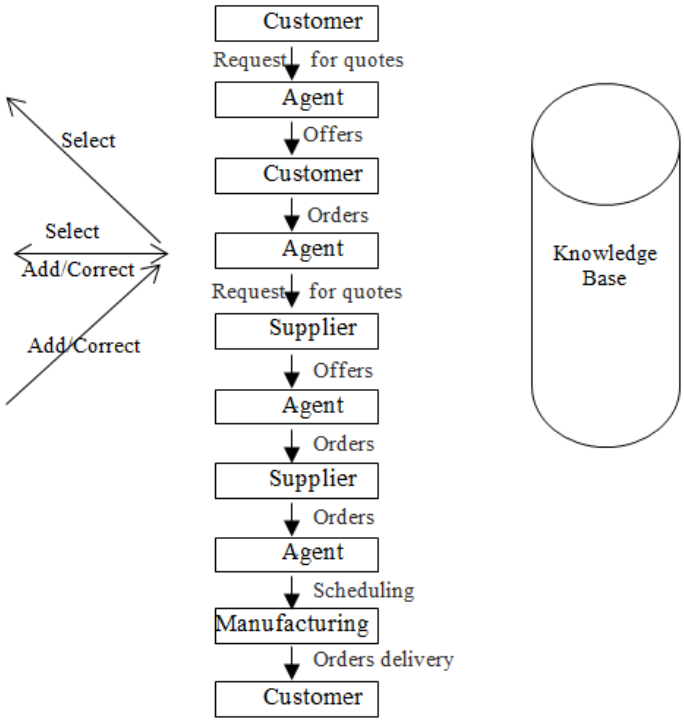
Example3:

Assume having following rule in the base:

IF days < 5	THEN	Reject_RFQ	with grading 8
IF penalty > 500	THEN	Reject_RFQ	with grading 2

Rule with the highest grading will be applied, as the first rule in this case.

Figure 4
Shows a single game cycle



Common knowledge

This type of knowledge also called game specific knowledge includes rules that describe the general game regulations. They are inserted in the knowledge base before the game start and are triggered by incoming events during the game. We call these incoming events triggers.

Examples:

IF	Receive_Offer	THEN	Send Order
IF	Receive_Order	THEN	Get Parts
IF	Send_Order	THEN	Bid_on_ProductX

GAME FLOWCHART

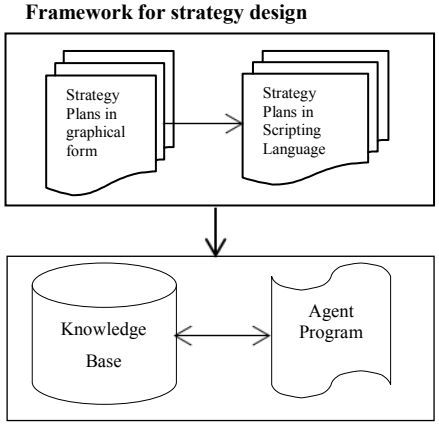
On every game cycle the agent sequentially deals with customers and suppliers and accesses the knowledge base. Figure 4 shows a single game cycle where the interactions between the agent program and the knowledge base are pointed out. In practice we will have numerous game cycles running in parallel during the simulation.

ARCHITECTURE

Strategy plans are created by the user and parsed into scripting language by the strategy builder tool.

The agent takes strategies as input and acts upon them. Figure 5 shows the architecture for strategy builder tool and the agent.

Figure 5
Shows the architecture for strategy design framework and the agent



CONCLUSION AND FUTURE WORK

A knowledge-based approach for organization and application of strategy plans for trading agent competition for supply chain management is represented. We have also employed a mechanism to optimize the knowledge base by inserting and updating knowledge elements in the base. We have presented methodology to address the issues for TAC SCM such as developing strategies, modeling markets and decision making.

We have employed TAC SCM (Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2007) game as a simulation platform for applying and proving our methodology. In addition, we have shown that it can also be employed by various simulation games such as SORAS beer game.

The key benefit for suggested strategy design framework is that it is easy for a non-programmer user to develop strategies using proposed graphical tool (Baghdasaryan T. 2007). The strategies in scripting language are easy to read and understand since the scripting language format is very close to natural language. Application of knowledge base technology our agent will increase its efficiency but decreasing its search space that further decrease its search time. This approach allows the agent to add new knowledge to its knowledge base and optimize it. Amount and quality of the knowledge increase with the agent's game experience.

The power of our approach lies in its ability to model and simulate all possible situations that may result from all possible moves of all participants (Pogossian E. 2005). Mentioned advantage moreover addresses the decision making issue which is a major issue in supply chain management domain.

We have also enhanced the quality of knowledge in the knowledge base by presenting an approach to deduce new knowledge to insert and update our base. The knowledge in the base is arranged by knowledge type (common and expert knowledge) and their degree of applicability and is represented as a set of rules in form of horn clauses (IF <antecedent> THEN <consequent>).

The downside of our approach is the huge branching of the game tree caused by the combination of high number of environmental parameters. This problem has been addressed by employing the Common Planning and Dynamic Testing (CPDT) algorithm (Pogossian E., Baghdasaryan T., 2003).

A game tree approach is suggested for modeling agent-customer and agent-supplier parts in TAC SCM domain (Baghdasaryan T., Danielyan E., Pogossian E. 2006). It simulates the behavior of an agent competing in both markets in order to find the best acting strategy.

It remains to accomplish the implementation phase of suggested approach and recommend a methodology for strategy assessment and selection. A strategy test framework is also highly required for testing and evaluating strategies.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to Professor Edward Pogossian for supervision of this work and to Emma Danielyan, Hugh M. Cannon and Chris Scherpereel

for their constructive comments.

REFERENCES

Chussil M., Reibstein D. (1994), "Strategy Analysis with Value War". *The SciPress*.

Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. (2007), "The supply chain management game for the 2007 trading agent competition". *Technical Report CMU-ISRI-07-100, Carnegie Mellon University,* Pittsburgh, PA 15213.

Baghdasaryan T. (2007), "Scripting Language and Design Tool for Trading Agents Strategy Plans", *International Conference CSIT2007*, 6p., Sep24-28, Yerevan.

Baghdasaryan T., Danielyan E., Pogossian E. (2005) "Testing Oligopoly Strategy Plans by Their On the Job Performance Simulation", *International Conference CSIT2005*, 8p., Sep19-23, Yerevan.

Baghdasaryan T., Danielyan E., Pogossian E. (2006) "Supply Chain Management Strategy Provision by Game Tree Dynamic Analysis", *Fifth International Conference SBM2006*, 2p., Sep04-08, Sevastopol.

Baghdasaryan T., Grigoryan A., Naghashyan Z. (2009) "Development of a Scripting Language Interpreter for Acquisition of Expert Knowledge in a Regular Way", *International Conference CSIT2009*, 5p., Sep28-Oct02, Yerevan.

Manabu Ichikawa, Yuhsuke Koyama, Deguchi, Hiroshi (2008). "Human and Agent Playing the Beer Game". *Developments in Business Simulation and Experiential Learning Volum 35*, P-36.

Moradkhani A. (2011) "On Trends in Supply Chain Management Trading Agent Competition", *Proceedings of the CSIT2011, 8th. International Conference in Computer Science and Information Technologies*, 190p, Yerevan.

Pogossian E. (2001) "Focusing Management Strategy Provision Simulation", *Proceedings of the CSIT2001, 3th. International Conference in Computer Science and Information Technologies*, 5p, Yerevan.

Pogossian E. (2005), "Combinatorial Game Models For Security Systems", *NATO ARW on Security and Embeded Systems"*, Porto Rio, Pataras, Greece, Aug. 20058-18p.

Pogossian E., Baghdasaryan T., (2003) "Cutting Time of Strategy Search By Sequential Quantification of Management Plans", *Proceedings of the CSIT2003, 4th. International Conference in Computer Science and Information Technologies*, 8p, Yerevan.

SOARS [Available at www.soars.jp].