An Autonomous Agent for Supply Chain Management

Shimon Whiteson and Saurabh Amin
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA
{shimon@cs,samin@mail}.utexas.edu
http://www.cs.utexas.edu/~{shimon}

1. INTRODUCTION

Supply chain management (SCM) is a set of approaches used to efficiently integrate suppliers, manufacturers, warehouses, and stores so that merchandise is produced and distributed in the right quantities, to the right locations, and at the right time, in a way that minimizes systemwide costs and satisfies service level requirements [13]. Two of the traditional challenges faced in SCM are: 1) system-wide optimal solutions are hard to obtain and 2) uncertainty is inherent in every supply chain. Traditionally, marketing, distribution, planning, manufacturing, and purchasing processes within a supply chain operate independently without much interaction. Since the stakeholders in a supply chain might have different goals and constraints, a system wide optimum in uncertain environments is very hard to obtain using conventional operations research techniques.

Recently, there has been great interest in managing supply chains using multi-agent approaches [6, 14]. Multi-agent approaches for SCM have several important potential benefits. First, local tasks can be designed to coordinate with each other so that global efficiency is achieved. Secondly, the approach can serve as a decision support test-bed which can test strategies in real time settings. Lastly, a multi-agent approach can provide control elements which communicate and control the flow of products in supply chains in an autonomous way.

In this paper, we present an autonomous trading agent for the 2003 TAC SCM game specification [1, 12]. TAC SCM is designed to capture a real world supply-chain scenario in the PC manufacturing market. The uncertainty, dynamism, and highly competitive environment of the game make analytical solutions infeasible [11].

Our agent capitalizes on using existing approaches in real supply chains to the best possible extent. We focus primarily on coordinating the agent actions between the component and computer markets. The actions in the component market include estimating demand for components, placing orders to suppliers specifying quantity and delivery times, making decisions on accepting supplier offers, and managing inventory during initial shoot-up, last wind-up, and normal day periods of the game. The actions in the customer market include making decisions on the fraction of customer demand to satisfy given factory capacity and resource constraints, and making bidding decisions that maximize profitability. Specifically, the actions in the component and computer markets are coordinated via the price and quantity information of the components in the inventory.

The rest of this paper is organized as follows: Section 2 discusses related work, Section 3 details the strategy our agent employs, and Section 4 displays the results of our controlled experiments. Section 5 discusses the implications of these results and Section 6 reviews possibilities for future work.

2. RELATED WORK

Traditional operations research and information systems literature dealing with SCM has focused mainly on a centralized decision making approach in which a single agent interacts with the environment assuming deterministic or stochastic models [10]. With recent interest in e-commerce technologies, much of the recent work focuses on designing autonomous self-interested agents with private information that can perform well in highly uncertain and competitive environments. A lot of this work deals with developing frameworks for multi-agent implementation. Different researchers have followed different approaches in various domains [4, 7, 18]. Some of the prior work also focuses on market-based approaches to complex domains [17, 15]. Auction theory has been a recent research focus in agent-based SCM and many research efforts are ongoing in the area of combinatorial auctions [16] and reverse auctions. Given the significant benefits of strategic interactions, the agent designers would also like to apply game theoretic tools to SCM. There has been some work in this direction too [3, 5].

This research aims at combining some of the current practices in SCM with experimental manipulation, rules of thumb, and empirical analysis to study the relationship between the component and computer markets. Through regular class competitions, we had a chance to monitor the agent's performance in different economies. In essence, our agent employs price and quantities of component inventory as the coordination mechanism between computer and component markets. We pay little attention to aggressive procurement strategy in the beginning of game and attempt to maintain the inventory to the levels that are actually required.

3. METHOD

The basic premise of our agent is based on a "pull" supply chain management strategy. We bid on customer RFQs as aggressively as our factory capacity can support and manufacture computers only in response to specific customer orders. However, we also make two important modifications to this general approach that are designed to improve the agent's robustness.

Firstly, we maintain buffers of component inventory. While we do not maintain any computer inventory and our component orders are based on estimates of customer demand, we do stock significant surpluses of each type of component. This increases the probability that our agent can perform well even when customer demand deviates from our estimates or supplier orders do not arrive on time.

Secondly, our agent computes projections of component inventory before responding to customer RFQs and restricts its bidding to those requests which it believes it will have the necessary components to fill. For the most part, our bidding strategy is based on available factory capacity and we rely on our component procurement strategy to respond to the orders we receive (not the other way around). However, this approach is tempered by the addition of an important safeguard: computing projections of component inventory and refraining from bidding on RFQs for which the necessary components will clearly be absent.

Sections 3.1 and 3.2 detail the two major parts of our agent strategy: procuring components and bidding on customer requests, respectively. These two parts are connected by a production and delivery scheduler lifted directly from the BaseAgent strategy. This connecting component uses a greedy algorithm to produce and deliver any outstanding orders.

3.1 Procuring Components

Procuring components consists of three tasks: forecasting demand, calculating order quantity and expected lead time, and accepting or rejecting supplier offers. The inventory management challenges which are posed by the game are:

1. Customer demand is highly uncertain.
2. Significant uncertainty also exists in component market because of limited capacity of suppliers and other agents competing for the supplies. This leads to uncertainty in the quantity to order, the costs at which the supplier offers can be accepted, and the supplier delivery times.

3.1.1 Demand Forecasting and Order Quantity Calculation

A typical supply chain has raw material inventory as well as finished product inventory. However, since our agent manufactures on a make-to-order basis, it only has to deal with raw material inventory.

The simulation year of 220 days can be considered to be comprised of 11 months each having 20 days. Each month can be further considered to be comprised of 4 weeks each having 5 days. Our agent uses a powerful inventory management strategy which is used in real supply chains. Whenever the component inventory is below a certain level s, the *reorder level*, the agent generates an RFQ for that component. RFQ generation is targeted to increase the inventory level of the components to level S, the *order-up-to level.*

Daily demand of PCs is generated in the product market in the form of customer RFQs which have information about the type of PCs required, quantity, due date, penalty, and reserve price. The actual demand is generated through a stochastic process described in the TAC specification. The daily demand of each component can be defined as the number of components required to manufacture the PC orders won by the agent. The agent aims to have inventory levels that satisfy 1/6 of the PC demand faced in the product market. We assume that the average daily demand $\bar{D}_c$ of each component c follows a normal distribution.

Every game day t, the number of units D_c^t of each component c required to satisfy the daily demand of PCs is calculated. If the day is not the beginning of month, the average daily demand of each component is updated using exponential smoothing with a locally constant mean [2] as follows:

$$\bar{D}_c^t = \bar{D}_c^{t-1} + \omega(D_c^t - \bar{D}_c^{t-1})$$

where ω is the smoothing constant. The choice of ω determines the weight given to the daily fluctuation from the average demand. It is set to 0.1 in our experiments. At the start of each day, the average demand of each component is reset to the *actual average daily demand* which the agent observed during the previous month. In order to calculate the actual average demand for each component, we use market report information. The market report is generated by the server at the start of each month (i.e. every 20 days) and has information about the quantity of each PC type ordered by the customers during the previous month m. The quantity of each component that was required to fulfill the entire customer demand during the previous month M_c^{m-1} is calculated from this information. The average daily demand during the previous month can be calculated as:

$$\bar{D}_c^m = \frac{\bar{M}_c^m}{20}$$

where $\bar{M}_c^m$ is the average monthly demand at the beginning of each month and is calculated as:

$$\bar{M}_c^m = \bar{M}_c^{m-1} + \omega(M_c^{m-1} - \bar{M}_c^{m-1})$$

The agent decides the order quantity of each component based on current average daily demand $\bar{D}_c^t$. In addition, we need information about the standard deviation of the daily demand σ_c^t faced by the agent, the expected replenishment lead time L_c^t from the component supplier in days, and current inventory position I_c^t. Finding a good estimate of the standard deviation of daily demand is very difficult; for the present implementation, it is calculated as:

$$\sigma_c^t = (\frac{\bar{M}_c^m}{20} - \bar{D}_c^t)$$

The expected lead time is set to 5 days at the start of each game and is updated whenever a supplier offer with due date T is accepted by the agent as follows:

$$L_c^t = min(20, 0.5(L_c^{t-1} + min(T - t, 20)))$$

The current inventory position for every component can be calculated as the actual inventory available plus the quantity that is ordered by the agent but not yet arrived, minus the back-ordered quantity. The back-ordered quantity is the quantity which the agent promises to deliver by a particular date but ends up supplying partial order due to its capacity constraints. We do not consider back ordered quantity in our implementation. After calculating $\bar{D}_c^t, \sigma_c^t, L_c^t$, and I_c^t the agent calculates the reorder level s_c^t and order-up-to-level S_c^t for each component as follows:

$$s_c^t = L_c^t \bar{D}_c^t + z\sigma_c^t \sqrt{L_c^t}$$

$$S_c^t = \sqrt{q\bar{D}_c^t} + s_c^t$$

The first term in the calculation of the reorder level is the average demand faced by the agent during the lead time. This will ensure that the agent has enough components to meet the expected demand during the lead time. The second component is the safety buffer to be kept by the agent to meet deviations from the average daily demand. The quantity z is the safety factor and corresponds to required service level which is the probability of not stocking out during the lead time. We keep required service level at 95% in this implementation. The safety factor to meet the required service level should satisfy: $P(\text{demand during lead time} \geq s_c^t) = 1 - 0.95$. The corresponding safety factor can be calculated from standard normal charts and comes out to be 1.654.

The order-up-to-level is calculated by adding economic order quantity [13] which accounts for average demand to the reorder level and variability in demand. The economic order quantity assumes constant demand, fixed order quantities, constant lead time, a fixed cost for placing an order, and fixed cost of holding an inventory. According to this model, the order quantity to be placed each day is $\sqrt{qD_c}$ where the parameter q depends on the cost of placing an order and inventory holding cost. Though the present game specification does not consider these costs, we still put bounds on our inventory levels and expect that the game specifications will incorporate these costs in the future. For the present implementation, we found a q value of 14000 to work well. This value can be calculated if the inventory holding costs are specified.

We also attempted to incorporate the standard deviation of component lead times σ_L^t in our implementation. However, as is the case with σ_c^t, it is tough to get a reliable estimate of σ_L^t. Our crude estimates of standard deviation of delivery lead times did not prove to be particularly helpful. If a good estimate of σ_L^t is available, the reorder and order-up-to levels can be calculated as follows:

$$s_c^t = L_c^t \bar{D}_c^t + z\sqrt{\sigma_c^{t^2} L_c^t + \bar{D}_c^{t^2} \sigma_L^{t^2}}$$

$$S_c^t = \sqrt{q\bar{D}_c^t} + s_c^t$$

After calculating the reorder and order-up-to levels every day, the agent generates its desired order quantity for component c if the current inventory position is less than the reorder level i.e. $I_c^t < s_c^t$. The desired order quantity for such components is given by:

$$Q_c^t = min(S_c^t - I_c^t, Q_{c,max}^t)$$

where $Q_{c,max}^t$ is the maximum order quantity that should be sent to the supplier. There are two reasons why it is necessary to put bounds on the number of components the agent can order each day. First, the suppliers have limited capacity to produce and so it is reasonable to send RFQs only to those suppliers who we expect to receive a reply from. Second, if the suppliers happen to make an offer in response to an RFQ with this quantity and the agent finds the order unacceptable, the agent's reputation will suffer. The maximum order quantity can be calculated from the total quantity of each component delivered by the suppliers during the last month. This information is available in the market report and is updated at the beginning of every month using exponential smoothing. The number of components delivered by the suppliers gives an indication of the additional inventory added to the existing inventory levels of the agent as well as the suppliers' current production capacity.

Note that customer demand, supplier costs, and capacities are symmetric which implies that each PC is equally likely to be produced. Each CPU appears in 1/4 of PCs and each non-CPU appears in 1/2 of PCs. However, since non-CPU components are manufactured by two suppliers, they share the production equally. Therefore, the desired order quantities of CPU and non-CPUs components can be calculated using the same formula.

3.1.2 Placing Orders to Suppliers

Whenever the desired quantity Q_c^t is generated for a component, an RFQ is sent to the supplier. However, since the supply procuring strategy at the beginning and end of the year will have significant impact on the agent performance, we distinguish these from all other days. In the present implementation, the agent has different RFQ generation policies for the first 10 days, the last 10 days, and all other days.

With the present game specification, day-0 procurement strategy has a significant impact on the inventory levels throughout the game and on the character of the game environment [5]. During the preliminary class competitions, little was known about the importance of start game procuring strategy or advantages that an agent can get by being aggressive on day-0. To counter the aggressive buying behavior, the original game specification was modified with starting prices of the components being randomly selected between 50-100% of the base price. However, this modification does not seem to have much influence on prevention of day-0 aggressive behavior and hence we also chose to be aggressive in buying components at the start of game. Our agent sends RFQs for 8000, 4000, 2000, and 1000 components to each supplier on day-0. For the remaining 9 starting days, the agent places RFQs for Q_c^t, $Q_c^t/2$, $Q_c^t/4$, and $Q_c^t/8$. All the RFQs for the first 10 days are due the next day.

On all other days, RFQ bidding is done using the expected lead time L_c^t which is updated whenever a supplier makes an offer, as described above. If there is sufficient monthly inventory present (i.e. current inventory levels are greater than $\bar{M}_c^m$ and $L_c^t < 5$), four RFQs are generated each for quantities $Q_c^t/4$ due after L_c^t days. Otherwise the due date remains at 5. The agent starts buying components more aggressively when it sees the current inventory level drop below the estimated monthly demand. If the inventory level of a particular component drops below 300, the agent immediately places RFQs for small quantities which are due over the next 5 days. For the last 10 days of the game, the agent does not place any RFQs if the inventory levels are above 300.

3.1.3 Accepting Supplier Offers

Ideally, the agent should place RFQs such that it will want to accept most offers made by the suppliers. However, when prices are low, component procurement is highly competitive. Also, there is significant uncertainty associated with supplier capacity. Therefore, the agent does not want to buy unusually expensive components especially when it has sufficient inventory to meet the short-term and near long-term expected demand. Our agent incorporates this decision making ability by defining the acceptable price under the current customer demand and inventory levels. In order to take the current customer demand level into account and decide on the acceptable price, the equilibrium price can be calculated from customer demand levels. Several approaches are available for determining the price equilibrium in a multi-agent setting. Recently these have been applied to supply chain management [8, 9, 11]. The equilibrium price P_c^t for component c is calculated such that on an average, the aggregate quantity of demanded at price level equals the quantity supplied. For the game specification, the steady state supply price-production quantity relationship and customer demand and reserve price relationship specified in [11] can be used to calculate the equilibrium price P_c^t and quantity relationship as follows:

$$P_c^t = \begin{cases} 0.5 + \frac{10.5}{4000} & \text{if } RFQ^t < 95, \\ P_c^{base}\left(\frac{2000+26.25RFQ^t}{4000+21RFQ^t}\right) & \text{if } 95 \leq RFQ^t < 320 \end{cases} \tag{1}$$

If the component inventory level is less than the reorder level s_c^t, the acceptable price is set to the base price of the component. If the current inventory levels are between s_c^t and $3s_c^t$, the acceptable price is the equilibrium price P_c^t plus a fraction of the component's base price. For higher current inventory levels, the equilibrium price is the acceptable price.

3.2 Responding to Customer Requests

On the customer side, the primary goal of our agent is to maintain a constant number of unproduced *order cycles.* Order cycles are simply the total number of factory cycles needed to complete a given set of orders. Unproduced order cycles are the order cycles for those orders that have not yet been sent to the factory.

On each turn, the agent computes how many more order cycles it needs to win on this turn to maintain the target, T (set to 6000 in our experiments). The number of needed order cycles, n, is simply:

$$n = T - (c - p - d)$$

where c is the order cycles for all outstanding orders, p is the number of cycles the factory will use today, and d is the order cycles for orders that are already completed but awaiting delivery. If the agent does not bid at all, the unproduced order cycles will become $c - p - d$ tomorrow. Hence, the number of order cycles the agent wants to win to reach T is simply the difference between T and $c - p - d$.

If n is negative, we simply do not bid at all. Otherwise, the agent can use n to estimate how aggressively it ought to bid by computing the *target fraction.* The target fraction is simply n/f, where f is the order cycles for *all* feasible RFQs received on this turn (the process for determining if an RFQ is feasible is described below). Since the agent will only bid on feasible cycles, the target fraction is simply what portion of the offers made on this turn need to be accepted in order reach T.

The target fraction gives the agent a rough gauge of what profit margin it should aim for when bidding on RFQs: the higher the target fraction, the lower the profit margin and vice-versa. The problem is that we do not know the function $m_i = f_i(n/f)$ that maps the target fraction to the profit margin that will cause the agent to win that fraction of its offers (m_i is the profit margin for computers of type i). If we did, the correct bid in response to any RFQ would be:

$$b_i = e_i + m_i = e_i + f_i(n/f)$$

where e_i is the estimated cost of obtaining the components necessary to make a computer of type i (we describe below how to perform this estimation).

Since this function f_i is likely to be highly dependent on the bidding behavior of other agents in the market, it does not seem feasible to learn it off-line using data from previous games. Instead, we must learn on-line by continually comparing the fraction of offers won to the target fraction and updating our function appropriately. As a simple approximation, we reduce the problem of learning f_i to the problem of learning a *cycle weight* c_i by defining the following linear relationship between the profit margin and the target fraction:

$$m_i = f_i(n/f) = C - c_i * n/f$$

where C is a constant *profit ceiling*, set to 750 in our experiments. Hence, the cycle weight c_i determines, given the target fraction, how much the profit ceiling should be reduced to obtain a desirable profit margin. Of course, f_i could easily be non-linear, in which case this representation could offer only an approximation. However, it is an approximation that is much more feasible to determine, given the constraints of learning on-line in a dynamic environment. In fact, the agent can constantly tune its approximation of f_i for each type of computer by modifying the corresponding c_i at every turn according to the following update rule:

$$c_i = c_i + \alpha * r_i * c_i$$

where α is a learning rate parameter, set to 0.25 in our experiments, and r_i is the error: the difference between yesterday's target fraction and the actual fraction of offers won. When too few orders were won, r_i is positive, which causes c_i to be increased. A larger c_i means a lower profit margin and a higher expected fraction of offers won in the future.

By estimating f_i, our agent can determine the appropriate profit margin for each type of computer. The only issues that remain are 1) how do we determine expected costs? and 2) how do we decide which RFQs are feasible? These questions are addressed in the following subsections.

3.2.1 Estimating Expected Costs

A good estimate of expected cost e_i of computer type i is important in order to decide the bid price in response to any customer RFQ. We calculate the expected price of each computer type as the sum of the expected prices of the components required to produce that computer. Our estimate of the expected price of a component is determined by exponential smoothing with a locally constant mean model (similar to the update made while calculating daily demand, monthly demand, and expected lead time). The estimate is updated every time a supplier offer is accepted. For this implementation, the weight given to the deviation of the current offer price from that of the previous expected price is 0.1.

3.2.2 Determining RFQ feasibility

An RFQ is *feasible* if the agent believes that, should it respond with an offer that is accepted, it will be able to complete the order on time. There are two factors that make an RFQ infeasible: 1) we are too close to the end of the game and 2) we do not have enough components in inventory. To address the former, we do not bid on any RFQs beyond four days before the end of the game. Since our agent does not maintain inventory of surplus computers, no orders received beyond that day could possibly be completed on time. To address the latter, we estimate, for each component type required in the computers requested, how much spare inventory we will have on the last day the computers could be produced for on-time delivery (the due date minus two days). If that spare inventory is below a given threshold (in our experiments, 50 for CPUs and 100 for all other components) for any of the necessary components, the RFQ is deemed infeasible.

To estimate, for any given type of component, how much spare inventory will be available on a given day, we consider the following four factors:

1. Current inventory: how many components of the given type are currently in our warehouse.
2. Expected additional inventory: how many more components we will have on the specified day if all current component orders arrive on time.
3. Components for unproduced orders: how many components we will need to complete the orders we have not yet produced.
4. Components for expected orders: how many components we will need to complete the orders we expect to win from bids already made today.

We can obtain an estimate of the spare inventory for a given component on a given day by adding the first two factors and subtracting the last two. This information allows the agent to estimate, for a given RFQ, whether inventory for all the relevant components will be available in time. Hence, it can determine which RFQs are feasible.

However, there is an important complication to this approach. To compute factor #4, the components required for expected orders, we need to estimate what fraction of the offers we have already placed today will be accepted. In other words, we need to know the target fraction. However, the target fraction is based on the how many feasible RFQs are available, which we are using factor #4 to compute. To avoid this circularity, the agent makes two different computations of each RFQ's feasibility. First, at the beginning of its turn, the agent iterates over all the RFQs and determines its feasibility with no regard to factor #4. These initial assessments of feasibility are used to compute the target fraction. Second, the agent iterates over the RFQs again, placing bids and updating its value for factor #4, based on the previously estimated target fraction. As bidding progresses, the value of factor #4 will increase such that RFQs encountered later may be deemed infeasible even though they were considered feasible on the first pass. The agent refrains from bidding on such RFQs. In this approach, the order the RFQs are examined is important: earlier RFQs are more likely to be deemed feasible. For this reason, we sort the RFQs before bidding such that those with the highest expected profit (i.e. lowest cycle weight) are bid on first.

4. RESULTS

Table 1 summarizes the results of the first seven phases of the class competition. Our agent, called "shimon", performs very competitively, generating on average more profit than its competitors except for "long" and "mazda." One feature that distinguishes our agent from its competitors is its consistency, as indicated by the low variance of its profit.

Table 2 shows the results of a small ablation experiment we ran to test the value of certain features of our agent strategy. The table shows the average scores (over three runs) of a TAC SCM competition held between four versions of our agent. The remaining two agent slots in each run were filled with the agents of various classmates or copies of the BaseAgent if no other agents were available. The four versions of our agent differ in whether or not they make use of two central features of our strategy: 1) aggressive bidding on day-0 and 2) checking component inventory to determine the feasibility of customer RFQs. The first version of our agent, which gets by far the best average performance, uses both of these features and is identical to the agent we used in the class competition. The second and third versions, which perform substantially worse, each have one of these two features ablated. The fourth agent, which performs still worse, uses neither of these elements in its strategy.

5. DISCUSSION

The strong performance of our agent relative to its competitors in the class tournament indicates that our strategy is both viable and effective. We suspect that the superior performance of "long" is due in large part to a more effective method of procuring components at the beginning of the game, which seems to have a dramatic impact on overall score.

Even when our agent does not get the highest score, it has the important advantage of consistency. The low variance in score for our agent in the class tournament suggests that our strategy is very robust: it can perform well, if not the best, against a wide variety of competitors.

Our ablation experiments show that two central features of our strategy: aggressive initial procurement of components and checking customer RFQs against inventory, are integral to our agent's success. Removing either of them results in dramatically worse performance.

6. FUTURE WORK

There are several ways in which our design strategy could be improved or elaborated. Firstly, our agent could be more aggressive about making use of available factory capacity. Currently, the agent builds computers only in response to specific orders and sits idle when there are no fillable orders, even if components are available to make computers. Using any extra factory capacity to build an inventory of completed computers could improve the agent's responsiveness and make it more feasible to complete orders with near due dates, as well as reduce the likelihood of suffering late penalties. However, such an enhancement is not without risk, since stocking computer inventory means giving up some flexibility: once a component has been put in one kind of computer, it can no longer be used to make another kind. Hence, such a change would require a careful analysis of these trade-offs.

Secondly, the estimations of profit margin for a given target fraction could be elaborated to allow for non-linear functions. The current assumption of linearity may be significantly limiting the accuracy of these estimates, though learning more elaborate functions on-line would be very challenging.

7. REFERENCES

[1] R. Arunachalam, R. Eriksson, J. Finne, N. Janson, and N. Sadeh. The tac supply chain management game. *Technical Report, SICS*, Draft version 0.62, 2003.

[2] G. E. C. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting and Control*. Prentice Hall, third edition, 1994.

[3] G. P. Cachon and S. Netessine. Game theory in supply chain analysis. *Supply Chain Analysis in the eBusiness Era*, 1994.

[4] P. Davidsson and F. Wernstedt. A framework for evaluation of multi-agent system approaches to logistics network management. *Multi-Agent Systems: An Application Science*, 2003.

[5] J. Estelle, Y. Vorobeychik, M. Wellman, S. Singh, C. Kiekintveld, and V. Soni. Strategic interactions in a supply chain game. *Technical report, University of Michigan AI Lab*, 2003.

[6] M. S. Fox, M. Barbuceanu, and R. Teigen. Agent-oriented supply chain management. *International Journal of Flexible Manufacturing Systems*, 12:165–188.

[7] D. Frey, T. Stockheim, P.-O. Woelk, and R. Zimmermann. Integrated multi-agent-based supply chain management. In *Proceedings 1st International Workshop on Agent-based Computing for Enterprise Collaboration (ACEC)*, 2003.

[8] T. Kaihara. Supply chain management with market economics. *International Journal of Production Economics*, 73(1):5–14, 2001.

[9] T. Kaihara. Multi-agent based supply chain management modelling with dynamic environment. *International Journal of Production Economics*, 85(2):263–269, 2003.

[10] K. Kalakota, J. Stallaert, and A. B. Whinston. Implementing real-time supply chain optimization systems. *Technical report, University of Texas at Austin, Dept. of MSIS*, 1996.

[11] C. Kiekintveld, M. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, and M. Soni, V. andRudary. Distributed feedback control for decision making on supply chains. *Tech. report, University of Michigan AI Lab.*

[12] S. N., R. Arunachalam, J. Eriksson, N. Finne, and S. Janson. Tac-03: A supply-chain trading competition. *AI Magazine*, 24(1), 2003.

[13] D. Simchi-Levi, P. Kaminsky, and E. Simchi-Levi. *Designing and Managing the Supply Chain*. McGraw-Hill, second edition, 2002.

[14] J. Swaminathan, S. F. Smith, and N. M. Sadeh. Modeling supply chain dynamics: A multiagent approach. *Decision Sciences*, 29(3), 1998.

[15] W. E. Walsh and M. P. Wellman. Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. *Journal of Artificial Intelligence Research*, 19:513–567, 2003.

[16] W. E. Walsh, M. P. Wellman, and F. Ygge. Combinatorial auctions for supply chain formation. In *ACM Conference on Electronic Commerce*, pages 260–269, 2000.

[17] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[18] D. Zeng and K. Sycara. Agent-facilitated real-time flexible supply chain structuring, 1999.