

Software Engineering and Strategies for Autonomous Trading Agents

Wolfgang Ketter

Dept of Computer Science and Engineering
University of Minnesota

Object Technology User Group
University of St. Thomas, St. Paul - March 16th, 2004

The MAGNET team: Amruddin Agovic, Alex Babanov, John Collins, Steven Damer,
Maria Gini, Ashutosh Jaiswal and Elena Kryzhnyaya.

Work supported in part by the National Science Foundation under awards

NSF/IIS-0084202 and NSF/EIA-9986042.

MinneTAC

An autonomous agent to compete in the Supply Chain Management Trading Agent Competition (TAC SCM).

- Agents, Markets, and Auctions.
- Game overview.
- Implications for the real world.
- Components of the game.
- MinneTAC architecture.
- Game analysis and Sales Strategies.
- Conclusions.

Agents, Markets, and Auctions

Our long term goal is to enable programs (“agents”) to do transactions on electronic markets on behalf of a user.

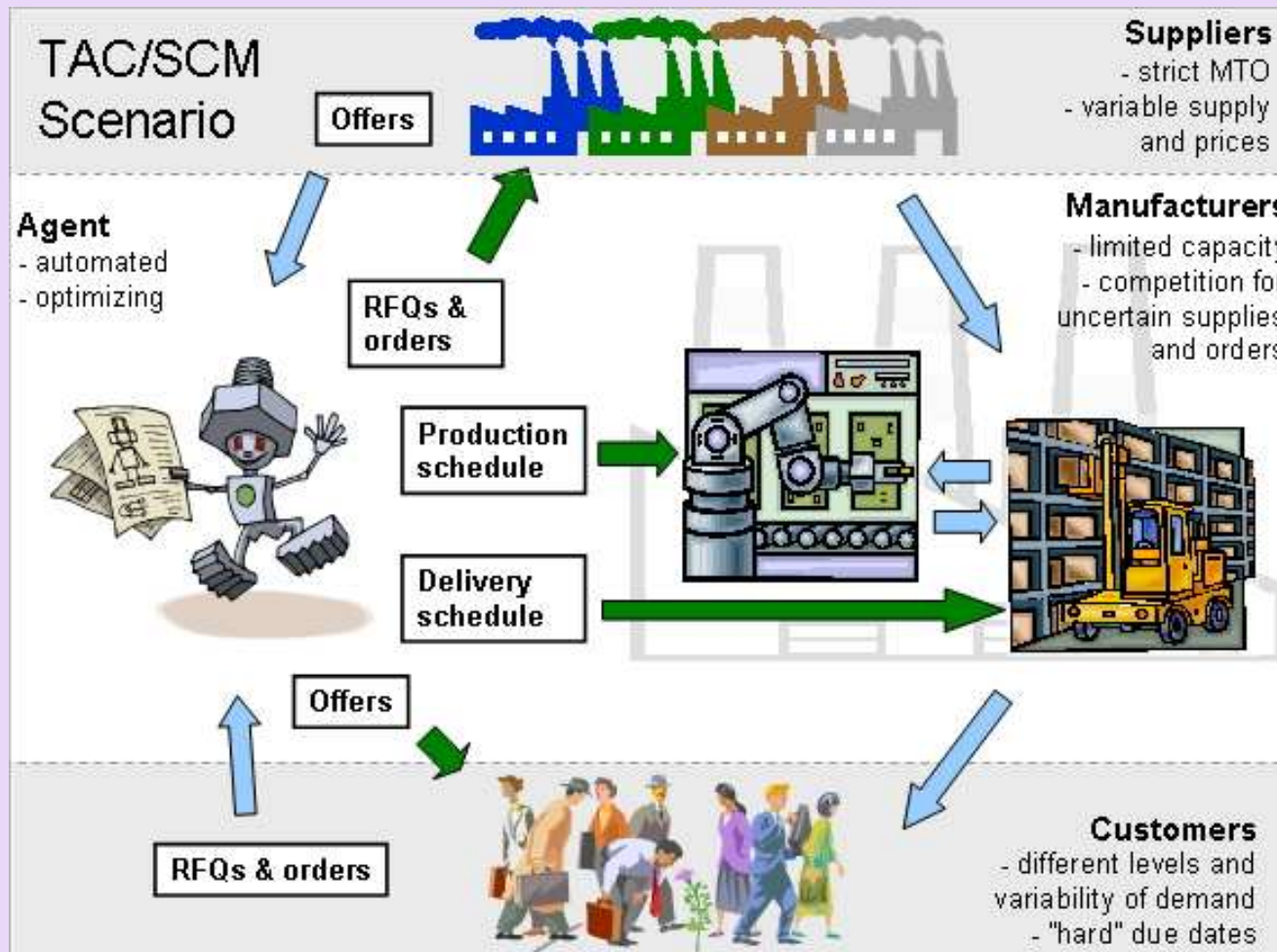
Why electronic markets and auctions?

- Electronic markets have the potential for reduced costs and increased access to world-wide markets.
- Auctions are a general and proven way to negotiate among rational entities.

TAC SCM - Game Overview

- Six autonomous agents compete to maximize profits in a computer-assembly scenario.
- Agents compete for customer orders and for procurement of various components.
- The simulation takes place over 220 virtual days, each lasting fifteen seconds of real time.
- At the end (game/tournament), the agent with the most money in its bank account is the winner.

TAC SCM - Scenario (1)



Description and rules at www.sics.se/tac.

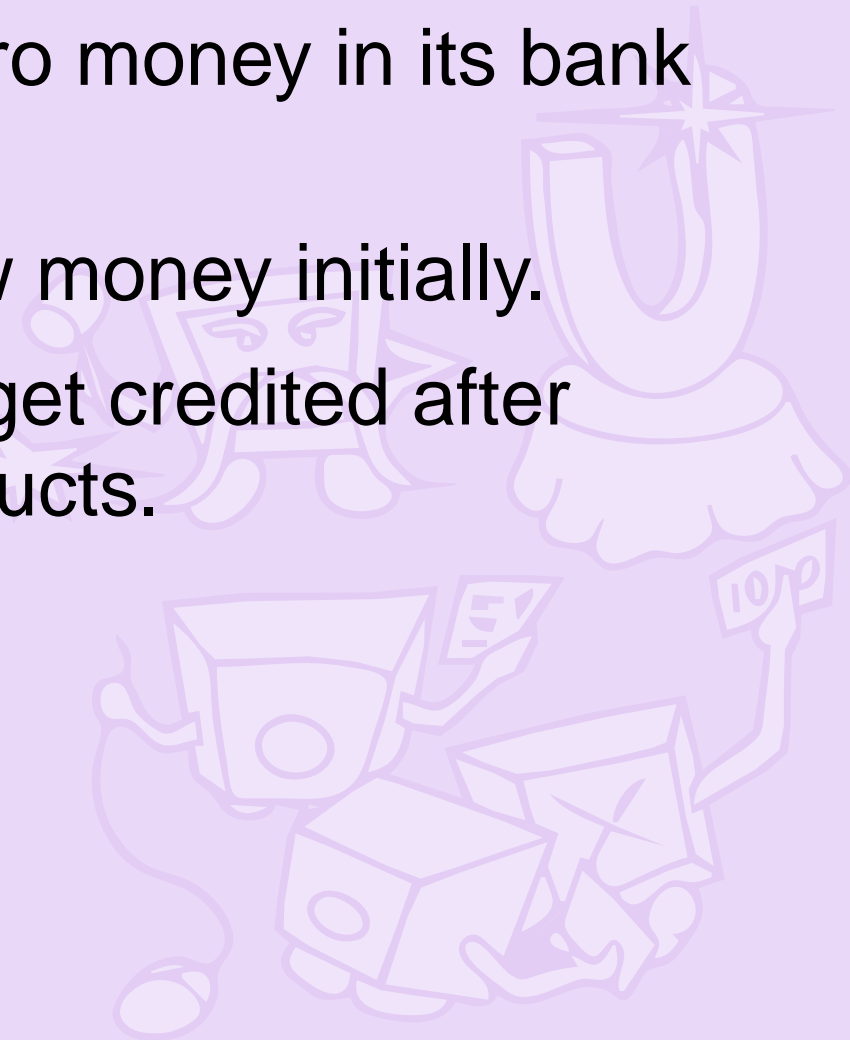
TAC SCM - Simulation of Business Environment

- Suppliers - produce raw materials
- Assembly agents
- Customers - buy assembled computers
- Raw materials market
- Finished goods market
- High degree of uncertainty
- Strategic
- Complex



TAC SCM - Simulation of the product life cycle

- Initially an agent has zero money in its bank account.
- The agent has to borrow money initially.
- Agents' bank accounts get credited after delivery of finished products.



Static vs. Dynamic Practices

- Today's supply chains are mostly static, relying on fixed, long-term trading partners. Not always optimal!
- Dynamic management allows for finding better matches between suppliers and customers as market conditions change.
- This is the goal of the Supply-Chain Management Trading Agent Competition (TAC SCM).

TAC SCM - Implications in the Real Business World

- Markets are changing quickly - Agents that have the ability to meet changing market demands in a timely and cost-effective manner will prosper.
- Decreased cost and performance improvement can be provided through new algorithms for procurement of components, production, and sales management.
- The algorithms can be used to provide information to a human decision maker.

TAC SCM - Components

- PCs are built from 4 component types: CPUs, motherboards, memory, and hard drives.
- CPUs and motherboards have two product families: Pintel and IMD.
- Given at the start of each game:
 - Component catalog: Information about the components (id, base price, supplier, name)
 - Bill of materials: Description of PC types (SKU, components, # assembly cycles)

TAC SCM - Suppliers (1)

- There are 8 suppliers in total, 2 for each component type.
- Each day an agent can send a maximum of 10 Request for Quotes (RFQs) to each supplier.
- This allows an agent to probe the supplier without swamping it with too many messages.

TAC SCM - Suppliers (2)

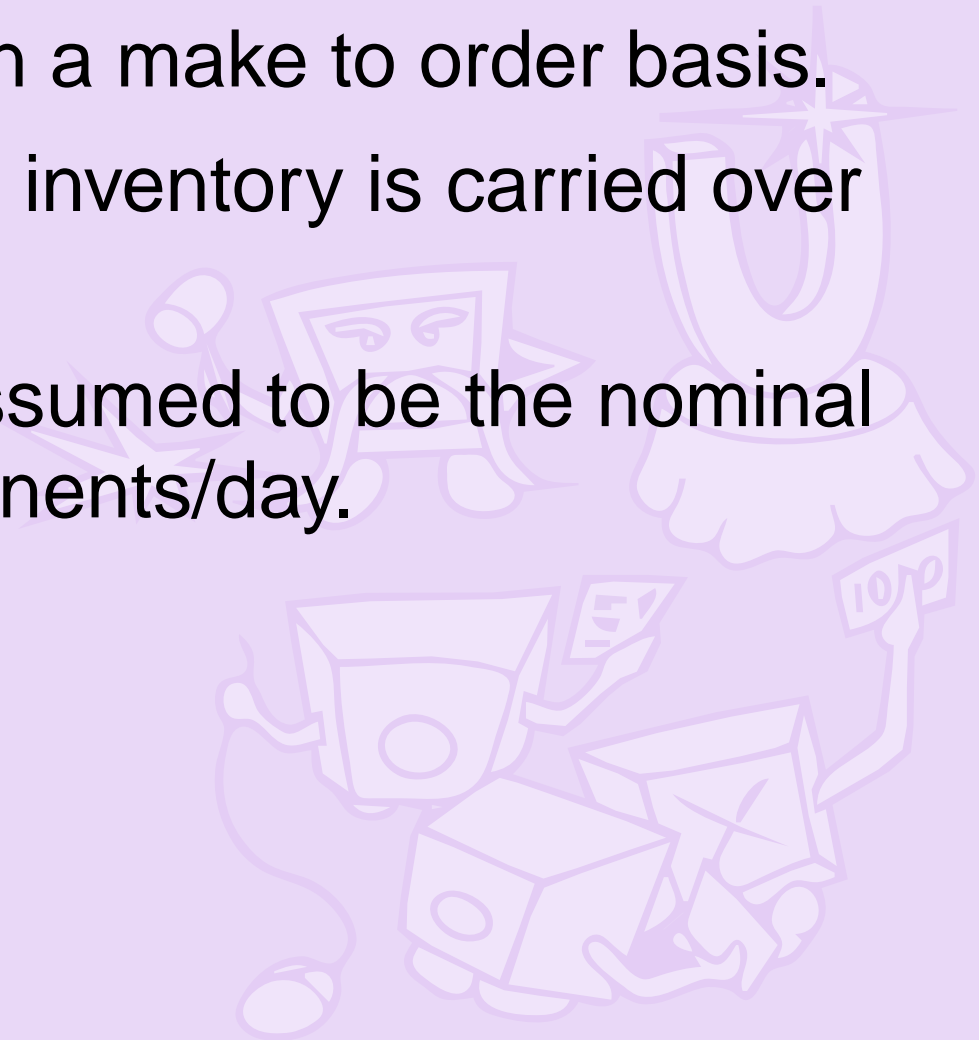
- The RFQs are bundled in an ordered list, going from highest to lowest priority.
- Each supplier collects all the RFQs from the agents and processes them at the end of the day.
- The supplier chooses an RFQ bundle and considers the next unprocessed RFQ in that bundle.

TAC SCM - Suppliers (3)

- RFQ response comes in the form of an offer:
<Offer-id, RFQ-id, Quantity, DueDate, Price>
- If the order cannot be filled in its entirety then two amended offers are sent: a partial offer and an earliest complete offer.
- The agent must then decide which offer, if any, to accept (it cannot accept both).

TAC SCM - Suppliers (4)

- Suppliers produce on a make to order basis.
- Excess capacity and inventory is carried over with no cost.
- Future capacity is assumed to be the nominal capacity, 500 components/day.



TAC SCM - Customers (1)

- Request PCs of different types to be delivered on a certain due date.
- The quantity of each order is chosen uniformly between [1,20].
- Agents must bid to satisfy the entire order (both in quantity and due date) for a bid to be acknowledged.

TAC SCM - Customers (2)

- Customer demand is expressed as $\#RFQ_s$.
- $\#RFQ_s = \text{Poisson}(\overline{\#RFQ} \text{ per day})$.
- $\overline{\#RFQ}$ per day varies using a trend updated with a random walk:
- $\overline{\#RFQ} = \min(320, \max(80, \overline{\#RFQ} \times trend))$
- $trend = \max(T_{min}, \min(T_{max}, trend + \text{random}(-0.01, 0.01)))$
- The start value of $\overline{\#RFQ}$ is chosen uniformly in the interval $[RFQ_{min}, RFQ_{max}]$
- The start value of $trend$ is 1.0 (reset at Opt.)

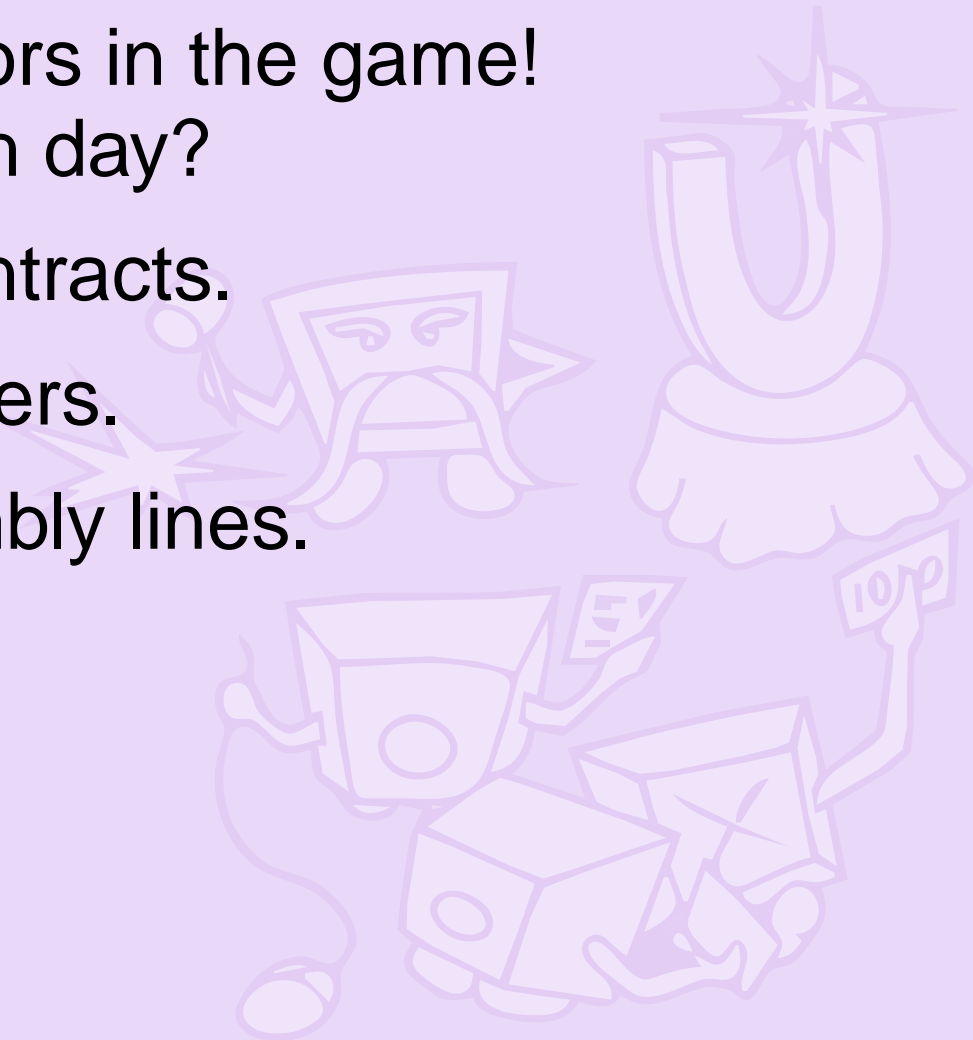
TAC SCM - Customers (3)

- All parameters of each RFQ are chosen uniformly between specified intervals.
- $(Qty_{min}, Qty_{max}) = (1, 20)$
- $(DueDate_{min}, DueDate_{max}) = (3, 12)$
- $(Penalty_{min}, Penalty_{max}) = 5\% - 15\%$ of customer reserve price.
- Customer reserve price = 75% - 125% of components base price.

TAC SCM - Assembly Agents (1)

These are the competitors in the game!
What do agents do each day?

- Negotiate supply contracts.
- Bid for customer orders.
- Manage daily assembly lines.



TAC SCM - Assembly Agents (2)

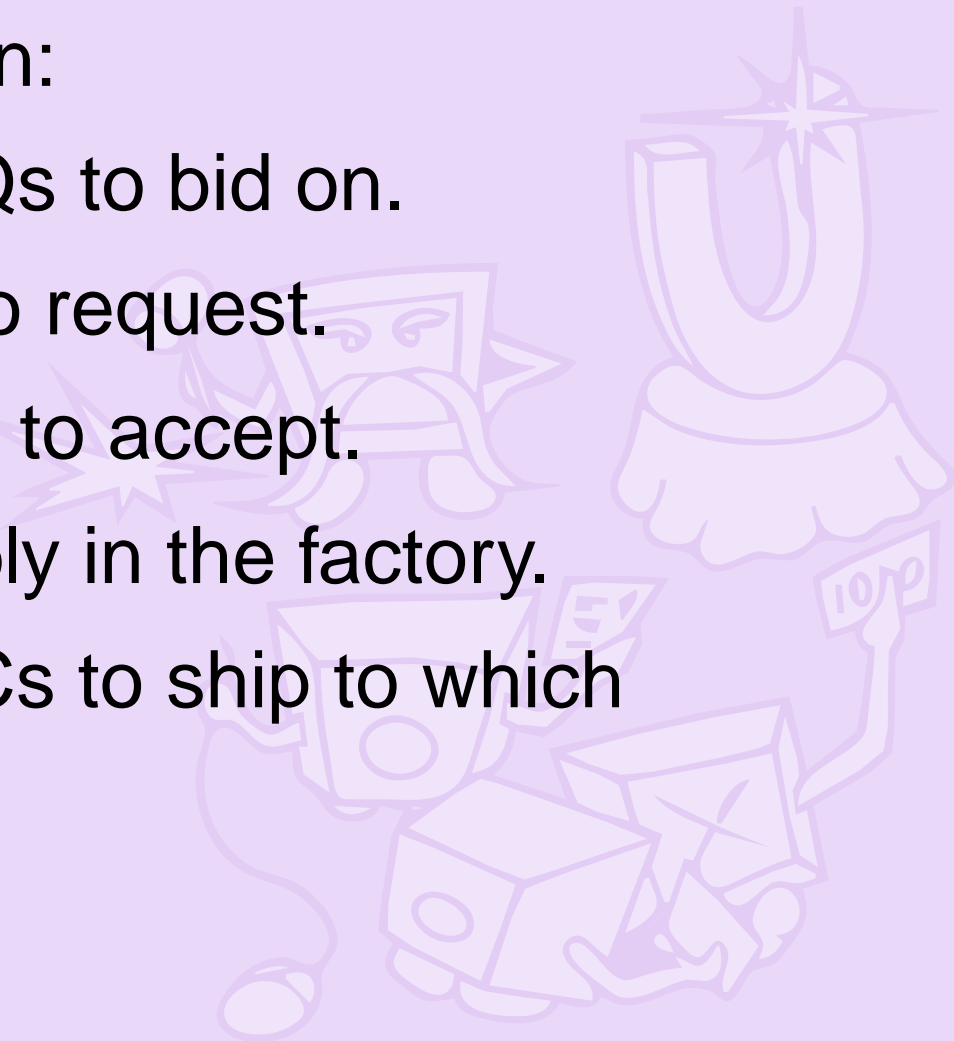
Each day the agents receive:

- RFQs and orders won from customers.
- Quotes and delivery of parts from component suppliers.
- Bank account statement.
- PC and component inventory report from the factory.

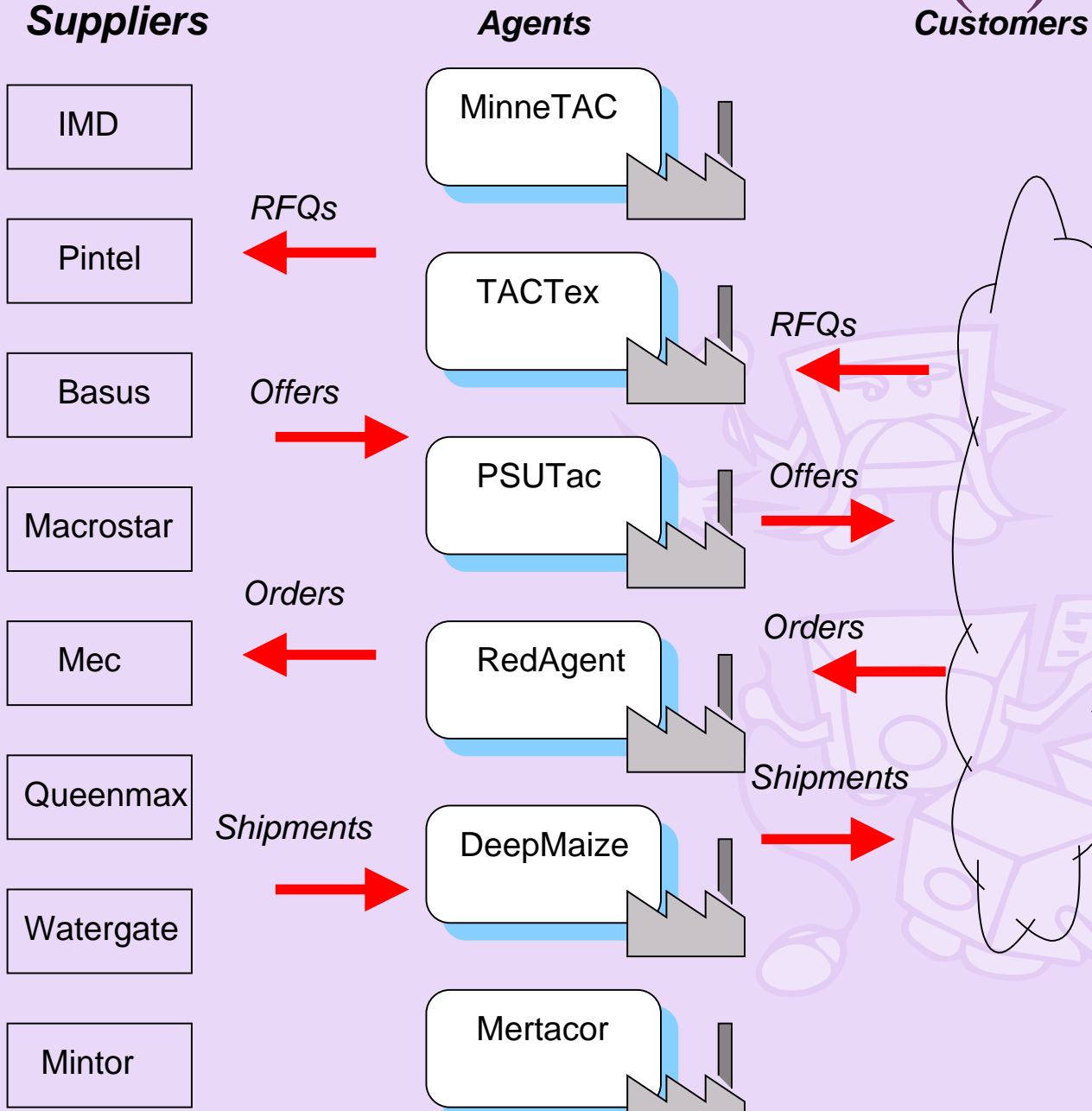
TAC SCM - Assembly Agents (3)

Each day the agents plan:

- Which customer RFQs to bid on.
- Which components to request.
- Which supplier offers to accept.
- What PCs to assemble in the factory.
- Which assembled PCs to ship to which customers.



TAC SCM - Scenario (2)



MinneTAC - Design Challenges

- Clean separation of infrastructure from decision processes.
- The design must support multiple independent developers pursuing their own lines of research.
- We need to be able to configure agents with different combinations of decision process implementations.
- Controllable generation of experimental data.

MinneTAC - The Design

We chose an component-oriented approach.^a
Two pieces form the foundation of MinneTAC:

- Avalon component framework
- “Dummy agent” distributed by the TAC SCM game organizers

J. Collins, A. Agovic, S. Damer, and M. Gini. “Component-based design for a trading agent.” submitted to Third Int’l Conf. on Autonomous Agents and Multi-Agent Systems, New York, 2004.

^aC. Szyperski, Component Software: Beyond Object-Oriented Programming, ACM Press, 1998.

MinneTAC - Avalon (1)

- Apache Avalon is a general-purpose component framework.
- For building complex and robust systems.
- Avalon components are independent entities - typically they have very few dependencies on each other, and minimal well-defined dependencies on the Avalon framework itself.
- The Avalon “container” loads components, sets up logfiles, configures the components, and starts any components that run independent threads.

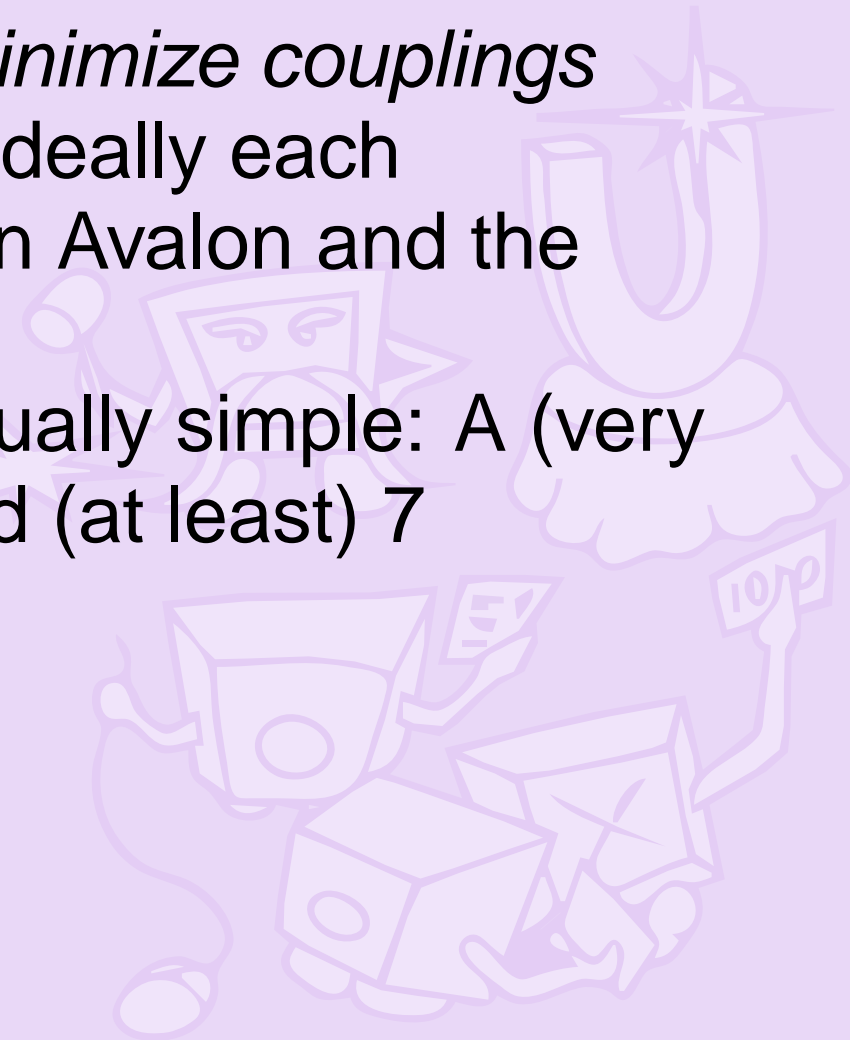
MinneTAC - Avalon (2)

- Each Avalon component is designed to fulfill a specific *role*, and an Avalon system is defined as a set of these roles.
- A role has a name, a set of responsibilities, and a well defined interface.
- In general an Avalon application is composed of the Avalon infrastructure, the specified components, and a “container” that reads the configuration files and starts the process running.

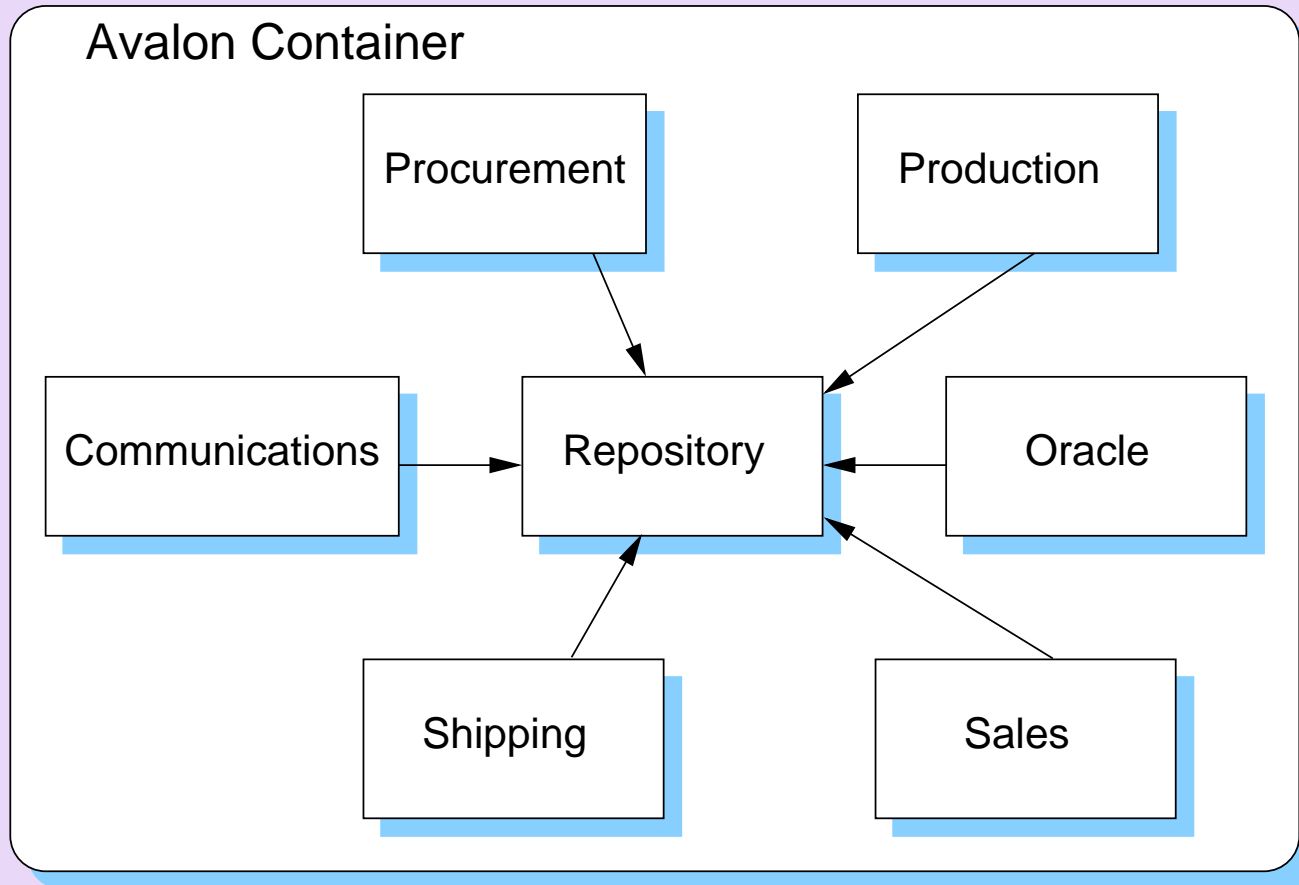
MinneTAC - Architecture (1)

Goal of this architecture: *Minimize couplings between the components!* Ideally each component depends only on Avalon and the Repository.

The architecture is conceptually simple: A (very small) Avalon container, and (at least) 7 components.



MinneTAC - Architecture (2)



Arrows indicate API dependencies.

MinneTAC - Architecture (3)

The MinneTAC container opens three configuration files when it starts.

System configuration file specifies the set of roles that make up the system.

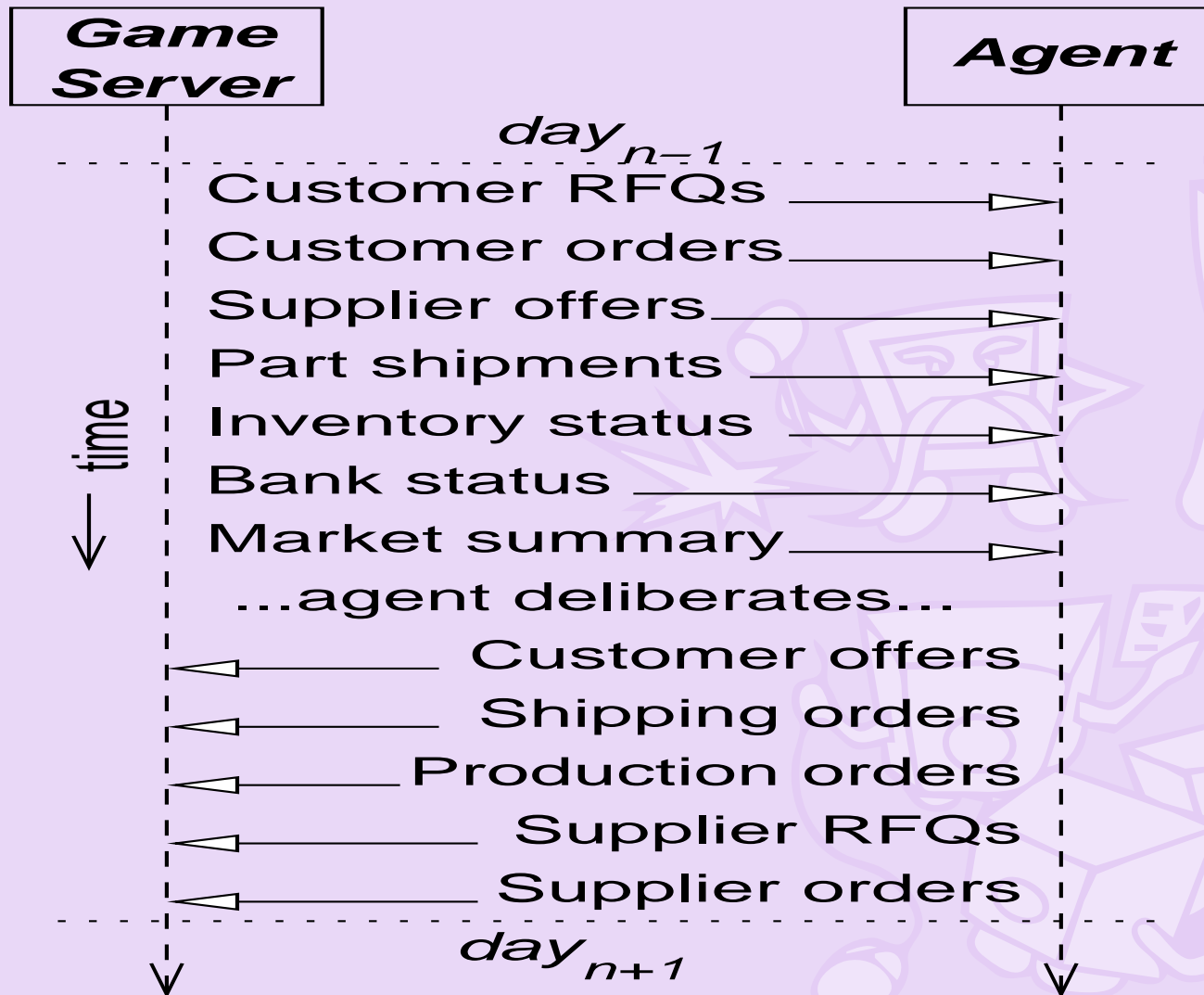
Component configuration file specifies runtime configuration options for each component.

Log configuration file controls, e.g., the names and locations of the log files, the level of detail to be logged, etc.

TAC SCM - Events (1)

- A TAC Agent is a “reactive system,” it responds to events coming from the game server.
- Events are in form of messages that inform the agent to changes to the state of the world: Customer RFQs and orders, supplier offers and shipments, etc.
- Each day two sets (bundles) of messages are exchanged between the game server and the agent.

TAC SCM - Events (2)



One day of communication activity.

MinneTAC - Events (1)

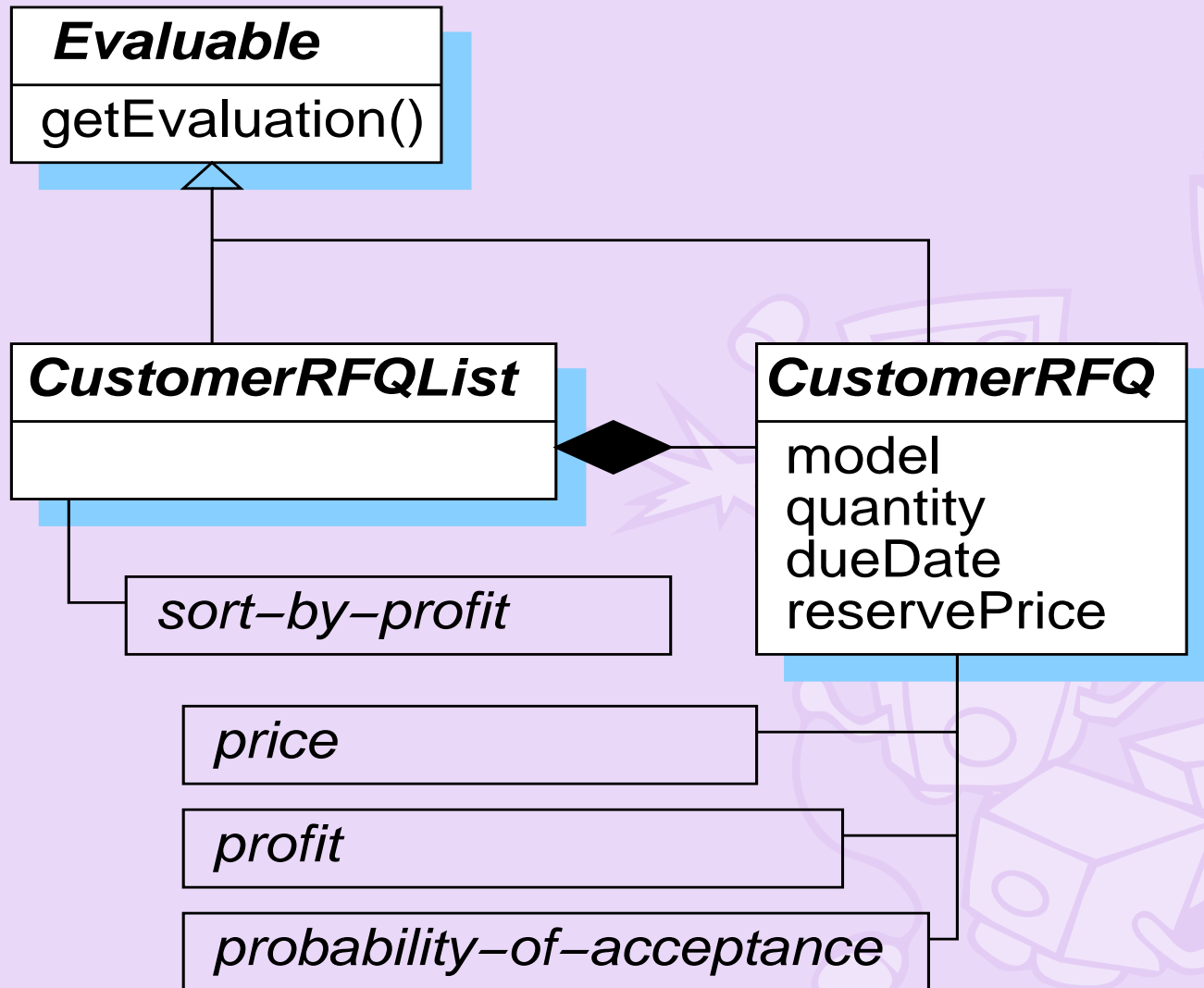
- The end-of-data message (not shown in previous figure) tells the agent that the day's input messages are complete.
- MinneTAC handles all data messages by storing them in the Repository.
- The Repository notifies the components, that have been asked to be notified, that the day's input is complete.
- The Repository acts as a Subject and the other components as Observers in the Observer pattern (see Gamma et al.).

MinneTAC - Events (2)

One of the design goals is to minimize coupling between the various components. How, then, do they communicate?

- Components use *evaluations* that are attached to the various data elements in the Repository.
- When a component needs to make a decision, it will inspect the available data and run some utility-maximizing function.

MinneTAC - Events (3)



RFQ evaluation example.

MinneTAC - Components (1)

Repository is the only component visible to all the other components. It serves as an internal database, maintains the state of the system, and notifies other components of changes in state.

Communications handles the communication with the game server, such as joining games, acquiring initial game parameter and the like.

Procurement procures parts. It may build and maintain target inventory levels, it may attempt to procure parts to meet customer orders, and the like.

MinneTAC - Components (2)

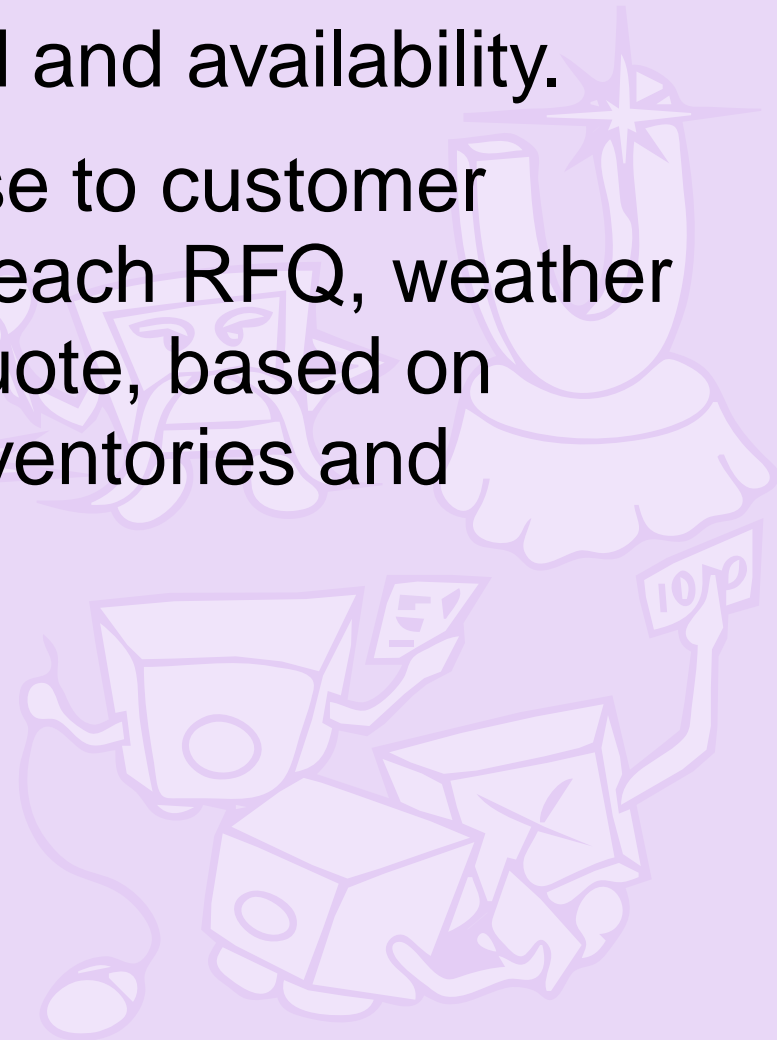
Production schedules the manufacturing facility. It may build and maintain target finished goods inventory levels, or it may build only to meet existing customer orders.

Shipping ships products to customers. In general, there is a benefit in shipping as late as possible, because this gives the agent an opportunity to minimize penalties for late deliveries.

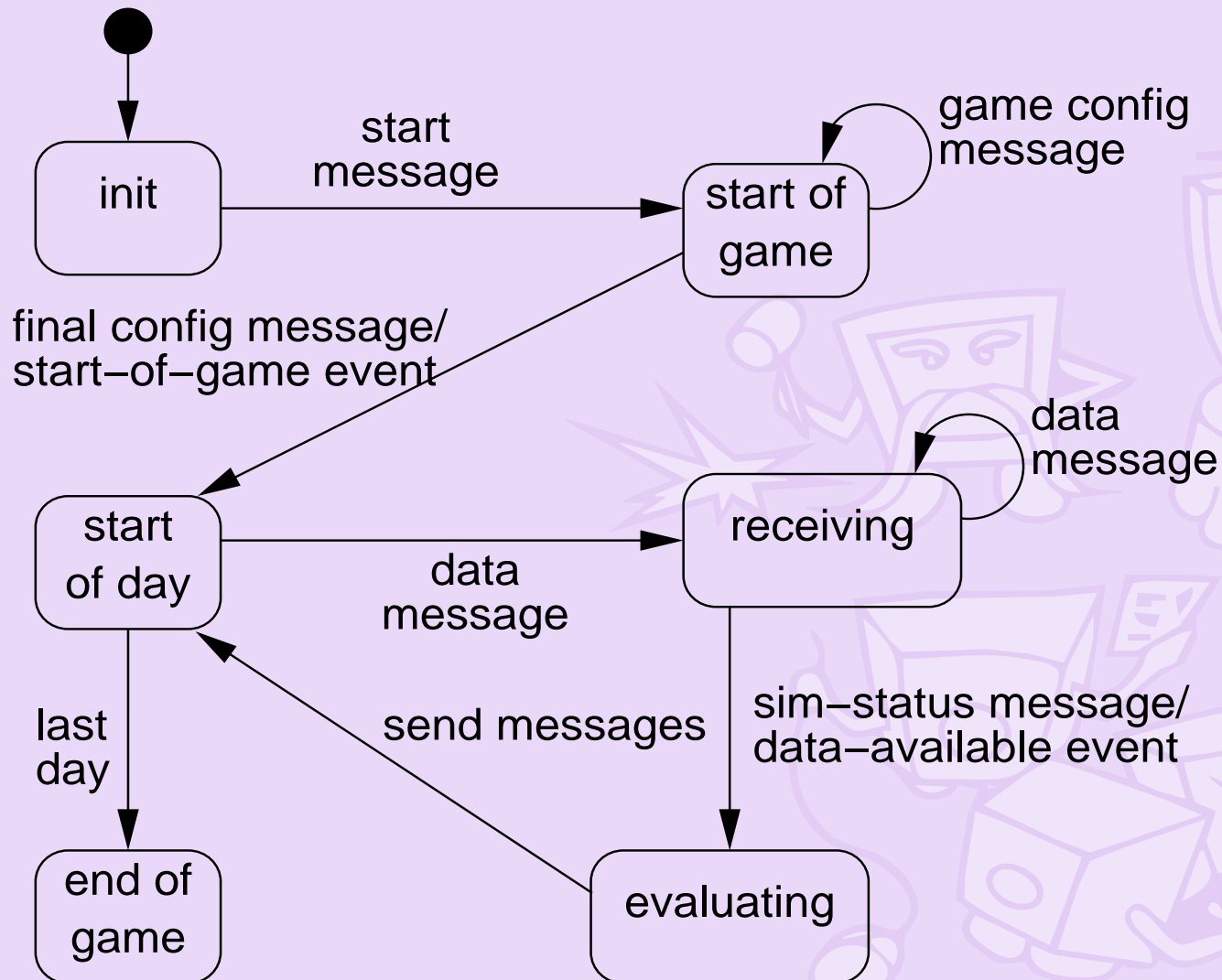
MinneTAC - Components (3)

Oracle predicts future demand and availability.

Sales makes offers in response to customer RFQs. It must decide, for each RFQ, whether to bid and what price to quote, based on available and predicted inventories and current market conditions.

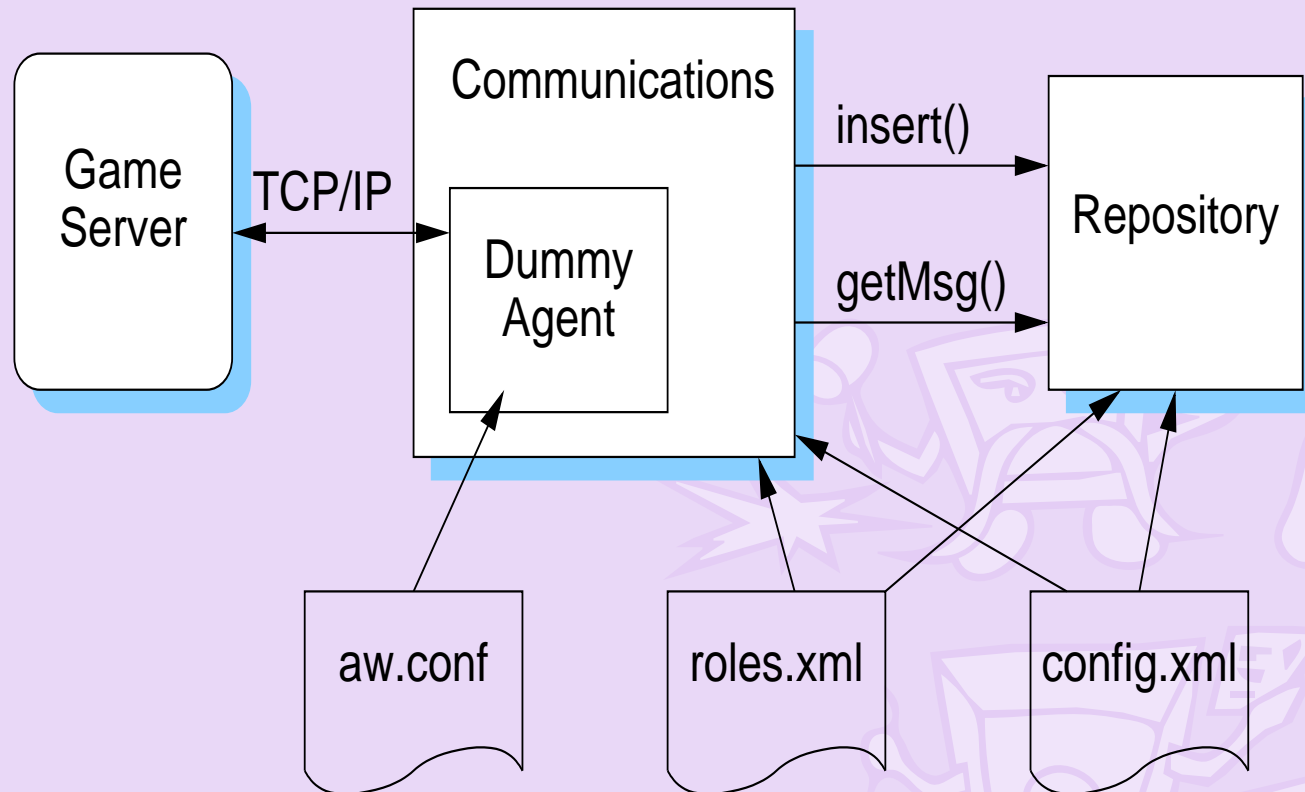


MinneTAC - Repository



States and transitions in the Repository.

MinneTAC - Communications



Communications component wraps the Dummy Agent. It acts as an Adapter (see Gamma et al., Design Patterns, Addison-Wesley, 1994).

TAC SCM - A Priori Analysis

Bottleneck is the factor which limits PCs production on a day. Types of bottlenecks:

Demand bottleneck if the demand for PCs is less than the agents' production capacities and the amount of available supplies.

Production bottleneck if the limiting factor is the agents' production capacities.

Supply bottleneck if the limiting factor is the amount of supplies.

A supply bottleneck is most likely (from game analysis).

MinneTAC - Performance Analysis

- The a priori analysis led us to decide on a *supply-driven* strategy.
- TAC SCM 2003 - 50% discount of parts on first day
- Order many components up front: good in high-demand games, but bad in low-demand games.

W. Ketter, E. Kryzhnyaya, S. Damer, C. McMillen, A. Agovic, J. Collins and M. Gini. "MinneTAC Sales Strategies for Supply Chain TAC." submitted to Third Int'l Conf. on Autonomous Agents and Multi-Agent Systems, New York, 2004.

Sales Strategies - MaxEProfit (1)

Determines offer price to maximize expected profit margin from a potential customer order x :

$$E[ProfitMargin(x)] = ProfitMargin(x) \times p_{order}(x)$$

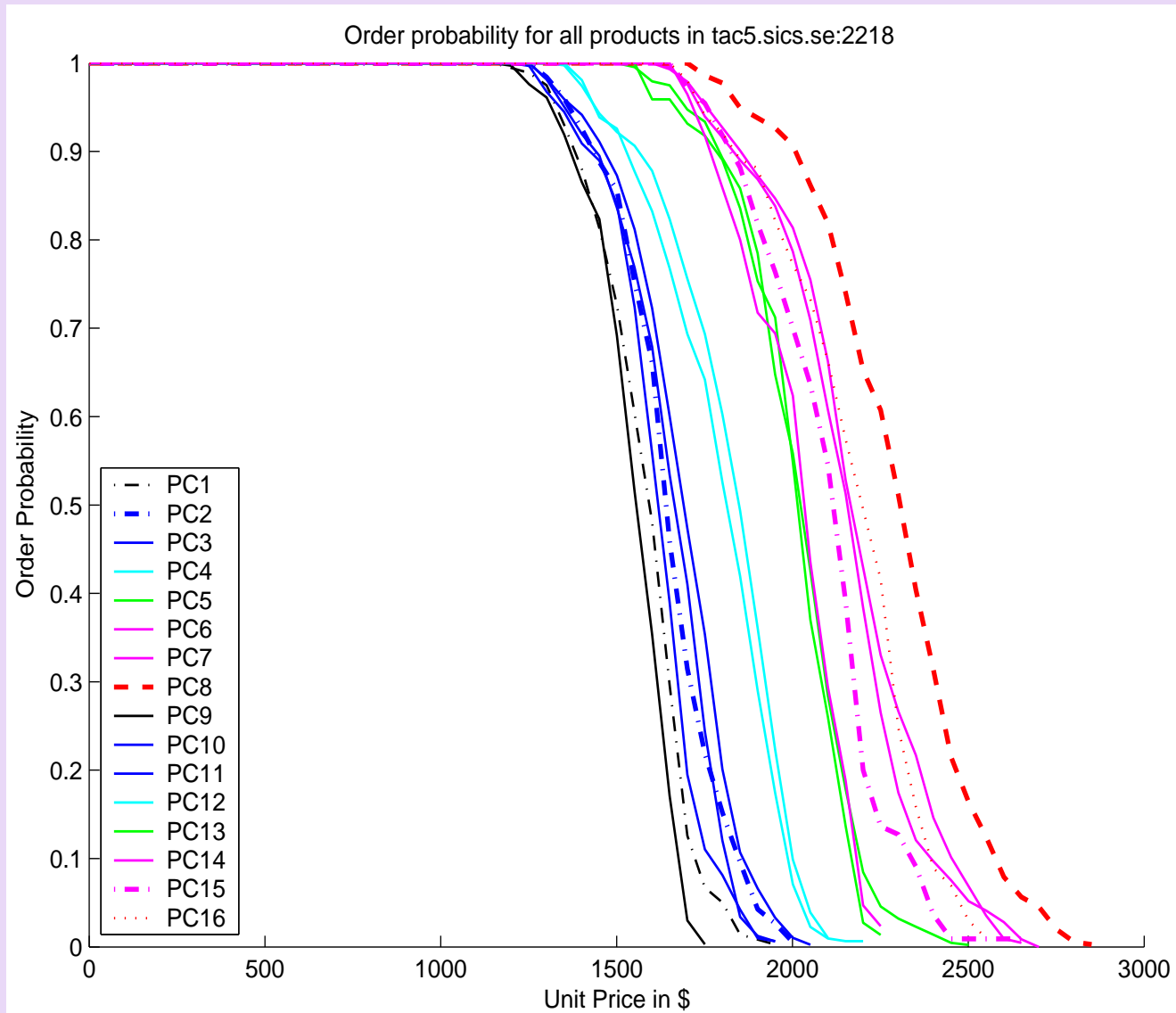
where $ProfitMargin = \frac{price - cost}{price}$ with the constraint that $price \geq targetAveragePrice$.

$ProfitMargin$ is calculated on the agent's moving average cost of the components. $targetAveragePrice$ is an internal parameter.

Sales Strategies - MaxEProfit (2)

- Offers are made only from uncommitted finished goods inventory and are sorted by Expected Profit Margin.
- The agent reserves a fraction of the PCs offered according to the estimated probability of receiving an order, p_{order} , i.e.
$$reserved_qty = RFQ_qty \times p_{order}(RFQ)$$
- Inventory is controlled by pulling stock.
- Procurement and production work to maintain target inventory levels until late in the game.

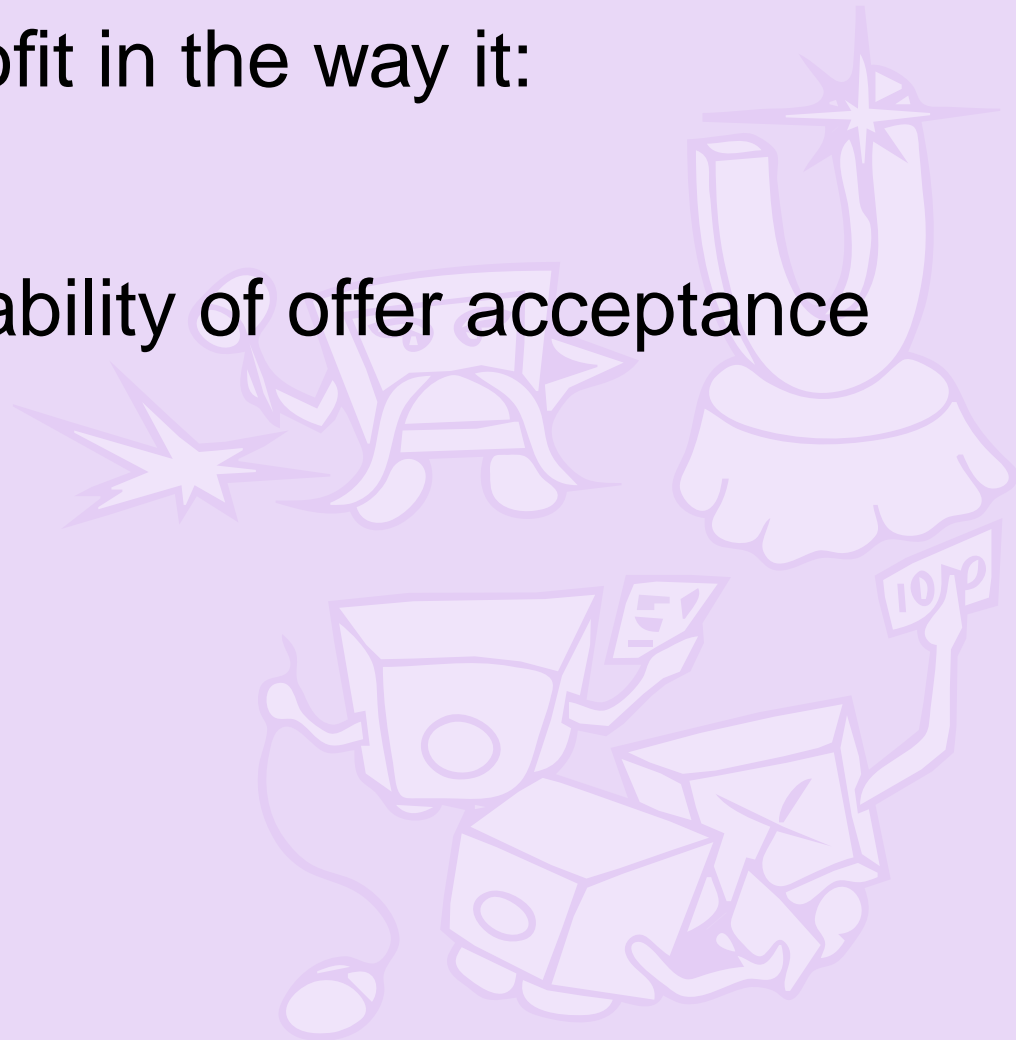
Reverse Cumulative Density Function (RCDF)



Sales Strategies - DemandDriven (1)

It differs from MaxEProfit in the way it:

1. sets offer prices
2. estimates the probability of offer acceptance



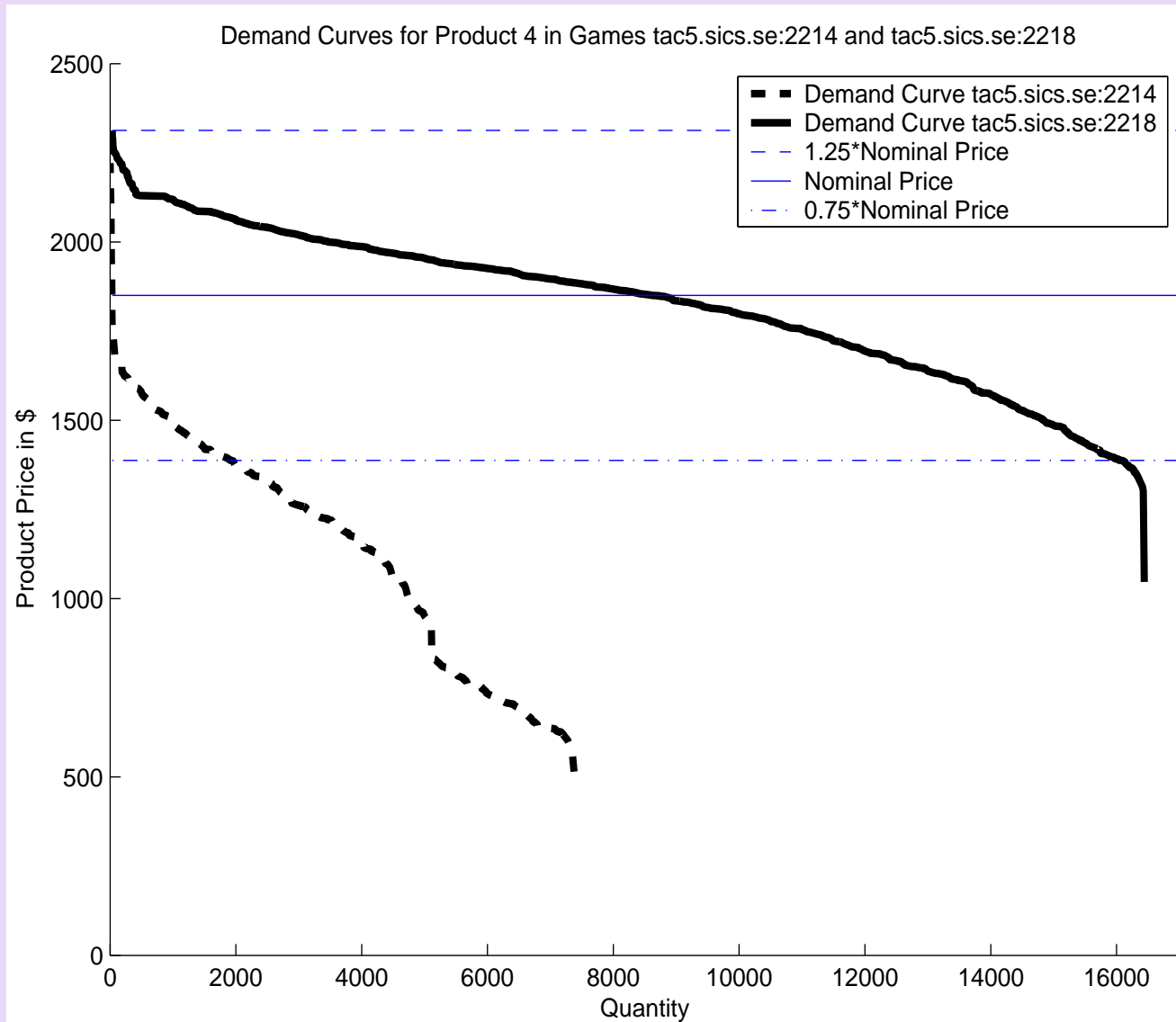
Sales Strategies - DemandDriven (2)

Determines the offer price based on a target probability of receiving a customer order. The *target_prob* is computed from the reverse cumulative density function (RCDF) that models order probability. The goal is to make offers to sell out the inventory by the end of the game.

$$target_prob = \min\left(1, \frac{avail_FG + \# built \times days_left}{estimated_demand}\right)$$

where *avail_FG* is the number of finished goods available for a PC type, *# built* is the number of units of the product built each day.

MinneTAC - High-Demand vs Low-Demand Games (1)



MinneTAC - High-Demand vs Low-Demand Games (2)

Minimum, average and maximum score of each strategy in a set of games (Int'l Conf. on Electronic Commerce 2003). MaxEProfit works better in high-demand games, but is generally worse in low-demand games.

<i>Strategy</i>	<i>High</i>			<i>Low</i>		
	<i>Values (in \$M)</i>					
	<i>Min</i>	<i>Avg</i>	<i>Max</i>	<i>Min</i>	<i>Avg</i>	<i>Max</i>
MaxEProfit	-12.02	12.30	35.99	-66.90	-44.44	-7.36
DemandDriven	-23.65	8.70	30.89	-57.15	-34.49	30.89

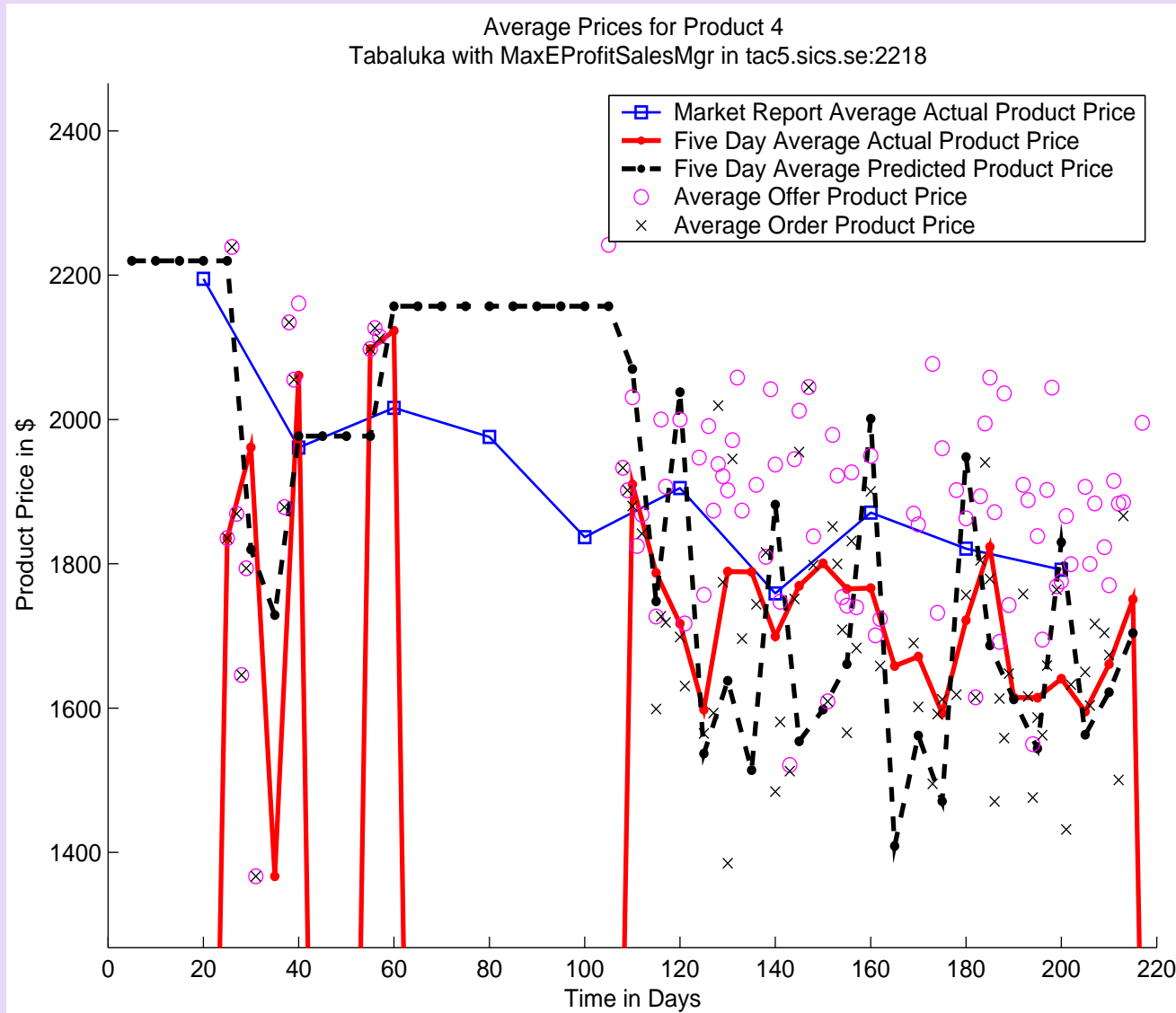
MinneTAC - High-Demand vs Low-Demand Games (3)

Game	Agents and their Result (in \$M)						Demand
	1	2	3	4	5	6	#RFQ/day
2214	team2	RedSox	MinneTAC	arnoch	RedAgent	Eini	99.44
	-10.43	-18.3	-31.06	-34.87	-38.08	-39.83	
2218	Tabaluka	RedAgent	arnoch	MinneTAC	team2	Eini	299.66
	31.23	30.69	23.24	20.8	8.86	7.89	

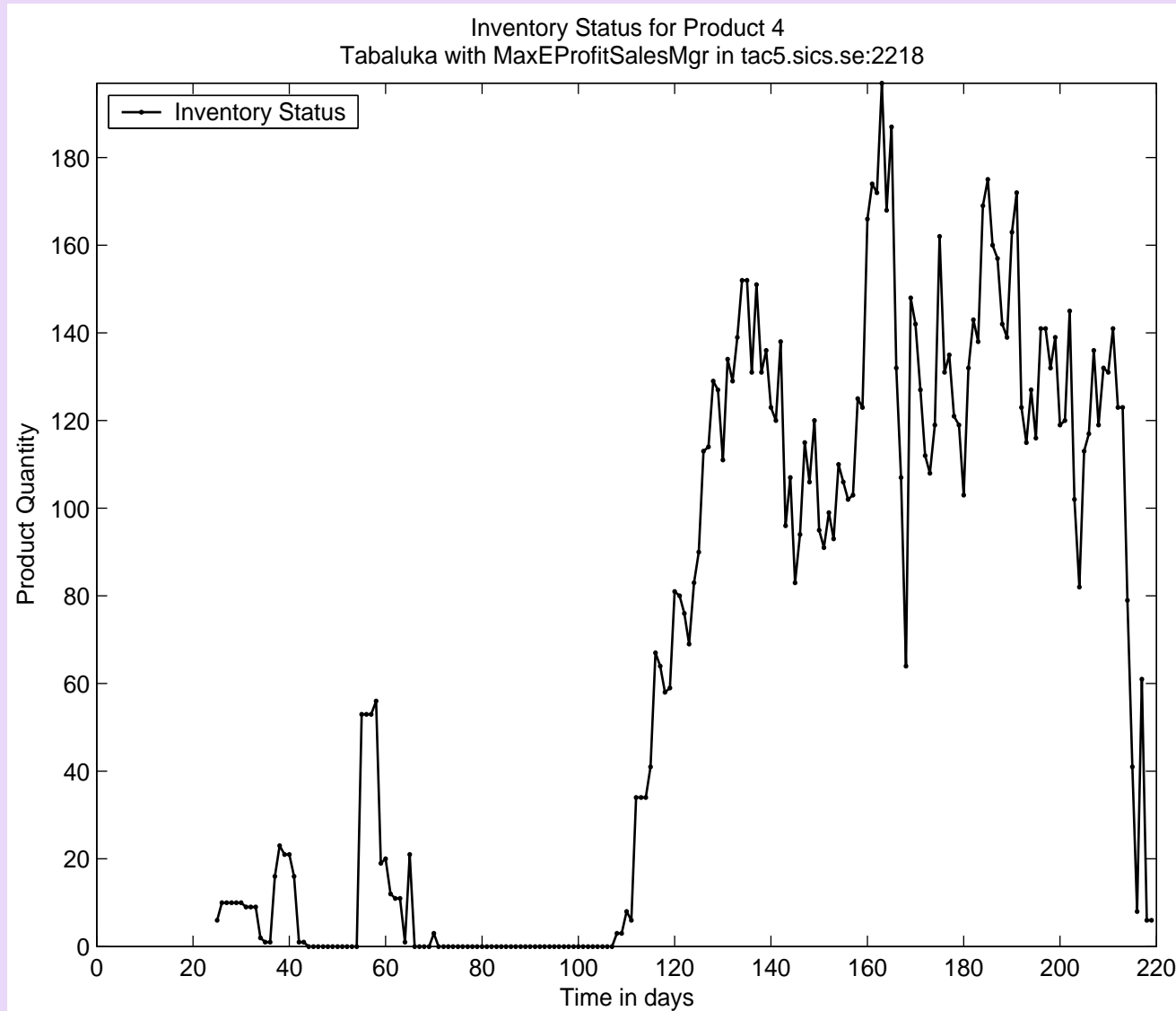
Table 1. Summary of two games. #RFQ/day is mean number of RFQs/day.

Customer RFQs are issued over 219 days in a game. Eini and MinneTAC use DemandDriven and Tabaluka uses MaxEProfit.

MaxEProfit - High-Demand (1)

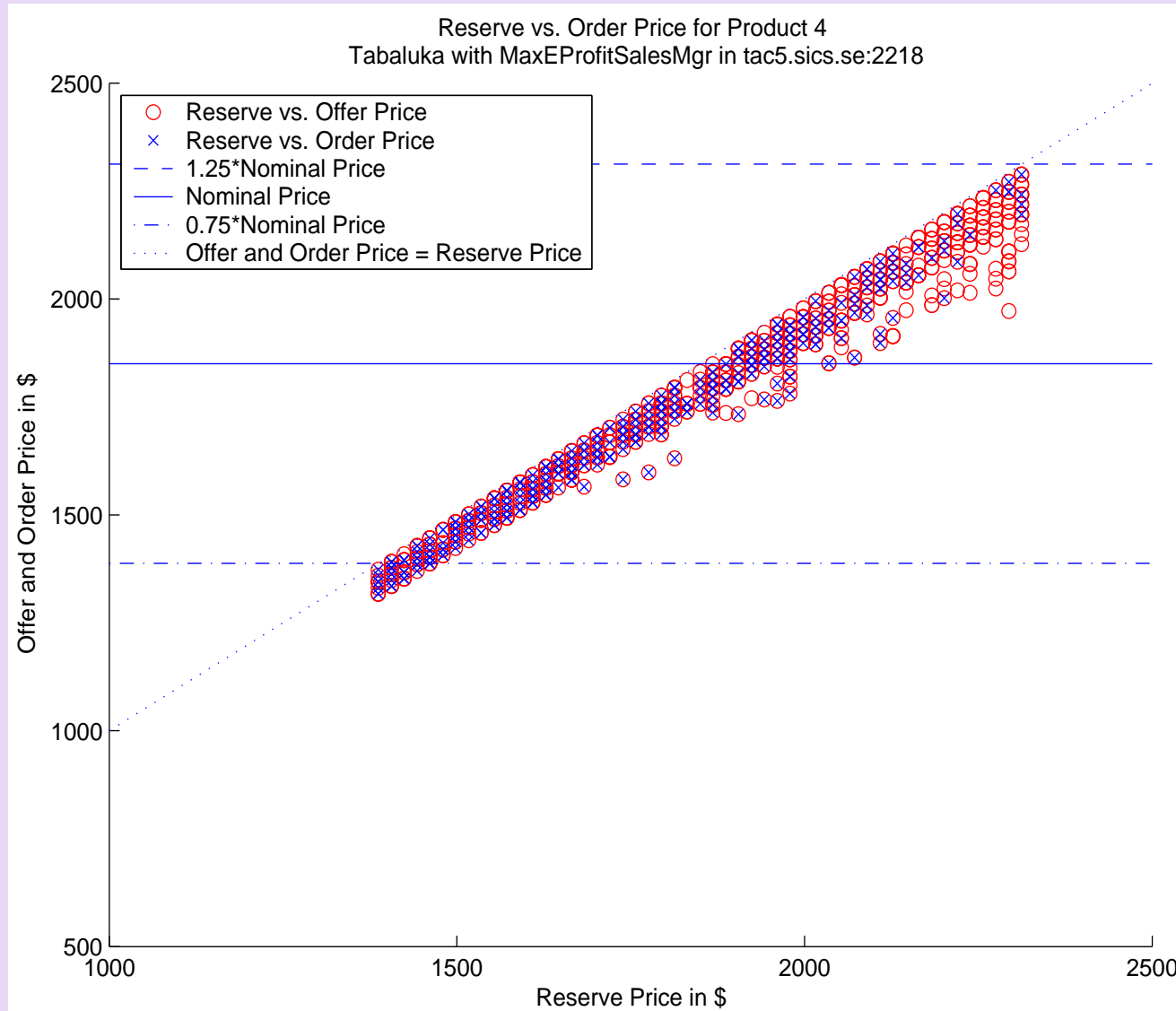


MaxEProfit - High-Demand (2)

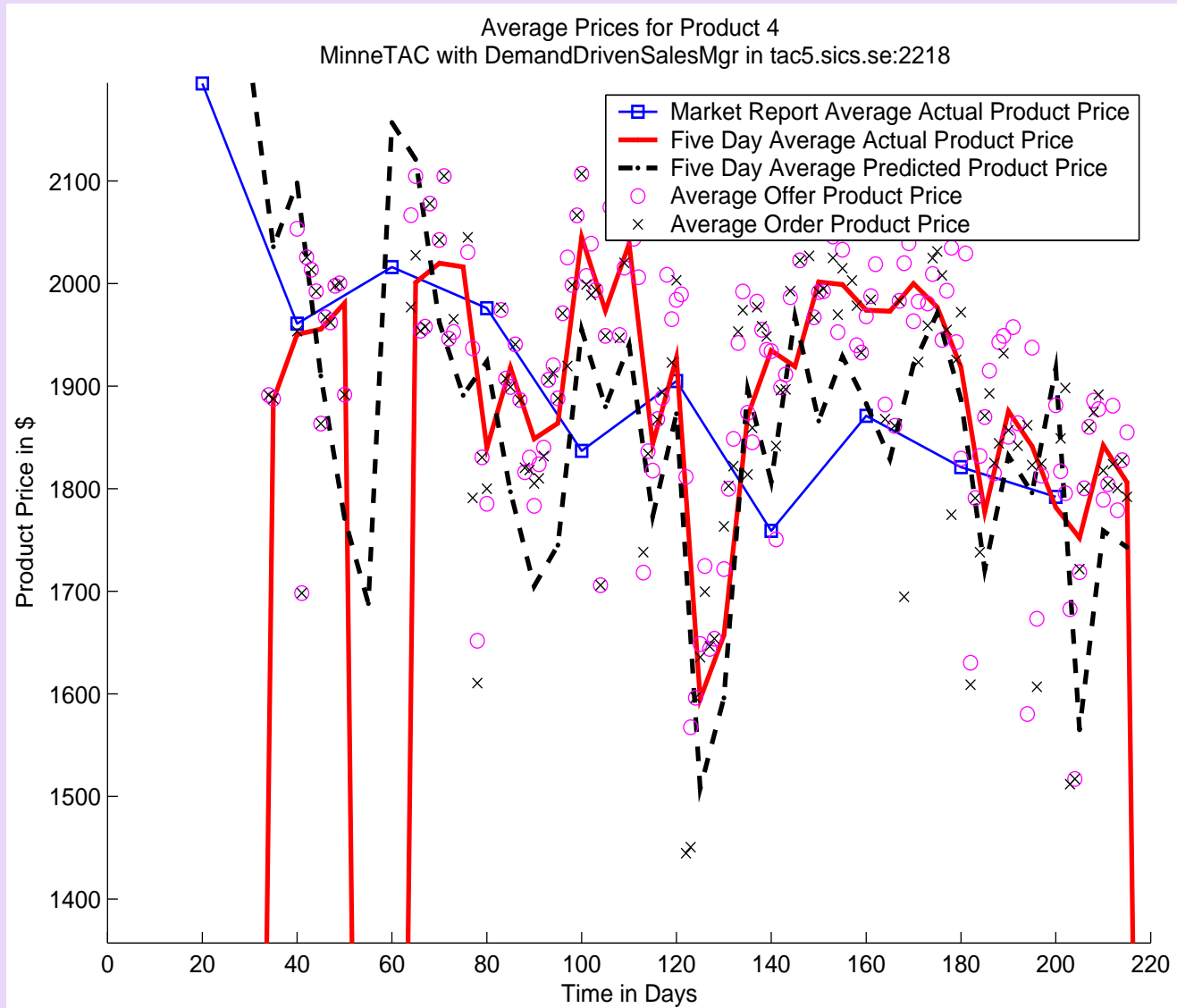


Game 2218 - inventory status: Tabaluka for product 4.

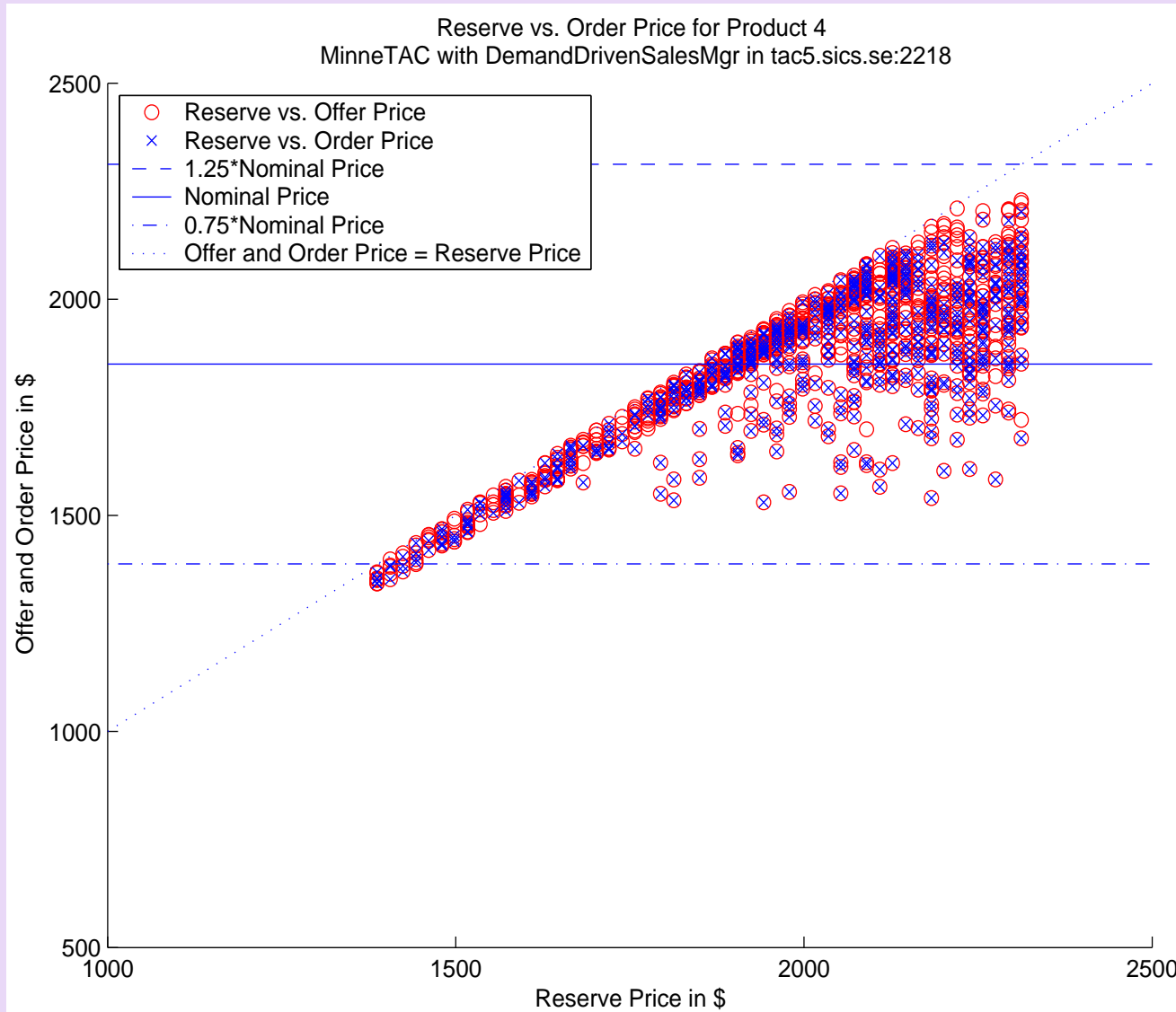
MaxEProfit - High-Demand (3)



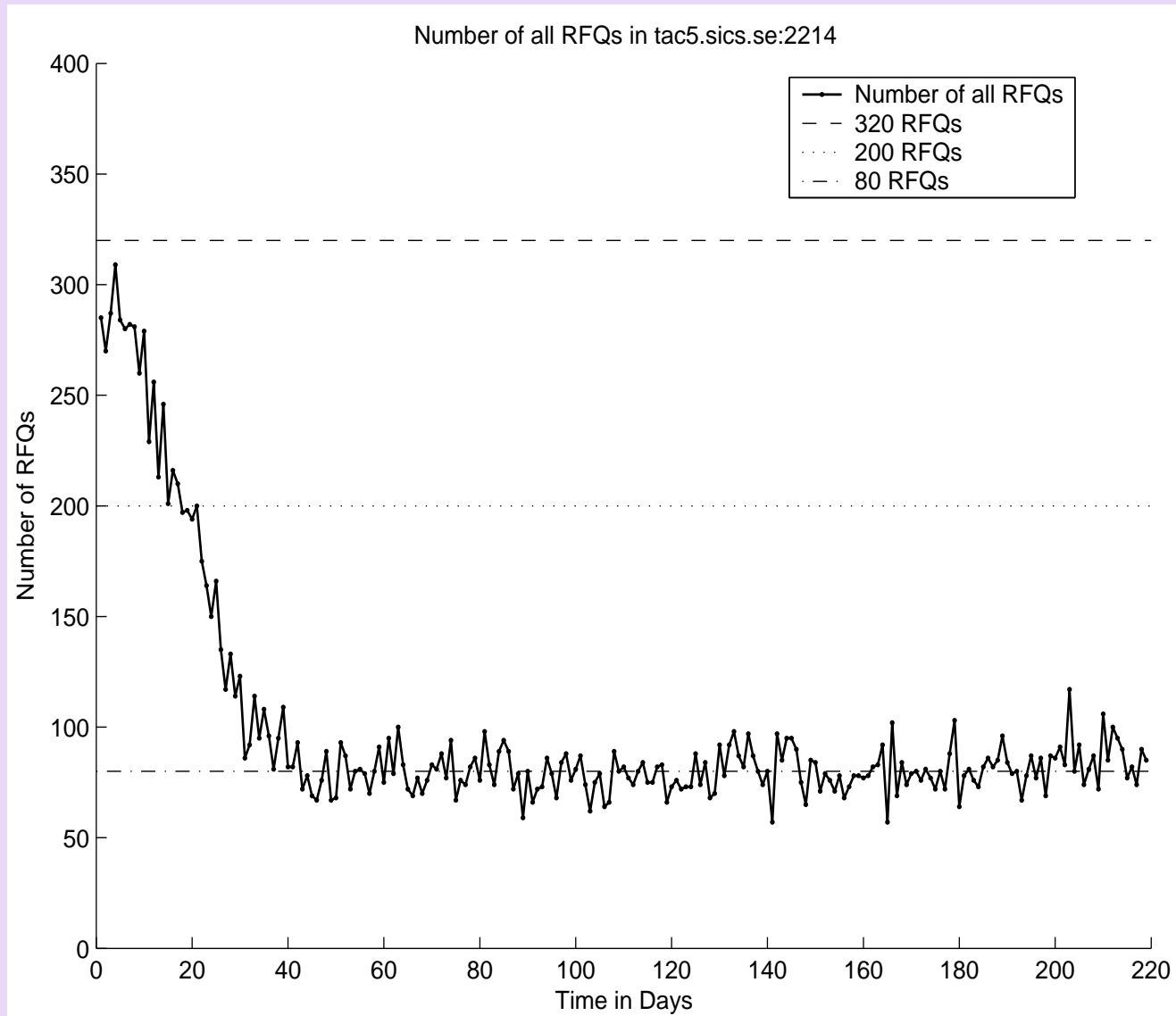
DemandDriven - High-Demand (1)



DemandDriven - High-Demand (2)

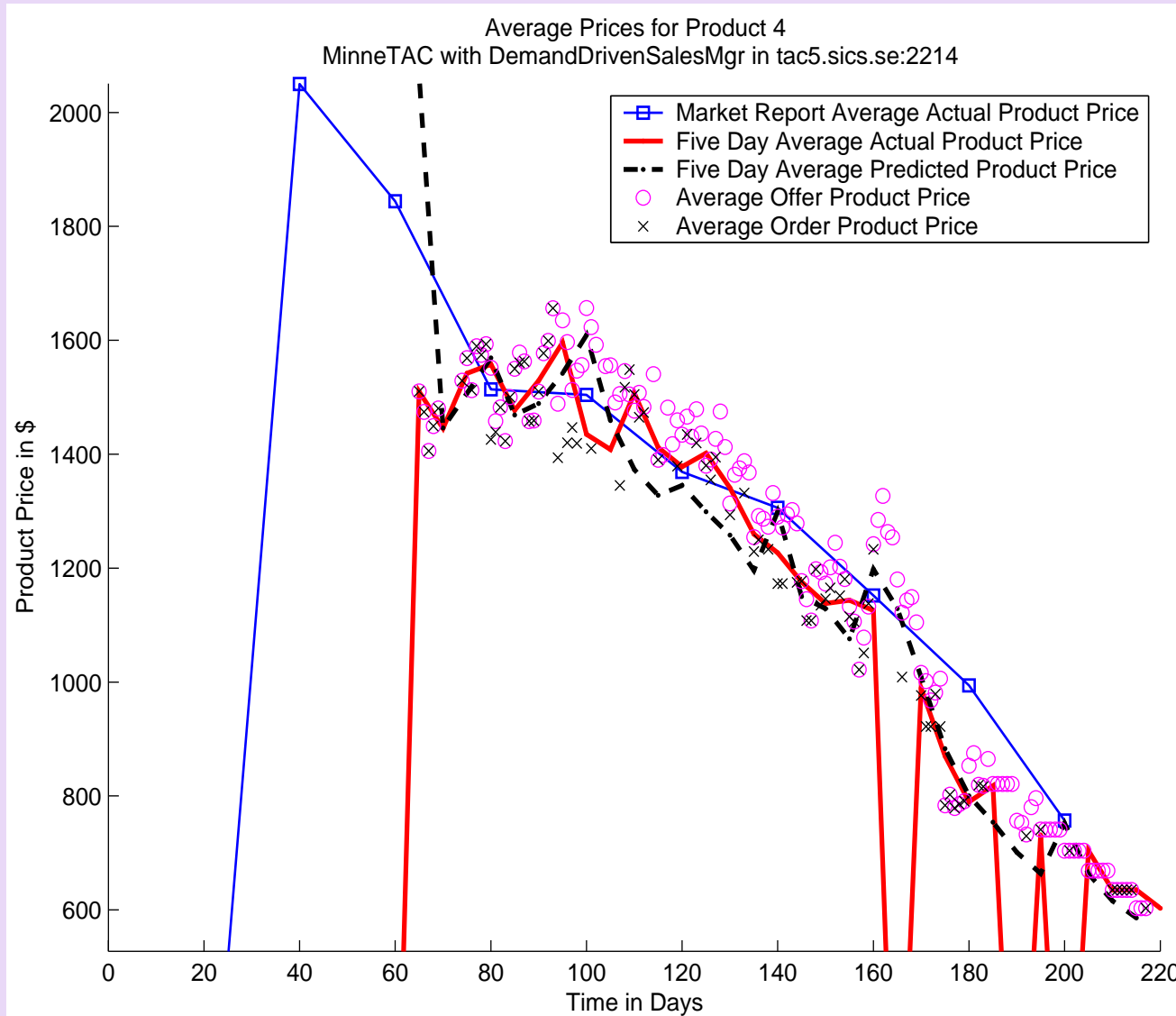


Low-Demand Games

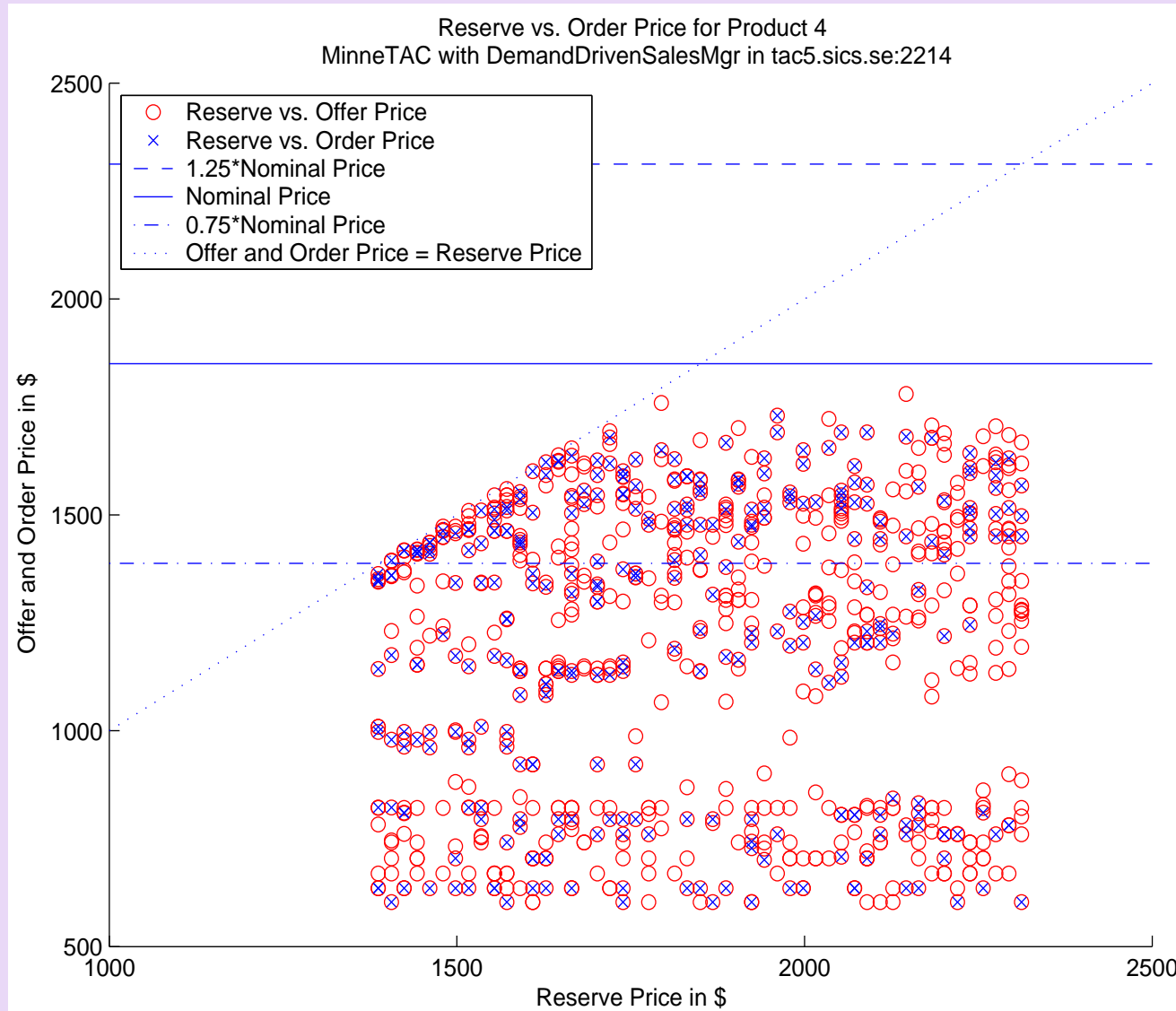


Game 2214 - Total number of RFQs.

DemandDriven - Low-Demand (1)



DemandDriven - Low-Demand (2)



Conclusions

- Broad study of electronic markets.
- Ability to automate the negotiation process.
- Strategies for online supply chain management

Contacts

email: magnet@cs.umn.edu

URL: www.cs.umn.edu/magnet