Flexible Decision Control in an Autonomous Trading Agent

John Collins, Wolfgang Ketter and Maria Gini

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2007-079-LIS
Publication	November 2007
Number of pages	34
Persistent paper URL	
Email address corresponding author	wketter@rsm.nl
Address	Erasmus Research Institute of Management (ERIM) RSM Erasmus University / Erasmus School of Economics Erasmus Universiteit Rotterdam P.O.Box 1738 3000 DR Rotterdam, The Netherlands Phone: + 31 10 408 1182 Fax: + 31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

Bibliographic data and classifications of all the ERIM reports are also available on the ERIM website:
www.erim.eur.nl

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

ABSTRACT AND KEYWORDS	
Abstract	An autonomous trading agent is a complex piece of software that must operate in a competitive economic environment and support a research agenda. We describe the structure of decision processes in the MinneTAC trading agent, focusing on the use of evaluators – configurable, composable modules for data analysis and prediction that are chained together at runtime to support agent decision-making. Through a set of examples, we show how this structure supports sales and procurement decisions, and how those decision processes can be modified in useful ways by changing evaluator configurations. To put this work in context, we also report on results of an informal survey of agent design approaches among the competitors in the Trading Agent Competition for Supply Chain Management (TAC SCM).
Free Keywords	autonomous trading agent, decision processes
Availability	
Classifications	

Flexible decision control in an autonomous trading agent

John Collins*, Dept. of Computer Science and Engineering
University of Minnesota
Wolfgang Ketter; Dept. of Decision and Information Sciences
Erasmus University, Rotterdam, NL
Maria Gini, Dept. of Computer Science and Engineering
University of Minnesota

Abstract

An autonomous trading agent is a complex piece of software that must operate in a competitive economic environment and support a research agenda. We describe the structure of decision processes in the MinneTAC trading agent, focusing on the use of *evaluators* – configurable, composable modules for data analysis and prediction that are chained together at runtime to support agent decision-making. Through a set of examples, we show how this structure supports sales and procurement decisions, and how those decision process can be modified in useful ways by changing evaluator configurations. To put this work in context, we also report on results of an informal survey of agent design approaches among the competitors in the Trading Agent Competition for Supply Chain Management (TAC SCM).

1 Introduction

Organized competitions can be an effective way to drive research and understanding in complex domains, free of the complexities and risk of operating in open, real-world economic environments. Artificial economic environments typically abstract certain interesting features of the real world, such as markets and competi-

*Contact: John Collins, fax: +1-612-652-0572, email: jcollins@cs.umn.edu, address: 4-192 EE/CS Bldg., 200 Union St SE, University of Minnesota, Minneapolis MN 55455, USA

tors, demand-based prices and cost of capital, and omit others, such as personalities, taxes, and seasonal demand.

Our primary interest in this paper is to examine a number of the software design tradeoffs in building an autonomous agent for the Supply-Chain Management Trading Agent Competition [11] (TAC SCM), and to describe in some detail the design of the MinneTAC trading agent, which has competed effectively in TAC SCM for several years.

We describe how we have attempted to respond both to the challenges of the game scenario as well as to the need to support multiple relatively independent research efforts that are focused on meeting one or more of those challenges. We also evaluate the success of our design both in terms of the competitiveness of the agents that have been implemented with it, and in terms of its ability to support our research agenda.

In Section 2, we review the TAC SCM game scenario, focusing on the decision challenges presented by that scenario. We review the literature on the design of autonomous economic agents in Section 3. We also discuss the results of a survey we have conducted into the design philosophy and features of other agents designed for the same competition scenario. Section 4 provides a high-level overview of the design of MinneTAC, focusing on the decision modules and their related evaluators. Section 5 provides several examples of evaluator-supported decisions processes, and shows how they may be reconfigured both manually through configuration files, and automatically by the agent itself using selectors. Section 6 presents a brief evaluation of the architecture.

2 Overview of the TAC SCM game

In a TAC SCM game, each of the competing agents plays the part of a manufacturer of personal computers. Agents compete with each other in a procurement market for computer components, and in a sales market for customers. A typical game runs for 220 simulated days over about an hour of real time. Each agent starts with no inventory and an empty bank account, and must borrow (and pay interest) to build up inventory of computer components before it can begin assembling and shipping computers. The agent with the largest bank account at the end of the game is the winner.

2.1 Game scenario

Customers express demand each day by issuing a set of RFQs for finished computers. Each RFQ specifies the type of computer, a quantity, a due date, a reserve price, and a penalty for late delivery. Each agent may choose to bid on some or all of the day's RFQs. For each RFQ, the bid with the lowest price will be accepted, as long as that price is at or below the customer's reserve price. Once a bid is accepted, the agent is obligated to ship the requested products by the due date, or it must pay the stated penalty for each day the shipment is late. Agents do not see the bids of other agents, but aggregate market statistics are supplied to the agents periodically. Customer demand varies through the course of the game by a random walk.

Agents assemble computers from parts, which must be purchased from suppliers. When agents wish to procure parts, they issue RFQs to individual suppliers, and suppliers respond with bids that specify price and availability. If the agent decides to accept a supplier's offer, then the supplier will ship the ordered parts on or after the due date (supplier capacity is variable). Supplier prices are based on current uncommitted capacity.

Once an agent has the necessary parts to assemble computers, it must schedule production in its finite-capacity production facility. Each computer model requires a specified number of assembly cycles. Assembled computers are added to the agent's finished-goods inventory, and may be shipped to customers to satisfy outstanding orders.

2.2 Agent decisions

An agent for the TAC SCM scenario must make the following four basic decisions during each simulated "day" in a competition:

1. It must decide what parts to purchase, from whom, and when to have them delivered (Procurement).
2. It must schedule its manufacturing facility (Production).
3. It must decide which customer RFQs to respond to, and set bid prices (Sales).
4. It must ship completed orders to customers (Shipping).

These decisions are supported by models of the sales and procurement markets, and by models of the agent's own production facility and inventory situation.

The details of these models and decision processes are the primary subjects of research for participants in TAC SCM. In particular, the Sales and Procurement markets are highly variable, and many important factors, such as current capacity, outstanding commitments of suppliers, sales volumes and price distribution in the customer market, are not visible to the agents. In addition, the number of competing agents in the competition scenario is relatively small (just 6 in a single simulation). This means that agents have oligopoly power – the actions of individual agents can have a significant effect on the markets, and agents are motivated to engage in strategic manipulation of the markets to the extent allowed by the rules of the competition.

Beyond the challenges presented by the TAC SCM problem domain, our research needs present several additional issues. The most important is that our design must support multiple independent developers pursuing their own lines of research. The TAC SCM scenario presents a number of relatively independent decision problems, and there are many possible approaches to solving them. A good design must make it relatively easy for a researcher to focus on a particular subproblem without having to worry about getting a whole agent to work correctly. In addition, we expect to continue participating in TAC SCM over several years, and we want to avoid redesign and re-implementation over that time, even though we expect significant details of the game scenario to change from one year to the next.

Experimental research requires data. The TAC SCM game server keeps data from each game played, which may be used to understand and compare the performance of competing agents. However, it is also necessary to integrate game data with information about the agent's internal state during the game, in order to understand the detailed performance of agent decision processes. This suggests a need for a data logging capability that can be easily configured to extract needed data from a running agent, while keeping the size of log files under control.

3 Related Work

In this section we first explore relevant work related to general agent design and to the design of trading agents. We follow up with a detailed exploration of the approaches used by other TAC SCM agent designers.

3.1 General Trading Agent Design

Most agent design efforts have focused on either the autonomous behavior aspects of the agents, or on interaction among agents. Kinny and Georgeff [25] suggest modeling agent systems from an *external* viewpoint, in which a system is decomposed into agents and other subsystems, and an *internal* viewpoint, in which an individual agent has beliefs, goals, and plans. JADE [27] is an agent framework that has been used to build trading agents, and could have been used for MinneTAC. However, its primary emphasis is on building multi-agent systems that comply with FIPA[1] specifications for inter-agent communications, and with flexible deployment in a network environment. This is not a requirement for the TAC SCM domain. The PROSOCS model [31], unlike JADE, addresses both the interaction issues and the reasoning issues of agent design by proposing a clear "mind-body" separation of concerns, and by proposing a KGP (Knowledge, Goals, Plan) breakdown for the "mind" portion, implemented as a logic program. Although MinneTAC does not use the PROSOCS model directly, much of the analysis underlying this approach are applicable to MinneTAC, and the mind-body distinction is quite clear in our approach. The MinneTAC design is *compositional* in the sense of Brazier *et al.* [7], but not hierarchically so. The DESIRE method from Brazier *et al.* recommends designing an agent as a hierarchy of elements or components that together compose a solution to the various elements of the problem being addressed, treating process requirements and knowledge requirements as separate compositional problems. The method places a strong focus on defining the shared ontology and communication processes in a multi-agent environment. It does not seem applicable to the MinneTAC situation, since we are dealing with a single agent in an existing environment, and the blackboard approach used in MinneTAC is not easily modeled with DESIRE. RETSINA [32] suggests both a multi-agent architecture with a variety of agent roles, and an architecture for individual agents that provides communications, planning, scheduling, and execution monitoring. This architecture could probably be adapted to the TAC SCM domain, but its planning and communication capabilities would not be especially useful. Vetsikas and Selman [34] describe a method for studying design tradeoffs in a trading agent. This approach could likely be used effectively in MinneTAC.

[1] Foundation for Intelligent Physical Agents – http://www.fipa.org/

3.2 TAC SCM Agent Design

This section presents results of an informal survey of research and development practices and related design issues for autonomous trading agents among participants in the Trading Agent Competition for Supply-Chain Management. The goal of the questionnaire was to understand commonality and variability among design principles for an agent competing in TAC SCM. We report our findings from the questionnaire which we sent to the TAC SCM community via the TAC SCM discussion email list in May 2007. The questionnaire was closed by September 2007 and was completed by the best teams in previous tournaments. We also supplement the practitioners' gained wisdom with relevant academic and industry papers.

In Table 1 we list the teams (team name and abbreviation, affiliation, main contact person, and the role of that person) who responded to the questionnaire.

After a review of the completed questionnaires, we categorized the results according to our understanding of the research agendas of the teams, and by the specific architectural emphases the teams identified in their agent designs that support those research agendas. Table 2 gives an overview of our findings.

We observe a convergence between the key issues identified in supply chain management in [24] and our independent findings from the TAC SCM questionnaire. After a thorough analysis of the questionnaire we were able to enrich those findings in relation to the different research agendas. In the following we deepen the discussion of research agendas and their related architectural emphases.

3.2.1 Constraint optimization

A supply-chain agent situated in a trading environment has to comply to many internal and external constraints. These constraints apply to different parts of the supply-chain, such as procurement (e.g. reputation effect), production (e.g. limited production capacity), sales (e.g. can't sell more than effectively demanded by the market), and shipping (e.g. can't ship more than currently in the finished goods inventory). Nearly all the teams who answered our questionnaire applied constrained optimization in some way, so we have listed here the ones who highlighted it in their papers and the questionnaire response. The teams who focus on realtime optimization, Botticelli [4], DeepMaize [23], Foreseer [8], MinneTAC [20] use mainly third party optimization packages, including CPlex[2], Ilog OPL[3], and lp_solve[4]. An exception is CMieux [3] which uses

[2]http://www.ilog.com/products/cplex/

[3]http://www.ilog.com/products/oplstudio/

[4]http://sourceforge.net/projects/lpsolve

Team	University	Team contact and role
Botticelli (B)	Brown University (USA)	Victor Naroditskiy (team member for 4 years)
CMieux (CM)	Carnegie Mellon University (USA)	Michael Benisch (team leader 2005/2006)
CrocodileAgent (CA)	University of Zagreb (Croatia)	Vedran Podobnik (team leader)
DeepMaize (DM)	University of Michigan (USA)	Chris Kiekintveld (team member for 5 years)
Foreseer (F)	Cork Constraint Computation Centre (Ireland)	David Burke (main developer)
Mertacor (M)	Aristotle University of Thessaloniki (Greece)	Andreas Symeonidis (team member 2006)
MinneTAC (MT)	University of Minnesota (USA)	John Collins (team leader and designer)
Southampton (S)	University of Southampton (UK)	Minghua He (designer and programmer)
TacTex (TT)	University of Texas (USA)	David Pardoe (main developer for 5 years)
Tiancalli (T)	Benemerita Universidad Autonoma de Puebla (Mexico)	Daniel Macas Galindo (team leader)

Table 1: Participating teams in the TAC SCM architecture questionnaire.

an internally-developed implementation of a search algorithm to solve a continuous knapsack problem for pricing customer offers.

Research Agenda	Team	Architectural Emphasis
Constraint optimization	B, CM, DM, F, MT	Using 3rd party packages
Machine learning	CM, MT, TT	External analysis framework
		Using 3rd party packages
Dynamic supply-chain	CM, F, M, MT, T	Flexibility
Telecommunication	CA	Scalability
Architecture	CA	IKB model for physical distribution
	MT	Blackboard architecture
		with evaluator chain
Empirical game theory	DM	External analysis framework
Decision coordination	CM, DM, M, MT, S	Modularity
Dealing with uncertainty	B, S	Modularity

Table 2: Research agendas of participating teams in the TAC SCM architecture questionnaire and their architectural emphasis.

3.2.2 Machine learning

Many agent designs depend on an external bootstrapping process to construct models and set parameters, using machine learning algorithms to learn from historical market data. An autonomous agent should also exhibit some learning ability during operation to adapt to changing situations. Successful teams are generally those who perform a thorough off-line bootstrapping as well as online machine learning. CMieux [6], MinneTAC [20], and TacTex [28] identified the need to support learning and adaptation as primary concerns in the design of their agents. To support research agendas with a strong emphasis on machine learning, both CMieux [6] and TacTex [28] use the Weka[5] [38] machine learning tool set. MinneTAC is using Matlab[6] in combination with the Netlab[7] neural network toolbox to develop and train market models, and to bootstrap the agent with the resulting models. At runtime, MinneTAC updates and adjusts those models using feedback and machine learning algorithms embedded in Evaluators (see Section 4.2.2) written in Java.

[5] http://www.cs.waikato.ac.nz/ml/weka/

[6] http://www.mathworks.com/

[7] http://www.ncrg.aston.ac.uk/netlab/

3.2.3 Management of dynamic supply chains

Traditionally, supply chains have been created and maintained through the interactions of human representatives of the various enterprises (component suppliers, manufactures, wholesalers/distributors, retailer and customers) involved. However, the recent advent of autonomous trading agents opens new possibilities for automating and coordinating the decision making processes between the various parties involved. The TAC SCM simulation is an abstract model of a highly dynamic direct sales environment [10], as exemplified by Dell Inc., for procurement, inventory management, production, and sales.

Given the dynamic nature of the TAC SCM environment and the new challenges in the real world, many teams have a strong research focus on dynamic supply-chain behavior. These include CMieux [6], Foreseer [8], Mertacor [26], MinneTAC [21], and Tiancalli [13]. As a consequence of this research agenda the teams strive for high flexibility in their agent design, so that they can easily accommodate changes arising from new theory or from changes in the game environment itself. These responses are coherent with classical and contemporary literature in software engineering, which recognizes the central role of flexibility in software design for environments where ability to adapt to changing requirements is a dominant factor.

3.2.4 Telecommunication

The CrocodileAgent [29] group at Zagreb is part of a larger group that is focused on autonomous agents for management of large-scale telecommunication networks. They view TAC SCM as an interesting challenge in building an agent that can successfully operate in a dynamic, competitive environment, but they are also concerned with scalability and other issues that go far beyond the TAC SCM scenario. They have chosen to base their design on the JADE[8] [2] agent framework, since it has been well-proven for large-scale multi-agent situations.

3.2.5 Architecture

CrocodileAgent [29] and Southampton SCM [15] have structured their agent decision processes around the the IKB model [35], a three layered agent-based framework for designing strategies in electronic trading markets. The first layer is the Information layer which contains data gathered from the environment. The Knowledge layer represents the knowledge extracted from the data stored in the information layer, and the

[8] http://jade.tilab.com/

Behavioral layer encapsulates the reasoning and decision-making components that ultimately drive agent behavior. As reported by the CrocodileAgent [29] team, an advantage of using JADE is that the separation of I, K & B layers enables physical distribution of layers on multiple computers. In such a design, information layer agents parse out information from the TAC SCM server messages, while information and knowledge flows are implemented as JADE agent communication (ACL[9]-based messages).

A MinneTAC agent is a component based framework. All data that must be shared among components are kept in the Repository, which acts as a blackboard [9]. For details on the MinneTAC design we refer the reader to Section 4. An interesting outcome of the questionnaire is that only the MinneTAC team strives specifically for a design that minimizes the learning curve for a researcher who wishes to work on a specific subproblem.

Half of the teams who have been in the competition for more than two years have found a need to make significant changes in their designs, often to correct problems with coordination among the modules that implement the various decision processes.

3.2.6 Empirical game theory

The DeepMaize [24, 36, 17] group at Michigan pursues empirical game-theoretic analysis as one of their major research cornerstones. They employ an experimental methodology for explicit game-theoretic treatment of multi-agent systems simulation studies. For example, they have developed a bootstrap method to determine the best configuration of the agent behavior in the presence of adversary agents [17]. They also use game-theoretic analysis to assess the robustness of tournament rankings to strategic interactions. Many of their experiments require hundreds to thousands of simulations with a variety of competing agents. To support their work they have developed an extensive framework for setting up and running experiments, and for gathering and analyzing the resulting data[10].

3.2.7 Decision coordination

Decision coordination is an important element of the research agendas for the DeepMaize [23], MinneTAC [21], and Southampton [16] teams. This problem is commonly viewed as one of enabling independent decision processes to coordinate their actions in useful ways while minimizing the necessity to share representation and

[9] Agent Communication Language

[10] P. Jordan, private communication

implementation details. This is important because of the difficulty in treating all the decisions an agent must make as a single problem. Indeed, real-world organizations often do a poor job of coordinating procurement and sales because they are functions of widely separated units within a typical industrial concern.

MinneTAC uses a blackboard approach to allow decision processes to coordinate their actions through a common state representation, and Southampton uses the hierarchical IKB approach, in which the Knowledge layer of the IKB model could be viewed as a type of blackboard. DeepMaize [23] treats the combined decisions as a large optimization problem, decomposed into subproblems using a "value-based" approach. The result is that much of the coordination among decision processes is effectively managed by assigning values to finished goods, factory capacity, and individual components over an extended time horizon.

A particularly interesting approach to the decision coordination problem was taken by RedAgent [18], which used loosely-coupled "sub-agents" competing with each other in internal auction-based markets for finished goods, production capacity, and components. This achieved a radical decoupling of the various components, but proved to be uncompetitive after the game design was adjusted in 2005 to defeat some of the simplest approaches that lacked adequate coordination among decisions. Specifically, agents that focused procurement only on keeping the factory in full production found themselves overproducing when the balance between factory capacity and expected customer demand was adjusted.

Every TAC SCM agent has to make decisions in support of sales, procurement, production scheduling, and sometimes inventory management. These decisions have to be coordinated, but there are many different ways to coordinate them. In this kind of setting it is advantageous to be able to replace individual decision-oriented components of an agent and compare their performances, e.g. two different sales modules compared on final profit. Many teams mentioned "modularity" as a separate goal for their designs, but we think that this is really a precondition that allows this sort of experimental flexibility.

3.2.8 Dealing with uncertainty

The TAC SCM competition scenario is designed to force agents to deal with uncertainty in many dimensions. Sodomka et al. [30] provide a good overview of the sources of uncertainty in the context of an approach to doing empirical study of agent performance. The Botticelli group clearly identifies the problem of dealing with uncertainty as one of their main research goals in [5]. SouthamptonSCM [16] employs a bidding strategy that uses fuzzy logic to adapt prices according to the uncertain market situation. SouthamptonSCM told us

in the questionnaire that the software package for the fuzzy reasoning on price adaptation will be released soon.

3.2.9 Published TAC SCM designs

Several participants in TAC SCM have described their agent designs. He *et al.* [16] have adopted a design consisting of three internal "agents" to handle Sales, Procurement, and Production/Shipping. Sales decisions use a fuzzy logic module. Some algorithmic details are given, but there is little further detail on the architecture of the agent. TacTex05, the winner of the 2005 competition [28] is based on two major modules, a Supply Manager that handles procurement and inventory, and a Demand Manager that handles sales, production, and shipping. These modules are supported by a supplier model, a customer demand model, and a pricing model that estimates sales order probability.

The overall survey outcome shows that there are common themes emerging from the different research groups on how to design a successful agent architecture. These include common software engineering quality criteria such as modularity, low coupling, and separation of concerns, in addition to more problem-specific approaches such as coordination of sales and procurement through internal models of inventory and prices, and assigning current and future value to inventory and production resources. There are also some strong differences such as how to organize the communication between the different modules and which modules should own the data for specific tasks. These findings, and the fact that after several years of competition there is still much to be learned, suggest that the recipe for a full competent supply-chain trading agent is still an unsolved problem, even for an abstract, constrained environment like TAC SCM.

4 The design of MinneTAC

In designing MinneTAC we follow a component-oriented approach [33]. The idea is to provide an infrastructure that manages data and interactions with the game server, and cleanly separates behavioral components from each other. This allows individual researchers to encapsulate agent decision problems within the bounds of individual components that have minimal dependencies among themselves. Two pieces of software form the foundation of MinneTAC: the Apache Excalibur component framework [12], and the "agentware" package distributed by the TAC SCM organizers. Excalibur provides the standards and tools to build components

and configure working agents from collections of individual components, and the agentware package handles interaction with the game server.

4.1 A brief overview of Excalibur

Apache Excalibur is a general-purpose component framework. It is widely used as a foundation for middleware and for server software, such as the OpenORB CORBA implementation[11] and the Cocoon web application framework[12], but its use in the implementation of autonomous agents is rare. It does not provide the "classic" facilities for agent design, such as knowledge representation, inter-agent communication, reasoning facilities, or a planning infrastructure. Instead, it provides a means to build complex, robust systems from sets of role-based, configurable components. This satisfies a primary goal of MinneTAC, allowing researchers to work independently on individual decision problems with minimal need for detailed coordination with each other.

Excalibur components are independent entities, in the sense that they typically have very few dependencies on each other, and minimal, well-defined dependencies on the Excalibur framework itself. Components are coarse-grained entities, each typically composed of a number of classes. Control inversion puts primary control in the Excalibur "container", which loads components, sets up logfiles, configures the components, and starts any components that run independent threads.

Each Excalibur component is designed to fulfill a specific *role*, and an Excalibur system is a set of roles, each of which is mapped to a specific Java class. A role has a name, a set of responsibilities, and a well-defined interface. An Excalibur application is composed of the Excalibur infrastructure, a container that initializes the system, and the components specified by the configuration. Configuration files specify the specific roles, the classes that satisfy those roles, and configuration parameters for those classes. The container reads the configuration files, loads the specified classes, and invokes the Excalibur interfaces on each component.

4.2 MinneTAC architecture

Following Bass et al. [1], we use the term "architecture" to refer to the set of components that make up our system, along with their properties and relationships. A MinneTAC agent is a set of components layered

[11] OpenORB.sourceforge.net

[12] cocoon.apache.org

on the Excalibur container, as shown in Figure 1. In the standard arrangement, four of these components are responsible for the major decision processes: Sales, Procurement, Production, and Shipping. In some configurations, an LpSolver component is included to provide optimization services. All data that must be shared among components is kept in the Repository, which acts as a blackboard [9]. The Oracle hosts a large number of smaller components that maintain market and inventory models, and do analysis and prediction. The Communications component handles all interaction with the game server. By minimizing couplings between the components, this architecture completely separates the major decision processes, thus allowing researchers to work independently. Ideally, each component depends only on Excalibur and the Repository.

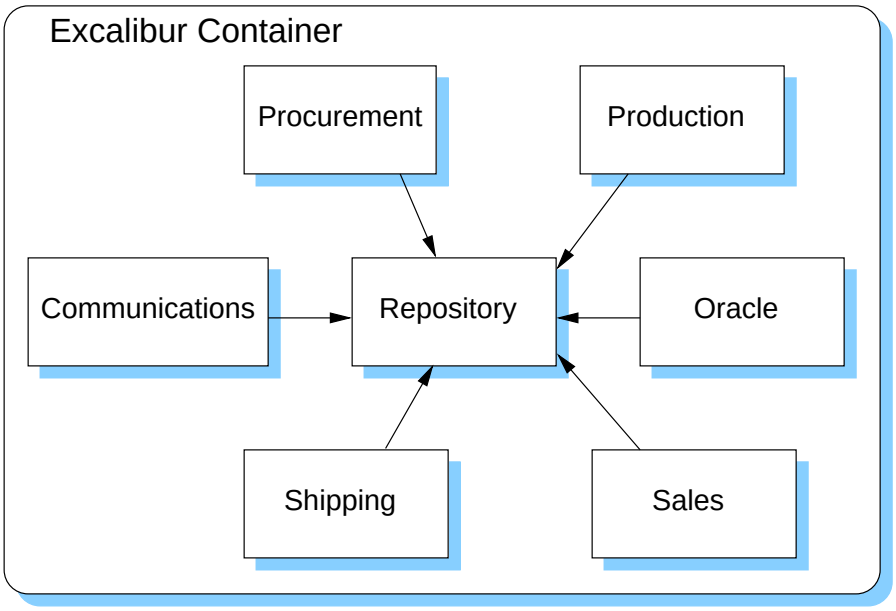

Figure 1: MinneTAC Architecture. Arrows indicate dependencies.

The agent opens four configuration files when it starts. Two of them are interesting in the context of this paper. The system configuration file specifies the set of roles that make up the system, along with the classes that implement those roles. This allows the major components (Sales, Procurement, Production, Shipping) to be swapped out with a simple edit. The component configuration file specifies runtime configuration options for each component. For example, the Sales component may have a parameter that controls the maximum level of overcommitment of its existing inventory or capacity when it makes customer offers. More importantly, the various Evaluators are specified and configured in this file (see Section 4.3.2).

4.2.1 Events

A TAC Agent is basically a "reactive system" in the sense that it responds to events coming from the game server [37]. These events are in the form of messages that inform the agent of changes to the state of the world: Customer RFQs and orders, supplier offers and shipments, etc. The game is designed so that each simulated day involves a single exchange of messages; a set of messages sent from the game server to the agent, and a set returned by the agent back to the server. For example, from the standpoint of the agent, each day's incoming messages includes the set of customer RFQs for the day, and the return set of messages includes the agent's bids for those RFQs.

More specifically, Figure 2 shows the communication activity for a game day. The general pattern is that the game server sends out a set of messages representing supplier and customer activity, as well as inventory and bank-account status data, the agent deliberates for some time, and then the agent responds with a set of messages that respond to the customer RFQs, and supplier offers for the current day. The agent must also specify the production and shipping schedules for the following day, and it may issue additional supplier RFQs each day. The length of a simulation day is fixed by the server; in the standard tournament configuration, days are 15 seconds long.

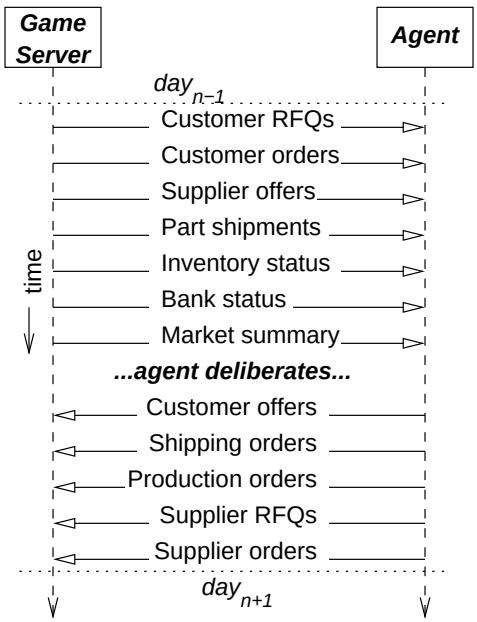

Figure 2: One day of communications activity between the game server and an agent.

As shown in Figure 2, the agent does not need to react to individual messages from the server. Instead, it waits until after all the day's messages have been received, and then considers all of them together. In fact, there is one additional end-of-data message not shown in the figure, which contains no data but simply tells the agent that the day's input messages are complete. MinneTAC handles all data messages by storing them in the Repository. When the end-of-data message is handled by the Repository, it notifies the other components that the day's data input is complete. Components use this notification as the signal to perform their deliberations and compose the day's return messages. In this interaction, the Repository acts as a Subject and the other components as Observers in the *Observer* pattern [14].

An important limitation of the Observer pattern is that the sequence of notifications is not controlled, although in most implementations it is repeatable. But the order of event processing is important for the MinneTAC decision processes. For example, it greatly simplifies the Sales decision process to know that the current day's Shipping decisions have already been made. To allow event sequencing without introducing new dependencies, two events are generated by the Repository for each day of a game. The *data-available* event is a signal to read the incoming messages and do basic data analysis. The subsequent *decision* event is a signal to make the daily decisions and post the outgoing messages back in the Repository. The decision event itself provides an additional level of sequence control by allowing components to "refuse" the event until one or more other components (identified by role names) have made their decisions. Components that have refused the event will receive it again once all other components have had an opportunity to process it. To ensure that Sales decisions are made after Shipping decisions, Sales must refuse to accept the decision event until after it sees "shipping" among the roles that have already processed it. No additional dependencies are introduced by this mechanism, since the role names are simply added to the event object itself, and the names come from a configuration file, not the code.

4.2.2 Evaluators

As indicated earlier in this section, a goal of the MinneTAC design is to minimize coupling between the various components. How, then, do they communicate, if they cannot depend on one another? One possible approach is the one used by the RedAgent team at McGill University [18], in which the components communicate through internal auction-based markets. Our approach is to use *evaluations* that are accessible through the various data elements in the Repository. The general idea is that when a component needs to make

a decision, it will inspect the available data and run some utility-maximizing function. The available data consists of any data it maintains internally, and the data in the repository. Any data reductions or analyses that are performed on Repository data can be encapsulated in the form of Evaluations, and made available to other components. These analyses are implemented by the Oracle component through a configurable set of *evaluators*.

All the major data elements in the Repository are Evaluable types. As shown in Figure 3, each Evaluable can be associated with some number of associated Evaluations. Also associated with each Evaluable is an EvaluationFactory, which maintains a mapping of Evaluation names to Evaluator instances, and is responsible for producing Evaluations when they are requested. It does this by inspecting the class of the Evaluable and the name of the requested Evaluation, and invoking the appropriate *evaluate* method on an associated Evaluator. Evaluators implement back-chaining by requesting other Evaluations in the process of producing their results. Evaluators are hosted by the Oracle component, which is responsible for loading and configuring Evaluators. Evaluators are registered with the Repository when they are configured, thus making them known to the EvaluationFactory.

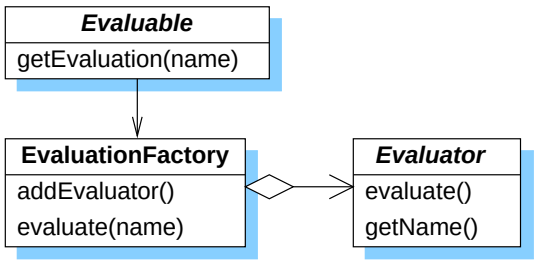

Figure 3: Evaluables and Evaluators.

Figure 4 shows a simple example of some Evaluable instances and a set of Evaluations that might be associated with them. The *price* evaluation might combine parts cost information with an estimate of current market conditions. The *profit* evaluation would need parts cost information and *price*. The *sort-by-profit* evaluation would need the *profit* evaluations on the individual RFQs. Extended examples of the power of this mechanism will be given later in Sections 4.3.3 and 5.

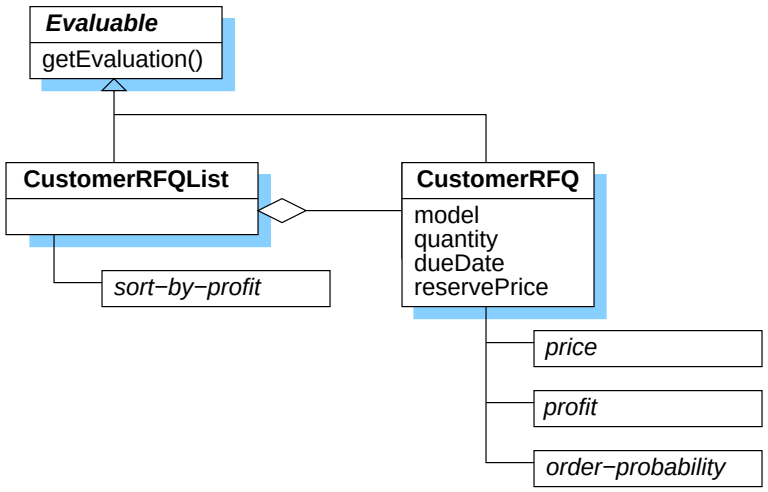

Figure 4: RFQ evaluation example.

4.3 MinneTAC components

A typical MinneTAC agent consists of the 7 components shown in Figure 1. Here we provide more detail on the three "core" components, the Repository, Communications, and Oracle components. The non-core components should be thought of as "role interfaces" since multiple implementations exist for each of them. To flesh out this concept, we also provide an in-depth look at two of our Sales component implementations, the price-driven sales manager used in the 2005 and 2006 competitions, and the quota-driven sales manager used in the 2007 competition.

4.3.1 Repository

The Repository is the one component that is visible to all the other components, as required by the MinneTAC architecture. At the beginning of each day of the game, new incoming messages are deposited into the Repository. The event subsystem described in Section 4.2.1 is then used to notify other components to perform their analyses and decisions. The decision components retrieve data and evaluations from the repository, and record their decisions back into the repository. Finally, the resulting decisions are retrieved from the Repository by the Communications component and returned to the server.

Events are generated in response to state transitions. Figure 5 shows the state transitions and associated events in the Repository. When a component receives the *data-available* event, it is able to inspect the incoming data for the day's transactions and perform whatever analysis is needed to update its models.

When a component receives the *decision* event, it is expected to finalize its decisions and record its outgoing messages back in the Repository.

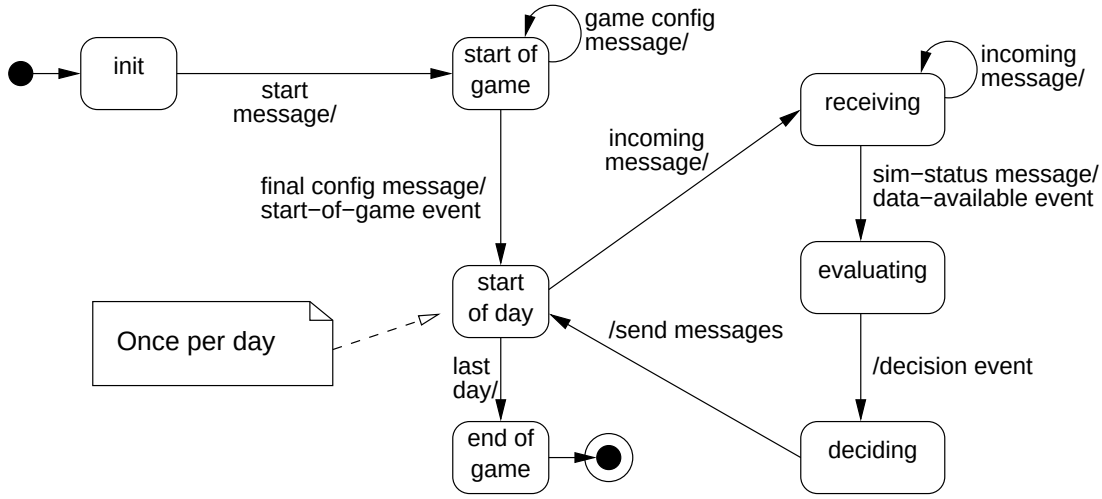

Figure 5: States and transitions in the Repository component. Arcs are labeled with event/action pairs.

From an architectural standpoint, the Repository plays the part of the Blackboard in the *Blackboard* pattern [9], and the remainder of the components, other than the Communications component, act as Knowledge Sources. However, the Control element of the Blackboard pattern is replaced by the Event and the Evaluable/Evaluator mechanisms.

4.3.2 Oracle

The Oracle component is essentially a meta-component, since its only purpose is to provide a framework for a set of small configurable components that may be used to perform analysis and prediction tasks. Most of these are Evaluators, but a few other types are supported as well. The Oracle itself simply reads its configuration data, and uses it to create and configure instances of Evaluators and other subclasses of ConfiguredObject. The top-level classes within the Oracle component are shown in Figure 6.

ConfiguredObject is an abstract class that has a name and an ability to configure itself, given an XML clause. The Oracle creates ConfiguredObject instances and keeps track of them by mapping their names to instances. When it starts, the Oracle processes a configuration clause that typically includes at least two subclauses. The first is a `<setup>` clause, which is processed at the time the Oracle is created, during agent initialization. At this time, objects can be created that do not need access to game parameters. A

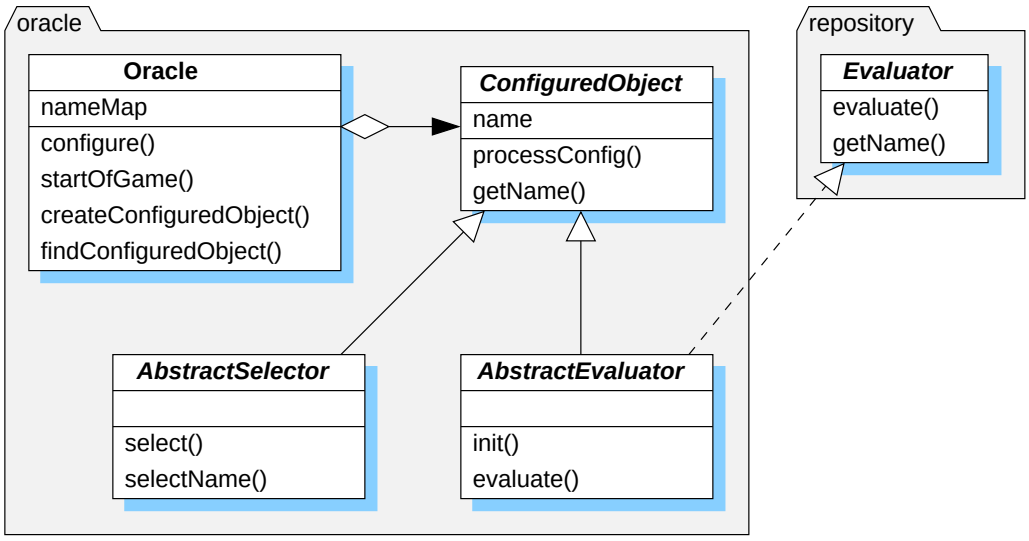

Figure 6: Principal classes in the Oracle component.

typical example is a model element that must read its initialization data from a file or database that has been created off-line, perhaps by analyzing prior games using machine learning techniques [19]. This must be done before the game begins simply because of the time required to set up these models; once the game begins, the agent must complete its work in less than 15 seconds each day. Other clauses are processed by the Oracle after the start-of-game event has been received, thus allowing objects to access game parameters from the Repository when they are created. For example, many evaluators need to initialize themselves using data from the server's Component Catalog or Bill of Materials, which are sent to the agent at the start of a game.

Figure 7 illustrates the configuration clause for an Evaluator called "order-probability" that estimates the sales order probability for each product (the *order-probability* evaluator in Figure 8) by combining a median price estimate with a slope estimate. By convention, the output of any evaluator that promises to estimate order probability is an object called a `Pricer` that has two methods: `getPrice()` returns the predicted median price, and `getPriceForProbability(p)` returns the price corresponding to the given probability p. Inputs to this evaluator are two other evaluators, named "median-price" and "slope-estimate."

The most common subclasses of ConfiguredObject are Evaluators and Selectors. We have discussed Evaluators at length in Section 4.2.2, and we shall see an extended example of their use in the next section. A Selector is simply a switch that can be used to select different models or evaluators in different game

```
<evaluator class="edu.umn.cs.tac.oracle.eval.LinearOPEstimator"
           name="order-probability">
    <inputs>
           <median source="median-price" />
           <slope source="slope-estimate" />
    </inputs>
</evaluator>
```

Figure 7: Configuration clause for an order-probability estimator that uses a median price and a slope estimate as data sources

situations. For example, the early part of a game is typically characterized by customer prices that start high and fall rapidly as agents acquire parts and begin building up inventories. Later in the game, prices are less predictable and more sophisticated models may be useful. A simple DateSelector can be used to switch between pricing models at a particular preset date, or a more sophisticated Selector could be used to switch models once the initial price decline bottoms out. An interesting subtype of Selector is Mixer, which blends one model into another over a period of time, thereby eliminating sharp transitions. This can be important when feedback loops are being used to track prices, as described in the following section.

4.3.3 Sales

To illustrate the power of Evaluators, we show in Figure 8 the evaluation chain that is used to produce sales quotas and set prices in a relatively simple MinneTAC configuration. Each of the cells in this diagram is an Evaluator. A version of the Sales component called PriceDrivenSalesManager is conceptually very simple – it places bids on each customer RFQ for which the randomized-price evaluator returns a non-zero value. The core of this chain is the allocation evaluator, which composes and solves a linear program each game day that represents a combined product-mix and resource-allocation problem that maximizes expected profit. The objective function is

$$\Phi = \sum_{d=0}^{h} \sum_{g \in \mathcal{G}} \Phi_{d,g} A_{d,g} \tag{1}$$

where Φ is the total profit over some time horizon h, $\mathcal{G}$ is the set of goods or products that can be produced by the agent, $\Phi_{d,g}$ is the (projected) profit for good g on day d, and $A_{d,g}$ is the allocation or "sales quota" for good g on day d. The constraints are given by evaluators *available-factory-capacity*, the current day's *effective-demand*, projected *future-demand*, and by Repository data, such as existing and projected inventories of parts and finished products, and outstanding customer and supplier orders. Predicted profit per unit for each product type is the difference between Evaluations called *median-price* and *cost-basis* for those products.

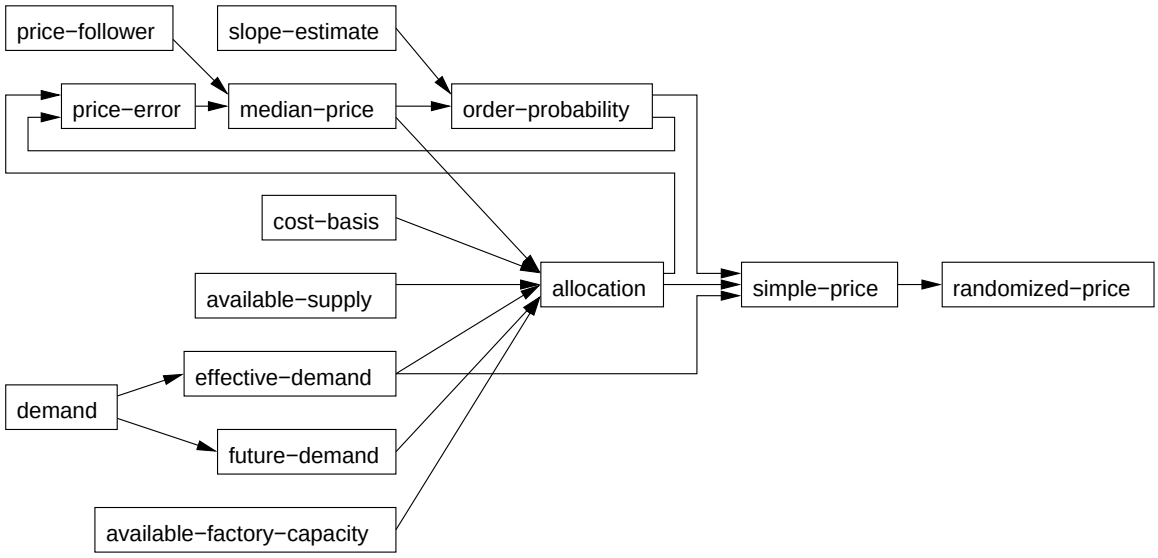

Figure 8: Evaluator chain for sales quota and pricing.

The Evaluation generated by the *allocation* evaluator gives desired sales quotas for each product over a time horizon. Given a sales quota for a given product and an *order-probability* function, the *simple-price* evaluator computes a price that is expected to sell the desired quota, assuming that price is offered on all the demand for that product. In other words, if the quota is 25 units and the demand is for 100 units, *simple-price* computes a price that is expected to be accepted by only 25% of the customers. Since there is some uncertainty in the predictions of price and order probability, randomized-price adds a slight variability to offer prices. This improves the information content and reduces variability of the returned orders.

Market prices are tracked by the *price-follower* evaluator, which observes the daily high prices reported by the server. The *price-follower* implements a straightforward double-exponential smoothing function

$$price_d^{sm} \quad = \quad \alpha(price_d^{max}) + (1-\alpha)(price_{d-1}^{sm} + trend_{d-1}) \tag{2}$$

$$trend_d \quad = \quad \gamma(price_d^{sm} - price_{d-1}^{sm}) + (1-\gamma)trend_{d-1} \tag{3}$$

where $price_d^{max}$ is the observed high price for a given product on day d (the highest price at which the product was sold on the previous day), $price_d^{sm}$ is the smoothed price for the current day d, $trend_d$ is the trend for the current day, and α and γ are the smoothing parameters. The resulting smoothed price estimate is too high to support sales, since it is tracking the daily high price, and it depends strongly on the detailed behavior of other agents. Therefore, we use a feedback mechanism to adjust our price estimates, as shown graphically in Figure 9. Each day d, the *order-probability* evaluator generates a pricing function $P_d(order|price)$, and $price^{est}$ is the price computed for yesterday's sales quotas Q_{d-1}. Yesterday's observed price $price^{obs}$ is computed by applying yesterday's pricing function to today's customer orders O_d. The correction computed by *price-error* is the difference between estimated and observed prices

$$err_p = price^{obs} - price^{est} \tag{4}$$

The *median-price* evaluator then computes a median price

$$price^{med} = price^{sm} + err_p \tag{5}$$

for the current day, giving the corrected output of *price-follower*.

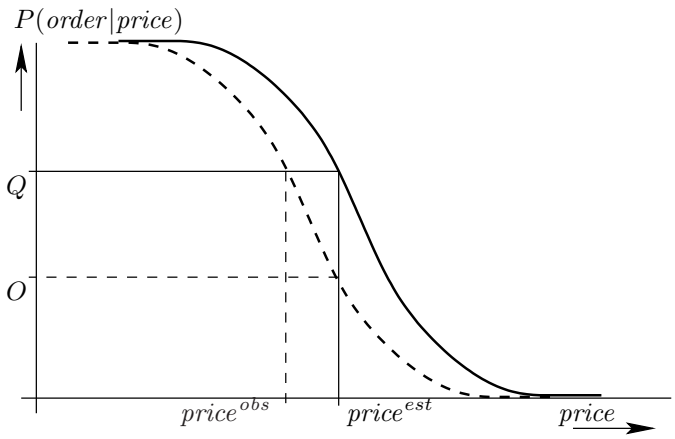

Figure 9: Estimating actual market price $price^{act}$, given sales quota Q, order volume O and an estimate of the order probability function P.

Another compelling example of the power of evaluators in the design of MinneTAC is the sales pricing and forecasting model based on Gaussian Mixture Models described by Ketter et al. in [19]. The elements of

this model replace the *price-follower*, *median-price*, and *order-probability* evaluators shown in Figure 8, and the *slope-estimate* evaluator is omitted in that configuration. There are also two ConfiguredObject types that load training data for this model when the agent starts. This is the configuration that ran in the 2006 TAC SCM tournament; the details are beyond the scope of this paper.

5 Examples

Here we describe in some detail two additional examples of the use and configuration of evaluators in support of significant agent decision processes. The first is the 2007 revision of the Sales component, and the second is the Procurement component that ran in the 2007 Trading Agent Competition.

5.1 Quota-driven Sales

In reality, the outline of the price-driven sales manager given in Section 4.3.3 is significantly oversimplified. This is because the uncertainty in price and order-probability estimates can cause over-selling or under-selling against a quota. Over-selling can be a significant problem when the actual sales volume violates inventory or capacity constraints, with the result that the agent is unable to ship customer orders on time. Late shipments are subject to substantial penalties, and therefore the price-driven sales component avoids overcommitment through detailed accounting of commitments against inventory constraints. The result is that the agent frequently fails to make enough offers to meet its intended sales quotas, in return for avoiding late-delivery penalties. This could be corrected by reducing prices slightly to account for a higher order/offer ratio, but this design computes prices for the original quotas. The feedback mechanism corrects for this, but the variable difference between intended and actual order/offer ratios introduces noise and reduces the effectiveness of the feedback.

The MinneTAC configuration for 2007 takes a somewhat different approach. Instead of avoiding any possibility of overcommitment, it avoids overcommitment in expectation, with an adjustable risk tolerance. The idea is that if we know something about the probability of overcommitment, we can keep that probability under control. Figure 10 shows a new feedback loop that was added to the 2007 configuration.

In this configuration, the *sales-performance* evaluator compares quotas for the previous day with orders on the current day, producing a ratio for each product. This information is smoothed using a decaying rolling

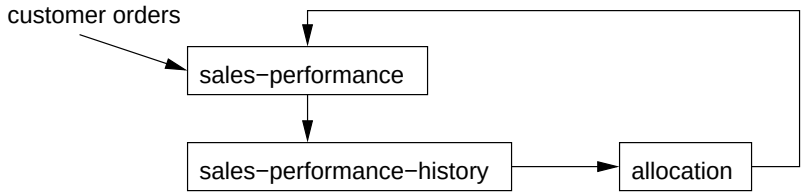

Figure 10: Evaluator feedback chain for managing overcommitment.

average in *sales-performance-history*, which generates mean and standard deviation data for the recent past. Under this scheme, *allocation* adjusts each of its current-day sales quotas $A_{d,g}$ as

$$A'_{d,g} = \min(A_{d,g}, \frac{constraint_{inv}}{\overline{perf}_g + rt \cdot \sigma_g}) \tag{6}$$

where $A_{d,g}$ is the quota value for product g produced by the linear program described in Section 4.3.3, $constraint_{inv}$ is the inventory constraint that (potentially) limits the value of $A_{d,g}$, $\overline{perf}_g$ is the rolling mean of sales performance for g, σ_g is the standard deviation of the recent performance history, rt is a risk-tolerance parameter (typically set to 0.5). Actually, the computation is a bit more complex than this, because inventory constraints due to individual parts can apply to multiple products that require those parts, and so the constraint is typically on the sum of the product allocations associated with any particular part.

We compared the performance of the 2006 and 2007 configurations with regard to this modification by computing the ratios of offers to quotas, orders to quotas, and orders to offers. As we can see in Table 3, the distributions of these ratios are strongly skewed. This is evidenced by the large difference between the mean and median values, and by the large standard deviations. There are a few very large outliers in both data sets. These can happen in situations with small quotas and high demand, if MinneTAC's price estimate is too low. Therefore we compared them with the Wilcoxon-Mann-Whitney two-sample rank-sum test. The p column in Table 3 represent the (two-tailed) probability that the corresponding data from the 2007 and 2006 games are from the same distribution. Clearly, the change from certain avoidance of overcommitment to avoidance in expectation has changed the behavior. In fact, the penalties paid due to overcommitments for the 2007 agent were under 1% of revenue, while the penalties for the 2006 agent were only slightly lower, about 0.8%. The 2006 agent used absolute control of overcommitment only with respect to inventory, and we experienced occasional overcommitment of production capacity.

This data comes from a set of games in the final rounds in the 2006 competition, and from the quarter-final rounds in the 2007 competition. We have omitted data from the first 10 days and the last 10 days

	MinneTAC 2006			MinneTAC 2007			
	mean	median	σ	mean	median	σ	p
offer/quota	3.41	1.33	8.86	4.43	1.35	12.47	< 0.01
order/quota	2.05	0.91	7.17	1.84	0.98	6.77	< 0.01
order/offer	0.58	0.80	0.44	0.57	0.69	0.40	< 0.01

Table 3: Performance comparison of the 2006 and 2007 MinneTAC configurations.

of each game, because those data tend to be dominated by the extreme instability of price estimates at the beginning and end of each game. The comparison is complicated by several factors, including the high variability in the game environment, and the fact that the 2006 data is from a different set of games, with a different set of competitors, compared to the 2007 data. Because the 2007 agent is presumably doing a better job of matching its optimized quotas to its actual sales behavior, we would expect the 2007 agent to exhibit more aggressive selling, as evidenced by higher ratios of offers to quotas and of offers to orders. However, assuming the price-tracking feedback is working well, we should see smaller differences in the relationship between quotas and orders. This is in fact what we observe. A more complete comparison of these configurations, one that will allow us to show the impact of this small change on agent profitability, will be performed using the method described by Sodomka et al. in [30].

5.2 Procurement

Procurement decisions in a supply-chain trading agent must balance a number of factors aside from ensuring that components are available to the manufacturing operation when they will be needed. In supply-chain situations generally, it is common for prices to be lower for longer leadtimes and larger commitments, but procurement must balance procurement cost against the cost to hold inventory. In many environments, including TAC SCM, reputation is also an issue. If suppliers are repeatedly asked to bid on large quantity orders, and then the offers are rejected, suppliers will likely raise their offer prices as a way of discounting the value of the uncertain business. Evaluator chain for the 2007 Procurement manager. In Figure 11, we show the evaluator chain that drives procurement decisions in MinneTAC.

The key elements of the procurement decision process are the *price-model* and estimates of needed supply

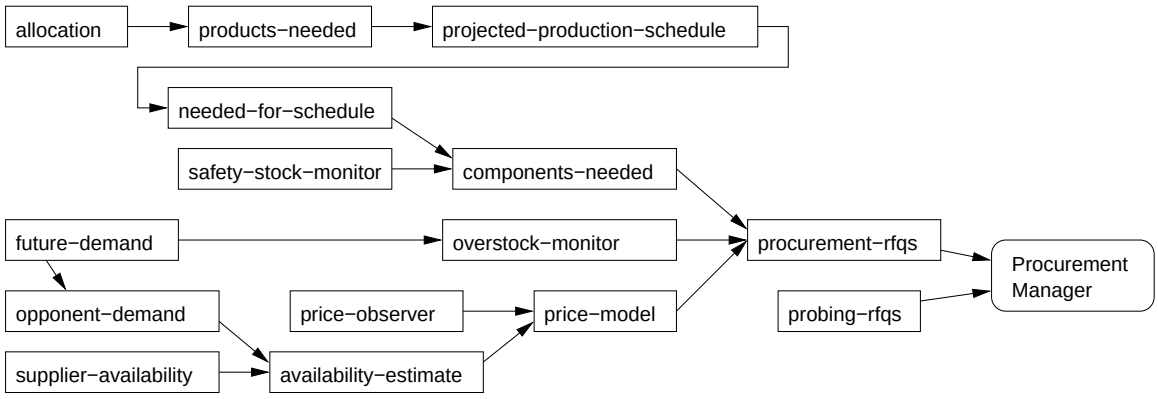

Figure 11: Evaluator chain for procurement.

over both short-term and longer-term time horizons. The *price-observer* evaluator monitors all interactions with suppliers that might produce price signals, and supplies a preliminary price model based on a weighted nearest-neighbors method similar to that used by Benisch et al. in the 2005 CMieux agent [6]. Some of those signals are produced from price-probes – simple requests to suppliers for future price estimates that carry no commitment to purchase. Probes are generated by *probing-rfqs* when the price model's data quality falls below a threshold, generally due to poor coverage of a potentially interesting future time period. Since supplier pricing in TAC SCM is a well-known function of demand and supply, another element of the pricing model is the *availability-estimate*, which estimates supplier capacity from information about expected future *opponent-demand* and from *supplier-availability* data derived from observed prices, and from situations where a supplier is unable to fulfill a request due to lack of capacity.

Demand for components must be estimated over a relatively long horizon in order to achieve good prices in the procurement market. Part of that information comes from the *allocation* module that was discussed in Section 4.3.3, and part comes from longer-term estimates based on general knowledge of the game environment. These are combined in *products-needed*. The capacity limits of the production facility are factored in by *projected-production-schedule*, and the conversion from products to parts is made by *needed-for-schedule*. This is combined with inventory monitoring from *safety-stock-monitor* to produce an integrated view of future component requirements in *components-needed*. Finally, *procurement-rfqs* combines future component needs with the price model and a second inventory monitor *overstock-monitor* that is focused on minimizing end-of-game inventory to produce sets of RFQs that are expected to procure the needed

components at the lowest possible price.

6 Evaluation

The software architecture of MinneTAC is strongly focused on strict control of dependencies, and on flexible configuration. We evaluate the success of this design by asking two questions: (1) Does the agent perform well, and to what extent does the design affect agent performance? (2) Does the design meet the "usability" challenges described in Section 2.2?

6.1 Performance

There are two measures of agent performance that could be affected by the design. One is related to overhead: Does the design impose an unacceptable runtime overhead? The other is how well the agent performs in competition against other agents that have been implemented with different designs.

The Excalibur framework does indeed impose some overhead when the agent starts up, since it must read configuration files, find and load code for components, and set up and configure the components. However, once the agent is running, there is essentially no overhead imposed by the framework. Event processing and evaluation is done by direct lookup, since event handlers and Evaluators are registered when components are loaded. We have run 6 MinneTAC agents (with a simple Sales component) on the same desktop machine (a 1 GHz Pentium), and all 6 agents are able to complete their daily decision procedures in less than 1 second. A Sales component that relies on solving a linear program each round takes longer, but its performance is almost entirely determined by the time required to compose and solve the linear program.

MinneTAC has done reasonably well in the official TAC SCM tournaments since 2003. In 2005 and 2006 it was a finalist. Each year, MinneTAC has been fielded with a new implementation of at least one of the decision components (Sales, Procurement, Production, and Shipping), and several others have been implemented but have not been entered into the competition. The ease with which these new components could be implemented and configured into the agent is a testament to the design we describe here.

6.2 Usability

The principal usability criterion is whether researchers can effectively work on the various decision problems independently, and whether they can extract the data they need to analyze performance and confirm or refute hypotheses.

There is considerable evidence that our design has met its goals.

- Inexperienced student programmers have been able to contribute significant functionality without needing to understand the entire system. Examples include two different Shipping components, two different Production components, five different Sales components, six different Procurement components, and over 80 different Evaluators, written by at least 22 students over a period of four years.
- The standardized log-message format produced by the Excalibur infrastructure makes data extraction relatively easy, even though MinneTAC generates approximately 5Mb of data for a typical game. A wide variety of analyses have been carried out with this data. An example of such an analysis is given in [22], where we were able to show that the original design of the game gave a large advantage to the agent that won a procurement lottery on the first day of the game.
- Selectors and Mixers (see Section 4.3.2) are very recent additions to the MinneTAC design, and they add considerable expressive power for constructing models that can learn and adapt to market conditions. Once the need was clear, they were added and tested in less than four hours, and required no other changes in the rest of the system.
- MinneTAC is an open-source project, available at `http://tac.cs.umn.edu`. The source release includes the full infrastructure, and relatively simple examples of each of the decision components, a few evaluators, and a sample set of configuration files. It has been downloaded almost 900 times since its initial release in April 2005. There have also been over 1000 binary downloads of the 2005 and 2006 competition versions of MinneTAC, which include some relatively complex evaluators that are not sufficiently documented or tested for source release.

7 Conclusions and Future Work

Experimental work with multi-agent systems requires an implementation. Often, the design qualities that best support experimental work are different from those normally considered "ideal" in industry. In complex

economic scenarios such as TAC SCM, the desired design qualities include clean separation of infrastructure from decision processes, ease of implementation of multiple decision processes, clean separation of different decision processes from each other, and controllable generation of experimental data. In a competitive environment, the ability to easily compose multiple agents with different combinations of decision process implementations makes it possible to test hypotheses about the effectiveness of competing decision models.

We show one way to construct such an agent, using a readily-available component framework. The framework provides the ability to compose agent systems from sets of individual components based on simple configuration files. We also show that two basic mechanisms, event distribution with the Observer pattern and our pipe-and-filter style Evaluator-Evaluation scheme, permit an appropriate level of component interaction without introducing unnecessary coupling among components. The ability to compose complex evaluator chains out of relatively simple, straightforward elements has greatly simplified the design of the decision components themselves.

There are many possible extensions to the basic design we show here. One that we are currently pursuing is to add an "executive" component that would allocate "resources" to competing implementations of basic decision processes within a single agent. This would allow a high degree of adaptability in the game environment, where the level of demand can fluctuate greatly, and where the actions of other agents can have a significant impact on the markets.

8 Acknowledgment

We would like to thank the organizers of the TAC SCM game for a stimulating research problem. Partial support for this research is gratefully acknowledged from the National Science Foundation under award NSF/IIS-0414466.

References

[1] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison Wesley, 1998.

[2] F. Bellifemine, A. Poggi, G. Rimassa, JADE–A FIPA-compliant agent framework, Proceedings of PAAM 99 (1999) 97–108.

[3] M. Benisch, J. Andrews, N. Sadeh, Pricing for customers with probabilistic valuations as a continuous knapsack problem, Fredericton, NB, 2006.

[4] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, M. Tschantz, Botticelli: A supply chain management agent, in: Proc. of the 3rd Int'l Conf. on Autonomous Agents and Multi-Agent Systems, ACM, ACM Press, New York, NY, USA, 2004.

[5] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, M. Tschantz, Botticelli: A supply chain management agent designed to optimize under uncertainty, ACM Trans. on Comp. Logic 4 (3) (2004) 29–37.

[6] M. Benisch, A. Sardinha, J. Andrews, N. Sadeh, CMieux: adaptive strategies for competitive supply chain trading, in: Proceedings of the 8th International Conference on Electronic Commerce, ACM Press New York, NY, USA, 2006.

[7] F. M. T. Brazier, C. M. Jonker, J. Treur, Principles of component-based design of intelligent agents, Data and Knowledge Engineering 41 (1) (2002) 1–28.

[8] D. Burke, K. Brown, S. Tarim, B. Hnich, Learning Market Prices for a Real-time Supply Chain Management Trading Agent, in: AAMAS06: Workshop on Trading Agent Design and Analysis (TADA/AMEC), 2006.

[9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: a System of Patterns, Wiley, 1996.

[10] S. Chopra, P. Meindl, Supply Chain Management, Pearon Prentice Hall, New Jersey, 2004.

[11] J. Collins, R. Arunachalam, N. Sadeh, J. Ericsson, N. Finne, S. Janson, The supply chain management game for the 2006 trading agent competition, Tech. Rep. CMU-ISRI-05-132, Carnegie Mellon University, Pittsburgh, PA (November 2005).

[12] A. S. Foundation, Apache excalibur, http://excalibur.apache.org/ (2006).

[13] D. Galindo, D. Ayala, Tiancalli06: An Agent for the Supply Chain Management Game 2006, Proceedings of the International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce.

[14] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Professional Computing Series, Addison-Wesley, 1995.

[15] M. He, A. Rogers, D. Esther, N. R. Jennings, Designing and evaluating an adaptive trading agent for supply chain management applications, in: IJCAI05: Workshop on Trading Agent Design and Analysis, Edinburgh, Scotland, 2005.

[16] M. He, A. Rogers, X. Luo, N. R. Jennings, Designing a successful trading agent for supply chain management, in: Proc. of the 5th Int'l Conf. on Autonomous Agents and Multi-Agent Systems, Hakodate, Japan, 2006.

[17] P. Jordan, C. Kiekintveld, M. Wellman.

[18] P. Keller, F.-O. Duguay, D. Precup, Redagent-2003: An autonomous market-based supply-chain management agent, in: Proc. of the Third Int'l Conf. on Autonomous Agents and Multi-Agent Systems, ACM, ACM Press, New York, NY, USA, 2004.

[19] W. Ketter, J. Collins, M. Gini, A. Gupta, P. Schrater, Identifying and forecasting economic regimes in TAC SCM, in: H. L. Poutré, N. Sadeh, S. Janson (eds.), Agent-Mediated Electronic Commerce: Designing Trading Agents and Mechanisms, vol. 3937 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2006, pp. 113–125.

[20] W. Ketter, J. Collins, M. Gini, A. Gupta, P. Schrater, A predictive empirical model for pricing and resource allocation decisions, in: Proc. of 9th Int'l Conf. on Electronic Commerce, Minneapolis, Minnesota, USA, 2007.

[21] W. Ketter, J. Collins, M. Gini, A. Gupta, P. Schrater, Detecting and Forecasting Economic Regimes in Multi-Agent Automated Exchanges, Decision Support Systems (2008) Forthcoming.

[22] W. Ketter, E. Kryzhnyaya, S. Damer, C. McMillen, A. Agovic, J. Collins, M. Gini, Analysis and design of supply-driven strategies in TAC-SCM, in: AAMAS04: Workshop on Trading Agent Design and Analysis, New York, 2004.

[23] C. Kiekintveld, J. Miller, P. Jordan, M. P. Wellman, Controlling a supply chain agent using value-based decomposition, in: Proc. of 7th ACM Conf. on Electronic Commerce, Ann Arbor, Mich., 2006.

[24] C. Kiekintveld, M. P. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, V. Soni, M. Rudary, Distributed feedback control for decision making on supply chains, in: Proc. Int'l Conf. on Automated Planning and Scheduling, Whistler, BC, Canada, 2004.

[25] D. Kinny, M. Georgeff, Modeling and design of multi-agent systems, in: M. J. Wooldridge, J. P. Müller, N. R. Jennings (eds.), Intelligent Agents III, vol. 1193 of Lecture Notes in Artificial Intelligence, Springer, Berlin, 1997, pp. 1–20.

[26] I. Kontogounnis, K. C. Chatzidimitriou, A. L. Symeonidis, P. A. Mitkas, A Robust Agent Design for Dynamic SCM environments, in: 4th Hellenic Conference on Artificial Intelligence (SETN'06), Heraklion, Crete, Greece, 2006.

[27] P. Moraitis, E. Petraki, N. Spanoudakis, Engineering JADE agents with the Gaia methodology, in: R. Kowalszyk, J. Miller, H. Tianfield, R. Unland (eds.), Agent Technologies, Infrastructures, Tools, and Applications for e-Services, vol. 2592 of Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 77–91.

[28] D. Pardoe, P. Stone, Tactex-05: A champion supply chain management agent, in: Proc. of the Twenty-First Nat'l Conf. on Artificial Intelligence, AAAI, Boston, Mass., 2006.

[29] V. Podobnik, A. Petric, G. Jezic, The CrocodileAgent: Research for Efficient Agent-Based Cross-Enterprise Processes, Lecture Notes in Computer Science 4277 (2006) 752–762.

[30] E. Sodomka, J. Collins, M. Gini, Efficient statistical methods for evaluating trading agent performance, in: Proc. of the 22nd Nat'l Conf. on Artificial Intelligence, AAAI, AAAI Press, Vancouver, BC, 2007.

[31] K. Stathis, A. C. Kakas, W. Lu, N. Demetriou, U. Endriss, A. Bracciali, PROSOCS: a platform for programming software agents in computational logic, in: J. Müller, P. Petta (eds.), Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation" (AT2AI-4 – EMCSR'2004 Session M), Vienna, Austria, 2004.

[32] K. Sycara, A. S. Pannu, The RETSINA multiagent system: towards integrating planning, execution, and information gathering, in: Proc. of the Second Int'l Conf. on Autonomous Agents, 1998.

[33] C. Szyperski, Component Software: Beyond Object-Oriented Programming, ACM Press, 1998.

[34] J. A. Vetsikas, B. Selman, A principled study of the design tradeoffs for autonomous trading agents, in: Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems, 2003.

[35] P. Vytelingum, R. Dash, M. He, N. Jennings, A Framework for Designing Strategies for Trading Agents, Proc. IJCAI Workshop on Trading Agent Design and Analysis (2005) 7–13.

[36] M. P. Wellman, J. Estelle, S. Singh, Y. Vorobeychik, C. Kiekintveld, V. Soni, Strategic interactions in a supply chain game, Computational Intelligence 21 (1) (2005) 1–26.

[37] R. J. Wieringa, Design Methods for Reactive Systems, Morgan Kaufmann, San Francisco, 2003.

[38] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Second Edition, Morgan Kaufmann, 2005.

Publications in the Report Series Research* in Management

ERIM Research Program: "Business Processes, Logistics and Information Systems"

2007

India: a Case of Fragile Wireless Service and Technology Adoption?
L-F Pau and J. Motiwalla
ERS-2007-011-LIS
http://hdl.handle.net/1765/9043

Some Comments on the Question Whether Co-occurrence Data Should Be Normalized
Ludo Waltman and Nees Jan van Eck
ERS-2007-017-LIS
http://hdl.handle.net/1765/9401

Extended Producer Responsibility in the Aviation Sector
Marisa P. de Brito, Erwin A. van der Laan and Brijan D. Irion
ERS-2007-025-LIS
http://hdl.handle.net/1765/10068

Logistics Information and Knowledge Management Issues in Humanitarian Aid Organizations
Erwin A. van der Laan, Marisa P. de Brito and S. Vermaesen
ERS-2007-026-LIS
http://hdl.handle.net/1765/10071

Bibliometric Mapping of the Computational Intelligence Field
Nees Jan van Eck and Ludo Waltman
ERS-2007-027-LIS
http://hdl.handle.net/1765/10073

Approximating the Randomized Hitting Time Distribution of a Non-stationary Gamma Process
J.B.G. Frenk and R.P. Nicolai
ERS-2007-031-LIS
http://hdl.handle.net/1765/10149

Application of a General Risk Management Model to Portfolio Optimization Problems with Elliptical Distributed Returns for Risk Neutral and Risk Averse Decision Makers
Bahar Kaynar, S. Ilker Birbil and J.B.G. Frenk
ERS-2007-032-LIS
http://hdl.handle.net/1765/10151

Optimal Zone Boundaries for Two-class-based Compact 3D AS/RS
Yugang Yu and M.B.M. de Koster
ERS-2007-034-LIS
http://hdl.handle.net/1765/10180

Portfolios of Exchange Relationships: An Empirical Investigation of an Online Marketplace for IT Services
Uladzimir Radkevitch, Eric van Heck and Otto Koppius
ERS-2007-035-LIS
http://hdl.handle.net/1765/10072

From Closed-Loop to Sustainable Supply Chains: The WEEE case
J. Quariguasi Frota Neto, G. Walther, J.Bloemhof, J.A.E.E van Nunen and T.Spengler
ERS-2007-036-LIS
http://hdl.handle.net/1765/10176

A Methodology for Assessing Eco-Efficiency in Logistics Networks
J. Quariguasi Frota Neto, G. Walther, J.Bloemhof, J.A.E.E van Nunen and T.Spengler
ERS-2007-037-LIS
http://hdl.handle.net/1765/10177

Strategic and Operational Management of Supplier Involvement in New Product Development: a Contingency Perspective
Ferrie E.A. van Echtelt, Finn Wynstra and Arjan J. van Weele
ERS-2007-040-LIS
http://hdl.handle.net/1765/10456

How Will Online Affiliate Marketing Networks Impact Search Engine Rankings?
David Janssen and Eric van Heck
ERS-2007-042-LIS
http://hdl.handle.net/1765/10458

Modelling and Optimizing Imperfect Maintenance of Coatings on Steel Structures
R.P. Nicolai, J.B.G. Frenk and R. Dekker
ERS-2007-043-LIS
http://hdl.handle.net/1765/10455

Human Knowledge Resources and Interorganizational Systems
Mohammed Ibrahim, Pieter Ribbers and Bert Bettonvil
ERS-2007-046-LIS
http://hdl.handle.net/1765/10457

Revenue Management and Demand Fulfilment: Matching Applications, Models, and Software
Rainer Quante, Herbert Meyr and Moritz Fleischmann
ERS-2007-050-LIS
http://hdl.handle.net/1765/10464

Mass Customization in Wireless Communication Services: Individual Service Bundles and Tariffs
Hong Chen and Louis-Francois Pau
ERS-2007-051-LIS
http://hdl.handle.net/1765/10515

Individual Tariffs for Mobile Services: Analysis of Operator Business and Risk Consequences
Hong Chen and Louis-Francois Pau
ERS-2007-052-LIS
http://hdl.handle.net/1765/10516

Individual Tariffs for Mobile Services: Theoretical Framework and a Computational Case in Mobile Music
Hong Chen and Louis-Francois Pau
ERS-2007-053-LIS
http://hdl.handle.net/1765/10517

Individual Tariffs for Mobile Communication Services
Hong Chen and Louis-Francois Pau
ERS-2007-054-LIS
http://hdl.handle.net/1765/10518

Is Management Interdisciplinary? The Evolution of Management as an Interdisciplinary Field of Research and Education in the Netherlands
Peter van Baalen and Luchien Karsten
ERS-2007-047-LIS
http://hdl.handle.net/1765/10537

Detecting and Forecasting Economic Regimes in Multi-Agent Automated Exchanges
Wolfgang Ketter, John Collins, Maria Gini, Alok Gupta and Paul Schrater
ERS-2007-065-LIS
http://hdl.handle.net/1765/10594

Emergency Messaging to General Public via Public Wireless Networks
P.Simonsen and L-F Pau
ERS-2007-078-LIS

Flexible Decision Control in an Autonomous Trading Agent
John Collins, Wolfgang Ketter and Maria Gini
ERS-2007-079-LIS

---

* A complete overview of the ERIM Report Series Research in Management:
https://ep.eur.nl/handle/1765/1

ERIM Research Programs:
LIS Business Processes, Logistics and Information Systems
ORG Organizing for Performance
MKT Marketing
F&A Finance and Accounting
STR Strategy and Entrepreneurship