Toward Human-Agent Competition in TAC SCM

Andrew Nelson*, **Dickens Nyabuti, John Collins, Wolfgang Ketter†, and Maria Gini**
Dept of CSE, University of Minnesota, Minneapolis, USA
†Rotterdam Sch. of Mgmt., Erasmus University, Rotterdam, NL

Abstract

We propose a variation of the TAC SCM supply-chain trading competition, in which human decision-makers compete with fully-autonomous agents. Because of the complexity and time pressures of the competition environment, humans may be assisted by semi-autonomous agents, which could be modifications of existing agents. The research goal is to discover what kinds of decision support will make a human decision-maker most effective in this environment. We show how an existing agent might be modified to operate in this new competition by updating our MinneTAC agent into a highly configurable, semi-autonomous agent that can support human users playing a variety of roles in the modified competition environment. The agent's decision processes are composed of networks of simple services that are described using an OWL ontology. The ontology describes the structure of the service network, along with the structure and semantics of the data elements that are produced and consumed by individual services.

Introduction

Organized competitions are an effective way to drive research and understanding in complex domains, free of the complexities and risks of operating in open, real-world environments. Artificial economic environments typically abstract certain interesting features of the real world, such as markets and competitors, demand-based prices and cost of capital, and omit others, such as human resources, secondary markets, taxes, and seasonal demand. The Trading Agent Competition for Supply-Chain Management (Collins *et al.* 2005) (TAC SCM) is an economic simulation in which competing autonomous agents operate in a simple supply-chain scenario, purchasing components, managing a factory and warehouse, and selling finished products to customers.

Supply chain management organizes the transfer of goods, information, and services between suppliers and buyers. Traditional supply chains are created and maintained through the interactions of human representatives of the various companies involved. Much progress in business processes comes from automating the elements that do not require human judgments, and from providing better and more timely information for those that do. Recently many companies have adopted Business Intelligence (BI) (Eckerson 2005) techniques to support information needs, and a variety of systems to automate trivial tasks like data entry to decrease the margin of human error.

Despite the many important research results arising from TAC SCM and the other Trading Agent Competitions, business people are not ready to trust fully autonomous agents with critical business decisions (Maes 1994), and existing BI systems are arguably not sufficiently flexible to produce the kinds of ad hoc information and analysis that would truly leverage their skills and experience and maximize their effectiveness (Collins, Ketter, & Gini 2008). One way to address both of these issues is through configurable, composable, and transparent decision processes that are fully described in terms that end users can understand. Our approach to the needed flexibility and transparency is called "ontology-driven decision support," in which information, analyses, and decisions are composed of a large variety of data views and small, single-purpose analysis modules that can be composed into dataflow networks to produce results with well-defined business meaning.

What is missing at this point is a clear understanding of exactly what kinds of information, and what level of automated assistance, will make human decision-makers most effective in complex, dynamic, and multi-faceted trading situations. It is also not clear whether an experienced human decision-maker, with appropriate information and assistance, can outperform the current generation of fully-autonomous agents. We believe that a modification to the Trading Agent Competition for Supply-Chain Management (TAC SCM), in combination with flexible mixed-initiative agent technology such as we describe here, can help answer these important questions.

The next section places our work in the context of related work. We then review the classification of decisions into strategic, tactical, and operational levels in both a general business setting and in TAC SCM. These distinctions are important to understand our approach to adjustable-autonomy decision making. We then discuss the ways in which the game and agents must be modified to operate in a human-agent competition environment. Next, we describe our approach to ontology-driven decision support and to human-agent interaction, providing a few dashboard examples that

*Work supported in part by National Science Foundation grant IIS-0414466

could support human decision making at multiple levels of abstraction. Finally, we conclude and present our future research agenda.

Related work

Advanced decision support systems and software agents promise to assist businesses by acting rationally on behalf of humans in numerous application domains. Examples include procurement (Sandholm *et al.* 2005), scheduling and resource management (Collins, Ketter, & Gini 2002), Internet shopbot agents (Montgomery *et al.* 2004), and personal information management (Myers *et al.* 2007). In real-world settings, fully-autonomous trading agents are inappropriate, so we have introduced an alternate style of semi-autonomous agents that we call "Advocate Agents" (Ketter *et al.* 2008). In addition to composing services and displaying data, these agents can help buyers to mine information from various sources, such as vendor web pages, and to harvest other facts.

There is a strong need to develop highly interactive, personalized dashboards so that managers are able to effectively scan and identify key information quickly (Ontrup *et al.* 2009). Creating performance and information dashboards is part of the new emerging field of BI (Eckerson 2005). BI systems provide functions such as real-time monitoring, performance reporting and support for exploring solution space with normative models, statistical techniques and visualization. Business intelligence software can crawl the web, mine data, and generate reports customized to user preferences. Our architecture fully supports BI and our dashboards are customizable for individual managers. BI is a customized solution, thus the purpose of the agents is to cover specific requirements of its users (Azvine *et al.* 2006). Gregg et al. (Gregg & Walczak 2006) claim that auction advisors like agents can keep buyers better informed about current market offers and maximize the outcome, even if the buyers have conflicting preferences (Ketter *et al.* 2008). In other words the main objective of the BI and agent is to process information and deliver in a reliable format for further consideration; according to Orlikowski and Iacono (Orlikowski & Iacono 2001) this type of technology can be basically viewed as "information processing tool".

The business world distinguishes between strategic, tactical, and operational decision making and their appropriate information needs (Eckerson 2005):

Strategic view lets executives and staff chart their progress toward achieving strategic objectives.

Tactical view helps managers and analysts track and analyze departmental activities, processes, and projects.

Operational view enables front-line workers and supervisors to track core operational processes.

Strategic dashboards emphasize management more than monitoring or analysis, tactical dashboards emphasize analysis more than monitoring or management, and operational dashboards emphasize monitoring more than analysis or management. The right approach to supporting these three decision levels varies from a focus on decision-support for knowledge workers to a focus on decision automation. Operational decisions are ideal for automation, and therefore to apply fully-autonomous intelligent agents. Tactical decisions, are also a good candidate for decision automation, but usually for partial automation guided by semi-autonomous agents. Strategic decisions might show results only over long time frames, suitable for reporting with standard business intelligence capabilities, and semi-autonomous agents. Performance management is particulary good at tracking the effectiveness of operational and tactical decisions, since the feedback times are much shorter than strategic decisions.

In prior work we show how to construct an intelligent trading agent based on dynamic service networks and present basic GUI components (Collins, Ketter, & Gini 2008). In the following we introduce an Ontology-Driven Decision Support which extends the basic architecture with an ontology, an inference engine, and connect to customizable user interfaces. To our knowledge the Michigan AuctionBot (Wurman, Wellman, & Walsh 1998) is the only other auction platform in the community that allows for the participation of human and software agents. AuctionBot was used as the underlying platform fr the design and implementation of TAC Travel (now called TAC Classic) (Stone & Greenwald 2005). AuctionBot was mainly designed to support research into auctions and mechanism design. Our architecture is designed to support research into auctions, business networks, ontologies, and human-agent interaction.

Game and agent modifications

Games in the current TAC SCM scenario run on a strict time schedule. Each agent must make thousands of decisions during each 15-second cycle. This strict, compressed schedule works well when the competitors are fully autonomous agents, because a full year of simulated supply chain interaction can be completed in under an hour. However, it appears to effectively eliminate the possibility of usable human interaction.

It seems likely that many interesting research questions into human-agent competition using the existing game parameters will require longer than an hour to complete. This leads to two important questions: how much longer, and how effectively could good decision support limit the time required. This leads to a set of goals for a modified game scenario:

- As far as possible, the competition should retain its full complexity and dynamic qualities, since the goal is to explore human decision-making in such a complex environment, and compare it with the performance of autonomous agents.
- Existing agents should be able to operate in a human-agent competition without modification. This means that the existing "agentware" software layer should be compatible with the new scenario.
- A human user should be able to "stall" a game cycle for a limited time, perhaps as much as a minute or two, in order to examine data and make decisions. However, the game should be able to run as quickly as possible given the cognitive limitations of the users. In other words, game cycles should stall only until the users have finished their decisions, or until they time out. Behavior of the agent in a

timeout situation should be well-defined.

- A given human-interface agent may support multiple users in different roles, such as procurement, production, sales, etc.
- Because even a shortened game may last for an uncomfortable duration, it may be necessary to have an ability to suspend a game and resume it at a later time. Unfortunately, this conflicts with the goal of using existing agents without modification.

We have implemented the basic modifications to the server and agentware that will allow for variable-length cycles. In order to retain compatibility with existing agents, there are additional data elements passed between agent and server when the agent logs in to the server. An human-interface agent requests a maximum daily timeout in its login message, and the server notifies agents of the maximum daily timeout in its response. This prevents an existing agent from timing out, but runs the risk that an existing agent is using the time/cycle information to control its computations, and therefore may miss cycles that are not stalled by users. There appears to be no fix for this that does not involve upgrading existing agents with a new agentware library.

Figure 1 shows the modified TAC SCM game with a dashboard that mediates the human-agent interaction. We have implemented a role-based system, so that every major decision component, i.e. procurement, production, and sales, can be carried by a team of people. The role-based structure enables us to use the system for teaching supply-chain management students, and to execute experiments with human subjects. =

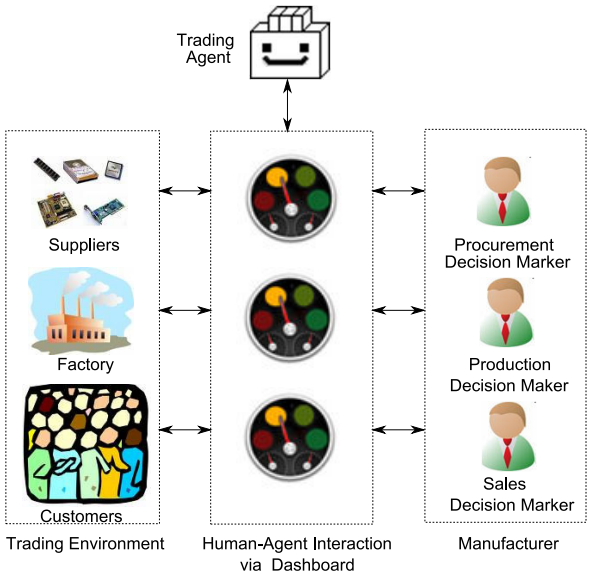

Figure 1: Human-agent interaction in TAC SCM.

There are (at least) three other pieces of the puzzle that must be in place before human-agent competitions are possible. First, agents running with human decision makers must notify the server when they are ready to proceed. In a game with multiple human-interface agents, the last agent to finish before the deadline controls the end of the cycle. Second, the game viewer must not run its clock independently of the server. Third, we need agents with usable human interfaces, and ideally those interfaces will be remote - they will not require that the user be sitting at the machine that is running the agent. We describe a design for such an agent in a subsequent section.

Ontology-Driven Decision Support

Our method for transforming an autonomous agent into a human-interactive agent hinges on its semantic description. To see how a human interface might use such a model of the agent we first discuss the architecture of our agent, MinneTAC (Collins, Ketter, & Gini 2009). The agent has a fine-grained modular design. All software of consequence to the agent's performance in the TAC SCM competition is either some structured data in a "repository," or is encapsulated in a small "data-flow service" module, each of which takes zero or more inputs and produces an output. We limit the topology of these services to simplify the problems of human understanding and of service composition. We define three types of data-flow services:

A **Source** is a simple data source; it provides a specific view of repository data.

An **Evaluator** takes input from sources and other evaluators, performs some transformation, and makes the result available to other services.

A **Sink** takes input from sources and evaluators in order to change the agent's state or affect the outside world. In the MinneTAC agent, this data is either stored back in the repository, or routed to the simulation server, where it represents the agent's decisions.

A working MinneTAC agent configuration is wholly defined by the sinks, sources and evaluators used; the data dependencies between them; and the parameter values sent to each. We visualize these data-flow services as a directed graph. Figure 2 shows a small portion of the graph in a typical MinneTAC configuration that determines sales prices for finished goods. The first component of a semantic description of our agent includes information intrinsic to the Java classes for each of our services and the data dependencies among them. To make these descriptions we use the World Wide Web Consortium's Resource Description Framework (RDF)[1]. Semantic models are built in RDF by way of triplet statements of the form <Subject> <Predicate> <Object>. Such a statement relates its subject to another object by a binary predicate, for example <order-probability> <has-output> <pricers>.

We can represent a collection of RDF triples as a directed graph, where each node represents some object and each arc represents a predicate relation between them. Figure 3 shows a graphical RDF description of an evaluator and a source which have "oracle.eval.SimplePriceEvaluator" and "oracle.eval.QuantityEvaluator" as respective class names. This figure also shows information that is extrinsice to the classes, including unique identifiers for each service, needed

[1] http://www.w3.org/RDF/

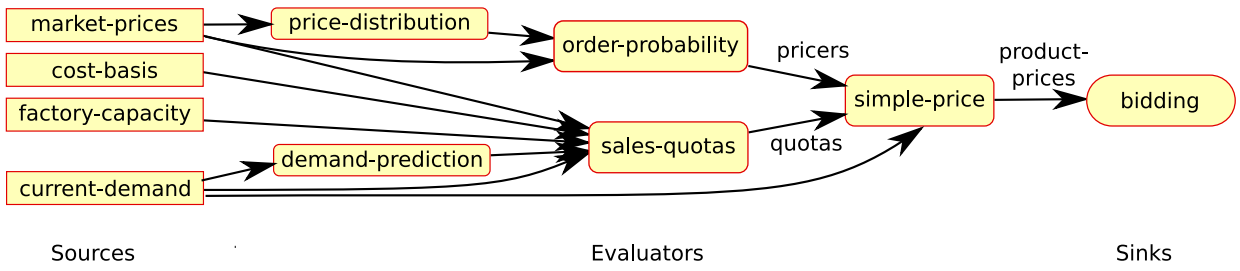

Figure 2: Evaluator network for computing sales prices.

since we often use the same class multiple times within an agent. We show the data dependency between the customer-quantity source and the simple-price evaluator as well as additional node labels, the white rectangles, which we explain shortly.

The second portion of our semantic model takes the form of an ontology. Ontologies are made up of concepts, such as classes of individual objects, which form a hierarchy through subsumption relationships. We compose our ontologies using the World Wide Web Consortium-defined Web Ontology Language (OWL)[2]. OWL is an extension of RDF which includes the special predicate "is-a" which allows us to make statements such as <Price> <is-a> <Number>, <Number> <is-a> <DataType>.

A partial view of our current ontology can be seen in Figure 4. We are using the OWL-DL subspecies of OWL, so named for its equivalence to description logics, because it is expressive enough for our purposes while its syntactic limitations allow for inference performed upon the ontology to guarantee computation completeness and decidability.Using the open source inference engine Pellet[3], our software can take the two statements above and conclude <Price> <is-a> <DataType>.

Our model relates its two sets of semantic information, which are commonly referred to as the terminological (TBox) and assertion (ABox) sets, via another special predicate in OWL, the "has-type" relationship <oracle.eval.SimplePriceEvaluator> <has-type> <Evaluator> These relations are shown in Figure 3 within the boxes next to each node.

Now that we have described our agent to a certain level of detail, how can we use it? Firstly, semantic information is used by developers and composers of the agent. The ontology can be used to validate individual service descriptions via another OWL construct, the class restriction. With OWL restrictions we designate which predicates must be used in describing a member of a class, what classes of objects these predicates can relate to and the cardinality of each predicate. The Pellet inference engine can determine if the semantic description of an individual violates the restrictions on its class. Once we are confident of the consistency of a description we can use it in turn to confirm that the behavior of some service matches its description, by testing it in a network of compatible services. Thus, while our semantic descriptions and the actual java code are duplicate representations of the agent's behavior, we enforce an order of precedence to resolve conflicts.

Secondly, our agent's descriptions can be queried and modified by users so they may know what information is moving between services and to inject changes as they see fit. These descriptions, being composed of human-understandable terms such as "price" or "product," are provided to the interface to decorate displays. For instance, from the semantic description of the simple-price evaluator the interface can determine that a graph of its output should have Product ID and Price as its axis labels. The semantics can also inform what type of display is appropriate. If we know that some source service provides English descriptions of each product it makes no sense to display this information as a pie chart. Services may be included simply to generate information for the user. For instance, we might insert a service that samples the output of another service each day to generate a time series.

Our model can also determine if a data dependency between two services is valid. Such validation is necessary when processing a configuration before the start of game or as the result of a user's change to the service network. We do so by comparing data types, dimensional cardinality and indexer types of a given input and output. We see in Figure 3 that these descriptions need not be exactly the same to form a valid dependency, Pellet must only show they are compatible. For instance, Pellet could confirm two differing data types are compatible by subsumption. This data-flow connection validation is done via rules written in the Semantic Web Rule Language (SWRL)[4]. Each rule can confirm, if queried, one part of a valid connection. For example, one rule states that if the input of some evaluator has a different dimensional cardinality then the output of another, we must conclude that a dependency between these services is invalid. When a valid dependency is established, rules also allow Pellet to conclude that the input involved has the output service's possibly more specific data type. The simple-price evaluator, for instance, will take any one dimensional array of numbers for its quantity input, but when we assign the output of the customer-quantity source to this input, Pellet will conclude that this input is an array of quantities representing customer demand. A proactive application of this

[2]http://www.w3.org/2004/OWL/

[3]http://clarkparsia.com/pellet/

[4]http://www.w3.org/Submission/SWRL/

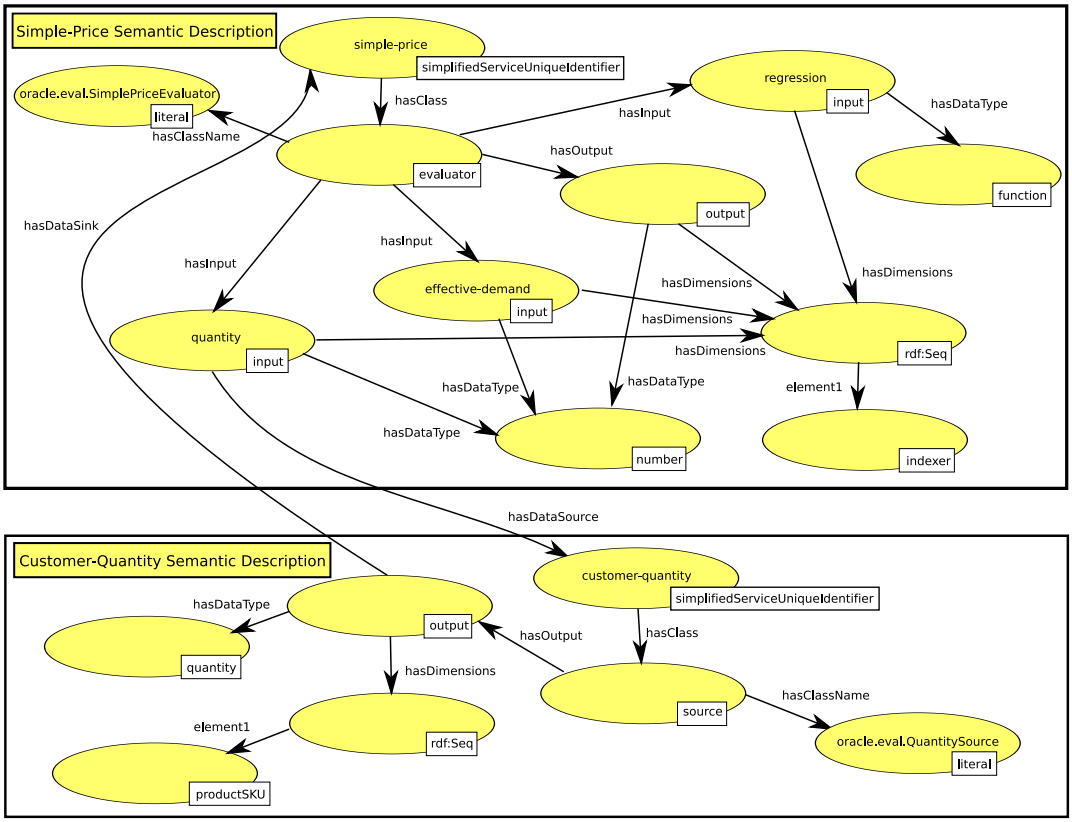

Figure 3: Representation of the semantic descriptions of two dataflow services, the data dependency between them, and their memberships in the ontological classes.

validation procedure can be used to suggest data-flow connections to the interface. If we combine proactive data-flow connection validation and the transfer of semantic information from outputs to inputs, we approach automatic composition. When the user requests some novel piece of information not currently available, a search of valid possible data-flow connections could result in an evaluator or chain of evaluators which, if added to the network, would produce the desired information.

While our model provides details of an agent's execution and an ability to precisely modify it, some issues remain unaddressed. Many of our services maintain internal information and therefore cannot be added or modified mid-game. Information identifying such services is not currently within the semantic model and would need to be included to restrict the interface in these cases. Also, our model does not handle anytime algorithms elegantly. Our model would be well-served by a scheduling problem solver with knowledge of each service's required and ideal computating times that could allocate processing time to each service efficently. By allocating processing time we could provide users with a timer to when their changes must be made for any given portion of the network.

Human Agent Interaction

In this section we discuss architectural elements that allow for Human Agent Interaction (HAI) within the MinneTAC agent. To support HAI, we add the HAIController that provides an interface between the agent and the dashboard. Figure 5 shows the architecture that supports HAI.

HAI Controller The HAIController component of the MinneTAC agent has four responsibilities:

Relay data from sources and evaluators to the dashboard: Once data is available from a source or an evaluator, the HAIController obtains its semantic description, decorates it and relays it to the dashboard for display.

Save data modification to sinks : Once a dashboard user submits data to the agent, the HAIController stores the users data into its appropriate sink for use by the evaluator.

Present agent configuration details : In the Ontology is information on the agent configuration. This information is made available for visualization by the HAIController through the HAIInterface.

Persist agent configuration details : Through the Dashboard users can made changes to the agent configuration.

Changes to Repository data will only occur during runtime for data that has been identified as modifiable. Once modifi-

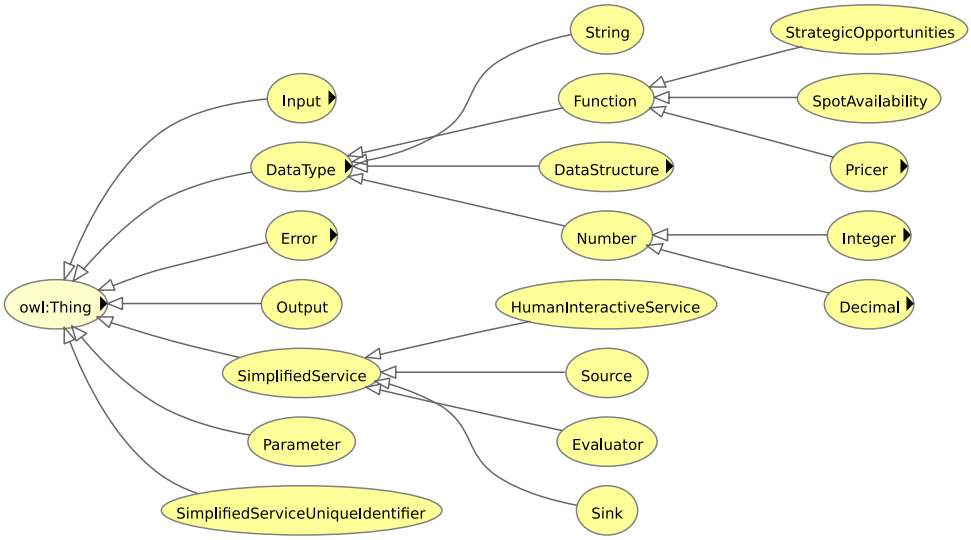

Figure 4: Top-level ontology for describing the MinneTAC agent.

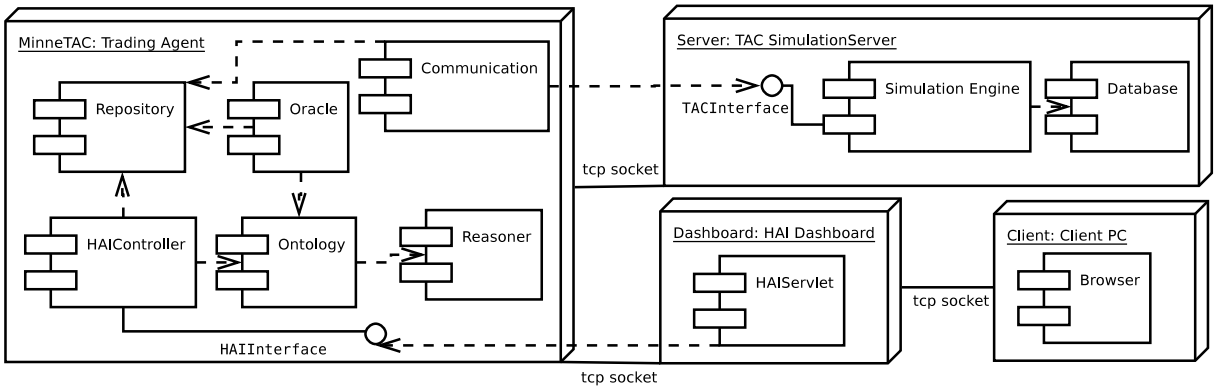

Figure 5: MinneTAC architecture with Human Agent Interaction.

able data is made available in the Repository, the HAIController is notified. Modifiable data can only be used after an approval has been received from the a user, however if a user does not provide an approval within a specified time interval, an automatic approval is provided by the controller.

Dashboard Human agent interaction occurs through the dashboard. Our dashboard can be configured to support a variety of dashboards for different purposes and stakeholders. The main component of the dashboard is the **HAIServlet** which provides communication services between the dashboard and the agent. Through the servlet data is transmitted back and forth between the browser and the agent. Below we describe three examples of dashboards that can be composed and provide details on their visuals and allowed user actions. A **strategic** dashboard can be configured to track agent performance against strategic objectives. Strategic objective are stored as configurations in the Ontology. See figure 6. Visualizations on the strategic dashboard include: (1) Market summary reports, (2) Evaluator scorecard, (3) Market projection reports, (4) Economic regime prediction. We describe the "economic regime" model in (Ketter *et al.* 2007).

Modifications on the strategic dashboard are executed in terms of configuration changes. These changes include: (1) Modifying an evaluator name, (2) Modifying an evaluator dependencies, (3) Determining whether an evaluator human interaction.

A **tactical** dashboard can be configured to track evaluators that need approvals from a user; configuration of this option is provided on the strategic dashboard. Through this dashboard, users are able to make changes and approve information that is available on a given evaluator or source. Modified evaluator and source data is stored into a sink. While in the tactical view active alerts are sent to users to help them better control the agent. See Figure 7. Visualizations on the tactical dashboard include, but are not limited to: (1) Daily factory allocation and usage, (2) Daily simple price values, (3) Daily predicted price densities, (4) Daily recommended

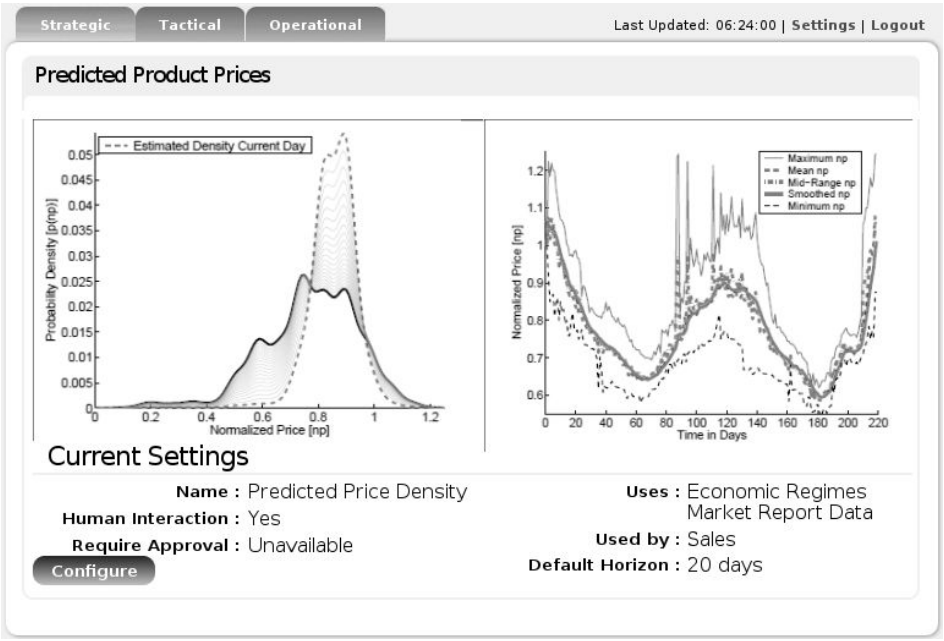

Figure 6: Strategic dashboard showing output and editable configuration parameters for the Predicted Product Prices evaluator.

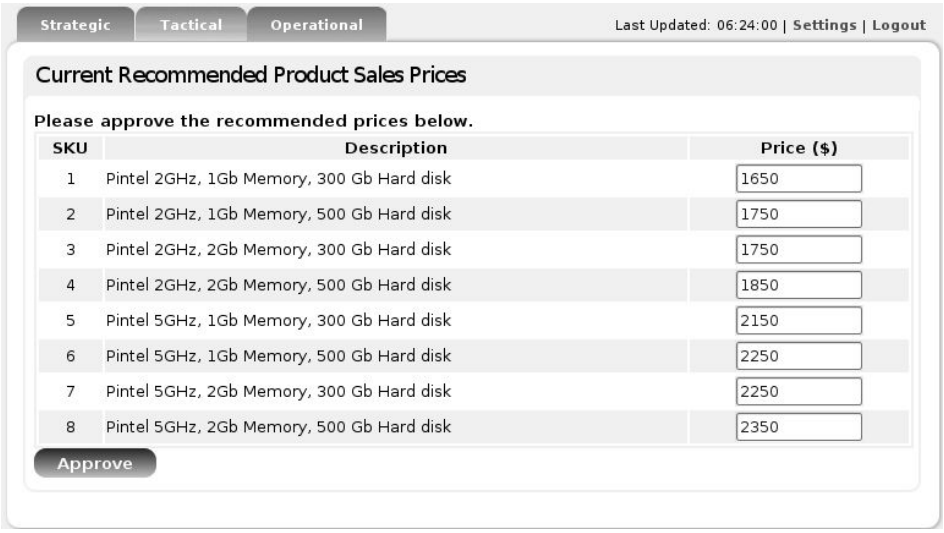

Figure 7: Tactical dashboard displaying current recommended product sales prices from a source that need to be approved.

price values, (5) Daily procurement summary, (6) Low procurement alerts.

Changes made in the tactical dashboard generate sink data that is used by other evaluators in the agent. Users are required to provide approvals within a specified time period from the time the data is made available to them. If an approval is not provided within the allotted time, the agent assumes an automatic approval. This helps prevent the user from holding up agent transactions. Changes in this dashboard include but are not limited to: (1) Modifying Recommended Prices, (2) Modifying the planning horizon for predicting the price density, (3) Controlling factory usage, (4) Making procurement orders and updates.

An **operational** dashboard can be configured to allow users track the daily outputs of evaluators. In this dashboard modification and approval options are not available. Unlike the tactical dashboard, this dashboard makes available data from all evaluators. See Figure 8. Visualizations in this dashboard include but are not limited to: (1) Time series graphs for evaluator outputs, (2) Table displaying maximum, mean mid-range, smoothed and minimum price of good sold in a given day.

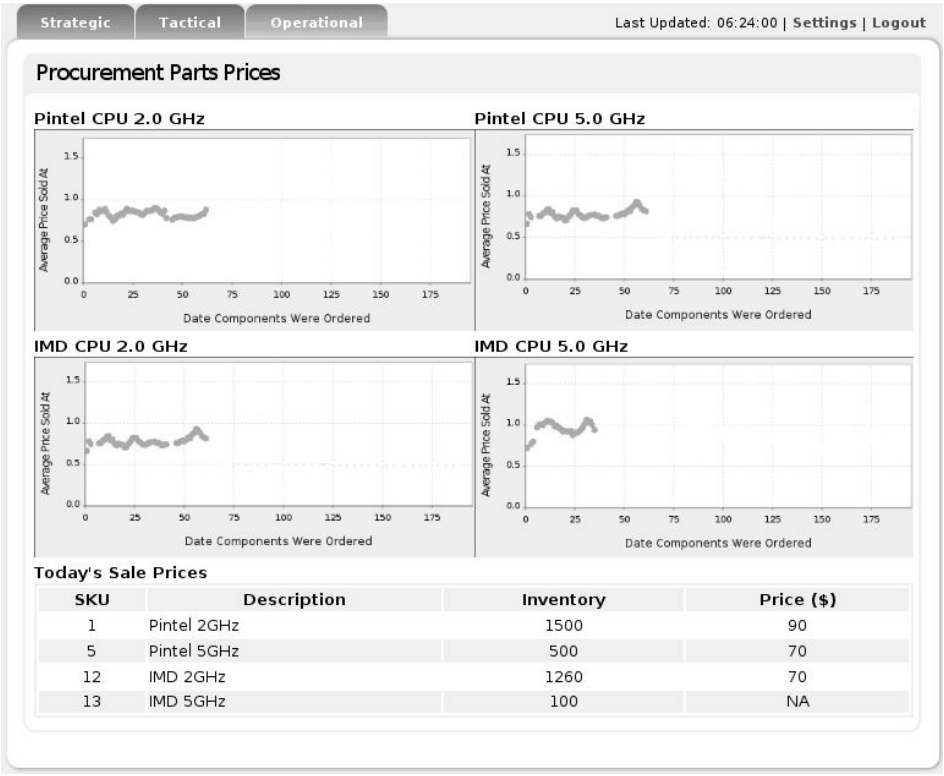

Figure 8: Operational dashboard displaying time series of procurement parts prices from a source with a table of today's prices.

Client Users access the dashboard through a client machine. The dashboard is accessible through a web browser interface. The **Browser** enables a user to view and interact with the dashboard visualizations. The presentation of data relies on JavaScript to dynamically update the visualizations with real-time data. Modifications made to the data are sent back to the dashboard via a POST request through the TCP socket between the client and the dashboard. One dashboard can support multiple clients at once.

Conclusion and Future Work

While TAC SCM is currently concentrating on the autonomous agents, business school games are focusing only on human interaction. We envision an agent-assisted human decision support paradigm that extends and bridges both approaches, so that humans may work in concert with agents to solve complex real-world problems. In the following we give an overview of the pressing research challenges which we are and will be working on.

Ontology-driven composition Given a set of services that are described in terms of our ontology, how do we compose them to produce useful results?

Service discovery How do we find services? This is probably not a UDDI problem. Do we need some sort of directory service? How simple could it be? How does it hook into the ontology?

Test support Semantic descriptions can presumably be used to generate test cases. Can we prove it with code?

Information visualization How do we best present complex decision attributes, and so tackle information rich environment?

Adjustable autonomy Determining if and when transfers of human control to the agent should occur.

We are currently working to create a web service wrapper around the evaluators, and integrate it in a real business network, such as the Dutch Flower auction (Kambil & van Heck 1998). Finally, we are planning to use our decision support system in teaching students about dynamic supply-chain management.

References

Azvine, B.; Cui, Z.; Nauck, D.; and Majeed, B. 2006. Real time business intelligence for the adaptive enterprise. In *IEEE Computer Society (Hrsg.): Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06), IEEE Computer Society, Los Alamitos*, 29–39.

Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2005. The supply chain management game for the 2006 trading agent competition. Technical Report CMU-ISRI-05-132, Carnegie Mellon University, Pittsburgh, PA.

Collins, J.; Ketter, W.; and Gini, M. 2002. A multi-agent negotiation testbed for contracting tasks with temporal and

precedence constraints. *Int'l Journal of Electronic Commerce* 7(1):35–57.

Collins, J.; Ketter, W.; and Gini, M. 2008. Flexible decision support in a dynamic business network. In Verwest, P.; van Liere, D.; and Zheng, L., eds., *The Network Experience – New Value from Smart Business Networks*. Springer Verlag. 233–246.

Collins, J.; Ketter, W.; and Gini, M. 2009. Flexible decision control in an autonomous trading agent. *Electronic Research Commerce and Applications* in publication.

Eckerson, W. W. 2005. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. Wiley.

Gregg, D., and Walczak, S. 2006. Auction advisor: an agent-based online-auction decision support system. *Decision Support Systems* 41(2):449–471.

Hevner, A.; March, S.; Park, J.; and Ram, S. 2004. Design science in information systems research. *Management Information Systems Quarterly* 28(1):75–106.

Kambil, A., and van Heck, E. 1998. Reengineering the dutch flower auctions: A framework for analyzing exchange organizations. *Information Systems Research* 9(1):1–19.

Ketter, W.; Collins, J.; Gini, M.; Gupta, A.; and Schrater, P. 2007. A predictive empirical model for pricing and resource allocation decisions. In *Proc. of 9th Int'l Conf. on Electronic Commerce*, 449–458.

Ketter, W.; Batchu, A.; Berosik, G.; and McCreary, D. 2008. A Semantic Web Architecture for Advocate Agents to Determine Preferences and Facilitate Decision Making. In *Proc. of 10th Int'l Conf. on Electronic Commerce*, 1–10.

Maes, P. 1994. Agents that reduce work and information overload. *Communications of the ACM* 37(7):30–40.

Montgomery, A.; Hosanagar, K.; Krishnan, R.; and Clay, K. 2004. Designing a better shopbot. *Management Science* 189–206.

Myers, K.; Berry, P.; Blythe, J.; Conley, K.; Gervasio, M.; McGuinness, D.; Morley, D.; Pfeffer, A.; Pollack, M.; and Tambe, M. 2007. An intelligent personal assistant for task and time management. *AI MAGAZINE* 28(2):47.

Ontrup, J.; Ritter, H.; Scholz, S.; and Wagner, R. 2009. Detecting, assessing and monitoring relevant topics in virtual information environments. *IEEE Transactions on Knowledge and Data Engineering* 21(3):415–427.

Orlikowski, W., and Iacono, C. 2001. Research commentary: desperately seeking the " IT" in IT research-A call to theorizing the IT artifact. *Information Systems Research* 12(2):121–134.

Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2005. Cabob: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science* 51(3):374–390.

Stone, P., and Greenwald, A. 2005. The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research* 5(1):229–64.

Wurman, P.; Wellman, M.; and Walsh, W. 1998. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the second international conference on Autonomous agents*, 301–308. ACM New York, NY, USA.