Design of supplier agents for an auction-based market*

Stefan Botman, Mark Hoogendoorn,
Vasile Bud, Ashutosh Jaiswal, Steve Hawkins,
Yelena Kryzhnyaya, Janice Pearce, Anne Schoolcraft, Espen Sigvartsen,
John Collins, and Maria Gini

University of Minnesota

Abstract. We are interested in supporting multi-agent contracting in which customer agents solicit the resources and capabilities of other self-interested supplier agents in order to accomplish their goals. Goals may involve the execution of multi-step tasks in which different tasks are contracted out to different suppliers.
In this paper we focus on the design of supplier agents. The agents are designed to operate in the context of the MAGNET (Multi AGent NEgotiation Testbed) system, but the design could easily be adapted to other situations in which agents interact through a market infrastructure.
MAGNET supplier agents can register their capabilities with the market, be notified of open and relevant requests for quotations, and submit bids that specify which tasks they are able to undertake, when they are available to perform those tasks, and at what price. Supplier agents attempt to maximize the value of their resources.
The paper describes the detailed design of supplier agents and presents preliminary experimental results.

1 Introduction

Online marketplaces offer benefits to both buyers and sellers. For buyers, a marketplace can significantly ease the process of searching for and comparing providers, while for sellers, marketplaces provide access to much broader customer bases. The major challenge in extending currently available online marketplaces comes from the necessity to go beyond simple buying and selling. A realistic system needs to incorporate time constraints, to enforce deadlines, to interact with a highly distributed web of suppliers with different capabilities and resources, to interact over long periods of time through the completion of the contracted work, and to deal with failures in contract execution.

The proliferation of business-to-business portals such as CommerceOne (`www.commerceone.com`) and VerticalNet (`www.verticalnet.com`) shows the need and industry demand for value-added services such as security, matchmaking, and trusted intermediaries. A framework which can successfully address

* Work supported in part by the National Science Foundation, awards NSF/IIS-0084202 and NSF/EIA-9986042

the full spectrum of the requirements mentioned above needs to provide support for contracting activities among participants, as well as provide support for automated agents that act on behalf of human participants.

In order to model these features the MAGNET (Multi-Agent Negotiation Testbed) system has been designed at the University of Minnesota [Collins *et al.*, 1998]).

2 The MAGNET system

The MAGNET architecture provides a framework for secure and reliable commerce among self-interested agents. What makes MAGNET unique is its ability to support negotiation of contracts for tasks that have temporal and precedence constraints [Collins *et al.*, 2002]. MAGNET shifts much of the burden of market exploration, auction handling, and preliminary decision analysis from human decision-makers to a network of heterogeneous agents.

2.1 A Motivating Example

For example, imagine that we need to construct a house. Figure 1 shows the tasks needed to complete the construction. The tasks are represented in a *task network* where links indicate precedence constraints. The first decision we must make is how to sequence the tasks in the *Request for Quotes* (RFQ) and how much time to allocate to each of them. For instance, we could reduce the number of parallel tasks, allocate more time to tasks with higher variability in duration or to tasks for which there is a shortage of laborers, or to allow more slack time. We assume that suppliers are more likely to bid, and to submit lower-cost bids, if given greater flexibility in scheduling resources [Babanov *et al.*, 2002], so time windows in the RFQ might overlap. A sample RFQ is shown in Figure 1. Note that the time windows in the RFQ do not need to obey the precedence constraints; the only requirement is that the accepted bids obey them.

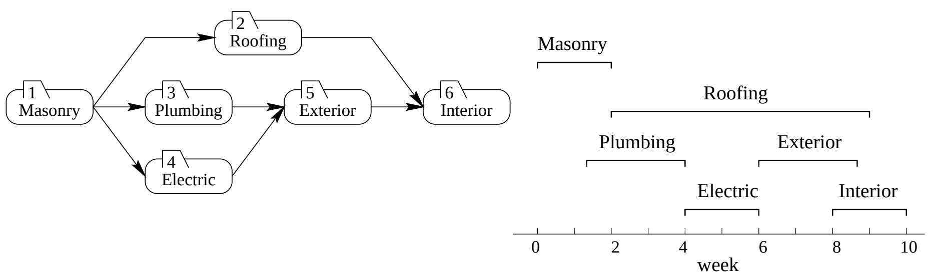

Fig. 1. A task network example and a corresponding RFQ.

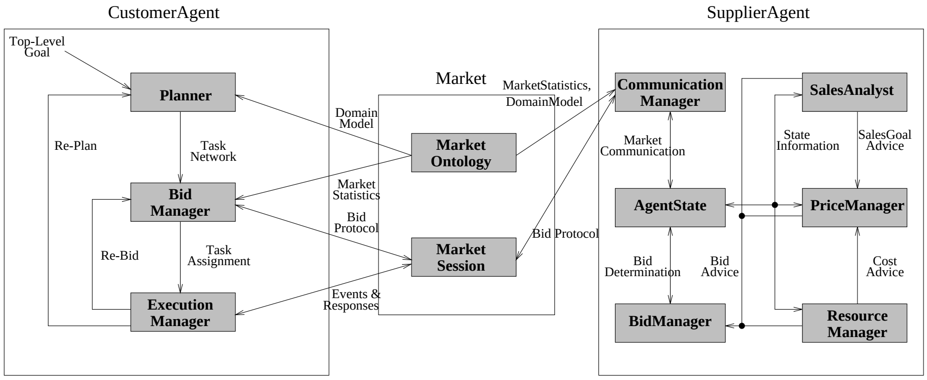

Fig. 2. The MAGNET architecture.

2.2 The MAGNET Architecture

The architecture of MAGNET is illustrated in Figure 2. The MAGNET system consists of *supplier* agents, *customer* agents, and a *market*. The supplier agent has resources to offer, while the customer agent has resource and/or service needs.

This is a schematic outline of the main interactions among agents.

- A customer agent issues a *Request for Quotes* (RFQ) which specifies tasks, their precedence relations, and a time line for the bidding process. For each task, a time window is specified giving the earliest time the task can start and the latest time the task can end.
- Supplier agents submit bids. A bid includes a set of tasks, a price, a portion of the price to be paid as a non-refundable deposit, and estimated duration and time window data that reflect supplier resource availability and constrain the customer's scheduling process.
- The customer agent decides which bids to accept. Each task needs to be mapped to exactly one bid (i.e. no free disposal [Nisan, 1999]), and the constraints of all awarded bids must be satisfied in the final work schedule. In MAGNET the customer can chose from a collection of winner-determination algorithms (A*, IDA*, simulated annealing, and integer programming).
- The customer agent awards bids and specifies the work schedule.

3 Architectural Design Principles

We now briefly outline the design principles behind MAGNET. We have adopted several design principles that make it easy to plug components together and to reconfigure the system. Examples are:

1. The system is written in Java and has been tested on multiple platforms. This makes it easy to adopt on whatever platform you happen to be using.

2. The system is organized into a set of components, and a set of systems that can be constructed from various subsets of components. Each system is constructed to serve a particular experimental purpose.
3. All the major behavioral modules are written as abstract classes, with (potentially) multiple implementations that can be "plugged in" to implement a particular behavioral variant.
4. Virtually every feature of the system is selectable and configurable from a configuration file, and many of these can be viewed and changed from a user interface. This includes the choice of behavioral plug-ins.
5. The interface between the agents and the Market is also abstracted. This allows connection with multiple types of markets (such as one that looks up price and availability info from a catalog or timetable) and through multiple communications protocols.
6. Much of the activity of the agent is agenda-driven, and development and maintenance of the agenda is an important activity in its own right. Agenda items can select plug-ins, update configuration details, evaluate options, interact with the market or other agents, update the agenda, and record results.
7. A pervasive logging and data collection system allows for both detailed examination of behavior and the generation of experimental data. The level of logging detail may be independently configured for different modules, and the various logging levels have well-defined meanings.

The MAGNET market exists as an EJB and interacts with customer and supplier agents through the use of the SOAP services described earlier. Market sessions exist also as EJBs and are created and accessed through the us of market-related SOAP services. Session persistence is addressed through the use of entity beans, which are persistent as needed through the use of an application server's database. Session-related EJBs are used by the RFQ, bid submission, and bid award SOAP services.

3.1 Security

The current MAGNET implementation does not have robust security, except for the rudimentary security provided by Java's sandbox model. The security needs for the system can be classified into three main components: (1) authentication, (2) authorization, and (3) non-repudiation.

Authentication is the process of identifying an entity, based on the information provided by it and verified by the system. In MAGNET, agents join and leave as needed to carry out transactions with other agents. The agents need to be authenticated before they join the system. We plan to carry out this process through the market. Absence of a secure authentication mechanism leaves the potential for rogue agents sneaking into the system.

An agent operates on behalf of its principal, which may be a person, a corporation or some other physical entity. Authenticating an agent would mean authenticating the principal indirectly. Public Key Infrastructure (PKI) and Kerberos are two acknowledged methods used for authentication. The first involves using digital certificates from certificate authorities, which are presented by an entity to the authenticating system each time its identity needs to be established. Kerberos involves using unique keys, called tickets, to exchange secure messages between two entities on an open network. The decision on which system to use requires analyzing the computational power needed to perform encryption of the channel.

Authorization is the process of granting or denying access to system objects to an entity based on its identity. This step is usually preceded by authentication. In the MAGNET system, agents utilize market resources to exchange messages with other agents, place and receive bids etc. In order to use the market resources, the agents need to have proper authorization. This can be done based on the origin of the code (signer) or the user executing the agent code.

Non-repudiation is a property achieved through cryptographic methods which prevents an entity from denying having performed a particular action related to a set of data [OECD, 1997]. What this means is that an agent would not be able to deny an operation after committing it. This is a desirable property to maintain trust in the system. If the first two components are executed properly, non-repudiation would just require the use of logs for these transactions.

3.2 Recovery of Agent State

To increase reliability, MAGNET agents periodically save their state in a database, which allows for recovery in case of a crash.

The database we selected is JDBM [1] (Java Data Base Manager). JDBM offers persistent storage, and is a relatively fast and simple database engine. All updates are transactionally safe. JDBM offers scalable data structures, such as Hash trees and B+ trees. All operations in the database have ACID properties, and the database uses a transaction log and can perform a recovery in case of a crash.

The JDBM database engine can be configured in different ways. For example, by choosing to synchronize the transaction log less frequently, the database performs better. The trade-off is that a recovery will be more time consuming. This feature is interesting when trying to keep the overhead as low as possible, as it is the case in MAGNET. In our current implementation, the agent state is saved between the time- and resource-consuming processes, such as the search algorithms for winner-determination.

4 Designing the Supplier Agent

Now the design of the supplier agent is discussed. The agent-oriented design and modeling methods used are discussed. The supplier Agent is designed to

[1] downloadable from `http://www.sourceforge.net`

make the MAGNET system able to handle fully automated business-to-business interaction.

4.1 Method for high-level design

The method we chose for high-level design is Desire, by Brazier, Jonker and Treur [Brazier *et al.*, 2000]. This approach makes it easy to specify all the information types and components. The actual framework consists of more than just a design method; it includes software tools to support system design, a formal specification language, an implementation generator to automatically translate specifications into code, and verification tools for static properties of components such as consistency, correctness, completeness. We used Desire only to specify the general structure of the supplier agent, and we chose Avalon for the implementation (described later in Section 4.2), since Desire does not support Java. More details on using Desire and other tools as an agent design tool can be found in [Shehory and Sturm, 2001].

Desire specifications are based on an architecture made of components with a hierarchical relationship between them. Each component has its own input and output information types and uses its own task control knowledge, so the system is structured in a decentralized manner. The structure within the components is hidden from the "outside world"—only the interface types are defined. A component can consist of multiple sub-components if the task the component performs is too complex for one component to manage.

4.2 Method for component design

We have chosen Avalon [Loritsch, 2001] to implement the supplier agent. Using Avalon, it is straightforward to have the components of the supplier agent interact, to instantiate different instances of the components, and to reuse code. Avalon allows us to switch components on the fly, which is very useful in testing. It is also possible to configure Avalon using XML files, which specify which components and which instances have to be included.

There are seven major interfaces that can be extended in order to fit a component inside the Avalon framework: *Activity*, *Component*, *Configuration*, *Context*, *Logger*, *Parameters*, *Thread* and *Miscellany*. Each of those categories represents a unique concern area. Each component should at least implement one of these interfaces, but can also implement several of them.

The lifecycle of a component is split into three phases: *Initialization*, *Active Service* and *Destruction*. These phases occur in sequential order.

The general structure of the Avalon architecture contains the following elements: *Components*, *Component Manager*, *Component Selector* and *Component Container*. The *Components* are the cornerstones of the Avalon framework and model components as used in design methods. The *Component Manager* provides a framework for allowing components to access each other. The *Component Selector* is responsible for managing the instances of the *Components*. In order to retrieve *Components* it needs a specification of the role and a ''hint" to tell it which version to use. Finally, the *Component Container* contains the *Components* it is responsible for.

In Figure 3 we show the proposed architecture of the supplier agent using Avalon.

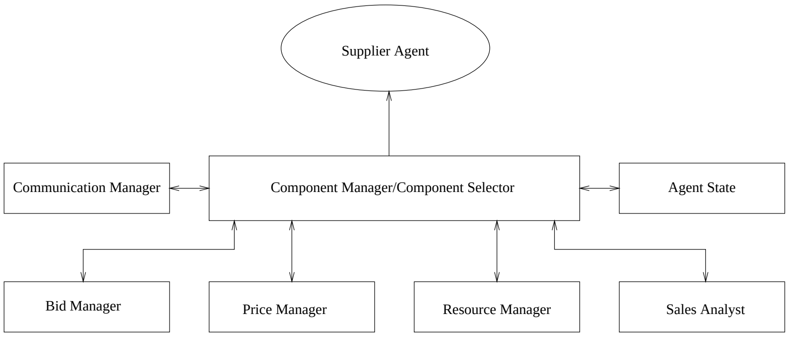

Fig. 3. Proposed architecture using Avalon

The SupplierAgent[2] is modeled as a *Component Container*. The SupplierAgent has no other function but to control the lifecycle of all the components. All six components are controlled by the ComponentManager. They all get a reference to the ComponentManager in order to be able to access the other components. If there are multiple instances of a component, the ComponentSelector can be retrieved from the ComponentManager and with this the right instance can be selected.

4.3 Specifying the components

We are now ready to specify the component architecture and the interfaces of the components.

Defining the component architecture The SupplierAgent is responsible for the components managed by the ComponentManager, including CommunicationManager, AgentState, BidManager, SalesAnalyst, PriceManager and ResourceManager

The task of the ComponentManager is to manage the components owned by the SupplierAgent. It can provide its components with references to other components it manages.

The AgentState is responsible for maintaining the state of the agent and the market. This consists of keeping track of time and events coming from the

[2] A note to clarify the text formatting: we use *italics* to denote Avalon interfaces and AllCaps to refer to implemented components.

market. It also includes the saving of all information that needs to be maintained, like bids that have been submitted and incoming RFQs. Furthermore, components can subscribe to certain types of events and be notified by the AgentState when they occur.

The task of the CommunicationManager is to receive information from and communicate information to the Market. This data includes RFQs, bids, bid awards and penalty payments.

The BidManager must decide whether to submit single or multiple bids, and whether to bid on individual tasks or block of tasks. Furthermore, it decides if the bid is to be submitted early in the bidding cycle or to wait until the last minute, and whether to bid on the entire available time windows or on some subset.

The task of the SalesAnalyst is to give other components advice on whether to engage in negotiation with the customer. It keeps track of its history and relationship with various customer agents in the market. It also keeps track of sales goals for the business represented by the supplier agent. Another duty is giving advice on which task(s) to make into milestones in the bids. Milestones are tasks that require a notification be sent to the customer agent upon completion of the task.

The ResourceManager is responsible for determining, given the current resources and the tasks, upon which tasks to bid. It also suggests upon which time windows to bid, and it is responsible for the management of the resources. This includes executing awarded tasks and handling the payments for these tasks.

The job of the PriceManager is to determine the price to bid. It can determine this price using the sales goals for these tasks, which it retrieves from the SalesAnalyst. In order to determine the price, it also needs to know the cost of the resources for these tasks. This information is retrieved from the ResourceManager.

4.4 Design using UML

The mapping between the higher-level design discussed previously and the more detailed design as presented here is completed as follows: the components become classes and the flow of information from one component to another component is accomplished through the parameters of a method call. Similarly, the information types have been modeled as classes and the information included in the information types can be accessed through method calls. When the flow of information is synchronous, the method call returns the result. When the communication is asynchronous, the information will be sent through a method call in the requesting component. A combination of both is also possible. For example, when you subscribe to a particular event you will get a confirmation through a synchronous return value. However, the event notification will be sent asynchronously.

Assume that the RFQ has already arrived at the CommunicationManager and that the BidManager has already received a reference to the CommunicationManager and the AgentState. If aggregate transactional statistics are desired, they will be retrieved from the market.

Next, the SalesAnalyst will be asked to give the BidManager advice concerning upon which tasks to focus, given its history with the customer. For this, the BidManager will have to retrieve the reference to the SalesAnalyst from the ComponentManager and, when multiple instances exist, the ComponentSelector is also used. After the reference has been received, the BidManager will pass the RFQ to the SalesAnalyst.

In our house-building example, the SalesAnalyst knows that the customer that issued the RFQ has issued a RFQ before concerning the "Interior" task, and did not pay in time. The SalesAnalyst decides not to include this task but to include the rest of the tasks in the advice. Another issue is to decide which milestones to include in the bids. The SalesAnalyst knows that the customer finds the masonry very important, so it advises that this task be flagged as a milestone.

The reference to the ResourceManager is retrieved in the same way as with the SalesAnalyst. The BidManager will pass the task plan through to the ResourceManager. After this has been done, the ResourceManager checks the availability of the resources needed to complete the five tasks that are left. The ResourceManager concludes that it is unable to provide the necessary resources for the Roofing task during the specified period. Therefore, it advises the BidManager to bid only on the other four tasks. The ResourceManager decides to recommend the same time windows for the remaining tasks as specified in the RFQ. The resulting task plan is retrieved and will be used to determine the number of bids. In our case, the BidManager decides to issue only one bid.

Now that the bid has almost been constructed, the price needs to be calculated. For this the PriceManager is asked for advice. The reference is retrieved in the same way as described above and the task plan and the customer agent's id are passed to the PriceManager. The PriceManager may ask for the sales goal for this customer from the SalesAnalyst and the ResourceManager will be consulted for the costs of the resources included in the task plan. The references are retrieved in the same way as with the BidManager. After the PriceManager has received the aforementioned information, it will determine the price and return it. Now the BidManager is able to make the collection of bids (one, in our case) and stores them until the time comes to issue them.

5 Current implementation of Supplier Agents

5.1 ResourceManager

Motivation. The consumption of resources by a task is analogous to the fulfillment of tasks in a task plan. The consumption of resources is considered to be a lower level of abstraction. It is therefore a necessary underpinning that allows for the modeling of complex task fulfillment. The motivation for the ResourceManager is to provide an abstraction for other supplier components to keep them from worrying about the details of production and consumption.

There is more to managing resources than simply modeling the consumption of consumable resources. Fixed resources, such as machines in a manufacturing cell, have a cost even when not in use. Because their use needs to be scheduled, a central part of the job of the ResourceManager is to maintain their schedule. An additional future goal is to learn cost-minimization strategies to contribute to the SupplierAgent's profit-maximization goal and to work on methods for dynamic allocation of tasks to different resources within the supplier agent (as, for instance, in [Fatima and Wooldridge, 2001]).

Design. The design of the ResourceManager encompasses the design of a resource production and consumption model and the design of the component.

The resource model makes two main simplifications in creating a lower level of abstraction. First, instead of considering time windows, the production and consumption of resources are quantized to discrete points in time. Second, a generic interface is presented that does not consider the explicit modeling of exotic resource attributes. Details such as price elasticity, storage, and purchasing are considered below this interface, but can still be modeled. This allows resources to simply be thought of as something the supplier agent has in time, not something the agent acquires in time.

The ResourceManager may control many Resources. Each Resource acts as a wrapper to a hash table of quantized production times and corresponding amounts of reserved resources. Using this hash table, given an implementation of the productionLevel method and a confidence level, getAvailable will return the likely amount of resources available.

For each SubtaskRequest in a task plan, a ResourceRequirement specifies the type of resource it requires and acts as an iterator to a set of evaluation times. At these evaluation times, the ResourceRequirement can be queried for the maximum and minimum requirements. We assume that after setting a start time, the set of evaluation times and resource requirements are deterministic.

In our current implementation, the design of the ResourceManager is straightforward. From the interfaces mentioned above, the ResourceManager steps through the set of consumption times using a naive strategy for resource allocation to determine satisfiability for the getTasks and getTimeWindows methods. Once a bid is made, the ResourceManager is responsible for speculatively reserving some amount of resources. Then, once a bid award event has been received from AgentState, the ResourceManager is responsible for the execution of those tasks. Presently, this amounts to tracking whether its resources meet the resource requirements and logging the successful or unsuccessful result.

5.2 PriceManager

Motivation. We believe that a smart PriceManager will help provide MAGNET suppliers with a competitive advantage and an incentive to join the marketplace. The lure for customers is clear: they can choose the "best deal" from those submitted. Suppliers might be hesitant to join such a market, for fear that prices will be driven too low. Our hope is that this price management technology will assure suppliers a profitable position in the market.

Design. The algorithm used by the PriceManager in this implementation endeavors to search through the space of prices for each task type and find the optimal price. That is, it looks for the price which maximizes profit as often as possible. We have developed two methods for conducting this search. One is a derivative-following (DF) technique, based on the work of Kephart [Kephart *et al.*, 2000]. The other is based on simulated annealing (SA)[Reeves, 1993].

For both methods, we track a markup percentage, so that prices reflect cost. Markups are tracked for each of the n task types the agent is interested in performing. Certainly, price tracking for sequences of tasks would provide a nice level of accuracy. However, the number of combinations would tend toward infinity as the size of the task plans increased. With long enough market exposure, the prices for individual tasks should be sufficiently shaped by their environment, so single-task schedules should provide a reasonable model.

The DF algorithm is initialized with a markup $M_i, 1 \leq i \leq n$ of 50% for each of the n task types, a maximum step size D of 50% and a price-movement direction δ of 1. When δ equals 1, the algorithm intends to raise the markup, if δ equals -1, it means to lower the percentage.

The PriceManager is called by the BidManager with a list of tasks the agent has decided to bid on (usually based on the ResourceManager's recommendations). A price for each task type is determined as follows. A step size is randomly chosen between zero and D. This value is multiplied by δ and added to the last bid's markup value. Next, the cost C_i of the task in question is retrieved from the ResourceManager. The price P_i for the task is stored as $M_i * C_i + C_i$. These individual P_i's are totaled for all the tasks in the bid and returned to the BidManager as the bid's price, P^t.

Once the time for awarding bids arrives, the DF PriceManager is notified by AgentState. The PriceManager takes stock of its performance and makes adjustments for the next round. A set of values $\pi_i^c, 1 \leq i \leq n$ represents the profit gained on the previous transaction. If the bid was awarded, the PriceManager calculates its profit, π_i^{new} and checks to see if $\pi_i^{new} > pi_i^c$. If so, δ remains unchanged. However, if profit decreased or the bid was rejected, δ is switched to its opposite. π_i^c is set to the value of π_i^{new} and stored for the next bid's comparison.

The SA PriceManager works in a somewhat similar way. Each task type's annealing schedule is initialized with a "current" markup $M_i^c, 1 \leq i \leq n$ of 50%. The PriceManager calls the Resource Manager and obtains from it the cost C_i of performing each task. Next a markup M_i^b is chosen for each task type. This is determined by taking a random step, less than an ever-shrinking distance D, away from the current price. (D is initialized to 0.5.) The price P_i for the task is stored as $M_i * C_i + C_i$. The prices for all the tasks are summed to give a total price, P^t, which is returned to the BidManager.

The SA PriceManager is notified by AgentState if has received an award for the work it bid on or not. The PriceManager now calculates its profit on this transaction, if any, and compares it to past performance. A set of values $\pi_i^c, 1 \leq i \leq n$ represents the best profit so far[3] for each of the n types. For each task type i, M_i^b and its cost are retrieved, and the profit π_i^{new} is calculated if the bid was awarded. For rejected bids, $\pi_i^{new} = 0$. We next find the value $E = \pi_i^{new} - \pi_i^c$. If E is positive, we have made an "uphill" step, an improvement in profit. Therefore, we accept the change and P_i^c is set to the value of P_i^b. If $E < 0$, it was a "downhill" step, so we will only take it with probability $e^{E/T}$. This allows us to avoid local minima and also filters out "fluke" high profits. T is the countdown timer, so smaller and smaller steps are taken as we zero in on the profit-maximizing price.

Research is currently underway to determine the market conditions where each of these performs best.

5.3 SalesAnalyst

Motivation. The SalesAnalyst can analyze customer agents and derive the most effective advising strategy, given the current customer agent. With this strategy, the profitability of the supplier agent will grow.
Design. We intend to include several features within the SalesAnalyst. First of all, we would like to use data mining techniques to derive information about profiles of the customers. The information on which the data mining tool is used should be maintained within the SalesAnalyst itself. This is because the information should include a history of payments and bid awards; this is considered to be private information. The profile can be used for several advising tasks: It can be used to decide on what sales goal to pass to the PriceManager. If the SalesAnalyst decides that this customer should become a regular customer, it can pass a sales goal to bid a lower price. The SalesAnalyst is also useful for giving advice on when not to bid, such as when the customer in question fits a "bad credit" profile. Finally, the profile of the customer can be used to decide which milestones to include in the bid.

5.4 Preliminary Experimental Results

In order to show that our design works, we ran our system using for the RFQ the example of building a house illustrated earlier in Figure 1.
Stage I The RFQ from Figure 1 was sent to each of the three suppliers.
Stage II Each supplier's ResourceManager chose the tasks and time windows of the bid. Since, in this case, all the suppliers bid on all the tasks, each PriceManager set a price that was near the market average price of $14,300. The BidManager of each agent took these pieces of data and formed a single bid.

[3] This is actually the profit for the last step that was accepted by the algorithm. It may not be the highest profit seen so far, if any "downhill" steps have been taken.

Stage III The bids were received and analyzed by the customer agent. It decided to accept the bid of Supplier 1.

We are testing more complex strategies for choosing which tasks to bid on and what price to bid.

6 Related Work

Markets play an essential role in the economy, and market-based architectures are a popular choice for multiple agents (see, for instance, [Chavez and Maes, 1996, Sycara and Pannu, 1998, Wellman and Wurman, 1998, Tsvetovatyy *et al.*, 1997, Karacapilidis and Moraïtis, 2001, Choi and Liu, 2001]. Most market architectures limit the interactions of agents to manual negotiations, direct agent-to-agent negotiation [Sandholm, 1996, Faratin *et al.*, 1997], or various types of auctions [Wurman *et al.*, 1998].

Existing architectures for multi-agent virtual markets typically rely on the agents themselves to manage the details of the interaction between them, rather than providing explicit facilities and infrastructure for managing multiple negotiation protocols. As discussed above, agents interact with each other through a Market. Our Market infrastructure provides a common vocabulary, collects statistical information that helps agents estimate costs, schedules, and risks, and acts as a trusted intermediary during the negotiation process.

Little research appears to have been done in bidding strategies for the style of auction used by MAGNET. However, Kephart, Hanson, and Greenwald have written a survey article aimed at understanding collective interactions among agents that dynamically price services or goods [Kephart *et al.*, 2000]. In this article, several useful pricing strategies for sellers are discussed. The game-theoretic computation, GT, chooses prices randomly from a distribution which is computed from buyer parameters as well as the number of sellers bidding. Like MAGNET, this pricing strategy assumes that no seller observes another seller's price before setting its own price. A second algorithm discussed is called the myoptimal, MY, or the best-response Cournot. Like the GT algorithm, MY requires perfect knowledge of the buyer population as well as the number of sellers in the Market. The third algorithm discussed is called the derivative-follower, DF. It does not require any knowledge about buyers or assumptions about the number of sellers. Rather it uses a learning technique by experimenting with increases or decreases in price. It continues to move its price in the same direction until the observed profitability level begins to fall, seeking a local maxima of profitability.

The first two pricing strategies discussed in this article could only be employed by supplier agents in MAGNET if the MAGNET server were to give out information on the number of other suppliers registered to bid on a given task. The MAGNET server shall certainly have this information, but it is not clear that this information should be given to supplier agents. The learning employed in the DF algorithm is designed to be used in situations where sales occur repeatedly. This algorithm was the inspiration for the simulated-annealing-based PriceManager.

7 Conclusion and Future Work

At this time we have a full implementation of the customer agent, but only a partial implementation of the supplier agent. In the Market, work is needed to develop mechanisms for transferring resources from the supplier to the consumer task and for tracking the monetary situation of the SupplierAgents. In the ResourceManager work is needed in implementing and testing the learning of cost minimization as well as resource reservation and allocation strategies. Finally, it might be interesting if the current use of task plans were to be made hierarchical. In this way, some ResourceManagers would act as consumers until an atomic level of a task was reached.

The system is not yet mature enough to test whether the PriceManager can "learn" a good pricing strategy. Once the system is ready, we would like to see if profit margins can be improved and which features of the environment have an effect on profits. For example, we want to see the difference between markets with homogeneous supplier agents and markets with heterogeneous ones. It might also be interesting to see how the size of RFQs affect profitability. The ratio of suppliers to customers could be another interesting parameter to study.

The system does not support payments yet, so at this time it is not possible to collect the data necessary for the SalesAnalyst. When these features are implemented, we would like to whether or not we can promote loyalty among good customers and see if we can avoid high-risk tasks and ill-behaved customers.

So far, the MAGNET system has been used for several types of studies. Recent work includes experiments with performance of winner-determination algorithms [Collins *et al.*, 2002], and studies of the RFQ composition problem [Babanov *et al.*, 2002]. We are currently studying how to use an evolutionary approach to let the market develop and stabilize so that which various supplier strategies can be studied. Our longer-term goal is to support studies of supplier strategies, and studies of mixed-initiative decision making with human users in realistic market simulations.

References

[Babanov *et al.*, 2002] Alex Babanov, John Collins, and Maria Gini. Risk and expectations in a-priori time allocation in multi-agent contracting. In *Proc. of the First Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002.

[Brazier *et al.*, 2000] F. Brazier, C. Jonker, and J. Treur. Design of multi-agent systems. Technical report, Vrije Universiteit Amsterdam, 2000.

[Chavez and Maes, 1996] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. of the First Int'l Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.

[Choi and Liu, 2001] Samuel P. M. Choi and Jiming Liu. A dynamic mechanism for time-constrained trading. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 568–575, 2001.

[Collins *et al.*, 1998] John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 285–292, May 1998.

[Collins *et al.*, 2002] John Collins, Maria Gini, and Bamshad Mobasher. Multi-agent negotiation using combinatorial auctions with precedence constraints. Technical Report 02-009, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, Minnesota, February 2002.

[Faratin *et al.*, 1997] Peyman Faratin, Carles Sierra, and Nick R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1997.

[Fatima and Wooldridge, 2001] S. Shaheen Fatima and Michael Wooldridge. Adaptive task resources allocation in multi-agent systems. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 537–544, 2001.

[Karacapilidis and Moraïtis, 2001] Nikos Karacapilidis and Pavlos Moraïtis. Intelligent agents for an artificial market system. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 592–599, 2001.

[Kephart *et al.*, 2000] Jeffrey O. Kephart, James E. Hanson, and Amy R. Greenwald. Dynamic pricing by software agents. *Computer Networks*, 32(6):731–752, 2000.

[Loritsch, 2001] B. Loritsch. Developing with Apache Avalon. Apache Software Foundation, 2001.

[Nisan, 1999] Noam Nisan. Bidding and allocation in combinatorial auctions. In *1999 NWU Microeconomics Workshop*, 1999.

[OECD, 1997] OECD. Guidelines for cryptography policy. http://www1.oecd.org/dsti/sti/it/secur/prod/crypto2.htm, 1997.

[Ossher and Tarr, 2000] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Proc. Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer, 2000.

[Reeves, 1993] Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY, 1993.

[Sandholm, 1996] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents*. PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1996.

[Shehory and Sturm, 2001] Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In *Proc. of the Fifth Int'l Conf. on Autonomous Agents*, pages 624–631, 2001.

[Sycara and Pannu, 1998] Katia Sycara and Anandeep S. Pannu. The RETSINA multiagent system: towards integrating planning, execution, and information gathering. In *Proc. of the Second Int'l Conf. on Autonomous Agents*, pages 350–351, 1998.

[Tsvetovatyy *et al.*, 1997] Maxsim Tsvetovatyy, Maria Gini, Bamshad Mobasher, and Zbigniew Wieckowski. MAGMA: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 11(6):501–524, 1997.

[Wellman and Wurman, 1998] Michael P. Wellman and Peter R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115–125, 1998.

[Wurman *et al.*, 1998] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second Int'l Conf. on Autonomous Agents*, pages 301–308, May 1998.