The UMDL Service Market Society

José M. Vidal, Tracy Mullen, Peter Weinstein,
Edmund H. Durfee
Artificial Intelligence Laboratory, University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110
{jmvidal, mullen, peterw}@umich.edu

September 30, 1997

Abstract

One of our goals when building the University of Michigan Digital Library (UMDL) was to provide an architecture for a digital library that can continually reconfigure itself as users, contents, and services come and go. This has been achieved by the development of a multi-agent infrastructure with agents that buy and sell services from each other using our commerce and communication protocols. We refer to the services/protocols offered by this infrastructure as the Service Market Society (SMS). The SMS allows for the decentralized (scalable) ongoing configuration of an extensible set of users and services. We describe the protocols, services and agents that embody the UMDL SMS and show how they achieve our stated goals.

1 Introduction

A library serves a community of users by making available information content and services that are valued by that community. A traditional, physical library is thus not simply a building that houses information, but rather a complex configuration of information goods and services that have been carefully selected and organized around the needs of a user community.

In a digital library, the content and services are electronically available, and user communities are no longer geographically defined. Realizing a digital library therefore includes difficulties in digitizing contents, computerizing services, and networking together users. But even if these difficulties are overcome, the result can well be an overwhelming tangle of possible information sources without the structure and selectivity that renders a library navigable. In other words, if the administration of a traditional library is challenging, the administration of a digital library can border on impossible due to the magnitude of content and services available, the rate of change in what is available, the size of a user population that is not bounded by physical proximity, and the evolving nature of that population.

One answer to this challenge is to rely on traditional methods of putting administrators at the center of the endeavor, to attract, register, and track a user community, to seek and include the content that will benefit the community, and to provide the most valuable services for tasks such as organizing, searching, abstracting, and disseminating the content. An alternative approach, however, is to move as much of the administration into the digital infrastructure as possible. The goal is to provide mechanisms by which a digital library can continually reconfigure itself as users, contents, and services come and go. These mechanisms should encourage: **Flexibility**: They should be able to embody a wide variety of policies to realize different flavors of libraries (public, corporate, university, personal,...) **Extensibility**: Providers and consumers of information goods and services should have incentives to join the library and abilities to find their counterparts. **Scalability**: As the plethora of users, goods, and services grows, the underlying, computerized administration of the library should not bog down.

Toward this end, the University of Michigan Digital Library (UMDL)[1] is structured as a collection of agents that can buy and sell services from each other using our commerce and communications infrastructure. While one of the emphases of the UMDL is to provide a working testbed to improve secondary education [9], a second emphasis is on the definition and design of the infrastructure, and the kinds of agents that exist in it, that allow decentralized (scalable) ongoing configuration of an extensible set of users and services. We refer to the services/protocols offered by this infrastructure as the Service Market Society (SMS).

The SMS requires the integration of numerous agent technologies for knowledge exchange, commerce, learning, and modeling. In this paper, we describe how we have brought these technologies together to create a prototype SMS in which a changing population of agents can find each other, enlist

[1] `http://www.si.umich.edu/UMDL/`

each other's aid (for a price), decide on the terms of an interaction, and learn to differentiate among providers. We use our prototype to demonstrate how these technologies contribute to providing a flexible, extensible, and scalable digital library. Before this, however, we start with an overview of SMS in Section 2, and the component technologies of the UMDL ontology (Section 3) and auctions (Section 4). These technologies allow flexibility in the UMDL configuration policies, extensibility in what can be bought/sold in the SMS, and scalability by using demand (as represented by price) as incentive for replicating services (Section 5). Robust performance in such a society, however, also requires that participating agents make informed buy/sell offers for the goods and services, and that they have the ability to recognize and learn from whether another party keeps its end of the bargain (Section 6).

2 The Service Market Society

The UMDL SMS implements a market-based multi-agent system where agents buy and sell services from each other. Agents can be added to or removed from the system at any time. Instead of having to rely solely on internally designed agents, UMDL can attract outside agents to provide new services. These agents in turn are motivated by the long-term profit they might accrue by participating in the system.

The raw resources of a digital library consist of collections of information, where a collection might be a database of journal articles or it might be a collection of related web pages. Collection Interface agents, (CIAs), provide access and search services for these collections. Various middlemen, or *mediator* agents, transform the raw information resources into finished products or services which the end-user desires. Users, or library patrons, each have an individual User Interface Agent, (UIA), which interacts with UMDL on their behalf to acquire these services.

Figure 1 shows a simple scenario which we will be using throughout this paper. In this scenario, users want to find sources of information for various topics (e.g. history, science or mathematics) and various audience levels (e.g. middle school, high school, or professional). Query Planning agents (QPAs) act as middlemen, accepting queries from users and returning collections related to the query. Some QPAs support general queries, some are specialized and support queries geared to a particular topic area or audience level, e.g. a specialized QPA for middle-school-science queries. Below we discuss in more

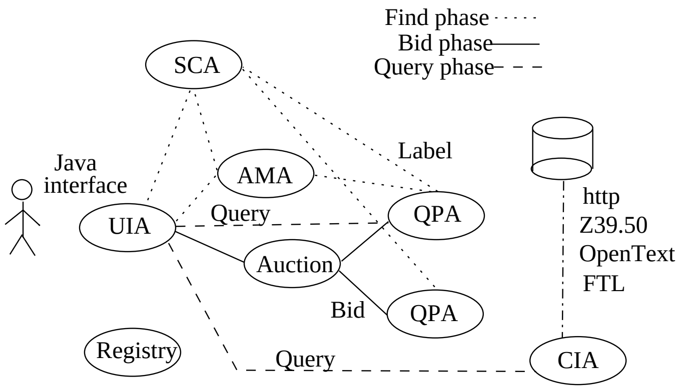

Figure 1: The UMDL Service Market Society. It supports any number of UIA, CIA, QPA and Auction agents.

detail the mechanics of how this scenario is implemented in UMDL.

Initially the user sends a query, via a web-based Java interface, to the User Interface Agent (UIA). The UIA must then find an agent that can service this query. In order to allow agents to find the services they need, we have implemented an ontology of services (Section 3), which is represented using description logic; specifically, Loom [4]. Every agent must be able to describe the service it wishes to buy and sell using Loom's query and concept languages, respectively, with terms from the UMDL ontology.

In order to join the UMDL every agent must register its unique agent-id with the Registry. An agent can then advertise by sending[2] its service description to the Service Classifier Agent (Section 3.1), as shown by the dotted lines. The SCA maintains a mapping between service descriptions and labels which are used by all agents as a convenience in communication.

The SCA automatically classifies the advertised service descriptions into a subsumption-based taxonomy. This organization enables the SCA to match requests for services to "semantically close" descriptions, to recommend the most appropriate services out of those that are currently available. We encourage agents to describe their needs with as much detail as possible. The SCA can then recommend the available services with the least general subsuming descriptions.

[2] Agent communications use KQML primitives implemented using CORBA objects, see [1] for details.

Next, buyers and sellers of a given service need to locate each other. Multi-agent systems often have specialized system agents which broker connections among other agents, called *facilitators*. Within UMDL, we use market facilitator agents, or *auctions* (Section 4). UMDL auctions operate by collecting offers from participating agents and determining agreements consistent with those offers. Many different types of auctions can be instantiated by using different auction parameters.

Once an agent has acquired the appropriate service-label from the SCA, and is ready to buy/sell that service, it sends the desired label to the Auction Manager Agent (AMA) (also shown by dotted lines) which returns a list of auctions which sell matching services (Section 4.1). The agent then sends buy or sell bids to a particular Auction agent (solid lines). The auction matches individual buyers and sellers and sets price of the transaction.

In our scenario, UIAs want to buy query planning services while QPAs want to sell it. Once a match is found for the UIA, it will send its query to the matching Query Planning Agent (QPA). The QPA [6] returns the appropriate Collection Interface Agents (CIA) and the UIA then forwards its query to them. The CIAs can translate UMDL queries to a variety of protocols (e.g. http, Z39.50, FTP, etc.) and return the appropriate documents.

The agents are free to return services of differing qualities or to disagree on the exact quality of a given service. Therefore, agents might also need to be able to assess the quality and value of a service received or the exact price to charge, in order to maximize their expected value/profit. This is achieved by giving agents the ability to learn (Section 6) and to make strategic bids. We find that the SMS market works to minimize the amount of strategic thinking that the agents can gainfully engage in, but it does not eliminate the need for having some agents with learning/strategic capabilities, even if these are not always used.

3 The UMDL Ontology

We use ontologies to encode declarative descriptions of complex agent services. Declarative descriptions are required to establish a space of services independent of the implementation of the current set of agents. Representing complexity in digital libraries, and any other "real-world" domain, demands expressiveness that is substantially more powerful than that of relational databases. Services have many descriptive dimensions (attributes), may be

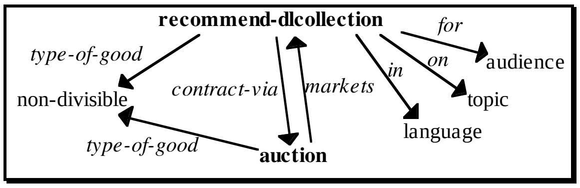

Figure 2: Multi-dimensional, multi-perspective complexity

partially described at any level of granularity (with any combination of dimensions), and may be viewed from many perspectives (accessed by different sequences of attribute values). For example, an agent might seek a service to recommend a collection of articles on volcanos, for high-school audiences, to be contracted for via auction. Alternatively, the agent might seek an auction that sells a service to recommend collections. (In Figure 2, follow links either from "recommend-dlcollection" at the top, or "auction" at the bottom of the figure). In an ontology, both perspectives can be represented simultaneously, and retrieval supports queries on any level of granularity. Declarative formalisms with less expressiveness than ontologies, such as relational databases, force a commitment to a particular ordering of dimensions and a particular perspective.

Because ontologies are costly to develop, many researchers are working on ways to define ontologies so that they may be reused in different problem solving contexts [2]. To promote reuse, the UMDL ontology is divided into nested modules (each of which is itself called an ontology) [8]. The most general includes library content and services that we consider to be part of a "generic" digital library. The second module adds concepts specific to the UMDL implementation, such as auctions. The third module describes agent services. We call this last ontology "dynamic" because agents define new service concepts at runtime. In contrast, "static" ontologies are either fixed, or are changed slowly over time by committees of persons.

3.1 The Service Classifier Agent

UIAs, QPAs and Auction agents use the same SCA, and thus subscribe to the same ontologies: the UMDL SMS is not terminologically heterogenous. Figure 3 illustrates the SCA's interactions with the other agents. Agents are in ovals, ontologies in rectangles, and the arrows connecting agents represent

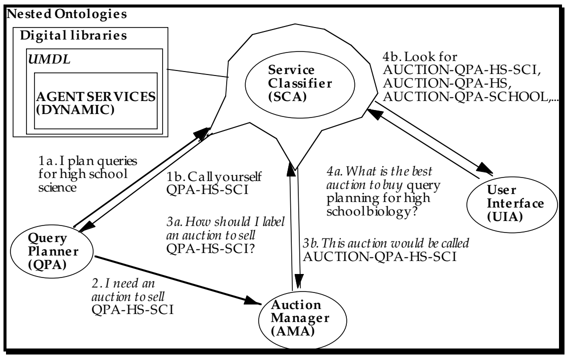

Figure 3: Agent interaction with the SCA in the SMS

messages, paraphrased in natural language. In the SMS these messages are expressed in Loom. To communicate with the SCA, agents use terminology from the nested ontologies. The thin line around the SCA is jagged to show that the set of available terms is dynamic; both QPAs and the AMA add concepts to the agent services ontology. Messages use font styles that correspond to the ontology labels (plain, italics, and upper-case), to highlight the source of their terminology.

Figure 3 illustrates the ability of the SCA to construct new terminology from existing terms at runtime. Concepts in the SCA's dynamic ontology hide knowledge, just as words chunk meaning in natural language. Auctions sell some service, but they don't need to know anything about that service. The AMA asks the SCA for a service label for an auction, using the QPA's service label, but it does not know anything about that service. The SCA classifies the auction service using characteristics inferred from the QPA's service label. Thus, the SCA can respond to the UIA's request for an auction, which is phrased in terms from the static ontologies, rather than the label which the AMA used to define the auction service. This appropriate hiding of knowledge reduces overall system complexity, and increases reusability and maintainability.

In future versions, auction managers will make extensive use of the SCA's knowledge. First, ontologies are the natural place to represent the type of auction that is appropriate for marketing a particular service. Second, auction managers may need to define markets for sets of services, rather than services with exactly the same classification. Efficient resource allocation depends on maintaining suitable numbers and ratios of seller and buyer agents. These quantities can be controlled by adjusting the generality or specificity of the service to be auctioned.

The most dramatic contribution of the SCA to the SMS derives from its declarative description of services, and its ability to rank available services given a target and search strategy. When a new QPA provides a service that better meets a UIA's needs, the UIA switches to buying services from the new QPA, without requiring any intervention to modify the UIAs behavior. This occurs because the UIA periodically repeats its search for the best available service. Furthermore, if the new service subsequently becomes unavailable, the UIA reverts to its previous supplier.

4 Market-based Negotiation– Auctions

Since UMDL cannot know at design-time what services will be available in the future or what the best negotiation mechanisms are for any given situation, its languages and protocols have been designed for flexibility and extensibility. In the case of negotiation mechanisms, UMDL uses a generalized auction specification to allow goods and services to be offered under a wide variety of terms [5].

While generalized auctions are not the only possible kinds of negotiation mechanisms, they offer many advantages. They provide a structured, yet flexible market infrastructure promoting automated negotiation due to the following characteristics:

1. **Mediated** (vs. Unmediated): Buyers do not have to separately find and contact every seller; a useful property in a large-scale and dynamic environment. Information about current status can be easily and uniformly disseminated.

2. **Price** (vs. Barter): Under certain circumstances, price minimizes communication between agents.

3. **Formal** (vs. Informal): Standardized offers simplify communication between agents.

Auction specifications consist of parameters which can be tuned to reflect the type of good being sold, timing requirements or mechanism properties desired. For example, a query planning service might be sold differently depending on how it is bundled (e.g., per-query, subscription), characteristics (topic, audience, timeliness), terms (redistribute, read-only) or to whom it is sold (individual, library, group). Different kinds of auctions will have advantages and disadvantages for different types of information goods and services.

Although evaluating tradeoffs between desirable properties of different auction types is important, the role of UMDL is not to require that a particular auction be used in a given circumstance, but rather to provide an open framework whereby the use of these market facilitator agents is supported. To automate the creation and management of auctions as much as possible, we use a system agent called the Auction Manager Agent.

4.1 The Auction Manager Agent

The purpose of the Auction Manager Agent (AMA) is both to generate and track auctions as well as to serve as a tool for exploring alternative market configurations. Currently the AMA is used to automate the process of creating a well-specified market. Agents request auctions for the services they sell/buy from the AMA. Using the SCA, the AMA determines which auctions sell these services. If one or more such auctions exist, then the agent is notified of their agent-ids. Otherwise, the AMA creates one and then informs the requesting agent of its location. Since the AMA uses the SCA to generalize service descriptions, the description sent by the agent does not have to exactly match the auction's service description.

This simple behavior provides a good deal of functionality. However, in the future, since the number of possible markets in UMDL is virtually unbounded, we will need to account for market overhead costs and recommend the most appropriate auctions.

5 Simple Market Scenarios– Price Takers

In this section, we discuss two simple scenarios implemented in UMDL and the design process and mechanisms used. The scenarios use the same agents and services as shown in Figure 1.

However, in this section, we make the assumption that the agents have been designed within UMDL to fulfill certain system-level objectives. In particular, the QPAs have been designed to behave competitively. Competitive agents take prices as given– ignoring any market power they might possess, thereby eliminating the need to reason strategically about other agents' bidding behavior and simplifying the QPA design.

5.1 Load Balancing

This scenario shows how, under certain conditions, a price mechanism can be used to distribute service load between multiple providers of the same service. Specifically, consider the case where we want to balance the service load between several high-school science QPAs running on separate machines. Each QPA provides the same service, in this case, *exactly* the same service since each one is just a spawned copy of the same code. We simplified this scenario in a number of ways, assuming that all sites provide free access, and all queries are the same.

In designing the QPAs, we assume they were competitive, and bid their marginal cost, namely what it would cost them to provide another unit of their product. This pricing policy directly reflects the cost to the system of using the input resources to provide this service. QPAs use computational and network resources to produce query planning service. Since these resources get congested as the load on a machine or network gets heavier, the marginal cost of adding another query will increase with the number of queries currently being processed. We modeled this technology in the QPAs using a quadratic cost function:

$$\begin{aligned} \text{Cost(query)} &= A * \text{load}^2 + B * \text{load} \\ \text{Marginal Cost(query)} &= 2 * A * \text{load} + B \end{aligned}$$

Each additional query the QPA produces is priced at its marginal cost. Note that this cost function does not represent any real computer load model, but it does capture the basic notion of using a congested resource. The parameters A and B can be set to reflect relative differences between QPA technologies. For example, a QPA with $A, B = 1$ could be thought of as running on a faster machine than a QPA with $A, B = 2$.

All sellers, or QPAs, make a offers based on their current marginal cost. Buyers, or UIAs, are assumed to want to purchase the query planning service immediately upon bidding. Query planning services are traded in an auction which matches the current lowest price seller with an incoming buyer, as long as the buyer's bid is above that price. Note that because of the way the agents have been designed, we know that this auction is incentive compatible for buyers, i.e., their best bidding strategy is to bid their true value for query planning services. This is because the QPAs who would normally have an incentive to try and strategize about bid amounts, have been engineered to be competitive[3]. Once a transaction occurs between a given seller and buyer, both offers are removed from the standing offer list. The QPA recomputes its marginal cost based on having an additional query to process and submits a new, higher, sell offer to the auction.

To see how this results in query load balancing, let's consider what happens where there are two QPAs, (QPA-1 and QPA-2). Given the same initial query load, the sell offers for each agent will be the same, and the first buyer will be matched up arbitrarily with one of them, say QPA-1. At this point, the query load for QPA-1 will rise, causing its marginal cost to rise, and it will submit a new offer at a higher price than before. The next incoming buyer will be matched with the current lowest price seller, QPA-2. QPA-2 now makes its new higher sell offer to reflect its increased load. Assuming that QPA-1 and QPA-2's previous queries are still being processed, both agents are again offering query planning service at the same, although higher, price. If one of QPA-1's queries finishes, then it sends in a new lower offer, and will be matched up to the next incoming buyer. Given a steady load of user queries, this system appears to settle down to an equilibrium price, although we have not done any formal analysis of these results. Notice that the dynamic addition or deletion of an agent does not affect the long-term running of the load balancing mechanism. For example, if any one of the agents were to die, it would only affect the queries that they were processing at that time. Future query planning requests would be matched with the remaining QPAs still participating in the auction. Similarly, spawning a new QPA means that it can start making offers to the query planning auction immediately.

Similar situations where this mechanism would be useful are in the provision of basic library services, which outside agents may not be interested in supplying. By making the agents competitive, UMDL can assure that the system costs are accounted for, that users get a low price, and agent designs are kept simple. Another example is distributing the access load to collec-

[3]If the QPAs had not been engineered, a better solution would have been to use a second price auction.

tions for which the library has a site license. Even though the site license may allow unlimited access, there are still associated network and compute costs, and for popular collections we may want to create mirror sites and distribute the load across them. A consistently high price of access may indicate when a given collection should be mirrored.

5.2 Price Monitor Agent

In a large-scale, dynamic system, information about the current status of the system can serve as feedback to different kinds of control mechanisms. In this section, we describe a simple monitor agent, called a Price Monitor Agent (PMA), which collects information about an auction's prices over time and uses that information to decide whether to spawn more service providers.

In our scenario, the PMA monitors the price of query planning services. The price in this auction reflects the load on the QPAs— a consistently high price means that the QPAs are heavily loaded. One way to reduce the load is to replicate QPAs (assuming that there are other less heavily loaded machines on the network). When the the load is reduced, the price should come down.

The PMA is initially set with a given minimum and maximum price bounds, although they can be changed later. When the price exceeds the upper bound, i.e., the load on each QPA is getting high, it spawns additional QPAs. This effectively distributes the load. When the price goes below the lower bound, i.e., the QPAs load has fallen, the PMA has a choice of either removing or just inactivating one of the extra QPA, depending on whether it expects the load to increase again in the near term. The effect of this behavior is to keep the load across QPAs (and therefore the response time to users) within specified bounds even though the user demand is dynamic.

Figure 4 shows an overview of how the PMA operates. Below is a description of what is occuring at each step:

1. A heavily loaded QPA sells its services for $0.13.

2. The PMA checks the current price at the Auction.

3. Since the price is above it's upper bound ($0.10), the PMA spawns another QPA.

4. The new QPA sells its services for $0.08.

5. The user is matched with the lowest priced (least loaded) QPA.

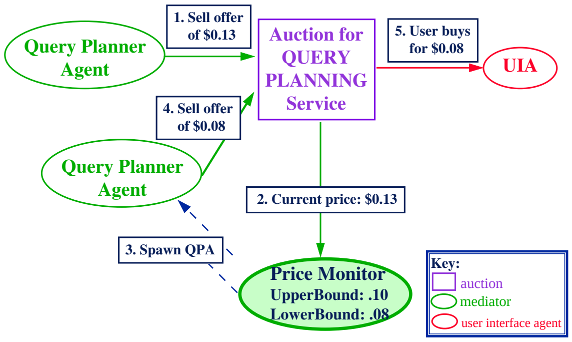

Figure 4: Price Monitoring Scenario

6. (not shown) When/if the price goes below $0.08, the PMA inactivates one or more of its spawned QPAs.

In our experiments, the PMA, given reasonable bounds for the prices and activity of the agents, was successfully able to keep the price within the given price bounds. However when the price bounds were too tight or the activity on the system was too dynamic, the PMA would oscillate. For example, if too many agents get added, the price may fall too low, causing too many agents to get removed, which causes the price to go too high, and so forth. Providing the monitoring agent with knowledge about usage patterns, or having it collect statistics about usage, is one way to address this problem.

6 A More Complex Market Scenario– Learning Agents

In this Section we extend the flexibility of the SMS by introducing auctions that reveal all information (i.e. bids) they have to all interested agents. The agents are free to ignore any of this added information. We also give the sellers the freedom to return a service of any quality. The only thing that will prevent them from doing this will be the fact that the buyers have the ability to learn, which they use for determining which seller to buy from and at what price. The scenario is the same as in Figure 1.

6.1 Learning in a market system

A real economic system provides monetary incentives to its participants and dynamically allocates the available system resources in part because the human beings and corporations that take part in it are smart. They can recognize when they are charging too little or too much, or when they are not getting the quality they expected. They also do not spend all their time thinking about the economy, but only do as much strategic thinking as is needed.

If we want our agent economy to be as robust as the real economy we will need to have at least marginally intelligent agents. These agents will need to know what to bid— both when price has reached an equilibrium (which is easy), and when the price is fluctuating. If an agent is the only seller of a service then it should be able to take advantage of its monopoly, while if the buyers find that a seller's prices are too high for the service it sells, they should be able to avoid buying from him. Agents will also need to have enough computational power left over to actually deliver their service.

Using learning agents frees us from having to implemented some sort of centralized "police" agents. While the UMDL ontology provides a way for agents to characterize the services they sell, there is no guarantee that all the goods sold at an auction for service x are indeed instances of service x. This guarantee could be provided by police agents that periodically check all the goods sold at all the auctions. Unfortunately, this would a) be computationally taxing solution on the system, b) give rise to thorny political problems given the fact that the police agents would have special rights over other agents and c) some agents might have specific subjective preferences over a good which can not be expressed in the ontology (and, therefore, not recognized by other agents). For example, while all agents might agree that QPA1 does sell service x, one agent might think that QPA1's service is faster, better, or more thorough than the same service x as provided by QPA2. This being the agent's subjective opinion, it is unlikely to be in agreement with all the other agents, but the agent might still be willing to pay more for service x from QPA1 than from QPA2. Giving agents the ability to learn is also a first step towards the implementation of "recommender" agents that gather together agents of similar tastes.

In essence, learning provides a way for agents to build *trust* among each other. For a market-system to work well, an agent needs to be able to trust that the other agent's view of good x is the same as his view of x. Similarly, an agent that uses recommender agents needs to trust the recommendations they give. This trust can be acquired by repeated iterations with the agents in question. Once the trust is acquired the learning is no longer needed, that is, until the trust is broken. This is why we argue that agents need the *capability* of learning, even if this capability is not always exercised.

Lastly, we propose that learning agents are not only useful, they are inevitable. In a society of selfish agents, we can expect that the designers will use every technology available to enhance the profits of their agents. Learning is one such technique. By implementing learning agents ourselves we can determine how much of an advantage they will have and how they will affect the system.

6.2 Experimental Results

In order to demonstrate the viability of the SMS with learning agents and under real-world heavy usage conditions, we ran several tests on the scenario shown in Figure 1. We implemented UIAs that periodically (every 16 seconds) buy a query from some QPA using the protocol described is Section 2. We also used one Auction, AMA, and SCA, along with several UIAs and QPAs. All the agents were deployed in machines all over our network. The UIAs kept track of how long it took for the QPA's reply to arrive and used this value in their learning. In general, the UIAs preferred fast and cheap service, and they were willing to pay more for faster service. A QPA's only preference was to increase its immediate profit, i.e. place its bid in order to maximize its expected profit (remember that failing to get a sale means the QPA gets zero profit).

We gave the agents different learning abilities (see [7]). 0-level agents used a simple form of reinforcement learning on the prices/values received. 1-level agents actually tried to model the other agents (as 0-level agents) and took actions based on predictions from these models.

6.2.1 0-level agents

For our first test we used 0-level UIAs and QPAs. They all began with no knowledge about what prices to bid or accept. UIAs quickly learned that they can expect to get higher value if they pick lower prices, while QPAs learned what price they can charge that will maximize their expected profit. As predicted by economic theory, this price was their marginal cost, i.e. the lowest price they can charge without losing money. This equilibrium was reached even while all the agents acted purely selfishly.

However, this equilibrium is not completely stable. Network and machine delays, along with the agents' occasional explorative actions[4], add noise to the system preventing the price from staying fixed at the marginal cost.

Even with only 0-level agents (i.e. no 1-level agents) we can see how the system behaves in a robust manner. Figure 5 shows the clearing price for an auction which starts with only one seller QPA. This QPA quickly "realizes" that he has a monopoly and starts to raise its prices. Other QPAs are then added (as marked by the small arrows on the x-axis), and we can see how their addition makes the price drop. The equilibrium price gets fairly close to the QPAs' marginal cost of 0. Towards the end the QPAs start to quit, since we set them to quit after a fixed amount of time, and the price starts to rise. This type of experiment gives us confidence that the agents will behave in a reasonable way even under boundary conditions, e.g. when a lot of the agents die, or when new services are added. If the agents where to determine their bid price based on a fixed utility function we might expect to find periodic or chaotic behavior [3], which we wish to avoid.

6.2.2 1-level Agents

1-level QPAs take advantage of price fluctuations by keeping models of the QPAs and UIAs and use these to make better predictions as to what they should bid. The 1-level models, while computationally expensive, allow QPAs to track the individual agents more closely, thereby identifying when a UIA is willing to pay more than the going rate. Previous research has shown that the advantages of 1-level models can be correlated to the price volatility (see [7]).

However, this strategic thinking is only successful against 0-level UIAs. When we tested the 1-level QPAs against the 1-level UIAs we found the QPA's performance on par with other similar 0-level QPAs. The 1-level QPA's advantage was eliminated only at the cost of making the UIAs keep 1-level models. From the systems' perspective, these computations are "wasted" effort in that they do not contribute in any way to the task of servicing the user. Notice, however, that they would only be needed when the 1-level QPAs participating in the auction are offering significantly different services.

[4]The agents were set to take a random action with probability of .05. This keeps them from converging on a local maxima.

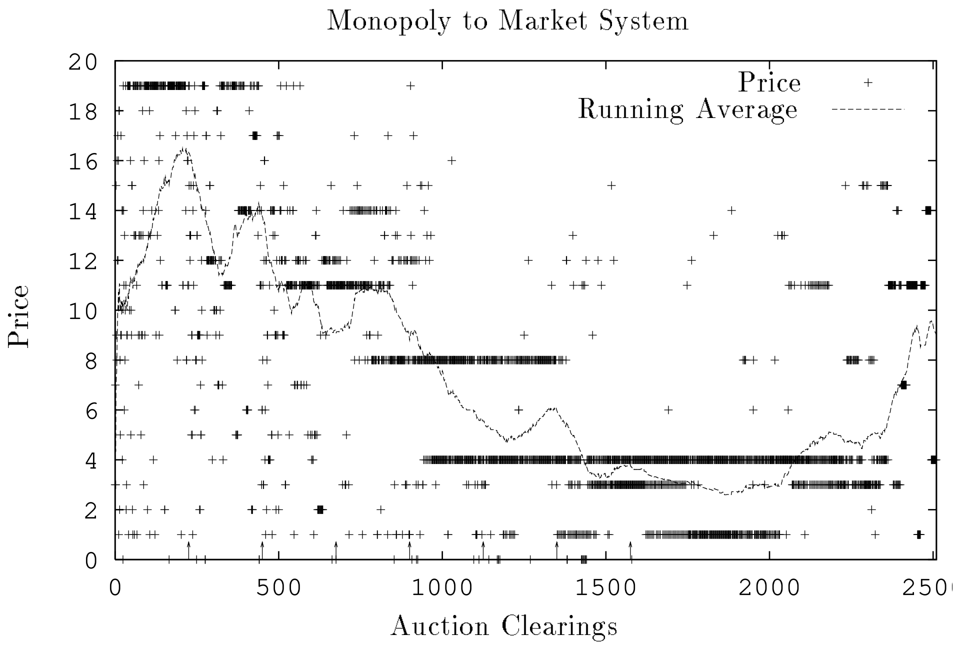

Figure 5: Clearing price for successive auctions. Initially there is only one seller QPA, other ones are added at regular intervals as noted by the arrows on the x-axis. QPAs' marginal cost is zero.

In these cases, the buyers will want to determine which one of the QPAs offers the "best" (in its own opinion) service. Once they have determined this then, perhaps, a new auction could be started that would service only these QPAs so that the UIAs could then go back to being simpler 0-level agents and avoid the heavy computational costs. The incentive/financing mechanisms for starting new auctions are still under study.

6.3 Conclusions from Learning Experiments

The results on learning deeper agent models show that the UMDL SMS benefits from the existence of agents that have the *capability* of keeping deeper models and of looking out for their best interests. Agents are encouraged to model other because this can bring them higher profit. However, there is also a computational cost associated with modeling, along with a decreasing return as other agents also start to build models. This means that that the SMS will likely evolve to a point where some agents build models, some of the time. This is a great scenario because it gives us a very robust system (i.e. one that can not be sabotaged by deviant agents), while using few resources for this purpose and distributing the resources it uses among the agents.

7 Summary

We have given a condensed overview of the UMDL SMS and the agents, protocols and languages that it encompasses. The system's benefits are its dynamic extensibility, the ability to write agent wrappers around legacy database systems (or even other existing web services), an effective allocation of system resource with little overhead, and a very open architecture that allows and encourages (via profits) any third-party to incorporate their agents into our society.

These results were achieved via the merging of different technologies which include: ontology design, market oriented design, and nested agent modeling/learning. Throughout the paper we mentioned research issues that still remain open and are under study. Still, we believe that the SMS has already demonstrated that it can meet most of the our original goals.

Acknowledgments The rest of the members of the UMDL SMS group all contributed to its design and development. They are: Anil Arora, Bill

Birmingham, Eric Glover, Anisoara Nica, Sun Park, William E. Walsh.

References

[1] Edmund H. Durfee, Daniel L. Kiskis, and William P. Birmingham. The agent architecture of the university of michigan digital library. *IEEE Transactions on Software Engineering*, 144(1), 1997.

[2] Thomas R. Gruber. A translation approach to portable ontology specifications. Technical report, Stanford University Knowledge Systems Laboratory, 1993.

[3] Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1325–1332, December 1991. (Special Issue on Distributed AI).

[4] Robert M. MacGregor. The evolving technology of classification-based knowledge representation systems. In John. F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann.

[5] Tracy Mullen and Michael P. Wellman. Market-based negotiation for digital library services. In *Second USENIX Workshop on Electronic Commerce*, 1996.

[6] José M. Vidal and Edmund H. Durfee. Task planning agents in the UMDL. In *Proceedings of the Fourth International Conference on Information and Knowledge Managment (CIKM) Workshop on Intelligent Information Agents.*, 1995. .

[7] José M. Vidal and Edmund H. Durfee. The impact of nested agent models in an information economy. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 377–384, 1996. `http://ai.eecs.umich.edu/people/jmvidal/papers/amumdl/` .

[8] Peter Weinstein and Gene Alloway. Seed ontologies: growing digital libraries as distributred, intelligent systems. In *Proceedings of the Second ACM International Conference on Digital Libraries*, June 1997.

[9] Michael P. Wellman and *et. al.* Toward inquiry-based education through interacting software agents. *IEEE Computer*, May 1996.