Decentralized Peer-to-Peer Auctions

Marcus Fontoura, Mihail Ionescu, Naftaly Minsky
fontoura@almaden.ibm.com, {ionescu,minsky}@cs.rutgers.edu
IBM Almaden Research Center
650 Harry Road, San Jose, CA, 95120
Department of Computer Science
Rutgers University, Piscataway, NJ, 08854

Abstract

This paper proposes a flexible architecture for the creation of Internet auctions. It allows the custom definition of the auction parameters, and provides a decentralized control of the auction process. Auction policies are defined as *laws* in the Law Governed Interaction (LGI) paradigm. Each of these laws specifies not only the auction algorithm itself (e.g. open-cry, dutch, etc.) but also how to handle the other parameters usually involved in the online auctions, such as certification, auditioning, and treatment of complaints. LGI is used to enforce the rules established in the auction policy within the agents involved in the process. After the agents find out about the actions, they interact in a peer-to-peer communication protocol, reducing the role of the centralized auction room to an advertising registry, and taking profit of the distributed nature of the Internet to conduct the auction. The paper presents an example of an auction law, illustrating the use of the proposed architecture.

1 Introduction

The Internet has made possible the creation of virtual auctions rooms, in which buyers and sellers scattered across the globe interact to close deals. Internet auctions allow faster and less expensive transactions with no geographical barrier [7]. Although Internet auctions have been successfully used [6], current auction applications still do not fully exploit the distributed nature of the Internet. They are based on centralized systems that necessarily make decisions about the auction process, taking away important choices from the auction participants. These choices include:

- The auction algorithm itself: several types of such algorithms can be used (like open-cry, dutch, sealed, etc.), and each of them may have many variations [4, 5];
- Certification: how to compute reputation and trust information about the auction participants;
- Auditing: what needs to be audited, and by whom;
- The treatment of complaints: how to handle complaints about inappropriate behavior of auction participants or about unsuccessful transactions.

Although some centralized auction systems provide flexibility in the choice of the auction algorithm (e.g. through parameters, as in the AuctionBot system [17]), the ability to make the other auction parameters flexible, such as certification, auditing, and treatment of complaints, is much more difficult, if at all possible, to be accomplished via a centralized system.

Let us consider two auction scenarios: in the first the seller wants to sell modern art items while in the second he or she wants to sell audio CDs. In the modern art case, it is very important for the seller to present some certificate that can be used in order to determine the authenticity of the items being sold. If this auction is being conducted by a centralized auction server, the buyer has to trust either the server or the certifying authorities appointed by the server. This might not be

desirable for the buyer, since different people trust different entities, which is especially true for transactions involving valuable items. Therefore, a buyer should be able to decide to participate in such an auction only if some particular certifying authorities (CAs) are involved. On the contrary, in the case of audio CDs, a sophisticated certification mechanism is not necessary since buyers can easily check the accuracy of the product description and complain afterwards, if not satisfied.

This simple example illustrates that the level of certification required in an auctioning system is product dependent, and cannot be fully implemented by a centralized server. Instead, auction participants should be able to indicate which certifying authorities they trust, customizing the auction process. The same argument is true for other auction parameters. For example, the management of user reputation in eBay (http://www.ebay.com) allows people to build a good reputation by doing false transactions. Buyers would be more willing to participate in some auctions if someone they trust knew the real identity of the seller. If such an entity exists, complaints can more easily be solved and the possibility of cheating decreases.

Having a centralized server to perform all the tasks of an Internet auction (advertising, enforcing the auction policy, controlling user reputation, and so on) is not suitable for online auctioning. Since different people trust different entities for performing each of the individual tasks in the auction process, we need a different model, in which all the entities involved can be dynamically specified in an *auction policy*.

In this model the auctions applications are reduced to an advertising registry, in which sellers and buyers register and search for auctions [1]. Each registered auction follows a given auction policy that is publicly available. Sellers and buyers can participate in a given auction as long as they obey the auction policy. Once they find out about the auction, the interested parties establish a peer-to-peer (P2P) communication, which is regulated by the specified policy. The policy is explicitly defined in a special file, called the auction law, using the Law Governed Interaction (LGI) paradigm [12].

LGI allows a group, or a *community*, of distributed heterogeneous agents to interact with each other with confidence that an explicitly specified policy, called the *law* of the group, is complied with by everyone in the group. LGI has been designed specifically to satisfy the following principles, which we consider critical for coordination in large heterogeneous systems: (1) coordination policies need to be formulated explicitly rather than being implicit in the code of the agents involved, (2) coordination policies need to be enforced, and (3) the enforcement needs to be decentralized for scalability. LGI has been implemented in a middleware system called Moses [12], which has been applied to a broad range of coordination and control applications, including: on-line reconfiguration of distributed systems [13], security [11], and electronic commerce [10].

The rest of the paper is organized as follows: Section 2 briefly describes the LGI model and its main concepts. Section 3 describes the architecture for law-based auctions and a scenario in which the proposed architecture can be applied. Section 4 presents an overview of related work. Section 5 concludes the paper and presents our future research directions.

2 Law-Governed Interaction—an Overview

Broadly speaking, LGI is a message-exchange mechanism that allows an *open group* of distributed agents to engage in a mode of interaction *governed* by an explicitly specified policy, called the *law* of the group. The messages thus exchanged under a given law $\mathcal{L}$ are called $\mathcal{L}$-messages, and the group of agents interacting via $\mathcal{L}$-messages is called a *community* $\mathcal{C}$, or, more specifically, an $\mathcal{L}$-community $\mathcal{C}_{\mathcal{L}}$. The concept of LGI has been introduced (under a different name) by one of the authors (Minsky) in [9], and has been implemented via a middleware called Moses [12].

By the phrase "open community" we mean (a) that the membership of this community can change dynamically, and can be very large; and (b) that the members of a given community can be heterogeneous. In fact, we make here no assumptions about the structure and behavior of the agents[2] that are members of a given community $\mathcal{C}_{\mathcal{L}}$, which might be software processes, written in

[1] Although the paper focuses on auctions, the proposed architecture also works for other kinds of online trading, such as brokerages and two party negotiations [5].

[2] Given the popular usages of the term "agent," it is important to point out that we do not imply by it either

an arbitrary languages, or human beings. All such members are treated as black boxes by LGI, which deals only with the interaction between them via $\mathcal{L}$-messages, making sure it conforms to the law of the community. (Note that members of a community are not prohibited from non-LGI communication across the Internet, or from participation in other LGI-communities.)

For each agent `x` in a given community $\mathcal{C}_{\mathcal{L}}$, LGI maintains, what is called, the *control-state* $\mathcal{CS}_x$ of this agent. These control-states, which can change dynamically, subject to law $\mathcal{L}$, enable the law to make distinctions between agents, and to be sensitive to dynamic changes in their state. The semantics of control-states for a given community is defined by its law, could represent such things as the role of an agent in this community, and privileges and tokens it carries. For example, under law $\mathcal{OC}$ to be introduced in Section 3, as a formalization of an open-cry auction policy, the term `role(seller)` in the control-state of an agent denotes that this agent has been certified as a seller.

We now elaborate on several aspects of LGI, focusing on (a) its concept of law, (b) its mechanism for law enforcement, and (c) its treatment of digital certificates. We do not discuss here several important aspects of LGI, including the *interoperability* between communities, and the treatment of *exceptions*. Nor do we discuss here the expressive power of LGI, its theoretical performance, and the performance of its current implementation. For these issues, and for a more complete presentation of the rest of LGI, the reader is referred to [12, 16, 1].

2.1 The Concept of Law

Generally speaking, the law of a community $\mathcal{C}$ is defined over a certain types of events occurring at members of $\mathcal{C}$, mandating the effect that any such event should have—this mandate is called the *ruling* of the law for a given event. The events subject to laws, called *regulated events*, include (among others): the *sending* and the *arrival* of an $\mathcal{L}$-message; the *coming due of an obligation* previously imposed on a given object; and the *submission of a digital certificate* (more about the latter two kinds of events, later). The operations that can be included in the ruling of the law for a given regulated event are called *primitive operations*. They include, operations on the control-state of the agent where the event occurred (called, the "home agent"); operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent.

Thus, a law $\mathcal{L}$ can regulate the exchange of messages between members of an $\mathcal{L}$-community, based on the control-state of the participants; and it can mandate various side effects of the message-exchange, such as modification of the control states of the sender and/or receiver of a message, and the emission of extra messages, for monitoring purposes, say.

On The Local Nature of Laws: Although the law $\mathcal{L}$ of a community $\mathcal{C}$ is *global* in that it governs the interaction between all members of $\mathcal{C}$, it is enforceable *locally* at each member of $\mathcal{C}$. This is due to the following properties of LGI laws:

- $\mathcal{L}$ only regulates local events at individual agents,
- the ruling of $\mathcal{L}$ for an event `e` at agent `x` depends only on `e` and the local control-state $\mathcal{CS}_x$ of `x`.
- The ruling of $\mathcal{L}$ at `x` can mandate only local operations to be carried out at `x`, such as an update of $\mathcal{CS}_x$, the forwarding of a message from `x` to some other agent, and the imposition of an obligation on `x`.

The fact that the same law is enforced at all agents of a community gives LGI its necessary global scope, establishing a *common* set of ground rules for all members of $\mathcal{C}$ and providing them with the ability to trust each other, in spite of the heterogeneity of the community. And the locality of law enforcement enables LGI to scale with community size.

"intelligence" nor mobility, although neither of these is being ruled out by this model.

Operations on the control-state	
`t@CS`	returns true if term `t` is present in the control state, and fails otherwise
`+t`	adds term `t` to the control state;
`-t`	removes term `t` from the control state;
Operations on messages	
forward(x,m,y)	sends message `m` from `x` to `y`; triggers at `y` an `arrived(x,m,y)` event
deliver(x,m,y)	delivers the message `m` from `x` to `y`
Miscellaneous	
`t@L`	returns true if term `t` is present in list L, and fails otherwise
imposeObligation(oType,dt)	causes the triggering of an `obligationDue(oType)` event after time interval `dt`.

Figure 1: Some primitive operations in LGI.

On the Structure and Formulation of Laws: Abstractly speaking, the law of a community is a function that returns a *ruling* for any possible regulated event that might occur at any one of its members. The ruling returned by the law is a possibly empty sequence of primitive operations, which is to be carried out locally at the location of the event from which the ruling was derived (called the *home* of the event). (By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.) Such a function can be expressed in many languages. Our middleware currently provides two languages for writing laws: Java, and a somewhat restricted version of Prolog [2]. We employ prolog in this paper.

When prolog is employed for this purpose, an LGI law is defined by means of a Prolog-like program `L` which, when presented with a goal `e`, representing a regulated-event at a given agent `x`, is evaluated in the context of the control-state of this agent, producing the list of primitive-operations representing the ruling of the law for this event. In addition to the standard types of Prolog goals, the body of a rule may contain two distinguished types of goals that have special roles to play in the interpretation of the law. These are the *sensor-goals*, which allow the law to "sense" the control-state of the home agent, and the *do-goals* that contribute to the ruling of the law. A *sensor-goal* has the form `t@CS`, where `t` is any Prolog term. It attempts to unify `t` with each term in the control-state of the home agent. A *do-goal* has the form `do(p)`, where `p` is one of the above mentioned primitive-operations. It appends the term `p` to the ruling of the law. A sample of primitive operations is presented in Figure 1.

The Concept of Enforced Obligation: Informally speaking, an obligation under LGI is a kind of *motive force*. Once an obligation is imposed on an agent—generally, as part of the ruling of the law for some event at it—it ensures that a certain action (called *sanction*) is carried out at this agent, at a specified time in the future, when the obligation is said to *come due*, and provided that certain conditions on the control state of the agent are satisfied at that time. The circumstances under which an agent may incur an obligation, the treatment of pending obligations, and the nature of the sanctions, are all governed by the law of the community.

Specifically, an obligation can be imposed on a given agent `x` at time `t0` by the execution at `x` of a primitive operation

```
imposeObligation(oType,dt),
```

where `dt` is the time period, after which the obligation is to come due (`dt` is specified as a pair `[n,u]`, where`n` is an integer and `u` is a unit of time, such as "second" or "hour"), and `oType`—the *obligation type*—is a term that identifies this obligation (not necessarily in a unique way). The main effect of this operation is that unless the specified obligation is *repealed* before its due time t=t_0+dt, the *regulated event*

```
obligationDue(oType)
```

would occur at agent `x` at time `t`. The occurrence of this event would cause the controller to carry out the ruling of the law for this event; this ruling is, thus, the *sanction* for this obligation. Note that a pending obligation incurred by agent `x` can be *repealed* before its due time by means of the primitive operation

```
repealObligation(oType)
```

carried out at `x`, as part of a ruling of some event. (This operation actually repeals *all* pending obligations of type `oType`).

For example, under law $\mathcal{OC}$, when an seller `d` starts an auctions, an obligation `expired(timeout(P))` is imposed on `d`, to come due at the expiration time of this auctions. When this obligation comes due, it will cause `d` to declare the current winner of the auction.

Note that there is a significant difference between this concept, and the concept of obligation under *deontic logic* [8], used for the specification of normative systems. The obligations of deontic logic allow one to reason about *what an agent must do*, but they provide no means for ensuring that what needs to be done will actually be done [14].

2.2 The Law-Enforcement Mechanism

We start with an observations about the term "enforcement," as used here. We do not propose to coerce any agent to exchange $\mathcal{L}$-messages under any given law $\mathcal{L}$, just as we cannot coerce doctors to issue their orders via a computer rather than via pen and paper. The role of enforcement here is merely to ensure that *any exchange of $\mathcal{L}$-messages, once undertaken, conforms to law $\mathcal{L}$*. More specifically, our enforcement mechanism is designed to ensure the following properties: (a) the sending and receiving of $\mathcal{L}$-messages conforms to law $\mathcal{L}$; and (b) a message received under law $\mathcal{L}$ has been sent under the same law (i.e., it is not possible to forge $\mathcal{L}$-messages).

Since we do not compel anybody to operate under any particular law, or to use LGI, for that matter, how can we be sure that all auction messages in a decentralized auction are entered under law $\mathcal{OC}$? The answer is that an agent may be *effectively compelled* to exchange $\mathcal{L}$-messages, if he needs to use services provided only under this law, or to interact with agents operating under it. For instance, if the sellers accept only $\mathcal{OC}$-messages, then anybody needing their services would be compelled to send $\mathcal{OC}$-messages to them.

Distributed Law-Enforcement: Broadly speaking, the law $\mathcal{L}$ of community $\mathcal{C}$ is enforced by a set of trusted agents called *controllers*, that mediate the exchange of $\mathcal{L}$-messages between members of $\mathcal{C}$. Every member `x` of $\mathcal{C}$ has a controller $\mathcal{T}_x$ assigned to it ($\mathcal{T}$ here stands for "trusted agent") which maintains the control-state $\mathcal{CS}_x$ of its client `x`. And all these controllers, which are logically placed between the members of $\mathcal{C}$ and the communications medium (as illustrated in Figure 2) carry the *same law* $\mathcal{L}$. Every exchange between a pair of agents `x` and `y` is thus mediated by *their* controllers $\mathcal{T}_x$ and $\mathcal{T}_y$, so that this enforcement is inherently decentralized. Although several agents can share a single controller, if such sharing is desired. (The efficiency of this mechanism, and its scalability, are discussed in [12].)

Controllers are *generic*, and can interpret and enforce any well formed law. A controller operates as an independent process, and it may be placed on any machine, anywhere in the network. We

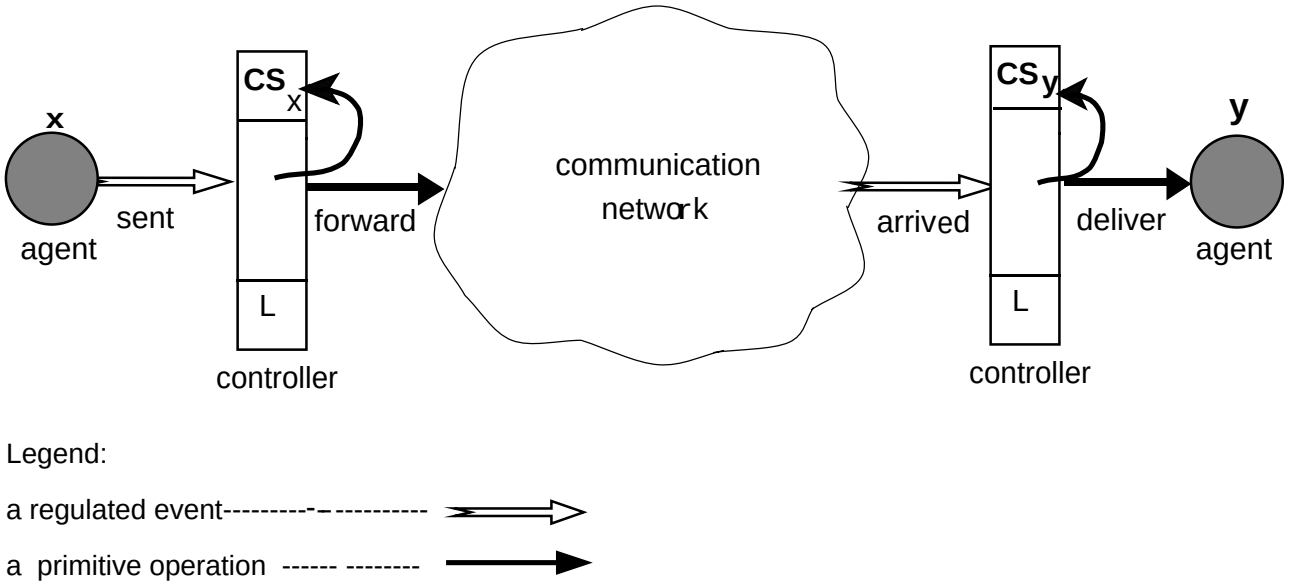

Figure 2: Enforcement of the law

have implemented a *controller-service*, which maintains a set of active controllers. To be effective in a widely distributed enterprise, this set of controllers need to be highly available, and well dispersed geographically, so that it would be possible to find controllers that are reasonably close to their prospective clients. One can envision such a service being provided by governmental agency, such as the US Post Office, or by a commercial company (or several of them), which charges for the use of its controllers, and assumes a degree of responsibility for their correctness, and a degree of liability for damages that might be caused by a malfunctioning controller.

On the basis for trust between members of a community: For a members of an $\mathcal{L}$-community to trust its interlocutors to observe the same law, one needs the following assurances: (a) that the exchange of $\mathcal{L}$-messages is mediated by controllers interpreting the *same law* $\mathcal{L}$; and (b) that all these controllers are *correctly implemented*. If these two conditions are satisfied, then it follows that if `y` receives an $\mathcal{L}$-message from some `x`, this message must have been sent as an $\mathcal{L}$-message; in other words, that $\mathcal{L}$-messages cannot be forged.

To ensure that a message forwarded by a controller $\mathcal{T}_x$ under law $\mathcal{L}$ would be handled by another controller $\mathcal{T}_y$ operating under the *same* law, $\mathcal{T}_x$ appends a one-way hash [15] `H` of law $\mathcal{L}$ to the message it forwards to $\mathcal{T}_y$. $\mathcal{T}_y$ would accept this as a valid $\mathcal{L}$-message under $\mathcal{L}$ if and only if `H` is identical to the hash of its own law.

With respect to the correctness of the controllers, if an agent is not concerned with malicious violations, then it can trust a controller provided by our controller-naming service, or a controller provided by the operating system – just like we often trust various standard services on the Internet, such as TCP/IP protocols. When malicious violations are a concern, however, the validity of controllers and of the host on which they operate needs to be certified. In this case, the controller-naming service needs to operate as a *certification authority* for controllers. Furthermore, messages sent across the network must be digitally signed by the sending controller, and the signature must be verified by the receiving controller, allowing the two controllers to trust each other. Such secure inter-controller interaction has been implemented in Moses ([10]).

2.3 The Treatment of Certificates under LGI

Under LGI, *all agents are made equal* at the time they join an $\mathcal{L}$-community. This is because the control-state of all new members is identical—and control-states provide the only means for a law

to make distinctions between agents. We now explain how an agent can acquire extra privileges, thus becoming *more equal than others* (with apologies to George Orwell), by submitting appropriate certificates.

The submission by an agent x, operating under law $\mathcal{L}$, of a certificate `Cert` to its controller, has the following effect: An attempt is made to confirm that `Cert` is a valid certificate, duly signed by an authority that is acceptable to law $\mathcal{L}$, i.e., an authority that is represented by one of the `authority-clauses` in the preamble to the law (See Figure 6 for an example). If this attempt is successful[3], then a *certified-event* is triggered. This event, which is one of the *regulated-events* under LGI, has as its argument the following representation of the submitted certificate:

```
[issuer(I), subject(S), attributes(A)].
```

Here `I` and `S` are internal representations of the public-keys of the CA that issued this certificate, and of its subject, respectively; and `A` is what is being certified about the subject. Structurally, `A` is a list of `attribute(value)` terms. For example, the attributes of a certificate might be the list `[name(johnDoe), role(seller)]`, asserting that the name of the subject in question is JohnDoe and his role in this community is a doctor. Additional components of the attributes field include the expiration time of the certificate, the URL of the server that maintains CRLs for this type of certificates, a certificate id (used to identify it in CRLs), etc. (Currently we support SPKI format of certificates [3]).

What happens when the `certified` event is triggered depends, of course, on the law. In the case of law $\mathcal{OC}$ of Figure 6, for example, the following would happen when a seller-certificate is presented, triggering the certified event: (a) the term `role(seller)` is set in the control-state of the agent in question, and (b) an obligation is imposed to deal with the eventual expiration of this certificate.

2.4 Engaging in an $\mathcal{L}$-Community

For an agent x to be able to exchange $\mathcal{L}$-messages with other members of an $\mathcal{L}$-community, it must: (a) find an LGI controller, and (b) notify this controller that it wants to use it, under law $\mathcal{L}$. We will discuss these two steps below.

Locating an LGI Controller: As already discussed, the Moses middleware includes a controller-service, which can be used to maintain a set of active controllers. This server provides the address (host and port) of the available controllers to any agent that wishes to engage in LGI. One may have any number of such servers so that controllers can be distributed in different regions of the Internet. Efficiency-wise, x would do best by selecting a controller closest to it (to minimize the overhead of forwarding $\mathcal{L}$-messages through the controller). But functionally, one is free to choose a controller anywhere on the Internet, and several agents may share a single controller, without knowing of each other. In Figure 3, we present a simple example of such an organization, as well as possible ways of interacting with it.

For simplicity, In Figure 3, we only presented one cluster of controllers, maintained by one organization. This is not mandatory. Under our architecture multiple such clusters can exist which can be maintained by one or multiple organizations. In order for agent x to communicate with agent y, x (and y) have first to locate an available controller. This is done by contacting the controller-server and obtaining from it the address of a controller (messages 1 and 2, respectively 1' and 2', in Figure 3). Then x needs to find out the address of y (message 3). In our example of law-governed auctions, the seller is publishing its own address so that the buyers are able to contact it. In other applications, discovering the address of an agent can be done either by using centralized name servers or by any other means of communication (like telephone or e-mail messages).

After x knows the address of y, every message that x sends to y (message 4 in Figure 3) goes first to x's controller, then to y's controller and finally, if the law allows, delivered to y.

[3]If the the certificate is found invalid then an *exception-event* is triggered.

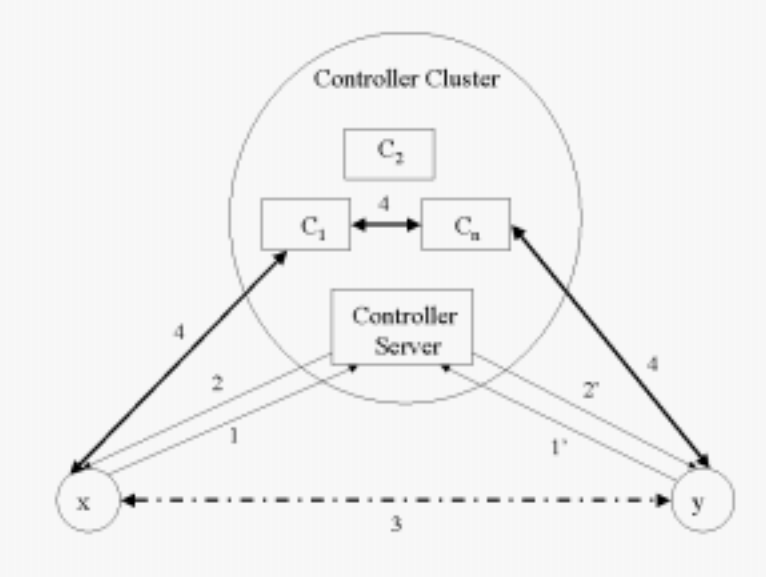

Figure 3: Deployment of an $\mathcal{L}$-Group

Adopting a law: Upon selecting a controller C, x would send C the message

```
adopt(law,name),
```

where `law` is the law that it wants to adopt, and `name` is the name that it wants to be known by. The argument `law` can take the form of either the text of the law to be adopted or the name of such a law, given to it by a specified *law-repository* service, which is another tool provided by Moses—we will not discuss here the details of this service but rather assume that the text of the entire law is always passed to the controller.

When controller C receives the `adopt` message, it checks the supplied law for syntactic validity, and the chosen name for uniqueness among the names of all current agents handled by C. If these two conditions are satisfied, and if C is not already loaded to capacity, it will set up an initial control-state for agent x, as specified in the preamble of the law adopted by x, allowing x to start operating under this law[4].

3 Decentralized Auctions and Scenarios of Use

In this section we present the LGI-based auctioning system. We start with the general architecture and we then describe an example law for an open-cry auction policy [5]. We also propose solutions for two different issues related to the auctioning in the Internet: auditing and treatment of complaints. We show how these issues, which can be viewed as auction parameters, are specified as part of the auction policy using LGI. The same approach is valid for other auction parameters.

3.1 LGI-based Auctioning Architecture

The three main entities in the LGI-based auctioning architecture are: auction registry, sellers, and buyers. The following paragraphs detail each of them.

- **Auction registry**. The auction registry is a separate agent that holds the selling offers as a tuple {ProductName, Description, SellerAddress, AuctionPolicy, Timeout}. Sellers can insert

[4]If any one of these conditions is not satisfied, then `x` would receive an appropriate diagnostic, and will be able to try again.

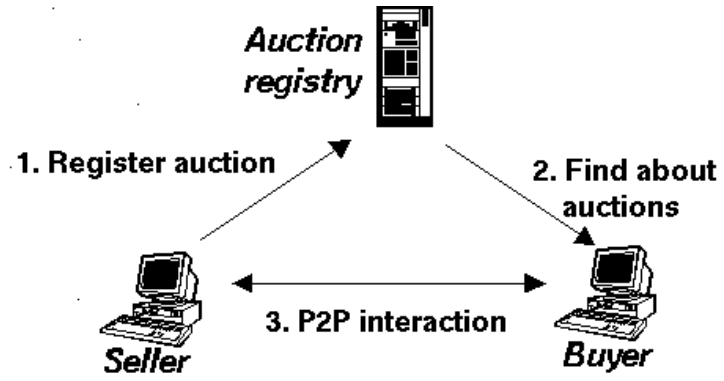

Figure 4: Interaction among sellers, buyers, and the auction registry

or delete tuples from the registry, while buyers can just query the registry about current auctions [5]. If there is no offer until the timeout for the auction expires, the registry withdraws the auction tuple.

- **Sellers and Buyers**. All the interaction between sellers and buyers is governed by LGI according to the auction policies (laws) specified in the registry tuples. We assume, for simplicity, that the actual exchange of product and money between the buyer that wins the auction and the seller is handled by the two parties involved, like in other auction systems, such as eBay (http://www.ebay.com). However, the law can also incorporate electronic methods of payment like electronic money or PayPall (http://www.paypal.com). After the auction is over, the seller deletes the associated tuple from the auction registry.

The following messages summarize the interaction between the different entities, as illustrated in Figure 4:

1 Sellers send messages to the auction registry to insert or delete auction tuples. Before a tuple is inserted in the auction registry, or after it is deleted, it is not possible for buyers to bid on the specified product.

2 Buyers make requests for offers that meet some conditions and the registry returns the list of such tuples (if any) back to the buyer. When a buyer discovers about an interesting auction, it can adopt the auction law and join the community that is conducting the auction.

3 Buyers and sellers exchange messages according to the law specified in the auction tuple. They interact directly, in a peer-to-peer communication model, meaning that there is no centralized auction room. The enforcement of the auction law is distributed, using LGI.

It is up to the agents involved to agree to participate in the auction or not, after examining the auction law. Please note that the auction registry does not necessarily belong to the community since interaction with the registry does not need to be governed by LGI. In fact, several implementations of the auction registry can be used, as long as they provide methods for registering and consulting auctions.

Other kinds of agents may also participate in the auctioning process, such as auditors and complaints agents. Figure 5 shows a snapshot of the system with two buyers and two sellers: buyer 1 interacts with seller 1 under law K and with seller 2 under law L. Buyer 2 only interacts with seller 2 under law L. Moreover, in the case of law K buyers and sellers interact with a given auditor, while

[5] In the case of reverse auctions, such as Priceline (http://www.priceline.com), the opposite situation is possible: the buyers post the auctions and the sellers query the registry for new auctions. For the sake of simplicity, in the rest of the paper our discussion does not consider reverse auctions. However, the proposed architecture can be used to model any kind online trading [5], as long as it can be represented as a law in the LGI paradigm.

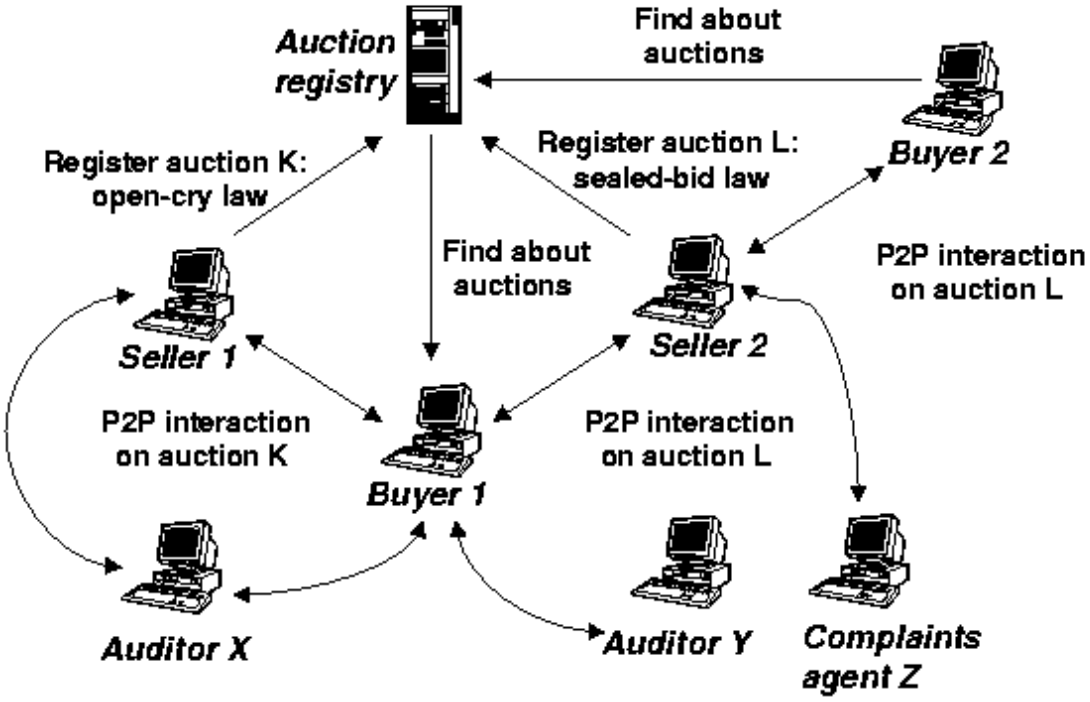

Figure 5: A snapshot of the system with 2 buyers and 2 sellers

in law L they interact with an another auditor and with a complaints agent. All interaction among buyers, sellers, auditors, and complaints agents, is peer-to-peer and enforced using LGI. Section 3.2.1 details auditioning, while Section 3.2.2 describes how complaints can be handled in the proposed architecture.

3.2 Auction Law Example

In this section we present an example of auction law (called open-cry auction [5]) written using LGI. The main idea is that buyers compete with each other to purchase the specified product. At each moment, a buyer can either make a bid or ask for the highest bid so far. If the bid value is greater than the current maximum value, this value is stored at the seller as the maximum current bid. If not, the bid is rejected and the current maximum value is sent back to the buyer. When a higher bid is accepted, the buyer that had the previous highest bid is notified that it was out-bided. At the end of the auction period the seller notifies the winning buyer that he or she actually won the auction. In this way, all of the involved buyers will be notified if they won the auction (by receiving a succeeded message) or if they were out-bided. The law is presented in Figures 6 and 7. The rules of the law are followed by comments (in italic) that, together with the following discussion, should provide the reader with some understanding of the nature of LGI laws. Using the techniques described in this paper, one can easily develop laws for different types of auctions.

The preamble of the law contains the following clauses: First there is the `law` clause that identifies the name of this law, and the public key of the CA to be used for the authentication of the controllers that are to mediate $\mathcal{OC}$-messages, as described in Section 2.4.Second, there is an `authority` clause that identifies `admin`, represented by its public key, as a CA for certifying various roles played in this community. Third, the `initialCS` clause defines the initial control-state of all agents in this community—it is empty in this case. And finally the `Auditor` specifies the auditor for this auction.

Rules $\mathcal{R}1$ and $\mathcal{R}2$ deal with the adoption of this law, allowing an agent to start operating either as

a seller or as a buyer by getting the term `seller` or `buyer` in its initial control-state. The adoption of the law is conditioned on presenting a valid certificate, previously obtained from a certifying authority. Any agent that presents a valid certificate gets the term `certified` in its control state, allowing him to participate (as a buyer or as a seller) in any auction. The certificates expire after a specified period of time (100 seconds in our example). When that happens, Rule $\mathcal{R}11$ is trigerred and the term `certified` is removed from the control state of the agent. The agent is then not able to participate in any auction until he or she presents a valid certificate to its controller. We will now discuss how a buyer can make an offer and what are the possible answers from the seller.

First, by Rule $\mathcal{R}3$, the seller actually starts the auction for the product with the name P. The maximum price is initially set to 0. An obligation is imposed in the sense that the auction should be finished after a specified period of time. Buyers can now send offers using Rule $\mathcal{R}4$. According to the rule $\mathcal{R}5$, the bid is checked if it is the best offer. If it is, the maximum price is set to this value and the buyer that had previously the best offer is informed that his offer was out-bided triggering the execution of Rule $\mathcal{R}10$. If it not the best offer, according to Rule $\mathcal{R}6$, the buyer is informed that his or hers bid was rejected and the execution of Rule $\mathcal{R}9$ is triggered. When the auction is finished (rule $\mathcal{R}7$) the seller informs the winning buyer through Rule $\mathcal{R}8$.

We also try to minimize the possibility of a seller to act as a buyer in the same auction to artificially increase the price. When an offer is received by the seller (in rules $\mathcal{R}5$ and $\mathcal{R}6$), the law checks whether the seller is also a buyer or not.

Agents have to exchange messages using the format required by the law. The current implementation of LGI provides an XML interface, allowing agents to communicate using XML as the main language for describing the data. In this case, the law plays also the role of the schema for the XML documents that are exchanged between the agents.

3.2.1 Auditing

The LGI paradigm naturally supports the concept of auditing. An auditor is basically an agent that is not involved in the auction but that receives copies of the messages that were exchanged. If an auditor exists, an agent can request copies of the messages exchanged during the auction and use this information to find out about the behavior of some of the involved agents. An auction can have more than one auditor and the auction law specifies the messages that are sent to each of them. An agent can choose not to participate in an auction if it does not trust its auditors. Although the law specifies what messages are sent to the auditors, it imposes no restrictions in the way they handle the messages they receive.

3.2.2 Treatment of Complaints

An agent can complain about another agent (A) if he or she thinks that A did not have a correct behavior. Examples of incorrect behavior include: not sending the item once the auction is over, sending a defect product, not sending the payment, and so on. To handle these situations an auction law can specify a complaints agent, which is an analogue of the Better Business Bureau from the real life. A complaints agent interacts with the auditors and with the certifying authorities to get copies of the messages exchanged and the IDs of the involved agents. By processing this information, it can check if a given complaint is accurate or not and it can inform the certifying authorities about the improper behavior of agents.

An example of how a complaints agent can be used to increase the trust of agents in the correctness of the auction is the prevention of the artificial increase of the price by the seller. As we detailed in the previous section, the law can check if the seller is registered at the same controller as a seller and as a buyer. However, the law cannot check whether the seller registered himself as a buyer on another controller and participates in the auction as a regular buyer. Moreover, it might happen that "friends" of the seller participate in the auction to artificially increase the price.

Once a complaint is filed, the complaints agent can talk to the auditor to retrieve copies of all the exchanged messages and the real IDs (as are written in the certificates) of the agents. With this information at hand, it can find out about the malicious behavior of such a seller. When the seller

$\mathcal{P}reamble$: law(name(oc),ca(publicKeyOfCAuth)).
authority(admin,URL(http://aramis.cs.rutgers.edu:9020))
initialCS([])
alias(auditor, auditor@enterprise.com)

$\mathcal{R}1$.
```
certified(X,certificate(issuer(admin),subject(Y),attributes([seller(N)])))
      :- do(deliver(X,certificate(issuer(admin),subject(Y),attributes([seller(N)])),
         X)), do(+certified),do(+role(seller)), repealObligation(endCertified(X)),
         imposeObligation(endCertified(X),100),
         do(deliver(X,attributes([seller(N)],auditor).
```

An agent can participate in the auction as a seller —i.e., having a term `role(seller)` *in its control state. In order to participate in the auction it has to present a valid certificate. An obligation is imposed to limit the duration of the validity of the certificate (in this case the certificate is valid for 100s). The content of the certificate, along with the seller name, is also sent to the auditor.*

$\mathcal{R}2$.
```
certified(X,certificate(issuer(admin),subject(Y),attributes([buyer(N)])))
      :- do(deliver(X,certificate(issuer(admin),subject(Y),attributes([buyer(N)])),
         X)), do(+certified),do(+role(buyer)), repealObligation(endCertified(X)),
         imposeObligation(endCertified(X),100),
         do(deliver(X,attributes([buyer(N)],auditor).
```

An agent can participate in the auction as a buyer —i.e., having a term `role(buyer)` *in its control state. Like in the case of the seller, an obligation is imposed to limit the duration of the validity of the certificate (the certificate is valid for 100s) and the content of the certificate, along with the buyer name, is also sent to the auditor.*

$\mathcal{R}3$.
```
sent(X,start(P,T),X)
      :- certified@CS, role(seller)@CS, do(+P),do(+max(P,0)),do(+winner(P,X)),
         do(imposeObligation(timeout(P),T)),do(deliver(X,start(P,T),auditor).
```

Only a seller can start the auction for the product P. From this moment on, offers are received and an obligation is imposed that the auction should stop after the specified period of time. The tuple corresponding to this product should have been already placed in the auction registry, otherwise the buyers cannot find about this auction. A copy of the message is sent to the auditor.

$\mathcal{R}4$.
```
sent(X,offer(P,M),Y) :- certified@CS, role(buyer)@CS, do(forward(X,offer(P,M),Y)),
         do(deliver(X,offer(P,M,Y),auditor).
```

The offer can be made by the buyer only if he or she has been certified. A copy of the message is also sent to the auditor.

Figure 6: Law $\mathcal{OC}$ for the Open-Cry Auction Part 1

certificate expires and he or she requests a new one from the certifying authority (CA), the CA can ask the complaints agent about the seller behavior and it can refuse to issue the new certificate.

4 Related Work

Currently several Web sites provide auction services. These sites can be classified into two categories: consumer-to-consumer (C2C) and business-to-consumer (B2C). In C2C auctions, such as Ebay (http://www.ebay.com), users can act as both sellers and buyers, posting new auctions and participating in existing auctions. In B2C auctions, such as Egghead (http://www.egghead.com), users act only as buyers, participating in the available auctions. B2C auctions are less "risky" for buyers, since the seller is usually a "brand name" company that can be trusted. In C2C auctions, although the action site acts as a mediator, there is no guarantee about the reputation of the seller. As discussed throughout the paper, centralized auction systems cannot solve this issue since they do not allow auction participants to engage the entities they trust as part of the auction process.

$\mathcal{R}5$. `arrived(X,offer(P,M),Y)`
`:- role(seller)@CS, max(P,Q)@CS,winner(P,Z)@CS, M>Q, not role(buyer)@CS,`
`do(-max(P,Q)), do(+max(P,M)), do(-winner(P,Z)), do(+winner(P,X)),`
`do(forward(Y,accepted(P,M),X)), do(deliver(Y,accepted(P,T,X),auditor),`
`do(forward(Y,outbid(P,M),Z)), do(deliver(Y,outbid(P,T,Z),auditor).`

If the offer is the best offer, it is recorded as the max bid for the item and the buyer becomes the current winner of the auction. We do not allow for the seller to bid on the same item, in order to prevent artificial increase of the price. Copies of the accepted and out-bid messages are also sent to the auditor.

$\mathcal{R}6$. `arrived(X,offer(P,M),Y)`
`:- role(seller)@CS, max(P,Q)@CS,winner(O,Z)@CS, not role(buyer)@CS, Q >= M,`
`do(forward(Y,rejected(P,Q),X)), do(deliver(Y,rejected(P,M,X),auditor).`

If the offer is not the best offer, a message is sent to the buyer indicating that his or hers bid was rejected and informing value of the current maximum bid. We do not allow for the seller to bid on the same item, in order to prevent artificial increase of the price. A copy of the message is also sent to the auditor.

$\mathcal{R}7$. `obligationDue(timeout(P)) :- max(P,M)@CS,M>0,winner(P,X)@CS, do(-P),`
`do(forward(Self,succeeded(P,M),X)), do(deliver(Self,winner(P,M,X),Self)),`
`do(deliver(Self,succeeded(P,M,X),auditor).`

When the auction is finished, the buyer with the biggest offer receives the succeeded message. This message also indicates the amount he or she has to pay. A copy of the message is also sent to the auditor.

$\mathcal{R}8$. `arrived(X,succeeded(P,M),Y) :- role(buyer)@CS, do(deliver).`

The buyer that receives the succeeded message is the winner of the auction. The actual exchange of money and products is done by the two parties independently of LGI.

$\mathcal{R}9$. `arrived(X,rejected(P,M),Y) :- role(buyer)@CS, do(deliver).`

The buyer is notified that his or hers bid was not accepted.

$\mathcal{R}10$. `arrived(X,outbid(P,M),Y) :- role(buyer)@CS, do(deliver).`

The buyer is notified that he or she was out-bided.

$\mathcal{R}11$. `obligationDue(endCertified(X)) :- do(-certified).`

The agent certificate expired without a replacement. When this happens, sellers are not able to start a new auctions (the current ones are not affected) and buyers are not able to send new offers.

Figure 7: Law $\mathcal{OC}$ for the Open-Cry Auction Part 2

The AuctionBot system [17] allows users to customize auction policies. The system is centralized, having a very similar structure to eBay (http://www.ebay.com) or other commercial systems. The main advantage of AuctionBot is that the user can parameterize the pre-defined auction policies (like dutch, sealed) to choose, for example, between the first prize or the second prize as the auction winner. The marketplace framework described in [5] also allows the definition of several auction and negotiation protocols. However, both systems do not support distributed enforcement and rely on a centralized marketplace, limiting the customization options of the auction participants.

The auction registry component of the proposed LGI-based architecture can be view as a yellow pages or discovery service, where buyers search for auctions. UDDI (Universal Description, Discovery and Integration) can be used to standardize the auction registry interface. UDDI is an "open industry initiative enabling businesses to (i) discover each other, and (ii) define how they interact over the Internet and share information in a global registry architecture" (http://www.uddi.org).

5 Conclusions and Future Work

Online auctions is an important application area - Forecast Research expects that in 2003 there will be a market of 14 million consumers and $19 billion in sales. In this paper we presented a novel system for supporting online auctions that takes full advantage of the distributed nature of the Internet.

In the proposed architecture, agents can setup their own auction policies and these policies are explicitly enforced using the LGI paradigm. Auctions are conducted in a totally distributed manner, through a peer-to-peer communication protocol among the several agents. There is no centralized authority that can act as a trusted mediator. However, we have shown how third parties, such as auditors and complaints agents, can participate on the auctioning process under a given law. Currently, none of the online auction services follow a similar architecture. Moreover, this architecture is not limited to auctions, but it can be applied to any online trading model.

We are currently working on the definition of laws for other types of negotiation. We are especially interested in studying the behavior of agents in the presence of several optional (and conflicting) laws. Another topic we plan to investigate is the integration of our system with the Web services paradigm, in which agents find out about each other using XML-based discovery mechanisms, such as UDDI (http://www.uddi.org), and exchange messages using XML and SOAP. We are particularly interested in investigating the relationship between laws and Web service description languages, such as WSDL (http://www.uddi.org). Another point of extension that we are currently working on is the development of a Web-based system for the definition of auction policies in a simple and straightforward manner. This interface allows users to define new laws visually, based on previously defined ones.

References

[1] X. Ao, N. Minsky, T. Nguyen, and V. Ungureanu. Law-governed communities over the internet. In *Proc. of Fourth International Conference on Coordination Models and Languages; Limassol, Cyprus; LNCS 1906*, pages 133–147, September 2000.

[2] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[3] C. Ellison. The nature of a usable pki. *Computer Networks*, (31):823–830, November 1999.

[4] Ralph Cassady Jr. *Auctions and Auctioneering*. Univ. California Press, 1979.

[5] M. Kumar and S. Feldman. Internet auctions. In *Fifth USENIX Workshop on Electronic Commerce*, August 1998.

[6] H. G. Lee. Do electronic market marketplaces lower the price of goods. *Communications of the ACM*, 41(1):73–80, January 1998.

[7] H. G. Lee and T. H. Clark. Impact of the electronic marketplace on transaction cost and market structure. *International Journal of Electronic Commerce*, 1(1):127–149, Fall 1996.

[8] J. J. Ch. Meyer, R. J. Wieringa, and Dignum F.P.M. The role of deontic logic in the specification of information systems. In J. Chomicki and G. Saake, editors, *Logic for Databases and Information Systems*. Kluwer, 1998.

[9] N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.

[10] N.H. Minsky and V. Ungureanu. A mechanism for establishing policies for electronic commerce. In *The 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 322–331, May 1998.

[11] N.H. Minsky and V. Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *7th USENIX Security Symposium*, January 1998.

[12] N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *TOSEM, ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.

[13] N.H. Minsky, V. Ungureanu, W. Wang, and J. Zhang. Building reconfiguration primitives into the law of a system. In *Proc. of the Third International Conference on Configurable Distributed Systems (ICCDS'96)*, March 1996. (available through `http://www.cs.rutgers.edu/~minsky/`).

[14] M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.

[15] B. Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.

[16] V. Ungureanu and N.H. Minsky. Establishing business rules for inter-enterprise electronic commerce. In *Proc. of the 14th International Symposium on DIStributed Computing (DISC 2000); Toledo, Spain; LNCS 1914*, pages 179–193, October 2000.

[17] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 301–308, New York, 9–13, 1998. ACM Press.