Towards robust multi-agent systems: Handling communication exceptions in double auctions

Simon Parsons[1,2] and Mark Klein[2]

[1] Dept of Computer and Information Science
Brooklyn College, City University of New York
Brooklyn, NY 11210, USA.
parsons@sci.brooklyn.cuny.edu

[2] Center for Coordination Science
Massachusetts Institute of Technology
Cambridge, MA 02142, USA
m_klein@mit.edu

Abstract. This paper addresses an important question in the development of multi-agent systems—how can we create robust systems out of the often unreliable agents and infrastructures we can expect to find in an open systems' context? Here we examine an approach based on distinct exception handling services, and apply it to systems performing resource allocation by means of a double auction. The exception handling system provides protocol-specific but domain independent strategies for monitoring the auction, and for ameliorating problems when they occur. We describe a number of experiments that suggest the exception handling approach works well for various kinds of message loss in double auctions.

1 Introduction

This paper studies the following question. "How can we develop robust multi-agent systems from the kind of unreliable agents and infrastructures—whether buggy, malicious, or just dumb—we can expect to have to deal with in the context of open systems?" This is an increasingly important question because of the emerging changes in the way that human organizations work.

One result of globalization, coupled with the increasing power and ubiquity of cheap telecommunications, is that organizations are under increasing pressure to re-configure within short time-frames. This can have the effect of bringing together partners who have never worked together before, and force these partners to make their infrastructure inter-operate in ways that it was never designed to. Examples of this requirement can be found in military coalitions, disaster recovery operations, open electronic marketplaces and virtual supply chains [7, 35, 37]. One way to deal with the challenge of enabling this interoperation is to build the infrastructure as a multi-agent system, and benefit from the ability of such systems to dynamically self-organize as their tasks and constituents change [17]. However, a critical problem remains.

Much of the work in multi-agent systems has considered *closed* systems in which well-behaved agents have run on reliable infrastructures in relatively simple domains

[13]. Both agents and infrastructure have been developed for a specific multi-agent system, and have been engineered to work together. These assumptions do not hold for the *open* systems described above, where agents may have been developed by many different organizations and must be able to operate on whatever infrastructure is provided.

For open contexts, we can expect to have to deal with the following problems:

– *Unreliable infrastructure* In large distributed systems like the Internet, unpredictable host and communication problems can cause agents to slow down or die unexpectedly, and messages to be delayed, garbled or lost. These problems become worse as the applications increase in size, because of the growth in possible points of failure.
– *Non-compliant agents* In open systems, agents are developed independently, come and go freely, and cannot always be trusted to follow the rules properly due to bugs or even outright malice. This can be expected to be especially prevalent and important in e-commerce or military scenarios where the incentives for fraud or malice can be considerable[1].
– *Emergent dysfunction* In large, multi-agent systems interactions between agents are complex and using the kinds of coordination mechanisms that have proved popular can lead to chaotic behavior and other emergent dysfunctions [4, 39].

These problems all give rise to *exceptions*, situations which fall outside the normal operating conditions of the multi-agent system.

This paper, building on previous work by the second author [24], considers the first of these problems in the context of agents engaged in resource allocation using a double auction. This is a scenario that one can imagine easily arising in an open electronic market or supply chain. We focus on the specific matters of message loss and corruption, complementing earlier work by the second author on agent death [24].

2 Exception handling

Now, one way to deal with exceptions is to elaborate the individual agents so that they are able to cope with all the exceptions that they might face. Most previous research on dealing with exceptions has taken this approach. For example the contract net [33] includes an "immediate response bid", which allows an agent to determine whether receiving no response to its request for bids is due to all eligible sub-contractors being busy (in which case a retry is appropriate) or due to the outright lack of subcontractors with the necessary skills (in which case some other action needs to be taken). This *survivalist* approach to exception handling faces a number of serious shortcomings.

First, developing survivalist agents greatly increases the burden on agent developers. For this to be an effective approach, all the agents have to be carefully coordinated and provided with potentially complex mechanisms for exception handling. Agent developers have to anticipate and correctly prepare for all exceptions an agent may face in any environment it may have to operate in. Changing existing exception handling techniques is equally hard, since it requires coordinated changes across many agents built

[1] Note that "fraud" here is meant in terms of collusive behavior rather than deviation from bidding at one's private value for a good—in other words in the sense that is usually not considered in market design.

by many developers, and in general agents become harder to maintain, understand, and reuse.

Second, the survivalist approach can lead to poor exception handling. In open systems it is always possible that some agents won't have the necessary exception handling code, or may violate some of the assumptions built into the exception handling operated by others. Agents might not, for example, meet the assumption that they are fully rational [31]—they may be buggy, or too computationally limited. In addition, the best interventions—like killing an agent that is broken—might not be easily implemented because agents do not have the necessary authority, while detecting emergent dysfunctions can be a problem without a global view of the system—something that it is hard for individual agents to acquire without heavy bandwidth requirements.

In order to overcome these limitations Klein *et al.* [24] suggested attaining robustness by off-loading exception handling to distinct domain-independent services. We refer to this as the *citizen* approach, by analogy with the way that exceptions are handled in human society. Citizens of such societies typically adopt relatively simple and optimistic rules of behavior, and rely on a range of social institutions (law enforcement, the legal system, disaster relief agencies, the UN, and so on) to handle most of the exceptions that arise. This results in generally better handling of exceptions than individual citizens can manage—because the exception handling institutions are specialised, widely accepted as legitimate, and benefit from economies of scale—while placing few demands upon them—like paying taxes and reporting crimes.

The key insight in the citizen approach is that highly reusable and *domain independent* exception handling expertise can be separated from the knowledge that agents use to achieve their main tasks. There is considerable support for the validity of this idea. In the expert systems field there is evidence that it is useful to separate domain-specific knowledge from generic control information [2, 12], and that the same is true in collaborative design conflict management [21] and managing exceptions in workflow applications [22]. Previous work on the citizen approach has found that every coordination protocol has its own set of domain-independent exceptions, and that these can be turned into domain-independent strategies for handling exceptions [24]. This paper extends this earlier work to a new set of coordination protocols—auction protocols—identifying a new set of exceptions and exception handling mechanisms. Due to the popularity of auctions in the agents community, we believe that these results will be interesting to a large number of agent developers.

3 Exception handling in double auctions

3.1 Exceptions and double auctions

Double auctions are markets that include both buyers and sellers. A classic example of a double auction was the trading pit at the old Chicago Board of Trade. Here buyers and sellers, or rather human agents operating on their behalf, would call out offers, *bids*—offers to buy a good at a given price—or *asks*—offers to sell a good at a given price. Although such markets have long since become electronic, the same basic principles apply with buyers and sellers "gathering" in a virtual space in which bids and asks are

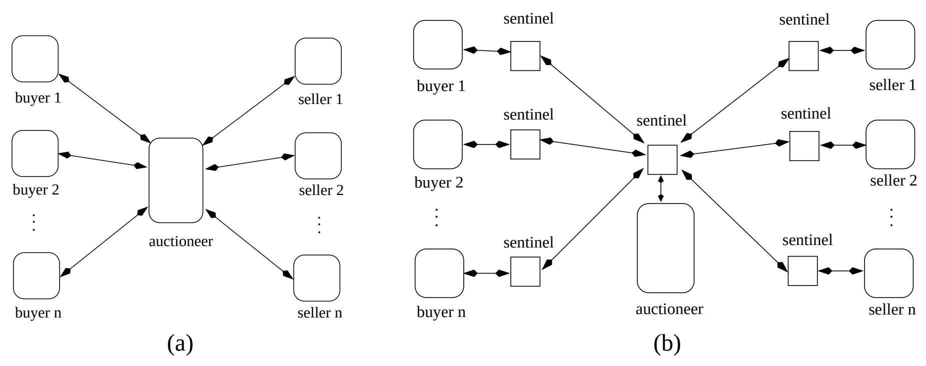


Fig. 1. Double auction archiectures, (a) basic, and (b) with exception handling facilities.

broadcast. When a bid is greater than an ask, a trade is possible, and a price between the *bid price* and the *ask price* is decided on as the *trade price*. This is a *continuous* double auction in which a trade is possible after every offer, another *periodic* variant of the double auction collects bids and asks until some deadline and then finds possible trades [8].

The wide applicability of auctions as resource-allocation mechanisms [5] and the fact that double auctions can be composed into supply chains have led to a great deal of interest in double auctions within the agents community. Indeed, following Smith [34], there has been much investigation of double auctions within both the fields of economics and computer science. This work has primarily tried to identify what makes double auctions effective [10, 25, 34], to find ways of analysing optimal behavior in a double auction [11, 32, 38], and to identify efficient bidding strategies for double auctions [5, 9, 27, 29]. The only work we are aware of on ensuring robustness in double auctions is that which looks to explain how such auctions are effective—that is provide high *allocative efficiency* and ensure prices are set close to the theoretical equilibrium—even with a small number of traders (since the underlying theory only guarantees such properties [10, 25, 34] for many traders).

As already stated, the investigation of exceptions explored in this paper concentrates on the first class of exceptions listed above—unreliable infrastructure. We considered that in practice[2] agents participating in an auction will be physically somewhat removed from the auction site and communicate with it through some form of message passing. The overall architecture of the kind of system we consider is shown in Figure 1 (a) with the auctioneer agent taken as embodying the functions of the auction market. There are several different types of message that need to be sent between auctioneer and the traders.

[2] This is in contrast to the experimental conditions used by most workers on double auctions—[36], where asynchronous communication is considered, being an honourable exception—who have run experiments in which communication between agents is considered to be synchronous, instantaneous, and completely reliable, understandably since their focus is entirely on the behavior of the bidding aspects of the agents.

- *Bid calls* Messages indicating that buyers should start bidding.
- *Ask calls* Messages indicating that sellers should start asking.
- *Bids* Offers to buy.
- *Asks* Offers to sell.
- *Quote Prices* Indicators of the price of the last trade.
- *Winner messages* Indicators that the addressee has provided a winning ask or bid.

Any of these messages can then be lost, delivered late, or corrupted, and these are exactly the exceptions that we consider in this paper.

To provide a citizen approach to exception handling we also define an exception handling infrastructure, which, following the approach in [24], associates a *sentinel* with every agent, resulting in a system like that in Figure 1 (b). These sentinels can then provide exception handling services. For example, a sentinel for a trader can identify the loss of a bid or ask call message intended for that trader (which would otherwise shut that trader out of the auction) by spotting messages (like quote prices) that indicate the auction is in progress. It can then work with the auctioneer sentinel to ensure that its agent is included in the auction by having the necessary call message be resent.

Note that in order to provide this kind of exception handling service, the sentinel need have no access to the internal state of the agent it is associated with. Indeed, the essence of the approach is that it does not have such access. In our first example, the sentinel needs only to know the type of the messages being transmitted, and in the second need only perform a parity check. Other exception handling services (such as detecting fraudulent behavior on the part of the auctioneer) may require sentinels to "look inside" messages, but to do this the sentinel has no more access to information than, for example, the auctioneer does.

The advantage of the citizen approach is that the mechanisms for detecting and resolving the exceptions, the exception *handlers*, are generic. Exactly the same mechanisms can be used for other classes of auction since (as described elsewhere [26]) the specific exceptions that are detected and resolved by for a double auction may be found across all kinds of auction, and so may be handled by the same mechanisms. Indeed, these kinds of exception—exceptions due to message delay, loss and corruption—will be common to all coordination mechanisms operating over unreliable infrastructures, and potentially the same handlers can be used for a wide range of multi-agent systems.

3.2 A general approach to exception handling

The way that we have built the auction exception handlers explicitly acknowledges this. Klein's previous work has described how knowledge about multi-agent coordination mechanisms can be described in the framework of the MIT Process Handbook[3], a repository of taxonomic information about general business processes—the tasks carried out and the exceptions that may occur. Building on this we have added knowledge of mechanisms for detecting and resolving exceptions in coordination mechanisms in this same framework, and furthermore have added knowledge about the specific exception handling mechanisms discussed here. The Process Handbook does more than

[3] `http://ccs.mit.edu/ph/`

provide abstract knowledge of the different process components and capture the relationship between them. It plays an active role in handling exceptions.

In essence we use the Process Handbook to perform model-based diagnosis when we encounter an exception. The Handbook itself is the model. Thus, when we spot an exception, we can use the Handbook to relate it to the task that the exception is an exception to, and hence to the right mechanism for dealing with the exception (the precise approach we use is detailed in [23]). In fact, in the segment of the Process Handbook that we built during our work on auctions, we even included the code that detects—the *detection handlers*—and the code that resolves—the *resolution handlers*—the exceptions in the Handbook. When the sentinels are created, they are informed of the kinds of exception that they should be trying to detect. They use the Handbook to identify what detection handlers should be used to do this for the specific kind of coordination mechanisms they are surveying, and then load the detection handlers. When the detection handlers first encounter a specific kind of exception, they use the Handbook to locate the correct resolution handler to resolve the exception, and then load and run it.

This method adds further generality to the citizen approach to exception handling. In our experiments the taxonomy was included as part of the local system on which all the agents were running, but it could equally well be in some remote location. This offers a number of advantages. The main advantage is that it makes the development of sentinels very easy. They do not need to be programmed with any exception handlers. They just have to be programmed with the knowledge that they need to load and apply handlers at certain points. For example, our sentinels are programmed to run any detection handlers relating to messages in every message they pass. The handlers themselves take care of knowing about, and calling, the requisite resolution handlers.

The second advantage is that sentinels (and thus the agents they survey) do need to be altered in any way when they switch between different coordination mechanisms. All they need to do is to load the new handlers. Indeed, they don't even need to know what handlers are required—this information can be stored with the handlers, and all the sentinel needs to do is to request a list of handler names for the new kind of mechanism. When the mechanism start running, the detection handlers are loaded, and when exceptions occur, resolution handlers are loaded. This lazy approach to loading the handlers then provides a third advantage. Updating handlers is straightforward. The new handler is just added to the central repository, and will be automatically uploaded into exactly the sentinel that needs it, when it is required.

All these advantages are, of course, just the classic advantages of making problem solving knowledge declarative as far as possible rather than purely procedural.

3.3 Some specific handlers

As examples of specific exception handlers, consider those that provide the basis of the experimental work in this paper.

The first pair of handlers deal with the loss of a bid call message (the loss of an ask call message can be handled in the same way, and the handlers for this are implemented). In the absence of any exception handling, the loss of a bid call means that a particular trader is effectively shut out of the auction. This may have no impact on the auction—if a buyer has a private value that is too low for it to be able to trade—but typically the loss of the bid call will cause some loss of trade.

A sentinel for a buyer can easily rectify the situation. Since many of the messages transmitted during the auction are broadcast (in the particular case on which we have been experimenting, these are just quote prices, but in many auctions bids and asks are also broadcast) these can be used as a proxy for the bid call. Since the auction must have started for these messages to be flying around, their existence can be taken as an indication that the bid call was somehow lost. It is simple to write a detection handler that spots this. The resolution handler is equally straightforward— it sends a message to the sentinel for the auctioneer, requiring it to send a further bid-call to the trader that did not receive the original message.

The second example of exception handlers, are those that deal with corrupted messages. Here we assume that messages are corrupted by some stochastic process on the communication lines between the auctioneer and the trader, and that corrupted messages are indechiperable by the traders. Under such circumstances corrupted messages will have a less severe impact than lost bid/ask calls—no trader will be shut out of the auction by such a message[4], but traders will lose valuable information like price quotes and some potentially winning bids will not be received by the autioneer. Both of these happenings may affect the efficiency of the auction.

Both the detection and resolution handlers for corrupted messages are very straightforward. The detection handler invoked by the seninel can detect the corrupted message by some simple mechanism like a parity check, and the resolution handler simply sends a message (which itself can be corrupted of course) to the sentinel of the agent from which the message was sent, asking for that message to be retransmitted.

Our current implementation also provides a detector for lost trade messages (the auctioneer sentinel receives a bid from an agent that has been sent a winner) and a resolver (the trade message is resent). The implementation also includes detectors for exceptions in auctioneer behavior—where the auctioneer matches bids and asks incorrectly, or sets the wrong price. These involve the auctioneer sentinel recording incoming bids and asks, duplicating the correct behavior of the auctioneer, and checking the outgoing messages against its own computation of which offers match. The exceptions can then be resolved by having the sentinel substituting a message based on its computation for that of the auctioneer (though this is not currently implemented).

4 Empirical work

4.1 Experimental setup

We tested this approach to exception handling in a small simulated double auction market. The auction simulator is extremely flexible, allowing many different auction configurations to be examined. We will report results across the full range of possibilities in due course. For now we describe some preliminary experiments.

[4] We do not allow bid/ask calls to be corrupted in our current experimental setup to allow us to determine the effect of the two kinds of error independently.

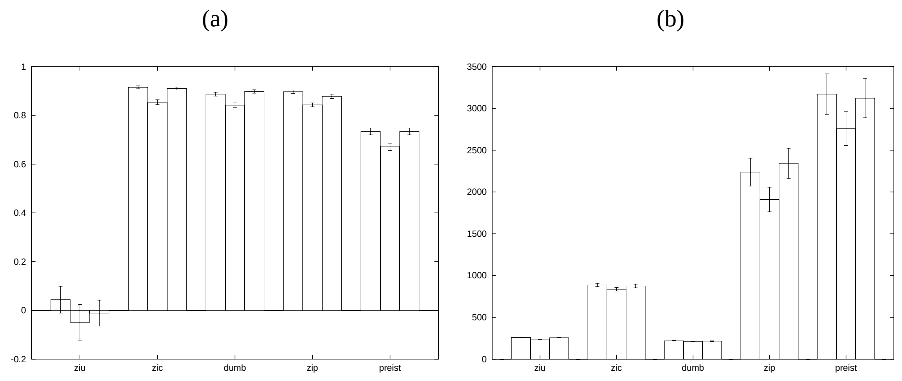


Fig. 2. Results of the bid-call loss experiments, (a) Efficiency, and (b) Messages

All the experiments we describe here involve a market of 10 buyers and 10 sellers, each of which is trading a single good and has a randomly selected private value. Each auction is run once, so we do not measure results over several periods. Because we are simulating an unreliable infrastructure, and so simulate that infrastructure, messages take time to travel from agent to auctioneer and from auctioneer to agent. Traders also bid asynchronously—each starts bidding/asking some random time after the receipt of a bid/ask call, and updates its bid/ask when it chooses to.

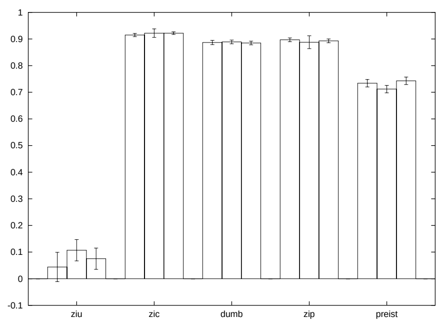
Our traders can use a range of mechanisms to pick their offer. While it is possible to create markets in which different agents use different mechanisms, so far we have only studied homogeneous markets. The strategies we use are the familiar zero-intelligence strategies of Gode and Sunder [10], both constrained ZIC and unconstrained ZIU[5], Cliff's zero-intelligence plus ZIP strategy [5], Preist and Van Tol's [27] variation on ZIP, and a variation of our own which we call the "dumb trader" (this was designed to be robust and easy to implement rather than particularly effective[6]).

With this setup, we ran experiments in which we introduced two main kinds of error. First we ran a series of auctions in which there were no errors. Then we introduced errors in the initial broadcast by the auctioneer of calls for bids, so that some agents were not informed of the start of the auction, but turned off the resolution handlers which responded to the detection of these errors (the detection handlers were still run, though equally well they they could be turned off). Then we ran a third set of experiments in which the same error was introduced at the same rate, but the resolution handlers were also run. We then repeated the process for an error in which the content of messages was corrupted.

[5] A zero-intelligence trader picks a random offer within a predefined range of possible prices, typically from 0 up to the maximum private value that any agent has for the good. The unconstrained ZIU trader bids or asks this price. The constrained ZIC buyer will bid no more than its private value, and the constrained seller will ask no less than its private value.

[6] It can be thought of as a simple-minded version of ZIP—it decreases an initial profit margin by a predetermined amount when a previous offer fails to win a good, so that bids rise and asks fall.

(a) (b)



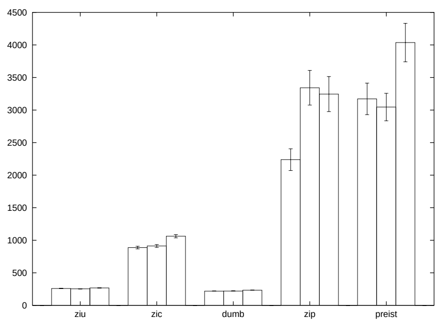


Fig. 3. Results of the corrupt message experiments, (a) Efficiency, and (b) Messages

4.2 Results

The results of these experiments are shown in Figures 2 and 3. For each experiment we use two metrics of market performance. One is *allocative efficiency*—a measure of how well the market runs. We measure this as a ratio of the profit made during the auction to the profit that could be made if the agents traded in the most efficient way (if each offered at its private value, and the traders were matched to maximise the profits obtained). This provides a measure of the effectiveness of the market in economic terms. The second measure is the number of messages sent during the course of the auction. This gives a computational measure of efficiency—how many resources the auction consumes in a run. All measures are plotted with standard deviations after 200 iterations (except for the ZIP results for corrupt data which are based on 75 iterations). These results show the broad effectiveness of the exception handling approach for auctions.

For all bidding mechanisms except ZIU, the loss of some bid calls causes a significant drop in efficiency (exactly as one might expect). This indicates that this is an exception that needs to be handled by the operators of the market—without fixes, traders in such auctions will lose profits, and may well move to other exchanges. When the exception is resolved, for all trading strategies except ZIP, efficiency is restored to a level that is not significantly different to the level without exceptions. For the case of ZIP, the efficiency may not be entirely restored by the resolution handler—we need more data (which we are currently collecting) to determine if this difference is significant.

ZIU is impervious, at least within the bounds of what is statistically significant, to the bid-call loss exception. Presumably this is just because it is so bad at extracting profit from the market, which in turn is because of its stochastic behaviour. Since agents generate bids randomly, and may trade at a loss as well as at a profit, taking agents out of the auction is as likely to reduce the chance to make a loss as it is to reduce the chance to make a profit.

Turning to the number of messages sent, the introduction of bid-call losses has the expected impact. Since some traders are taken out of the market, the number of messages falls. The important conclusion we can reach is that the resolution handler does not create a significant computational overhead (not that we would expect it to). When the handler is run, the number of messages returns to a level that is not significantly different to that without the exceptions.

Corrupt message errors at this rate of incidence have a much smaller effect on efficiency than bid call losses. This is understandable. As described above, the agents are all bidding, so the only real impact on efficiency will be when (a) an trader makes what would have been a winning offer, only to have that offer corrupted, and (b) a quote is lost that would otherwise have caused a trader to make a new bid/ask that would then have been a winning offer. We would only really expect these exceptions to hurt strategies like ZIP and the Preist and van Tol strategy since they are the only ones that try to take detailed account of how failure to win a good, or a change in quote price, should affect the next offer. These strategies show some loss of efficiency, but nothing too significant. ("Dumb" takes account of failure to win a good too, but in such a naive way that it is not surprising that a subtle effect is missed.) When the corrupt message exception handlers are turned on, any loss in efficiency is erased. Again this confirms the effectiveness of the handlers.

Looking at the number of messages passed for this second experiment confirms what one would expect. When messages are getting corrupted, then more are sent (because the total includes the corrupt messages and extra messages that are needed to replace the corrupted messages that would otherwise have secured trades) and this number broadly rises again when the handlers are switched on. The effect is particularly marked for the adaptive ZIP and Preist and van Tol strategies. This indicates that there can be an expense in running the handlers (unlike the case for the bid-call loss handler).

Two other observations, more general than those about the exception handlers, are worth making here. One is that the Preist and van Tol strategy does surprisingly badly across the board. We suspect this is due to the random values that agents have for their goods since efficiencies overall are below what one would expect in more structured markets. However, if this is the case, it makes it surprising that the similar ZIP strategy does comparatively well. This is something else we will investigate more in the future. The second observation is the message cost of running ZIP and Preist and van Tol traders compared with the simpler traders in our markets—the adaptation achieved by such strategies comes a price. This may be significant in some resource allocation scenarios (such as those where bandwidth is a limited or valuable resource).

4.3 Discussion

One might ask how the approach we have adopted would work better than classical networking techniques such as parity checks and message acknowledgement. In other words, why bother with exception handling? There are at least four answers.

The first answer is that we are providing exception handling at the system level, the auction infrastructure level, rather than at the networking level. This is important because we are interested in open systems. In such systems traders will be wandering in and out of electronic institutions (possibly in the middle of auctions—agents that are bidding in several different auctions will have to leave some of them when they secure the goods that they want) and so the transmitter of a message may well not know whether it is appropriate for all the intended receivers to respond. Network-level mechanisms like waiting for acknowledgement simply won't work under such circumstances, whereas the sentinel approach will.

The second answer is that where we use the same mechanism as can be handled at the network level, as is the case with the corrupt message handler, we abstract away the need to alter the trader by altering the handler. In other words, relying on network level handlers means we have to settle for a survivalist approach to exeception handling, and lose the advantages of the citizen approach that we argued for above.

The third answer is that the sentinel level solution can be more efficient because it is more intelligent. Adopting a solution to bid call loss that forces every message to be acknowledged will double the number of messages passed. In contrast the sentinel level solution requires a modest increase in the number of messages.

The fourth answer is that we can provide handlers for exceptions that can't be caught at the network level. While lost and corrupt messages are easy to detect at the network level, other kinds of exception cannot be detected without domain knowledge. We have already mentioned handlers that detect invalid trades—these clearly could not be handled by any network-level mechanism. Similiarly, network-level mechanism would not be able to deal with exceptions caused by agent death of the kind discussed in [24].

5 Related work

The work described here is clearly very close to Kaminka's work on execution monitoring [18–20] (and also [3] which builds on it). In execution monitoring, agents have a normative model of a process (which is basically a plan that all the agents are meant to be following) and use this to detect variations from the norm by some of their peers. Since team members are assumed to be both cooperative and under the control of the same organization, this work is set in the context of closed rather than open systems. That is one point of contrast with out work.

Now, the execution monitoring problem requires the agents identifying exceptions to infer the plan of the peer they are considering (since the former agents don't necessarily know which plan the latter is meant to be following), and this means carrying out successful plan recognition. To do this the agents require a lot of detailed knowledge about the domain and the intentions of their peers, whereas the exception handling approach only requires very general information about the infrastructure and the coordination mechanism. As a result, execution monitoring can probably do more in the way of handling problems in a narrow domain (while requiring a whole new knowledge base to handle a different domain) while exception handling can do less in a particular domain, but is applicable across a wide range of domains with little alteration.

In this respect our work, just like Kaminka's (as acknowledged in [20]) is an outgrowth of work on planning [6] that sought to ensure the correct outcome of a plan by checking that it was unfolding successfully. This work unfolded through a series of papers [30], [14], [28], and by the end the basic model was exactly that which we follow. Each keystone provides a checkpoint where expected progress is checked against actual progress, and a problem flagged up if one exists. However, our work goes further in fixing the exceptions once they are detected.

Since our work is concerned with finding handlers to overcome the exceptions, and to then get the system being monitored running again, our work is very much in the same domain as CIRCA [1]. However, an important difference is that while CIRCA tries to deal with states that are "unplanned for" our approach has the long term aim of engineering the very idea of "unplanned for" states out of existence by providing a compact knowledge base of general purpose processes to handle *any* unplanned states that emerge. Another important difference between CIRCA and our approach is that the former assumes access to the internal state of the agents being monitored.

We can also relate our approach to [15, 16]. Once again there is a close relation between the two approaches, but with a rather different emphasis. Both approaches make use of a causal model of a system in order to diagnose problems with it, and both make use of this diagnosis to overcome the problem. However, [15, 16] once again seems intended to operate in a closed system, and only considers a single exception—inadequately managed dependencies between a set of fully cooperative agents. It can therefore be subsumed by our approach.

6 Conclusions

This paper studied the question "How can we develop robust multi-agent systems from unreliable components?", and proposed the use of domain-independent exception handling services as a solution. In the context of multi-agent systems that implement double auctions, we showed empirically that the particular exception handling approach we describe here is able to provide this robustness.

While the idea of using exception handling services is not novel *per se* since it was suggested in previous work by the second author, there is still considerable novelty in this paper. First, we have extended the kinds of exception handling service beyond handling agent death exceptions to handling infrastructure issues like the unreliability of communication. This provides support for the generality of the exception handling approach. Second, we have extended the kinds of coordination mechanism covered by exception handling from the rather specific contract net model, to the much more general double auction model (indeed the same framework could be used to handle single sided-auctions without much alteration). We have also extended the range of handlers from very specific handlers for agent death to much more general handlers for dealing with message loss. This provides results that will be of interest to the large number of researchers who are interested in auctions.

Acknowledgments

This work was partially supported by NSF IIS-0329037, and by HP under the "Always on" grant.

References

1. E. Atkins, E. H. Durfee, and K. G. Shin. Detecting and reacting to unplanned-for world states. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 1997.

2. J. A. Barnett. How much is control knowledge worth? A primitive example. *Artificial Intelligence*, 22:77–89, 1984.
3. B. Browning, G. A. Kaminka, and M. M. Veloso. Principled monitoring of disributed agents for detection of coordination failure. In *Proceedings of Distributed Autonomous Robotic Systems*, 2002.
4. M. H. Chia, D. E. Neiman, and V. R. Lesser. Poaching and distraction in asynchronous agent activities. In *Proceedings of the Third International Conference on Multi-Agent Systems*, Paris, France, 1998.
5. D. Cliff and J. Bruten. Minimal-intelligence agents for bargaining behaviours in market-based environments. Technical Report HP-97-91, Hewlett-Packard Research Laboratories, Bristol, England, 1997.
6. R. Doyle, D. Atkinson, and R. Doshi. Generating perception requests and expectations to verify the execution of plans. In *Proceedings of the National Conference on Artificial Intelligence*, 1986.
7. K. Fischer, J. P. Müller, I Heinmig, and A-W. Scheer. Intelligent agents in virtual enterprises. In *Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-AgentTechnology*, Blackpool, UK, 1996.
8. D. Friedman. The double auction institution: A survey. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, Santa Fe Institute Studies in the Sciences of Complexity, chapter 1, pages 3–25. Perseus Publishing, Cambridge, MA, 1993.
9. S. Gjerstad and J. Dickhaut. Price formation in double auctions. *Games and Economic Behaviour*, 22:1–29, 1998.
10. D. K. Gode and S. Sunder. Allocative efficiency of markets with zero-intelligence traders: Market as a partial sustitute for individual rationality. *The Journal of Political Economy*, 101(1):119–137, February 1993.
11. D. K. Gode and S. Sunder. Lower bounds for efficiency of surplus extraction in double acuctions. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, Santa Fe Institute Studies in the Sciences of Complexity, chapter 7, pages 199–219. Perseus Publishing, Cambridge, MA, 1993.
12. T. R. Gruber. A method for acquiring strategic knowledge. *Knowledge Acquisition*, 1:255–277, 1989.
13. S. Hägg. A sentinel approach to fault handling in multi-agent systems. In *Proceedings of the Second Australian Workshop on Distributed AI*, Cairns, Australia, 1996. Workshop held in conjunction with the Fourth Pacific Rim Conference on Artificial Intelligence.
14. D. Hart, S. Anderson, and P. Cohen. Envelopes as a vehicle for improving the efficiency of plan execution. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.
15. B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structure. In *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.
16. B. Horling, V. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Technical Report 99-03, Department of Computer Science, University of Massachusetts, January 1999.
17. N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–36, 1998.
18. G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring deployed agent teams. In *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.
19. G. A. Kaminka and M. Tambe. What is wrong with us? improving robustness through social diagnosis. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 1998.

20. G. A. Kaminka and M. Tambe. I'm ok, you're ok, we're ok: Experiments in distributed and centralized socially attentive monitoring. In *Proceedings of the 3rd International Conference on Autonomous Agents*, 1999.
21. M. Klein. Supporting conflict resolution in cooperative design systems. *IEEE Systems, Man and Cybernetics*, 21:1379–1390, 1991.
22. M. Klein. Exception handling in process enactment systems. Working paper, MIT Center for Coordination Science, Cambridge, MA, December 1997.
23. M. Klein and S. Parsons. Diagnosing faults in open dstributed systems. Working paper, MIT Center for Coordination Science, Cambridge, MA, February 2003.
24. M. Klein, J. A. Rodriguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(1/2), 2003.
25. J. Nicolaisen, V. Petrov, and L. Tesfatsion. Market power and efficiency in a computational electricity market with discriminatory double-auction pricing. *IEEE Transactions on Evolutionary Computation*, 5(5):504–523, 2001.
26. S. Parsons. Exception analysis for double auctions. Working paper, MIT Center for Coordination Science, Cambridge, MA, May 2002.
27. C. Preist and M. van Tol. Adaptative agents in a persistent shout double auction. In *Proceedings of the 1st International Confernece on the Internet, Computing and Economics*, pages 11–18. ACM Press, 1998.
28. G. Reece and A. Tate. Synthesizing protection monitors from causal structure. In *Proceedings of the Conference on Artificial Intelligence Planning Systems*, 1994.
29. J. Rust, J. H. Miller, and R. Palmer. Characterizing effective trading strategies. *Journal of Economic Dynamics and Control*, 18:61–96, 1994.
30. J. Sanborn and J. Hendler. A model of reaction for planning in dynamic environments. *Artificial Intelligence in Engineering*, 3(2):95–102, 1988.
31. T. Sandholm, S. Sikka, and S. Norden. Algorithms for optimizing levelled commitment contracts. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
32. M. A. Satterthwaite and S. R. Williams. The Bayesian theory of the *k*-double auction. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, Santa Fe Institute Studies in the Sciences of Complexity, chapter 4, pages 99–123. Perseus Publishing, Cambridge, MA, 1993.
33. R. G. Smith and R. Davis. Distributed problem solving: The contract net approach. In *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978.
34. V. L. Smith. An experimental study of competitive market behaviour. *The Journal of Political Economy*, 70(2):111–137, April 1962.
35. A. Tate, editor. *Proceedings of the International Workshop on Knowledge-based Planning for Coalition Forces*, Edinburgh, Scotland, 1999.
36. G. Tesauro and R. Das. High-performance bidding agents for the continuous double auction. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 206–209, 2001.
37. M. B. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. Magma: An agent-based virtual marketplace for electronic commerce. *Applied Artificial Intelligence*, 11:501–524, 1997.
38. W. E. Walsh, R. Das, G. Tesauro, and J. O. Kephart. Analyzing complex strategic interactions in multi-agent systems. In P. Gymtrasiwicz and S. Parsons, editors, *Proceedings of the 4th Workshop on Game Theoretic and Decision Theoretic Agents*, 2001.
39. M. Youssefmir and B Huberman. Resource contention in multi-agent systems. In *Proceedings of the First International Conference on Multi-AgentSystems*, San Francisco, CA, 1995.