An Exception-Handling Architecture for Open Electronic Marketplaces of Contract Net Software Agents

Chrysanthos Dellarocas
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
+1 (617) 258-8115
dell@mit.edu

Mark Klein
Center for Coordination Science
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
+1 (617) 253-6796
m_klein@mit.edu

Juan Antonio Rodriguez-Aguilar
Center for Coordination Science
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA
+1 (617) 253-2430
jarjar@mit.edu

ABSTRACT
Software agent marketplaces require the development of new architectures, which are capable of coping with unreliable computational and network infrastructures, limited trust among independently developed agents and the possibility of systemic failures. In analogy with human societies, agent marketplaces will benefit from the introduction of appropriate electronic exception handling institutions, whose role will be to help guarantee efficiency and fairness in the face of these challenges. This paper presents a research methodology for designing and evaluating such electronic institutions. It also describes how the methodology has been applied in order to design and evaluate an exception handling architecture for robust software agent marketplaces based on the contract net protocol.

Keywords
Software agents, electronic markets, electronic institutions, exception handling, contract net, failure management.

1. INTRODUCTION
Software agent technologies promise substantial increases in productivity by automating several of the most time-consuming stages of electronic commerce processes. Agents are software systems, which are capable of interacting with other agents in a flexible and autonomous way, in order to meet the design objectives of their creators [12].

Electronic agent marketplaces are formed by collections of software agents, which interact with one another in order to negotiate and form partnerships or trade products and services through the Internet. In the emerging model of 21st century electronic commerce, a variety of open software agent marketplaces will be competing with one another for participants. Independently developed agents will be entering and leaving marketplaces at will, in pretty much the same way that human investors enter and leave different financial markets today. The stakeholders of electronic marketplaces will, therefore, have an interest in making them as attractive to prospective "customers" (buyers and sellers) as possible. One expects that the most successful marketplaces will be the ones that provide the best "quality of service" guarantees (in terms of security, fairness, efficiency, etc.). The proper design of open electronic marketplace infrastructure thus emerges as an important research and practical question.

Designing efficient and robust open electronic marketplaces, whose participants will be independently developed software agents, is a difficult problem. Some of the most important challenges include:

- *Unreliable Infrastructures.* In large distributed systems like the Internet, unpredictable node and link failures may cause agents to die unexpectedly, messages to be delayed, garbled or lost, etc.
- *Non-compliant agents*. In open systems, agents are developed independently, come and go freely, and thus can not always be trusted to follow the rules properly due to bugs, bounded rationality, programmer malice and so on. This can be expected to be especially prevalent and important in electronic marketplaces where there may be significant incentives for fraud.
- *Emergent dysfunctions*. Emerging multi-agent system applications are likely to involve complex and dynamic interactions that can lead to emergent dysfunctional behaviors with the relatively lightweight multi-agent coordination mechanisms that have proved most popular to date. This is especially true since agent societies operate in a realm where relative coordination, communication and computational costs and capabilities can be radically different from those in human society, leading to behaviors with which we have little previous experience. It has been argued, for example, that 1987's stock crash was due in part to the action of computer-based "program traders" that were able to execute trade decisions at unprecedented speed and volume, leading to unprecedented stock market volatility [26].

All of these departures from "ideal" multi-agent system behavior can be called *exceptions*, and the results of inadequate exception

EC'00, October 17-20, 2000, Minneapolis, Minnesota.

handling include the potential for poor performance, system shutdowns, and security vulnerabilities.

The standard approach to dealing with exceptions in closed systems has been to "compile in" complicated and carefully coordinated exception handling behaviors into all problem-solving agents. We call this approach the "survivalist approach", because agents in such systems are expected to contain enough intelligence to be able to fend for themselves in the face of, say, unreliable infrastructure and systemic dysfunctions.

We argue that survivalist approaches to exception handling are not viable in the context of open systems. First, they raise the bar for participation into the system by requiring that all agents contain sophisticated exception handling mechanisms built into them. This is especially undesirable in applications, such as electronic marketplaces, where participation is voluntary and the market maker's incentive is to reduce the "barriers of entry" into the system as much as possible. Second, and most important, even if a sophisticated "survivalist" exception handling behavior could be agreed upon, in an open system where agents are developed independently there can be no guarantee that they will all correctly follow it. Agents could deviate from the specified exception handling behavior because of ignorance of the specification, programming bugs or malice. In such a system, naïve agents would be left unprotected in the face of exceptions. Furthermore, even sophisticated agents could be harmed by transacting with agents of variable quality. For example, suppose that a sophisticated and highly reliable agent receives a contract from another agent and contracts out a subtask to a third agent, who happens to be buggy and failure prone. Further suppose that the third agent encounters a bug before it has completed its assigned subtask. Because its implementation is flawed, it fails to notify its contractor of the problem. The contractor will then fail to receive the expected results on time (probably having to reassign the subtask to another agent) and will itself be late in returning its results to its contractor, thus damaging its reputation for being a reliable agent.

Civilized human societies have successfully coped with similar challenges by developing *social institutions* that set and enforce laws (e.g. courts, police), monitor for and respond to emergencies (e.g. ambulance system), prevent and recover from disasters (e.g. coast guard, firefighters), etc. In that way, societies allow citizens to utilize relatively simple, optimistic and efficient rules of behavior, offloading the prevention and recovery of many problem types to social institutions that can handle them efficiently and effectively by virtue of their economies of scale and widely accepted legitimacy.

In an analogous manner, we believe that the design of the right *electronic exception-handling institutions* will be a crucial success factor in the new universe of open electronic marketplaces. More specifically, our claim is that, through the proper division of labor between problem-solving agents and institutions, successful open electronic marketplaces will achieve a number of desirable outcomes, including:

- Decreasing the "barriers to survival" for each agent, simplifying their implementation requirements and allowing them to focus on their core problem-solving functionality.
- "Leveling the playing field" by offering a basic set of security, fairness and efficiency guarantees, which provide consistent system behavior in the presence of agents of varying sophistication, reliability and benevolence.
- Increasing the efficiency of the system as a whole.

In this paper we present an experimental evaluation of a set of domain-independent exception handling services we have developed to address these challenges, applied to the well-known "Contract Net" multi-agent coordination protocol. We show that these services produce more effective exception handling behavior than standard existing techniques, while allowing simpler agent implementations. The remainder of this paper will introduce the contract net protocol, outline our exception handling approach, describe the experiments used to evaluate it, consider the contributions of this work, and discuss directions for future research.

2. THE CONTRACT NET PROTOCOL

The "Contract Net" (henceforth called CNET) is a protocol for matching up tasks with agents in multi-agent systems [22]. CNET and its many variants is probably the most widely used agent system protocol, presumably because of its intuitiveness, direct applicability to many common problems, simplicity and relative efficiency. CNET has been applied to many domains including manufacturing control [1], tactical simulations [3], transportation scheduling [4], and distributed sensing [22]. CNET is also an abstract version of the one-round-sealed-bid auction protocol used in a number of today's B2B exchanges.

The CNET protocol operates as follows (Figure 1):

> An agent (hereafter called the "contractor") identifies a task that it cannot or chooses not to do locally and attempts to find another agent (hereafter called the "subcontractor") to perform the task. It begins by creating a Request For Bids (RFB) which describes the desired work, and then sends it to potential subcontractors (typically identified using a matchmaker that indexes agents by the skills they claim to have). Interested subcontractors respond with bids (specifying such issues as the time needed to perform the task) from which the contractor selects a winner. The winning agent, once notified of the award, performs the work (potentially subcontracting out its own subtasks as needed) and submits the results to the contractor.

CNET is prone to a wide range of potential exceptions from all three of the categories (unreliable infrastructure, non-compliant agents, emergent dysfunctions) described in Section 1. A more exhaustive analysis of these failure modes will appear in a forthcoming paper. For now we will limit ourselves to three examples:

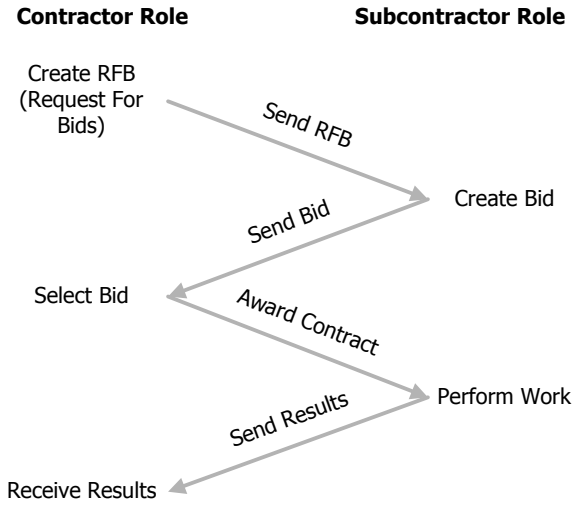

Figure 1. A simple version of the Contract Net protocol.

- *Agent death:* If a CNET agent dies there are several immediate consequences. If the agent is acting as a subcontractor, its customer clearly will not receive the results it is expecting. In addition, if the agent has subcontracted out one or more subtasks, these subtasks and all the sub-sub-… tasks created to achieve them become "orphaned", in the sense that there is no longer any real purpose for them and they are uselessly tying up potentially scarce subcontractor resources. Finally, if the system uses a matchmaker, it will continue to offer the now dead subcontractor as a candidate (a "false positive"), resulting in wasted message traffic.
- *Fraudulent [sub]contractor:* A buggy or intentionally malicious CNET agent can wreak havoc through fraudulent advertising, bidding or subcontracting.
- *Resource poaching:* It is typical for CNET systems to annotate tasks with priorities, so that when a subcontractor is considering several RFBs, it will bid (first) for the RFB with the greatest priority. One emergent dysfunction that can occur in such contexts is "resource poaching", wherein a slew of low-priority but long-duration tasks tie up the subcontractors, thereby freezing out resources needed for the higher-priority tasks that arrive later [6].

This paper concentrates on the "agent death" exception. As we mention in Section 6, we are currently working on electronic institutions for handling other CNET exceptions as well, but the results of that work will be reported in a forthcoming paper.

The standard "agent death" exception handling mechanism used in CNET, as in many distributed protocols, is timeout/retry: If no results are received by the deadline the subcontractor promised, for example, a contractor will re-start the subcontracting process for that task, sending a new RFB. This approach does handle the agent death exception, but rather inefficiently, since it does not eliminate orphaned tasks, does not remove false positives from the matchmaker, and is prone to an "unzippering" effect, wherein the death of an agent performing a subtask can cause cascading timeouts and retries for its customers, the customers of its customers, and so on, all the way up to the CNET agent at the top of the task decomposition tree. The timeout/retry approach will not, of course, prevent a contractor from repeatedly falling prey to a fraudulent CNET agent, nor will it help with resource poaching.

It is certainly imaginable that the CNET protocol could be elaborated to allow agents to handle a wider range of exceptions, and most agent system exception handling research has in fact taken this direction. Even the original CNET protocol [22] included such augmentations as an "immediate response bid', which allowed a contractor to determine whether the lack of bids was due to all eligible subcontractors being busy (in which case a retry is appropriate) or due to the outright lack of subcontractors with the necessary skills (in which case presumably the system manager/user should be informed). This "survivalist" approach to multi-agent exception handling faces, however, a number of serious shortcomings:

First of all, it greatly increases the burden on agent developers. It is predicated upon "compiling in" potentially complicated and carefully coordinated exception handling behaviors into all problem-solving agents. Perhaps more seriously, this approach is not viable in the context of open systems where agents are developed by independent third parties. Some agents may not comply properly with these more sophisticated protocols, or violate some of their underlying assumptions. Some protocols, for example, are based on game-theoretic analyses [21] and assume that all agents will be rational utility maximizers, which obviously may not always be the case. All agent interactions are slowed down by the overhead incurred by these heavyweight protocols. Some kinds of interventions (such as "killing" a broken agent that is uselessly monopolizing scarce resources) may be difficult to implement because the agents do not have the established legitimacy needed to apply such interventions to their peers. Finally, finding the appropriate responses to some kinds of exceptions (typically emergent exceptions such as resource poaching) requires that the agents achieve a more or less global view of the system state, which is notoriously difficult to create without heavy bandwidth requirements.

3. DOMAIN-INDEPENDENT EXCEPTION HANDLING SERVICES

It is for this reason that we have been creating a set of services that offload the exception handling burden from problem solving agents. We call this the "citizen" approach by analogy to the way exceptions are handled in human society. In such contexts, citizens adopt relatively simple and optimistic rules of behavior, and rely on a whole host of exception handling institutions (provided by the infrastructure) in order to handle most problems.

3.1 Capturing Domain-Independent Exception Handling Expertise

The key insight that makes this approach workable in the context of multi-agent systems (MAS) is the simple but powerful notion that the characteristic exceptions and applicable exception handling techniques for a multi-agent system can be usefully treated as dependent on the market mechanism used but *independent of the vertical domain the agents work in*. This is a key insight because it

means that we can build exception handling knowledge bases (and associated run-time services) that are generic and thereby highly reusable.

Early work on expert systems development revealed that it is useful to separate domain-specific problem solving and generic control knowledge [2]. Analogous insights were also confirmed in the domains of collaborative design conflict management [14] and workflow exception management [17].

The exceptions that characterize a given MAS protocol can be uncovered using an emerging technique we call Role Commitment Violation (RCV) analysis [16]. RCV analysis is based on the insight that coordination fundamentally involves the process of agents making commitments to each other. Exceptions can thus be viewed as the ways in which the agents in a MAS can fail to achieve the commitments underlying their coordination protocol.

We have used RCV analysis to uncover the consequences of agent death in the context of a CNET. In summary, if a CNET agent dies there are several immediate consequences. If the agent is acting as a subcontractor, its customer clearly will not receive the results it is expecting. In addition, if the agent has subcontracted out one or more subtasks, these subtasks and all the sub-sub-… tasks created to achieve them become "orphaned", in the sense that there is no longer any real purpose for them and they are uselessly tying up potentially scarce subcontractor resources. Finally, if the system uses a matchmaker, it will continue to offer the now dead subcontractor as a candidate (a "false positive"), resulting in wasted message traffic.

Once we have identified the exceptions that characterize a given MAS protocol we need to uncover the handlers that are appropriate for dealing with them. Unlike exceptions, the range of possible handlers is not a closed easily enumerable set but seems to be limited only by human ingenuity.

We have found that there are four main classes of handlers; those suitable for *anticipating* and *avoiding* exceptions before they occur, or *detecting* and *resolving* them after they occur [8]. The following is a domain-independent set of handlers for dealing with unexpected agent death.

To *detect* agent death, periodically poll active subcontractors. Consolidate polling in order to minimize the number of "are you alive?" messages CNET agents must respond to. To *resolve* a situation where an agent has died, clear the agent record from the matchmaker(s), and immediately instruct the contractors for that agent to re-run the bidding process for the failed tasks. One can cancel the orphaned sub-sub-tasks if any, or else (if there is a standard task decomposition for this kind of problem) be prepared to offer these results to the new CNET agent that takes on the sub-task previously assigned to the dead agent. Finally, to *avoid* or minimize the number of agent death exceptions, keep track of agent reliability statistics (as a function of mean time between failures) and help agents use them when making task assignment decisions.

3.2 A Domain-Independent Architecture for Handling Agent Death Exceptions

Our approach instantiates these ideas in an open MAS setting using the following functional architecture:

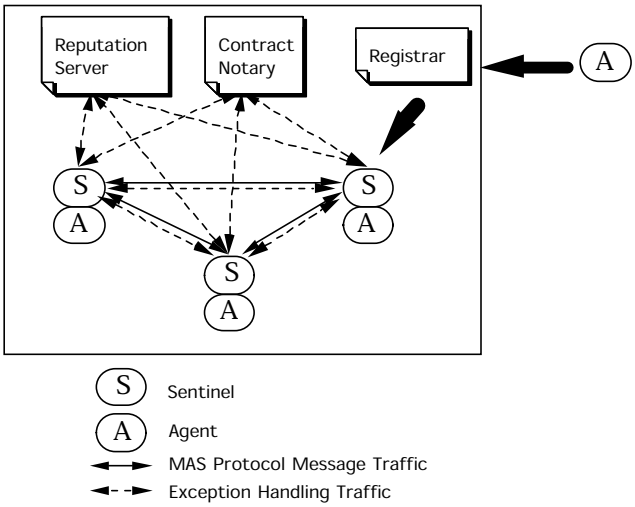

Figure 2. Functional architecture for open MAS with exception handling services.

When an agent joins an open MAS served by the exception handling (EH) services, it must register with a *registrar* responsible for assigning it a *sentinel* that will mediate all of the agents' further interactions with other agents in the system. The agents so 'wrapped' can include problem solving agents as well as components such as matchmakers that support the protocols they enact.

Sentinels are the central element in this approach. They can be viewed as "commitment monitors" whose role is to observe and influence agent behavior as necessary to ensure the robust functioning of the system as a whole. Each sentinel includes a repository of domain-independent EH expertise that describes the characteristic exceptions and associated handlers for the protocol(s) enacted by the agents in that MAS. Sentinels monitor message traffic to develop a model of the commitments their agent(s) are involved in, use the appropriate anticipation and/or detection handlers to uncover when these commitments are violated, diagnose the underlying causes to identify the appropriate avoidance and/or resolution handlers, and enact these handlers to help re-establish the violated commitments, or at least minimize the impact of them having been violated. Ancillary services such as the contract notary and reputation server keep track of global state information such as commitment structures and reliability statistics. Agents, for their part, must be able to respond appropriately to a relatively small set of EH directives to support the action of the sentinels.

4. EXPERIMENTAL EVALUATION

We ran a series of experiments to test these claims in a multi-agent marketplace running the CNET protocol. The experiments all take place in a discrete event based multi-agent system simulator built on top of the Swarm Simulation System [19]. Our system allows one to emulate a world consisting of multiple host computers, each running one or more agents and connected by network links, all with controllable speed and failure frequency. The scenario consists of several dozens CNET agents, one per host, interacting over a reliable network. Contractor agents send out an RFB with a specified timeout period: potential subcontractors bid only if they

become available during this period (i.e. subcontractors perform only one task at a time). Bids are binding, which means that subcontractors will bid on a new RFB only after the timeout for its pending bid expired without an award being received (presumably because some other subcontractor won the task). Contractors select the winning bids based solely on how quickly the bidders claimed they could perform the task. Contractors re-send RFBs if no bids have been received by the timeout period (presumably because no subcontractors with the needed skills were available at that time). This CNET protocol is modeled on the one described in [22] and was chosen because it is simple and was shown by Smith to represent a reasonable design tradeoff in several test domains.

Our experiments explored the effect of three experimental conditions. The key independent variable, of course, was whether the agents took a "survivalist" or "citizen" approach to handling agent death. Survivalist agents rely on the standard timeout/retry mechanism to handle agent death: If a subcontractor does not return results to its contractor by the agreed-upon deadline, the contractor issues a new RFB for the task. Citizen agents, by contrast, rely entirely on the EH services. Whenever a task has been awarded to a subcontractor, the EH service begins periodic polling of the subcontractor to check whether it is still alive, which continues until the agent has died or returned the task results to its contractor. If an agent dies, the EH service takes a series of coordinated actions:

1. It notifies the matchmaker that this agent is dead and should therefore be removed from the list of available subcontractors. This handles the "false matchmaker positive" problem.
2. If the agent is subcontracting to someone else, it immediately informs the contractor that it should re-send the RFB for that task, thereby ensuring that the contractor does not waste time waiting for results from a dead agent. Note that this avoids the "unzippering" effect described above.
3. If the agent is a contractor for some pending subtasks, a *proxy agent* is created to try to find new customers for those "orphaned" subtask results. The proxy registers itself with the matchmaker, so that it becomes eligible to receive RFBs. It then waits for an RFB for the orphaned tasks, and submits a bid whose estimated completion time accounts for the amount of time that has already been spent processing those tasks, and is therefore likely to be highly competitive. This is a reasonable strategy in domains where there is a standardized task decomposition, so the replacement for the dead agent is apt to require the same subtask results that the dead agent did. If the proxy wins the anticipated RFB, it forwards the results as it receives them. Otherwise it keeps responding to RFBs until it wins or until the task results become obsolete. This strategy is thus designed to minimize wasted work on orphaned tasks. In domains where results get obsolete very quickly, or where there is no standard task decomposition, it may be more appropriate to do without the proxy-bidding agent and simply kill all orphaned tasks when the ultimate customer for them has died.
4. Finally, it reports the agent death to the reputation server. This way the reputation server keeps track of the mean time between failures (MTBF) for every agent in the system. Whenever a contractor is receiving bids as a response to a previously sent RFB, the reputation server instructs sentinels to filter out incoming bids from agents whose MTBF is substantially lower than the marketplace mean unless they are the only bids received. This *bid filtering* scheme is designed to *avoid* agent death exceptions by protecting contractors from subcontracting unreliable agents, if other, more reliable agents are also available.

Our central hypothesis is that the "citizen" exception handling approach will significantly reduce the average amount of time needed to complete tasks in which exceptions occur (due to quicker detection of agent death, and the avoidance of the unzippering effect), as well as reduce the overall system effort needed to perform tasks (by avoiding wasting resources on orphaned tasks). We also explored the related hypotheses that the impact of the EH services will depend on the nature of the task decompositions needed to perform a task. More specifically, tasks that have deep task decompositions should benefit more because in those cases the unzippering effect will be more severe with survivalist agents.

In order to validate the above hypotheses, we tested the contract completion performance of five different agent configurations:

a) Failure-free environment (baseline case)

b) Failure-prone environment, "survivalist" agents (timeout-and-retry)

c) Failure-prone environment, "citizen" agents supported by EH services which poll subcontractors and, upon detection of agent death, inform the contractor of the dead agent and *kill all orphaned tasks*.

d) Failure-prone environment, "citizen" agents supported by EH services which poll subcontractors and, upon detection of agent death, inform the contractor of the dead agent and *create a proxy agent* to try to reassign orphaned tasks.

e) Same as d) with the addition of *filtering of bids from unreliable agents*.

In all configurations, top-level contractor agents execute a loop where they announce a new top-level task, wait for bids, award the contract to the best bidder, wait to receive the results and then stay idle a random amount of time before repeating the above steps.

In order to be completed, top-level tasks require the creation of task trees with depth 4 and branching factor 2. In other words, in order to complete a top-level task, a top-level contractor has to seek two level-2 subcontractors, each of which has to seek two level-3 subcontractors, and so on. Therefore, a single top-level task may involve up to 15 agents working simultaneously. To simplify the experiment, it is assumed that any available subcontractor is capable of performing any task in a given task chain.

In the failure-prone cases, subcontractor agents were divided into three reliability classes. All subcontractor agents had a "lifespan" (time until death) that was selected for each agent randomly from a geometric distribution with mean time between failures (MTBF) equal to:

10·(task duration)	for low reliability agents
50· (task duration)	for medium reliability agents
100·(task duration)	for high reliability agents.

When an agent dies, a new one is created with the same skills but with a different unique ID and is registered with the matchmaker. This is done to keep the subcontractor population from shrinking over the course of the experiment, thereby emulating a large and

dynamic agent pool where the population of subcontractors remains roughly constant.

All simulations were run until a 90% confidence interval could be computed for each of the completion time estimates with a width of less than 15 percent of the estimated mean.

Figure 3 summarizes the mean contract completion time relative to the failure-free (baseline) case for survivalist and each of the three configurations of citizen agents described above.

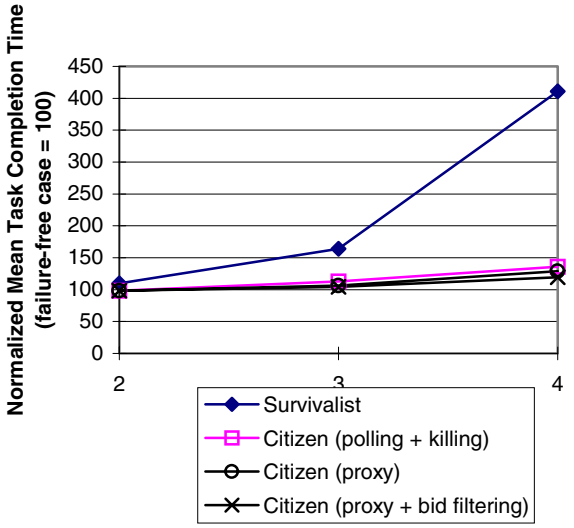

Figure 3. Mean task completion times relative to the failure-free case.

As expected, citizen agents with EH support clearly outperformed survivalist agents and managed to greatly reduce the effects of agent death exceptions (mean completion time in all citizen configurations was less than 140% of the failure-free mean). The difference in performance was particularly dramatic for longer tasks chains. The explanation in this case is that, for longer tasks, the probability of multiple agent deaths in the same task tree is correspondingly higher. In the survivalist case, each death may trigger the "unzippering" effect described in Section 2, which would effectively double the task completion time. In the case of multiple deaths, the "unzippering" effect would be repeated, thus multiplying the mean completion time even more.

Figure 4 compares the relative performance of the three different configurations of EH services tested. Although the differences are not very dramatic, we can see that the creation of proxy agents has a positive effect in overall efficiency, as does the addition of bid filtering. The combination of polling, creation of a proxy agent and bid filtering is the approach that gave the best overall results.

In addition to the mean task completion time, we were also interested to compare the standard deviation of task completion times in the presence of agent deaths. Our rationale is that, in most environments, consistency is equally important to efficiency. A system with a low mean completion time, but where some task instances take a very long time to complete is bound to make some users extremely unhappy.

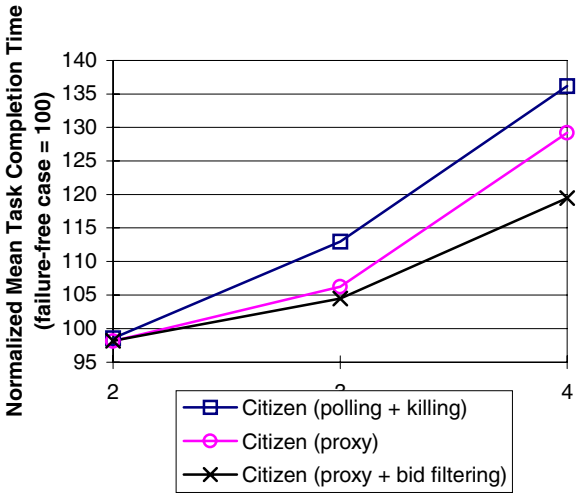

Figure 4. Comparative performance of the three configurations of EH services tested.

Figure 5 summarizes the standard deviations of task completion times in each of the tested configurations. From a study of the charts it is clear that citizen agents had a lower maximum observed completion time in all four configurations.

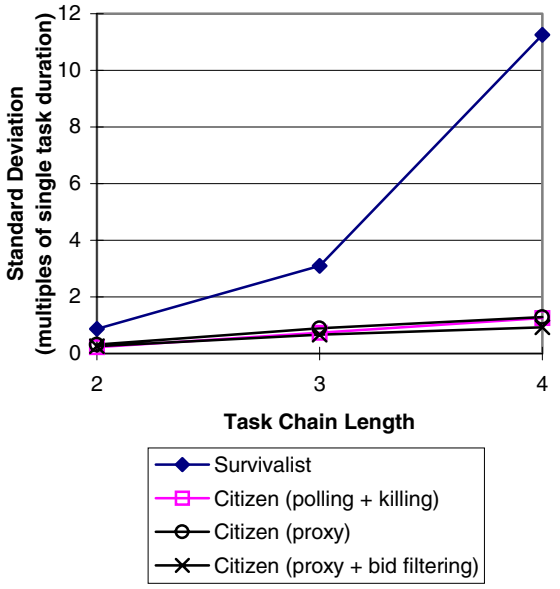

Figure 5. Standard deviation of task completion times in the four failure-prone agent configurations tested.

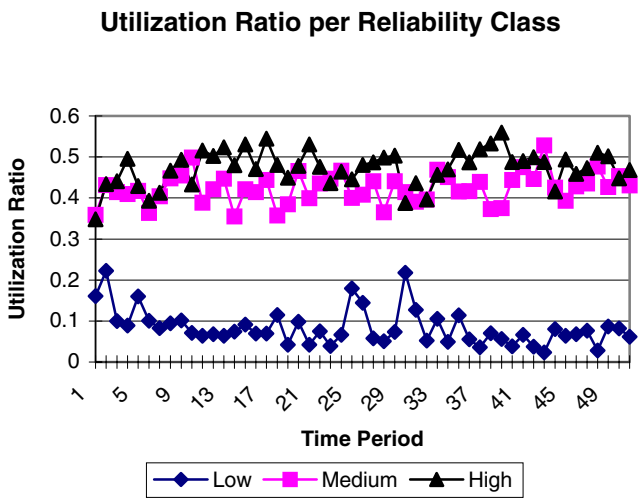

Figure 6. Effects of bid filtering in the utilization ratio of agents of different reliability.

Finally, Figure 6 depicts the effect of bid filtering in creating a "fair" marketplace where high reliability agents get more contracts and therefore have a higher utilization ratio than low reliability agents.

In conclusion, citizen agents have proven to perform more efficiently than survivalist agents both in terms of lowering the mean as well as improving the consistency of contract completion time in the face of exceptions.

5. CONTRIBUTIONS OF THIS WORK

From a general perspective, the "exception analysis" approach presented in this paper is an interesting complement to the field of economic mechanism design. The field of mechanism design uses the tools of economics and game theory to design "rules of interaction" for economic transactions that will yield some desired outcome [24]. In order to apply these analytical tools, mechanism designers usually make assumptions (such as the existence of rational, risk-averse agents, zero communication costs, perfectly reliable infrastructure, etc.), which are not always realistic in open systems. Our work begins by considering what might happen if one or more of the assumptions on which a particular game theoretic analysis is based are violated. It then proceeds to propose, and experimentally evaluate, additional mechanisms (exception handling institutions) whose goal is to maintain the desirable outcomes intended by the original mechanism in the complex, messy environments of open electronic markets.

Several lines of research have begun to explore concepts similar to those presented here, but none as far as we know have explored the combination of domain-independent exception handling implemented as distinct services. Hägg [10] presents the concept of sentinel agents; these are distinct services, which monitor the agent system and intervene when necessary by selecting alternative problem solving methods, excluding faulty agents, or reporting to human operators. This approach is not domain-independent, however: sentinels must be customized for each new application. Kaminka et.al. [13] present Social Attentive Monitoring (SAM), an exception handling approach wherein agents detect exceptions via uncovering violations of normative relationships with their teammates, and exploit a teamwork model to diagnose and fix these problems. This approach does have generic elements, but it is limited to teamwork protocols like TEAMCORE [23] and requires domain-dependent customization of the exception detection procedures. Horling et al. [11] have explored the use of domain-independent tools to detect and resolve the exception wherein the agents have a harmfully inaccurate picture of the inter-agent dependencies in their current context. This approach is limited to a single exception type, however, and like SAM applies to just one class of coordination protocol. Finally, Venkatraman et al [25] describe a generic approach to uncovering agents that do not comply with coordination protocols. This approach only addresses one subclass of exception types, however, and does not include a resolution component.

Distributed and real-time systems research has produced useful techniques such as checkpointing and rollbacks [5, 20], but these "one size fits all" techniques achieve generality at the cost of the efficiencies that can result from coordination-mechanism specific, albeit domain-independent, exception handling mechanisms.

6. FUTURE WORK

We plan to pursue two concurrent lines of development in this work. One line will include empirically and analytically evaluating different "survivalist" and "citizen" exception handling approaches for a wider range of exception types and market mechanisms. For example, we are currently looking at exceptions (intentional and unintentional) related with reputation mechanisms and ways to avoid or detect and resolve them. We are also planning to look at exceptions related to intentional contract violations and the associated electronic dispute resolution and sanctioning infrastructures.

A second line of work will be to increase the power and scope of our generic exception handling technologies. The RCV approach is currently being refined and formalized. Furthermore, we have developed a prototype repository of exceptions and associated handlers, built as an extension of the MIT Process Handbook [18]. As we are accumulating analytical and experimental evidence with market mechanisms, associated exceptions and relevant handlers, our repository will grow into an invaluable reference for electronic marketplace designers.

The long-term goal of these efforts is to integrate these lines of work, and thereby provide electronic marketplace system developers with a comprehensive knowledge base of well-founded design guidelines, along with a suite of domain-independent component technologies that enable them to much more easily develop more robust open electronic markets.

7. ACKNOWLEDGMENTS

This work was supported by NSF grant IIS-9803251 (Computation and Social Systems Program) and by DARPA grant F30602-98-2-0099 (Control of Agent Based Systems Program).

8. REFERENCES

[1] Baker, A. (1988). "Complete manufacturing control using a contract net: a simulation study." *1988 International Conference on Computer Integrated Manufacturing.* IEEE Comput. Soc. Press, pp.100-9. Washington, DC, USA.

[2] Barnett, J. A. (1984). "How Much Is Control Knowledge Worth? A Primitive Example." *Artificial Intelligence* 22(1): 77-89.

[3] Boettcher, K., D. Perschbacher, et al. (1987). "Coordination of distributed agents in tactical situations." *IEEE 1987 National Aerospace and Electronics Conference: NAECON 1987 (Cat. No.87CH2450-5).* IEEE, pp.1421-6 vol.4. New York, NY, USA.

[4] Bouzid, M. and A.-I. Mouaddib (1998). "Cooperative uncertain temporal reasoning for distributed transportation scheduling." *Proceedings, Third International Conference on Multi Agent Systems, Paris, France.* IEEE Comput. Soc. 1998, pp.397-8.

[5] Burns, A. and A. Wellings (1996). *Real-Time Systems and Their Programming Languages*, Addison-Wesley.

[6] Chia, M. H., D. E. Neiman, et al. (1998). "Poaching and distraction in asynchronous agent activities." *Proceedings, Third International Conference on Multi-Agent Systems*, Paris, France. IEEE Comput. Soc. Press, 1998, pp. 88-95.

[7] Dellarocas, C. and M. Klein (1999). "Designing robust, open electronic marketplaces of contract net agents." *Proceedings of the 20th International Conference on Information Systems (ICIS-99)*, Charlotte, North Carolina USA.

[8] Dellarocas, C. and M. Klein (2000) "A knowledge-based approach for handling exceptions in business processes" *Information Technology and Management* 1 (3): 155-169.

[9] Gruber, T. R. (1989). "A Method For Acquiring Strategic Knowledge." *Knowledge Acquisition* 1(3): 255-277.

[10] Hägg, S. (1996). "A Sentinel Approach to Fault Handling in Multi-Agent Systems." *Multi-Agent Systems, Methodologies and Applications. Second Australian Workshop on Distributed Artificial Intelligence. Selected Papers*. Springer-Verlag. 1997, pp.181-95.

[11] Horling, B., V. Lesser, et al. (1999). "Diagnosis as an Integral Part of Multi-Agent Adaptability." *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00.* IEEE Comput. Soc. 1999, vol. 2, pp.211-19.

[12] Jennings N.R., Sycara K. and Wooldridge M. "A Roadmap of Agent Research and Development", *Autonomous Agents and Multi-Agent Systems* (1:1), 1998, pp. 7-38.

[13] Kaminka, G. A. and M. Tambe (1998). "What is Wrong With Us? Improving Robustness Through Social Diagnosis." *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* Madison, Wisconsin, 1998.

[14] Klein, M. (1991). "Supporting Conflict Resolution in Cooperative Design Systems." *IEEE Systems Man and Cybernetics* 21(6): 1379-1390.

[15] Klein, M. and C. Dellarocas (1999). "Exception Handling in Agent Systems." *Proceedings of the Third International Conference on Autonomous Agents (Agents '99)*. Seattle, Washington, pp. 62-8.

[16] Klein, M. and C. Dellarocas (2000) "Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems." ASES Working Report ASES-WP-2000-02. Cambridge MA USA, Massachusetts Institute of Technology.

[17] Klein, M. and C. Dellarocas (2000). "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems." *Journal of Computer-Supported Collaborative Work* 9 (3/4) August 2000.

[18] Malone T.W., Crowston K., Lee J., Pentland B., Dellarocas C., Wyner G., Quimby J., Osborn C.S., Bernstein A., Herman G., Klein M., O'Donnell E. (1999) "Tools for inventing organizations: Toward a handbook of organizational processes." *Management Science* 45 (3), pp. 425-43.

[19] Minar, N., Burkhart, R., Langton, C., Askenazi, M., *The Swarm Simulation System: A Toolkit for Building Multi-Agent Systems*, Santa Fe Institute Working Paper 96-06-042, Santa Fe, NM, 1996.

[20] Mullender, S. J. (1993). *Distributed systems*. ACM Press, New York.

[21] Sandholm, T., S. Sikka, et al. (1999). "Algorithms for Optimizing Leveled Commitment Contracts." *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)* Stockholm, Sweden, vol.1, pp. 535-540.

[22] Smith, R.G. (1980) "The contract net protocol: high level communication and control in a distributed problem solver." *IEEE Transactions on Computers* 29 (12): 1104-13.

[23] Tambe, M. (1997). "Towards flexible teamwork." *Journal of Artificial Intelligence Research* 7: 83-124.

[24] Varian, H.R. "Economic Mechanism Design for Computerized Agents" *Proceedings of the First USENIX Workshop of Electronic Commerce*. USENIX Assoc. 1995, pp.13-21. Berkeley, CA, USA.

[25] Venkatraman, M. and M. P. Singh (1999). "Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems." *Autonomous Agents and Multi-Agent Systems* 3(3).

[26] Waldrop, M. "Computers amplify Black Monday", *Science* (238), Oct. 30 1987, pp. 602-604.