
Autonomous Agents and Multi-Agent Systems, 7, 179–189, 2003

Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death

MARK KLEIN m_klein@mit.edu
Center for Coordination Science, Massachusetts Institute of Technology NE20-336, Cambridge, MA 02139

JUAN-ANTONIO RODRIGUEZ-AGUILAR jar@isoco.com
iSoco S.A., C/Alcalde Barnils, 64-68, 08190 Sant Cugat del Valles, Barcelona, Spain

CHRYSANTHOS DELLAROCAS dell@mit.edu
Sloan School of Management, Massachusetts Institute of Technology E53-315, Cambridge, MA 02139

Abstract. This paper addresses a simple but critical question: how can we create robust multi-agent systems out of the often unreliable agents and infrastructures we can expect to find in open systems contexts? We propose an approach to this problem based on distinct exception handling (EH) services that enact coordination protocol-specific but domain-independent strategies to monitor agent systems for problems ('exceptions') and intervene when necessary to avoid or resolve them. The value of this approach is demonstrated for the 'agent death' exception in the Contract Net protocol; we show through simulation that the EH service approach provides substantially improved performance compared to existing approaches in a way that is appropriate for open multi-agent systems.

Keywords: multi-agent systems, failures, reliability, exception handling.

1. The challenge: enabling robust open multi-agent systems

This paper addresses one simple question: how can we develop robust multi-agent systems out of the often unreliable (buggy, malicious, or simply "dumb") agents and infrastructures we can expect to encounter in open system contexts? This is becoming an increasingly critical question because of emerging changes in the way human organizations work. Globalization, enabled by ubiquitous telecommunications, has increasingly required that organizations be assembled and re-configured within small time frames, often bringing together partners that have never worked together before. Examples of this include international coalition military forces, disaster recovery operations, open electronic marketplaces and virtual supply chains [1–3]. Multi-agent systems (MAS) represent one of the most promising approaches for supporting these kinds of applications, because of their ability to use multi-agent coordination protocols to dynamically self-organize themselves as their problems and constituent agents change [4, 5]. A critical open challenge remains, however. The vast majority of MAS work to date has considered *closed* systems with well-behaved agents running on reliable infrastructures [6], in contrast to the *open* and potentially unreliable contexts most applications present. For these contexts we can expect, in contrast, to find:

- *Unreliable Infrastructures*. In large distributed systems like the Internet, unpredictable host and communication problems can cause agents to slow down or die unexpectedly, messages to be delayed, garbled or lost, etc.
- *Non-compliant agents*. In open systems, agents are developed independently and thus can not always be trusted to follow the rules, especially in where there may be significant incentives for fraud or malice.
- *Emergent dysfunctions*. Most multi-agent coordination mechanisms are susceptible to emergent dynamical dysfunctions, such as chaotic behavior [7–10].

All of these departures from "ideal" multi-agent system behavior can be called exceptions, and the results of inadequate exception handling include the potential for poor performance, system shutdowns, and security vulnerabilities.

2. Our approach: distinct domain-independent exception handling services

It is certainly imaginable that agents could be individually elaborated so that they could handle all exceptions they are apt to face, and most MAS exception handling research has in fact taken this direction. This "survivalist" approach to multi-agent exception handling faces, however, a number of serious shortcomings. It greatly increases the burden on agent developers by requiring the implementation of potentially complicated and carefully coordinated exception handling behaviors in all agents. Developers must anticipate and correctly prepare for all the exceptions the agent may encounter, which is problematic at best since the agent's operating environments may be difficult to anticipate. Making changes in exception handling behavior is difficult because it potentially requires coordinated changes in multiple agents created by different developers. Agents become harder to maintain, understand and reuse because a potentially large body of exception handling code obscures the relatively simple normative behavior of an agent.

Perhaps more seriously, this approach can result in poor exception handling performance. In open systems it is always possible that some agents will not comply properly with these more sophisticated protocols or may violate some of their underlying assumptions. Some exception handling approaches, for example, are based on game-theoretic incentive analyses [11] that assume all agents are fully rational and share a particular class of utility function (typically profit maximization), but this obviously may not always be the case. Some agents may be buggy, face severe computational limitations that preclude full rationality, or have radically different utility functions (e.g. cause as much damage to a particular vendor as possible). All agent interactions are potentially slowed down by the overhead incurred by the more heavyweight 'exception-savvy' protocols. Some kinds of interventions (such as "killing" a broken or malicious agent) may in addition 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 (notably emergent exceptions) often requires that the agents achieve a more or less global view of the multi-agent system state, which is notoriously difficult to create without heavy bandwidth requirements.

It is in order to address these limitations that we have been defining an approach that enhances MAS robustness by offloading exception handling from problem solving agents

to distinct, domain-independent services. We call this the "citizen" approach by analogy to the way exceptions are handled in human society. Citizens typically adopt relatively simple and optimistic rules of behavior, and rely on a whole host of social institutions (the police, lawyers and law courts, disaster relief agencies, the Security and Exchange Commission, the Better Business Bureau, and so on) to handle most exceptions. This is generally a good tradeoff because such institutions are able, by virtue of specialized expertise, widely accepted legitimacy, and economies of scale, to deal with exceptions more effectively and efficiently than individual citizens, while making relatively few demands (e.g. pay your taxes, obey police officers, report crimes).

The key insight underlying the "citizen" approach is the simple but powerful notion that highly reusable, *domain-independent* exception handling expertise can be usefully separated from the knowledge used by agents to do their "normal" work. There is substantial evidence for the validity of this claim. Early work on expert systems development revealed that it is useful to separate domain-specific problem solving and generic control knowledge [12, 13]. Analogous insights were also confirmed in the domains of collaborative design conflict management [14, 15] and workflow exception management [16]. In our work to date we have found that every coordination protocol has its own characteristic set of domain-independent exceptions, which in turn can be mapped to domain-independent strategies potentially applicable for handing (anticipating and avoiding, or detecting and resolving) them. We shall see some examples of such strategies below; for others please see [17, 18].

3. Case study: handling agent death in the contract net protocol

Let us illustrate this approach by considering how it can be applied to an important scenario: handling agent death in the Contract Net protocol (CNET), a widely-used market-based task allocation protocol [19]. CNET operates as follows (Figure 1):

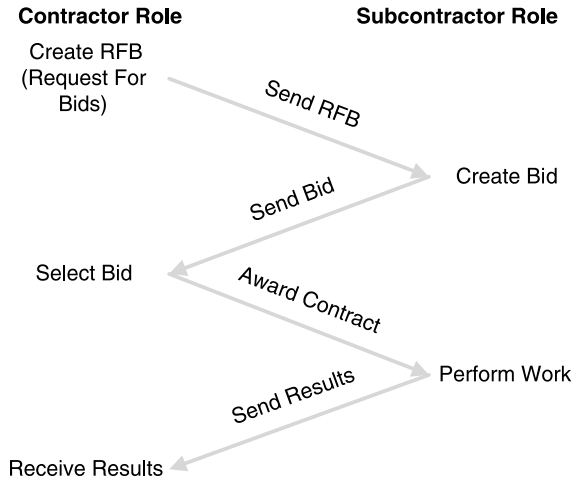

Figure 1. A simple version of the Contract Net protocol.

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 creates 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 described above [20].

Let us consider what happens when a CNET agent ‘dies’. Agent death can be common in a large distributed system: even the most carefully crafted code has been estimated to include an average of three bugs, mostly intermittent ones, per 1000 lines of code [21]. If a CNET agent dies there are several immediate consequences. If the agent is acting as a subcontractor, its customer 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’’, uselessly tying up potentially scarce resources. Finally, if the system uses a matchmaker, it will continue to offer the now dead agent as a candidate (a ‘‘false positive’’), resulting in wasted message traffic and a misleading picture of what skills are available in the MAS. CNET agent death presents a surprisingly rich source of challenges and helps reveal, we believe, many of the important issues involved in exception handling in open agent systems.

The standard mechanism used to handle agent death in CNET is a classic ‘‘survivalist’’ approach: timeout/retry. If no results are received by the deadline the subcontractor promised, a contractor will re-start the subcontracting process for that task, sending a new RFB. This approach does work but rather inefficiently, since it does not eliminate orphaned tasks, does not remove false positives from the matchmaker, and is prone to an ‘‘timeout cascade’’ 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, resulting in needless delays and wasted work.

Contrast this with a ‘‘citizen’’-style approach to handling the agent death exception. In our implementation of this approach, when an agent joins the MAS, the EH service begins periodic polling of the agent. If an agent dies (does not respond to polling in a timely way), the EH service takes a series of coordinated actions to resolve the problem

- It notifies the matchmaker that this agent is dead and should therefore be removed from the list of available subcontractors. This handles false matchmaker positives.
- If the dead agent was performing tasks for some customer(s), the EH service immediately asks these customers to re-allocate the tasks assigned to the dead agent. This avoids the ‘‘timeout cascade’’ effect described above, since contractors only reallocate tasks when the subcontractor has actually died.
- If the dead agent had allocated tasks to other agents, the EH service tries to find new customers for these orphaned tasks by acting as a proxy. The proxy waits for an RFB for the orphaned tasks, and submits a bid that is likely to be highly competitive since the tasks are either already in process or actually completed. 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 subtasks that the dead agent did. If the proxy wins the anticipated RFB, it forwards task results as they are generated. Otherwise it keeps responding to RFBs until it wins or the task results become obsolete. This strategy thus minimizes the work wasted on orphaned tasks. In domains where the proxy approach in inappropriate (e.g. results get obsolete very quickly, or there is no standard task decomposition) the EH service can simply kill all orphaned tasks.

- An agent reliability database is notified so it can keep up to date information about the mean time between failures for each agent type.

The EH service can also help *avoid* agent death problems exception via bid filtering. Whenever a contractor sends out an RFB, the EH service can transparently filter out the bids that come from the most failure-prone of the bidders, thereby reducing the probability that a task will be assigned to an agent that dies during its enactment.

The EH service makes two assumptions about agents in order to provide these capabilities. One is that it can transparently monitor and, if necessary, modify the domain-independent aspects (message types as well as task and agent IDs) of all inter-agent messages. This is straightforward to achieve if the EH service is realized using "sentinels" integrated into the communications infrastructure (Figure 2).

In this architecture, every agent (including the matchmaker if any) is "wrapped" with a sentinel through which all of its in- and out-going message traffic is routed. Sentinels can communicate with each other as well as with the agent reliability database. The distributed nature of this architecture allows us to avoid performance and reliability bottlenecks. The overhead of passing messages through sentinels can be minimized if sentinels are located on the same hosts as their agent "clients". Most EH messages go between a client and its sentinel; messages are interchanged between sentinels only when killing orphaned tasks.

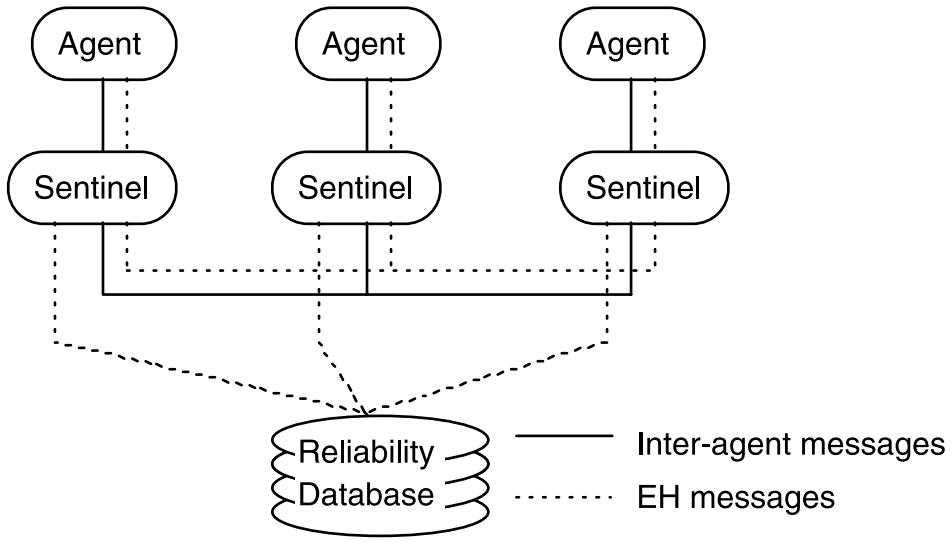

Figure 2. Sentinel architecture for EH service.

The reliability database is accessed only when an agent dies or a sentinel is filtering a set of bids, and standard distributed database techniques can be used to avoid bottlenecks and reliability problems in accessing/updating this information [21]. Sentinel death can be dealt with using techniques such as that described in [22].

The second assumption is that, when agents enter a multi-agent system supported by the EH service, they indicate the kinds of exception handling behavior they can support. This 'EH signature' specifies for that agent how agent death can be detected (i.e. whether or not that agent responds to the "are you alive?" message), how dead subcontractor problems are resolved (i.e. whether or not an agent responds to the "resend RFB" message), how dead customer problems are resolved (i.e. whether the agent allows orphaned task proxying and/or responds to the "cancel task" message), and how dead subcontractor problems are avoided (i.e. whether or not the agent allows bid filtering). 'Full' citizens support all options, while pure survivalists support none. Other agent types come somewhere in between. This allows the EH services to account for the agent heterogeneity we can expect to find in open systems.

Note that we are not claiming that this particular architecture and set of agent death handling strategies is the optimal, or even the only way in which agent death can be offloaded to an EH service. Our claim, rather, is that, at least for this particular exception, the citizen approach can provide significant advantages over survivalist approaches to exception handling in open multi-agent systems.

4. Evaluating the exception handling services approach

We ran a series of experiments to test this claim in a multi-agent system running the CNET protocol. The experiments all take place in a discrete event based MAS simulator built on top of the Swarm Simulation System [23]. The scenario consists of several dozen 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 [19] and was chosen because it is simple and was shown to represent a reasonable design tradeoff in several test domains.

Configuration	# of Agents	Task duration
Short tasks, abundant subcontractors	50	10
Short tasks, scarce subcontractors	16	100
Long tasks, abundant subcontractors	50	10
Long tasks, scarce subcontractors	16	100

Figure 3. Summary of agent configurations tested.

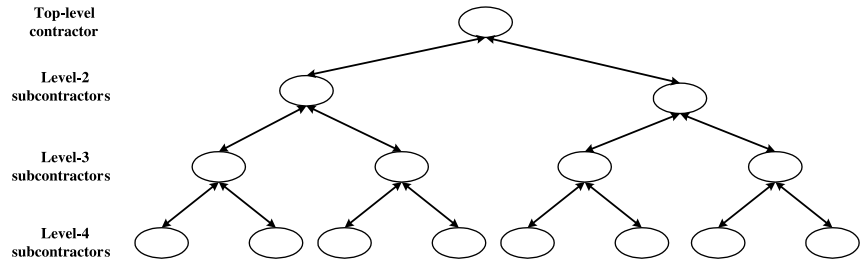

Figure 4. Top-level tasks require the creation of a 4-level task tree.

We designed the experiments to evaluate exception handling performance in a range of domain types. Three independent variables were selected to capture what we judged to be key domain-dependent elements affecting exception handling performance: task tree topology, task length, and agent scarcity. Task completion performance was measured for four different agent configurations, whose parameters are summarized in Figure 3:

In all of these configurations there were three top-level contractors, each executing a loop wherein they announce a new top-level task, wait for bids, award the contract to the best bidder, wait to receive the results and then repeat the above steps. Every top-level task involved the completion of task trees with depth 4 and branching factor 2, thereby requiring the combined contribution of 15 agents (Figure 4),

This allowed us to study the effects of the EH service for tree topologies of differing depths, ranging from ''flat'' (one level of decomposition, as in a client-server setup) to ''deep'' (three levels of decomposition, as we might expect to find in more complex information supply chains). Tree width was not varied because it does not affect total task completion time for any particular agent death instance. To simplify the experiment, all subcontractors were capable of performing any task in a given task tree.

In our initial set of experiments, three simulation runs were performed for each of the four configurations described above:

- Failure-free environment (baseline case)
- Failure-prone environment, ''survivalist'' agents using timeout/retry.
- Failure-prone environment, ''citizen'' agents fully supported by the EH services

In the failure-prone cases, subcontractor agents were divided into three reliability classes. All subcontractor agents had a ''lifespan'' (time until death) selected randomly from a geometric distribution with mean time between failures (MTBF) equal to 10 times the task duration for low reliability agents, 50 times the task duration for medium reliability agents, and 100 times the task duration for high reliability agents. When an agent dies, a new one with the same skills and reliability class but a different unique ID is created and 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 5 below summarizes the mean task completion times for deep task trees, normalized relative to the failure-free (baseline) case, for survivalist and citizen agents in each of the four agent configurations described above.

As expected, citizen agents supported by the EH services produced faster task completion times than survivalist agents. In the deep tree case shown above, mean completion times for citizen agents were as much as 3.4 times faster than for survivalist agents. Remarkably, citizen agents gave times for failure-affected trees no more than 27% longer than the failure-free mean. The advantage of citizen agents was much less dramatic for shallow task trees (not shown above): they were only about 50 to 60% faster than survivalist agents. This is because the 'timeout cascade' effect that plagues survivalist but not citizen agents only appears for deeper task decompositions. The size of the citizen advantage was only mildly affected by task length and subcontractor scarcity for the configurations tested.

We also noted that citizen agents had, for most conditions, a much smaller variation in task completion times than did survivalist agents (Figure 6):

The variance reduction represents, we believe, a significant benefit since in many environments we can expect that consistency will be equally as important as efficiency.

The benefits of the citizen approach are achieved in a way that is well-suited to open multi-agent systems. Recall that the most salient aspect of open systems is that we can make only minimal assumptions about the agents that compose it, since they were not developed under centralized control. Agents are thus likely to be heterogeneous with respect to their exception handling behavior. The EH service can work with a broad range of such behaviors by using the 'EH signature' concept described above. It requires at most that agents support three very simple directives ("are you alive?", "resend RFB", and

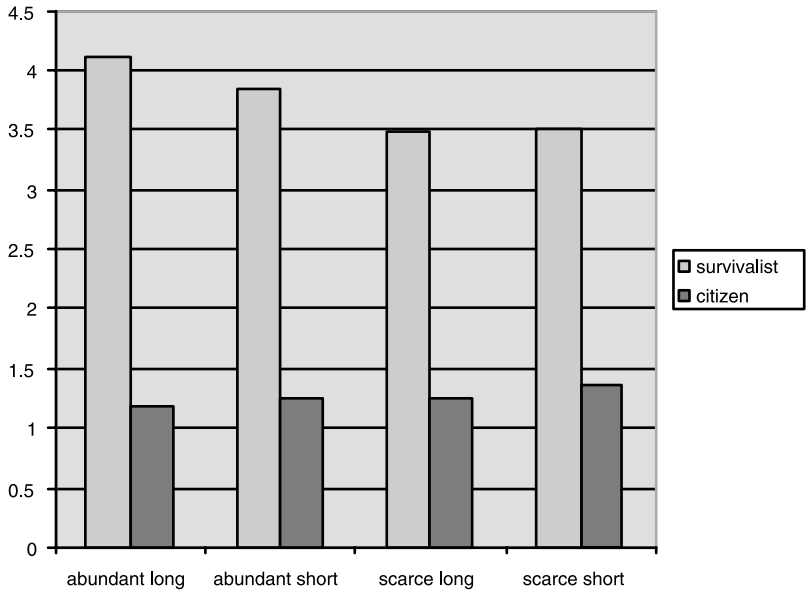

Figure 5. Normalized mean task completion times.

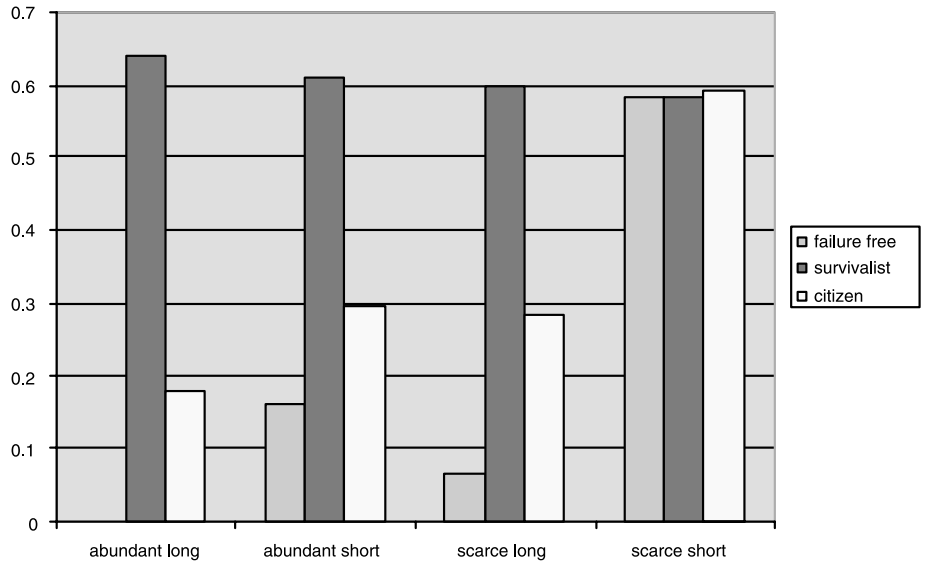

Figure 6. Normalized standard deviations of task completion times.

"cancel-task") and can provide significant support for agents (e.g. survivalists) that implement none of these messages, since bid filtering and orphaned task proxying operate in a way that is completely transparent to the affected agents. The EH service can use timeouts to detect agent death if the agent does not support the "are you alive?" message, avoiding timeout cascades by being aware of the inter-agent task commitment structure. The EH service can notify an agents' customer to resend an RFB if the agent itself does not support the "resend-RFB" message. Finally, if an agent does not support "cancel-task", at worse we are unable to avoid wasting some computational resources. We can expect that the degree of benefits derived from the EH service will be a function of which subset of the full 'EH signature' the agents support, which is born out by our experiments (Figure 7):

These experiments explored the mean task completion time performance for the "long tasks, abundant subcontractors, deep trees" configuration for agents that support differing subsets of the maximal EH contract, normalized relative to the failure free case. As we can see, mean completion times decrease as the scope of the EH signature supported by the agent increases: the 'full citizen' (with polling, proxying and bid filtering) is the fastest.

5. Contributions of this work

Narrowly construed, this work offers improved failure tolerance in open MAS. Almost all previous efforts in the MAS community have taken a "survivalist" approach, or (e.g. [6, 24–27]) have been specific to a small range of exceptions or coordination mechanisms. More general techniques (notably mirroring and rollbacks [28, 29]) from the distributed systems community are not well suited to open systems in that they assume cooperative behavior by all agents (e.g. that agents mirror their state or cooperate with rollback

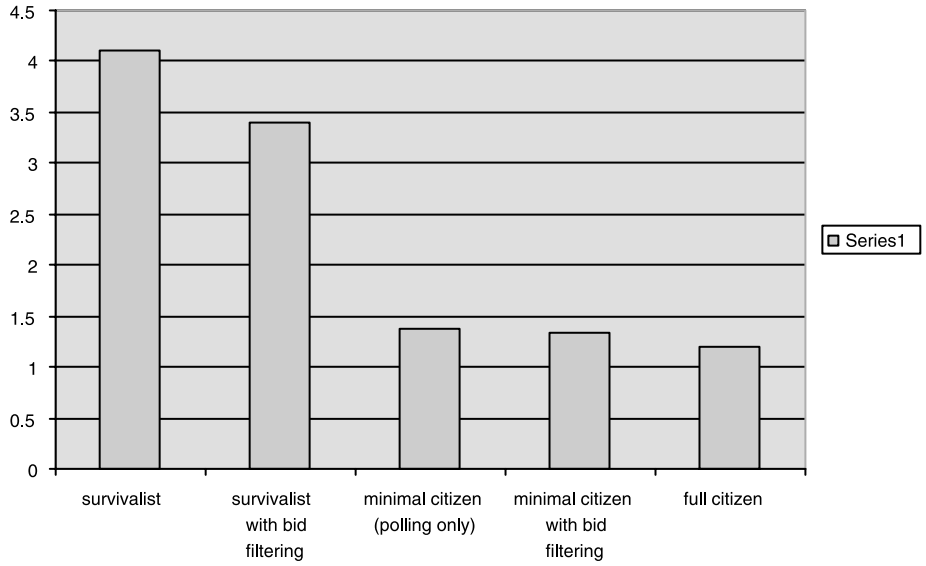

Figure 7. Normalized mean task completion times as a function of EH options supported.

directives) and can be inefficient (full rollbacks discard already completed work). Our approach avoids these limitations. More broadly speaking, this work presents a detailed example of the potential value of a domain-independent EH services approach to increasing robustness in open agent systems.

6. Future work

We are pursuing two concurrent lines of work: [1] defining techniques for handling other important exception types, particularly in market-based mechanisms, and [2] exploring diagnosis techniques suited for open systems where agents are black boxes and may lie.

References

1. *Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces*, Edinburgh, Scotland, 1999.
2. K. Fischer, et al., "Intelligent agents in virtual enterprises," in *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, Blackpool, UK, 1996.
3. M. B. Tsvetovatyy, et al., "MAGMA: An agent-based virtual marketplace for electronic commerce," *Applied Artificial Intelligence*, vol. 11, no. 6, pp. 501–524, 1997.
4. N. R. Jennings, K. Sycara, and M. Wooldrige, "A roadmap of agent research and development," *Autonomus Agents and Multi-Agent Systems*, vol. 1, pp. 275–306, 1998.
5. M. Wooldridge, N. R. Jennings, and D. Kinny, "A methodology for agent-oriented analysis and design," in *Proceedings of the Third Annual Conference on Autonomous Agents (AA-99)*, Seattle WA USA: ACM Press, 1999.
6. S. Hägg, "A sentinel approach to fault handling in multi-agent systems," in *Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96)*, Cairns, Australia, 1996.

7. M. Youssefmir and B. Huberman, "Resource contention in multi-agent systems," in *First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, CA, USA: AAAI Press, 1995.
8. J. D. Sterman, *Learning in and About Complex Systems*, Alfred P. Sloan School of Management, Massachusetts Institute of Technology: Cambridge, Mass., 1994, 51.
9. 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.
10. M. Waldrop, "Computers amplify black monday," *Science*, vol. 238, pp. 602–604, 1987.
11. T. Sandholm, S. Sikka, and S. Norden, "Algorithms for optimizing leveled commitment contracts," in *Proceedings of IJCAI-99*, Stockholm, Sweden, 1999.
12. T. R. Gruber, "A method for acquiring strategic knowledge," *Knowledge Acquisition*, vol. 1, no. 3, pp. 255–277, 1989.
13. J. A. Barnett, "How much is control knowledge worth? A primitive example," *Artificial Intelligence*, vol. 22, no. 1, pp. 77–89, 1984.
14. M. Klein, "Conflict resolution in cooperative design," Ph.D. thesis, Computer Science, University of Illinois: Urbana-Champaign, IL, 1989.
15. M. Klein, "Supporting conflict resolution in cooperative design systems," *IEEE Systems Man and Cybernetics*, vol. 21, no. 6, pp. 1379–1390, 1991.
16. M. Klein, *Exception Handling in Process Enactment Systems*, MIT Center for Coordination Science: Cambridge MA, 1997.
17. M. Klein and C. Dellarocas, *Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems*, Massachusetts Institute of Technology: Cambridge MA USA, 2000.
18. M. Klein and C. Dellarocas, *Towards a Systematic Repository of Knowledge about Managing Multi-Agent System Exceptions*, Massachusetts Institute of Technology: Cambridge MA USA, 2000.
19. R. G. Smith and R. Davis, "Distributed problem solving: The contract net approach," in *Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978.
20. C. Dellarocas and M. Klein, "An experimental evaluation of domain-independent fault handling services in open multi-agent systems," in *Proceedings of The International Conference on Multi-Agent Systems (ICMAS-2000)*, Boston, MA, 2000.
21. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann series in data management systems, Morgan Kaufmann Publishers. xxxii, 1070: San Mateo, Calif. USA, 1993.
22. S. Kumar, P. R. Cohen, and H. J. Levesque, "The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams," in *International Conference on Multi-Agent Systems (ICMAS-2000)*, Boston MA USA, 2000.
23. N. Minar, et al., *The Swarm Simulation System: A Toolkit for Building Multi-Agent Systems*, Santa Fe Institute: Santa Fe, New Mexico, USA, 1996.
24. 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 (AAAI-98)*, 1998.
25. M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, vol. 7, pp. 83–124, 1997.
26. B. Horling, et al., *Diagnosis as an Integral Part of Multi-Agent Adaptability*, University of Massachussets at Amherst Department of Computer Science: Amherst, Massachussets, 1999.
27. M. Venkatraman and M. P. Singh, "Verifying compliance with commitment protocols: Enabling open web-based multiagent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 3, 1999.
28. A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages*, Addison-Wesley, 1996.
29. S. J. Mullender, *Distributed Systems*, second edition, ACM Press: New York, 1993.