An Experimental Analysis of Exception Handling Services for Multi-Agent Systems

by

Athicha Muthitacharoen

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science

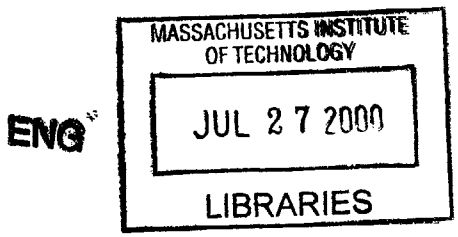
and

Bachelor of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000



ENG

Author .....

Department of Electrical Engineering and Computer Science

May 17, 2000

Certified by.....

.....

Chrysanthos Dellarocas

Douglas Drane Career Development Assistant Professor of Management

Thesis Supervisor

Accepted by .....

.....

Arthur C. Smith

Chairman, Department Committee on Graduate Students

An Experimental Analysis of Exception Handling Services for Multi-Agent Systems

by

Athicha Muthitacharoen

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2000, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering and Computer Science
and
Bachelor of Science in Computer Science

Abstract

This thesis evaluates the performance of exception handling services operating in multi-agent systems. The hypothesis is that agent systems with exception handling services will outperform agent systems where each individual agent detect and resolve errors on their own. Agents communicate using the Contract Net Protocol [16, 3]. The experiments focused on one exception, namely "agent death". Four configurations of agent systems have been simulated under three exception handling strategies (Simple Survivalist, Simple Citizen, Complex Citizen) and one base case (no exceptions). The results showed that exception handling service is beneficial to agent system performance in the average case. The result was quite remarkable in the long task case. However, system performance under exception handling service is not very consistent. Future work is required to resolve the consistency problem.

Thesis Supervisor: Chrysanthos Dellarocas
Title: Douglas Drane Career Development Assistant Professor of Management

Acknowledgments

I am blessed to be surrounded by numerous people, all of whom I am grateful for.

I would like to thank my thesis supervisors, Dr. Mark Klein and Professor Chrysanthos Dellarocas, who supervised and sponsored the work done for this thesis. Mark has been a truly devoted mentor, whose insight has guided me through uncertainty in the world of research. I have many thanks for Juan Rodríguez-Aguilar for spending endless days and nights building and polishing the simulation system. I would also like to thank Lijin Aryananda and David Shue for their work on the original simulation system, and for their patience while answering my countless questions.

Many many thanks to Matthew Grein for always being there whenever I need an ally. Thanks to Joy Rusmintratip–my spiritual advisor, and Tai - Nitsara Karoonuthaisiri for being the one to whom I can confess anything.

This thesis is dedicated to my parents, Amnuay and Prapaiwan Muthitacharoen. Thanks for making my future your first priority.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The complexity in open multi-agent systems

Software agents often work together to carry out complicated task execution. In a system where a lot of interaction among different components persist, there tend to be high complexity within. Solving errors once they occur in multi-agent systems, thus becomes an arduous task. Agents can die without a warning, resources can be taken up by idle agents, deadlock can occur in the system, communication links can break down, for instance. The problem poses a serious threat in open multi-agent systems, where agents with different implementations communicate and coordinate.

Due the variety and complexity of these problems, agents in dynamic systems have to be equipped with their own problem solving strategies, which has made implementing agents a complicated effort. We have hypothesized that these exceptions can be resolved much more efficiently if they are handled by the appropriate system component.

1.2 Coordination Mechanism: The Contract-Net Protocol

In an environment with limited resources, agents must coordinate their activities with each other to further their own interests or satisfy group goals [6]. Therefore, agents in a dynamic society need a common protocol in order to coordinate with each other.

The Contract Net Protocol (CNP) [16, 3] was originally developed for use in distributed sensor systems. The Protocol consists of contractor agents and subcontractor agents. When a contractor is assigned a task that it does not have the ability to carry out, it announces the task qualification to a system of agents. Agents who qualify for the task and are free to do the job will submit bids to the contractor. When the expiration time of the task announcement has been reached, the contractor then reviews the bids and awards the task to the agent with the most promising qualifications. If no bids have been received after the task announcement expires, the contractor may wait for a time interval before reinitiating the bidding process. Waiting before re-announcing the task increases the chance of agents submitting bids because it allows qualified, but busy, agents to finish executing their tasks and become available for bidding. If a subcontractor is assigned a task that is composed of subtasks which it does not have the skills to complete, it takes the role of a contractor agent and begin the search for qualifying agents.

Because it serves as a foundation of many other coordination protocols, the Contract Net is one of the most important paradigms developed in distributed artificial intelligence(DAI). The agents in this experiment thus coordinate under this protocol.

1.2.1 Extension to the Contract Net Protocol

In order to design experiments for this thesis, and because it has not been specified in the original paper [16, 3], some assumptions have to be made regarding the contract

net protocol. They are listed below:

1. Subcontractors can be involved in negotiation of only one task at a time, and can only execute one task at a time.

2. When a subcontractor is waiting for any sort of response from the contractor during negotiation phase, if the waiting period has passed and no response has been received, then the subcontractor can abandon the negotiation and free up itself.

Figure 1-1 outlines the Contract Net Protocol.

1.3 Approaches to Tolerate Failure

There have been several attempts to come up with the most effective approach to handle exceptions in multi-agent systems. (See Chapter 4 on related works.) Most of the current work on exception handling under the Contract Net Protocol are oriented towards solving domain-specific problems [14, 17]. This thesis, on the other hand, focuses on generic exception handling (non-domain-specific). Two approaches to tolerate failure are presented below: one is a classical approach from the creators of the CNP [3], the other is the approach that forms the hypothesis of this thesis.

1.3.1 Survivalist Agents: Self-contained Agents

Survivalist agents have their own error detection and resolution mechanisms. Some agents have prevention-avoidance mechanisms built-in to their coordination protocol [17]. While it results in effective exception handling capabilities, survivalist agents are complicated to implement due to the extra exception handling component. Furthermore, message overhead problems might arise due to messages sent by each survivalist agents while performing detection and prevention mechanisms.

Originally, survivalist agents operating under the CNP adopt the timeout-retry mechanism when faced with an error. This mechanism is the simplest form of exception

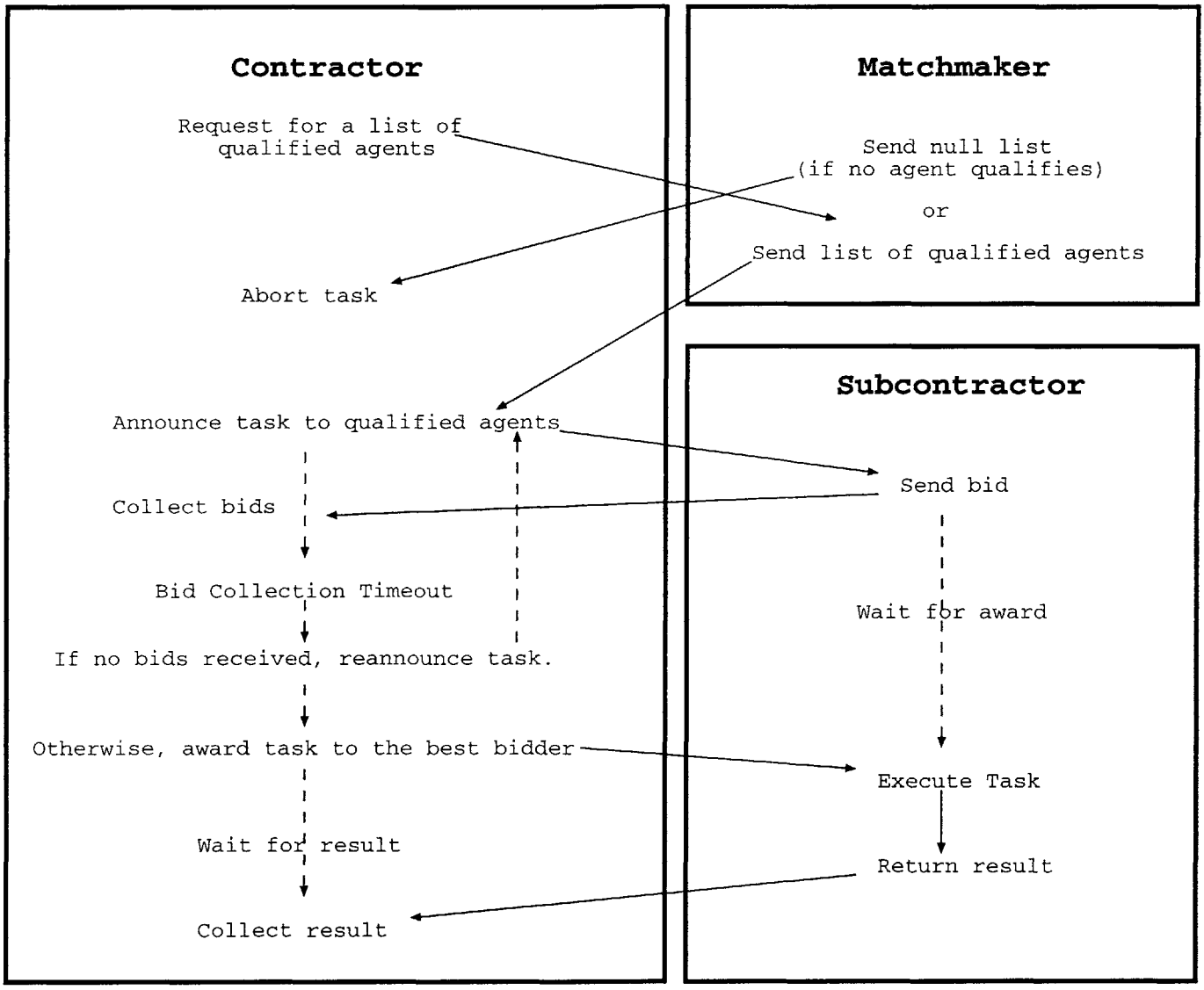


Figure 1-1: The Contract Net Protocol

handling. An agent detects an error by simply waiting for a reasonable amount of time, if nothing happens (when it is supposed to), then the agent assumes that there has been an error. The resolution is to restart the protocol again.

To illustrate the mechanism in terms of the CNP, take for example, an contractor agent who is waiting for the result from its subcontractor agent. If the deadline for task completion that was promised by the subcontractor has passed, then the contractor assumes that the result will never arrive (due to the subcontractor's death, communication failure, etc.). The contractor simply restarts the process of finding the next available and eligible subcontractor.

1.3.2 Citizen Agents: Systems with Exception Handling Service

There is a special class of software agents that is used to anticipate, avoid, detect, and resolve conflicts among multiple agents in the system called the exception handling (EH) agents. These agents detect problems in the system and prescribe solutions by accessing a knowledge base of possible solutions to various problems. EH agents also anticipate exceptions and adjust the infrastructure to avoid them. Klein and Dellarocas [8] have hypothesized that systems with multiple agents can perform better with citizen agents using shared exception handling services (provided by EH agents) than survivalist agents (those without the help of EH agents).

Citizen agents operate in systems that include EH agents. The implementation of a citizen agent does not necessarily include the exception handling component, unlike the case of a survivalist agent. Nevertheless, citizen agents need to be able to communicate with EH agents in order to exchange diagnostic information, and receive instructions should an exception arise. For a detailed description of the architecture of the exception handling service, see [8] and [1].

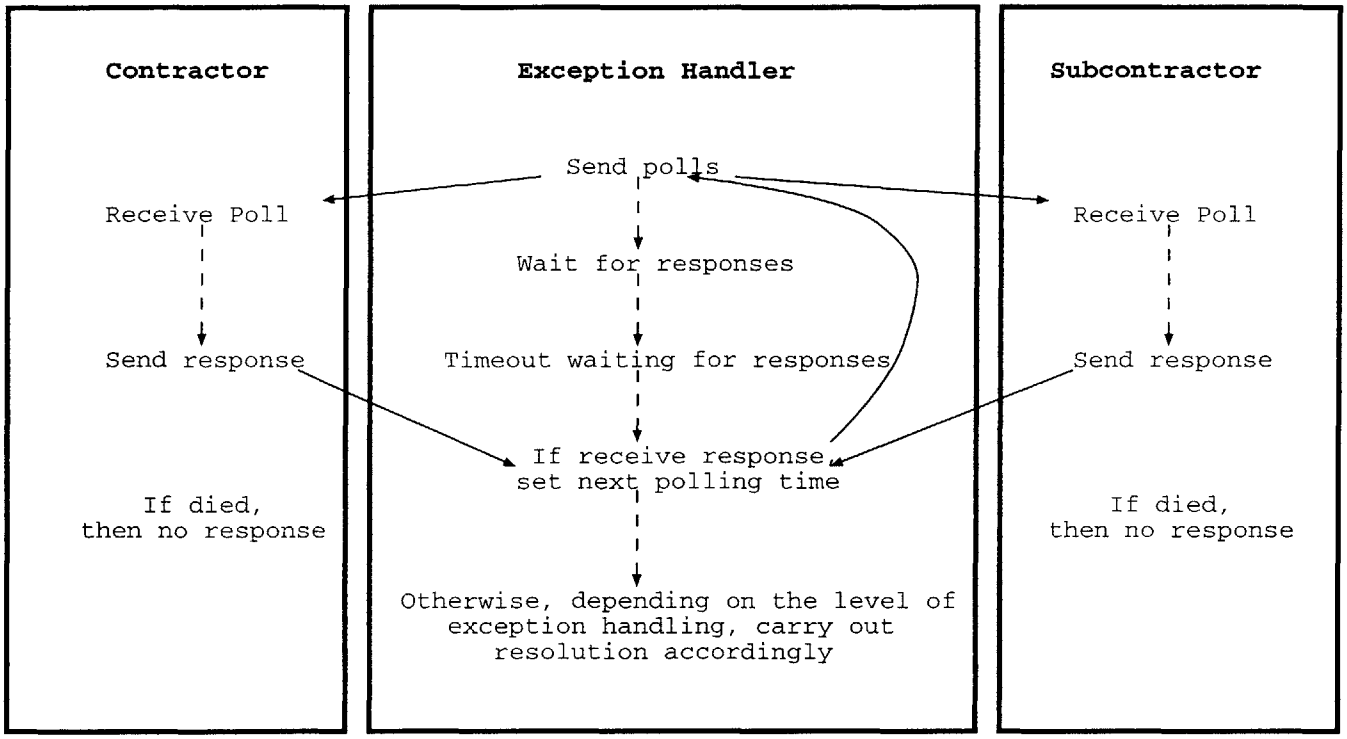


Figure 1-2: General Exception Handling Service – Agent Death Exception

Figure 1-2 outlines the general exception handling service for the agent death exception.

1.4 Thesis Goal

This thesis attempts to identify interesting regions within a design space of the Contract Net marketplaces, and explore the effects of exception handling service for each region. Our hypothesis is that the exception handling service will improve the performance of multi-agent systems. The parameters that compose the design space are explained in Chapter 2. The results will be a potentially useful compilation of the different regions within a design space of the CNP agent community, and an analysis of the performance of survivalist versus citizen agents in those regions.

1.5 Outline

The rest of the thesis is organized as follows: chapter 2 describes the rationale of the experiment design; chapter 3 includes the evaluation criteria and outline the results and the analysis; contributions to related work and future extensions are discussed in chapters 4 and 5, respectively.

Chapter 2

Experiment Design

2.1 Survey of Multi-Agent Systems Domain

The experiments in this thesis are designed in the attempt to resemble existing multi-agent systems. In order to accomplish the intention above, several literatures have been reviewed to explore the possible domains of agent marketplaces. Currently, there are four major domains: electronic commerce, information servers, supply chain, and virtual enterprise. Each of which is described below:

2.1.1 Electronic Commerce

Activities in the electronic commerce domain comprise of one-to-one on-line purchase transactions, possibly with an extended series of interactions to carry on a negotiation. Customers (contractors) initiate the purchase transaction, with the requirements for purchase (i.e. maximum price requirement, quality of goods). Retailers (subcontractors) reply with the details of their products (i.e. lowest price estimate). The contractor can then pick the most desirable subcontractor, if there is any. Otherwise, the contractor can re-announce the purchase transaction in the hope that additional subcontractors will reply, or existing subcontractors will lower their prices (or increase product quality, etc.). Once a subcontractor is chosen to engage in transaction, the task is a simple exchange of money and goods. Task decomposition trees within this

domain are one level deep with only one branch. Each contractor and subcontractor can be modelled as an agent, whose goal is to complete purchase transactions according to some constraints. Due to the distributed nature of the internet, there are currently a large number of on-line customers and retailers.

2.1.2 Information Servers

In this domain, customers (contractors) request information from web servers (subcontractors). The tasks are simply information retrieval activities. Web servers (which provide the information) have largely unique capabilities, and handle a large number of short tasks. The task structure for this domain has depth of one, and can have many leaf tasks. The parent task, which is to analyze information, takes a moderate amount of processing time. The leaf tasks, which are purely information retrieval, take up much less processing time. Since there are not that many qualified subcontractors, and because subcontractors can only process one task at a time (see section 1.2.1), the contractor needs to wait long enough for bids from any potentially busy subcontractor.

2.1.3 Supply Chain

The supply chain model consists of agents delivering products from suppliers to customers. These products can be used as inputs for some other larger, more complicated products, as in the case of manufacturing shop floor control[9]. On the other hand, the purpose of delivery can be just to transport goods to consumers, as in transportation scheduling. The tasks in this model consists of relatively large tasks in high volumes. The task decomposition trees have multiple levels (since the nature of the tasks in this model is concerned with transporting objects from source nodes, which can be more than two levels from the destination node), and a small number of branches (since there is not much variety in the source nodes).

2.1.4 Virtual Enterprise

This domain refers to a type of agent community where a vast number of heterogeneous agents coordinate (potentially on-line) in a virtual organization to carry out complicated tasks. Examples of applications in this domain are coalition operations, rescue operations, financial institutions and large scale collaborative designs, for instance. Tasks in this model can include subtasks which belong to any of the domains mentioned above. Task completion times range in size from small (10-20 times communication delay) to large (300-500 times communication delay). Task decomposition trees also range in size from moderate to large in both depth and breadth.

2.2 Exception Handling Strategy

We only consider the "agent death" exception. The reason for selecting this particular type of exception is that it is domain-independent, is fairly common among agent systems, and poses a significant performance degradation.

The basis for these exception handling strategies are based on the survey of approaches to tolerate failure in section 1.3.

2.2.1 Survivalist Strategy

In this strategy, each agent avoids and resolves exceptions on its own, using timeout-retry mechanism. We tested the case of **Simple Survivalist**. In this case, if a subcontractor does not deliver results by the deadline set by a contractor, the contractor simply restarts the contract.

2.2.2 Citizen Strategy

Citizen agents rely on central monitoring agent(s) (Exception Handling (EH) Agents) to detect and resolve errors. EH Agents can also prevent and avoid exceptions in the system. This thesis will focus only on detecting and resolving errors, and not

on prevention or avoidance. The following are two levels of exception handling for citizen agents:

I **Simple Citizen:** When the exception handler detects an agent death, it instructs the corresponding contractor and subcontractor to abort or re-announce properly in order to avoid abandoned or wasted tasks.

II **Complex Citizen:** Once an agent dies, the exception handler:

- Creates a proxy agent for every subcontractor (if any) of the dead agent. The proxy agent should have the same skills as the subcontractor to which it correspond.
- Instructs the subcontractors of the dead agent (if any) to send the results to their corresponding proxy agents.
- Informs the dead agent's contractor (if any) to reannounce.

See Appendix A for more details on different exceptions, and their detection-resolution mechanisms.

2.3 Experiment Parameters

From the survey in section 2.1, we can extract several important features of multi-agent systems. General characteristics of a multi-agent system can be defined using three independent parameters:

2.3.1 Task Tree Topology

Each contractor agent is assigned a task at the beginning of each experiment. Each task that an agent has to execute is represented by a tree. The structure of the tree represents the complexity of the task (ie. how many subtasks are needed to complete the whole task, and how much dependency exists among the subtasks). There are both vertical and horizontal dependencies among subtasks in a task decomposition tree.

Since the agent death exception have the most impact on tasks that are vertically-related, the experiments in this thesis only include the vertical dependency between subtasks. In addition, since short task chains (flat tree) are a subset of long task chains (deep trees), we consider one type of task tree–**Deep and Narrow (depth = 4, branch = 2)**.

2.3.2 Task Execution Time

Task execution time can be determined relative to the time needed to communicate among agents in the system (network latency).

a) Long execution time (500 * network latency)

b) Short task execution time (50 * network latency)

2.3.3 Subcontractor Resource-boundedness

Since the number of subcontractors with a particular skill varies depending on the type of agent system, we use the subcontractor resource-boundedness to account for this characteristics in the experiments.

a) Abundant skills (many qualified subcontractors available to perform a task at one time)

b) Scarce skills (few qualified subcontractors available to perform a task at one time)

Table 2.1: Experiment Configurations

Experiment Parameters	Task Tree	Number of Subcontractors	Task Duration
deep & narrow short task scarce skills	branch = 2 depth = 4	18	1000
deep & narrow short task abundant skills	branch = 2 depth = 4	48	1000
deep & narrow long task scarce skills	branch = 2 depth = 4	18	10000
deep & narrow long task abundant skills	branch = 2 depth = 4	48	10000

2.3.4 Exception Handling Strategy

In addition to the variables that constructs the design space, another variable is used to indicate the presence of exceptions, and exception handling service.

a) Base case: No exceptions

b) Exceptions with Simple Survivalist

c) Exceptions with Simple Citizen

d) Exceptions with Complex Citizen

All experiments are run until the variance of the task completion time falls within a predetermined confidence interval. The rationale is that we would like the simulation to run until the system is stable, so any measurements we make is a good representation of the system. The experiments are simulated in a discrete event-based multi-agent system simulator implemented by Juan Rodríguez-Aguilar. The simulator is built on top of the Swarm Simulation System (Minar, Burkhart, Langton and Askenazi [12]). The initial design was based on the theses of two former Master stu-

dents: Lijin Aryananda [1] and David Shue [15], under the supervision of Professor Chrysanthos Dellarocas and Dr. Mark Klein.

Chapter 3

Experiment Analysis

This chapter defines the criteria for analysis as well as outline the results from the experiments and their evaluation.

3.1 Evaluation Criteria

Performance of the citizen and survivalist cases will be compared according to the following dependent variables:

3.1.1 Task Completion Time

The task in this section refers to the top-level task assigned to each contractor when the contractor becomes active. The task completion time when there is no exception in the system is the shortest time that agents in a particular system configuration could achieve. However, when errors do occur, it may disrupt task processing, causing some of the subcontractors to submit their results late. This, in turn, will delay the completion time of the top-level task. We use the following variables to observe this behavior and compare it across different exception handling strategies:

a) **Maximum** Refers to the maximum top-level task completion time among all task instances and all contractors in a single simulation.

b) **Average** Refers to the average top-level task completion time across all task instances and all contractors in a single simulation.

c) **Variance** Refers to the variance of the top-level task completion time across all task instances and all contractors in a single simulation.

3.2 Results and Evaluation

3.2.1 Deep Trees, Short Tasks

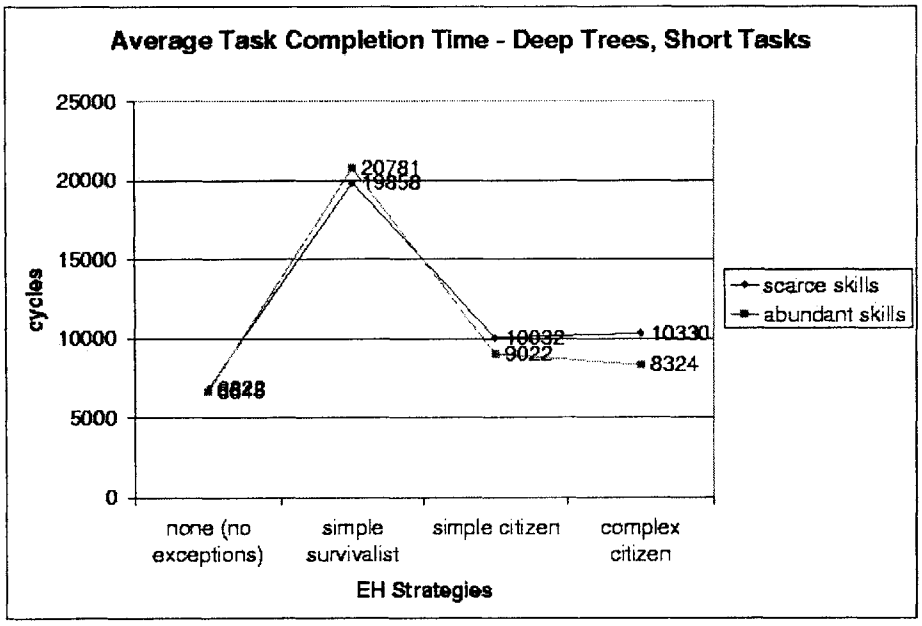


Figure 3-1: Average Task Completion Time for Deep Trees and Short Tasks

According to Figure 3-1, the average completion time of the tasks improved in both of the citizen cases, compared to the simple survivalist case. Citizen agents finished tasks 2-3 times faster than survivalists agents. Between the two citizen cases, complex citizens performed roughly as well as simple citizens in the scarce agents case, and

slightly better than the simple citizens in the abundant agent case. Because the task process time is short, it is less likely that an agent who dies will have subcontractors performing jobs for it at the time of death (the subcontractors could finish the tasks and return the results to the contractor in a short time). Therefore, this configuration is less likely to benefit from the complex citizens' strategy of saving orphaned tasks.

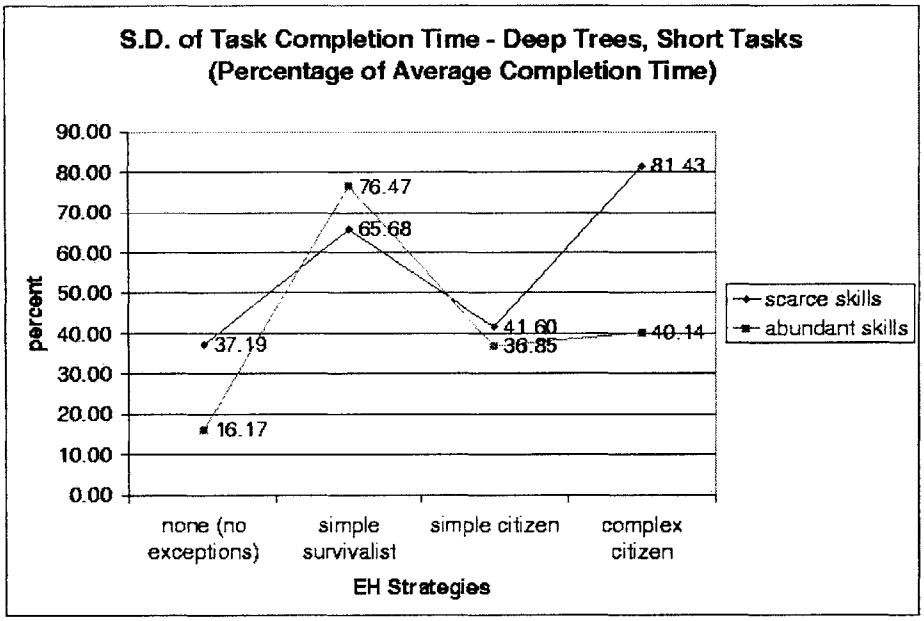


Figure 3-2: Standard Deviation of Completion Time for Deep Trees and Short Tasks

The standard deviation (Figure 3-2) follows almost the same pattern as the average task completion time. The significant difference is that the S.D. in the scarce agent case of complex citizen agents is much higher than that of simple citizen agents, even higher than survivalist agents. Scarce agents have a more fluctuating task completion time than abundant agents in the citizen cases. The magnitude of the S.D. of the scarce and complex citizens is large due to the highly fluctuating nature of agents in this case. When the complex citizens' strategy of salvaging orphaned tasks succeeded, the performance was greatly improved. However, when the strategy failed,

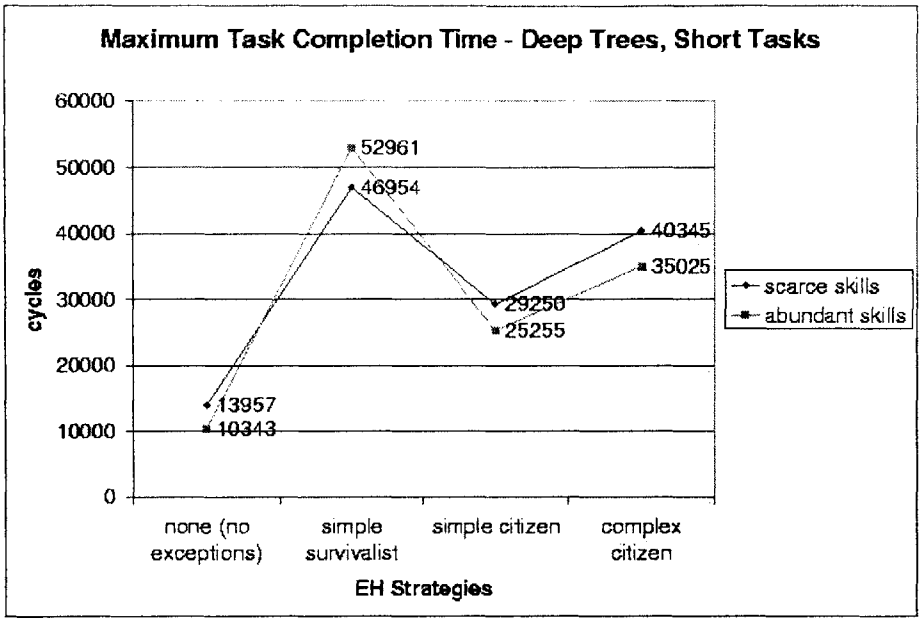


Figure 3-3: Maximum Task Completion Time for Deep Trees and Short Tasks

system performance severely declined, resulting in a high standard deviation. A plausible description of why the complex citizen strategy failed was due to some extreme cases, where an agent who was at a very high level in the task execution chain died after having received all the results from its subcontractors. Complex citizens could not apply the task salvaging strategy, since there was no subcontractor processing tasks from the dead agent. All the results, thus, are lost with the dead agent.

The results for the maximum task completion time (Figure 3-3) followed the same pattern as the standard deviation (Figure 3-2). Abundant citizen agents outperformed scarce citizen agents. Simple citizens outperformed complex citizens, which in turn, outperformed simple survivalists. Although the shape of the graph does not exactly resemble that of the standard deviation plot, they have similar qualitative interpretations.

3.2.2 Deep Trees, Long Tasks

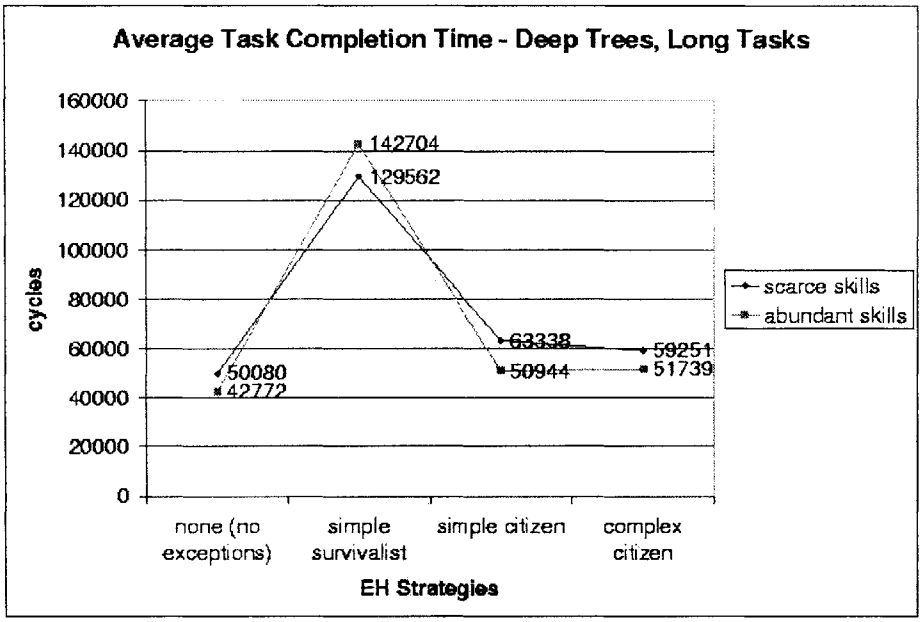


Figure 3-4: Average Task Completion Time for Deep Trees and Long Tasks

Citizen agents significantly reduced the average task completion time that the agent system achieved under the survivalist strategy. The improvement is remarkable in the case where there is a long chain of task execution (Figure 3-4). The average task completion time of citizen agents turned out to be only 18-19 percent longer than that of the base (failure-free) case.

Complex citizens delivered slightly better performance than simple citizens. This is surprising, since we were expecting complex citizens to dramatically improve the results compared to simple citizens. A plausible explanation for this phenomenon follows the description of why citizen agents fail in Section 3.2.1.

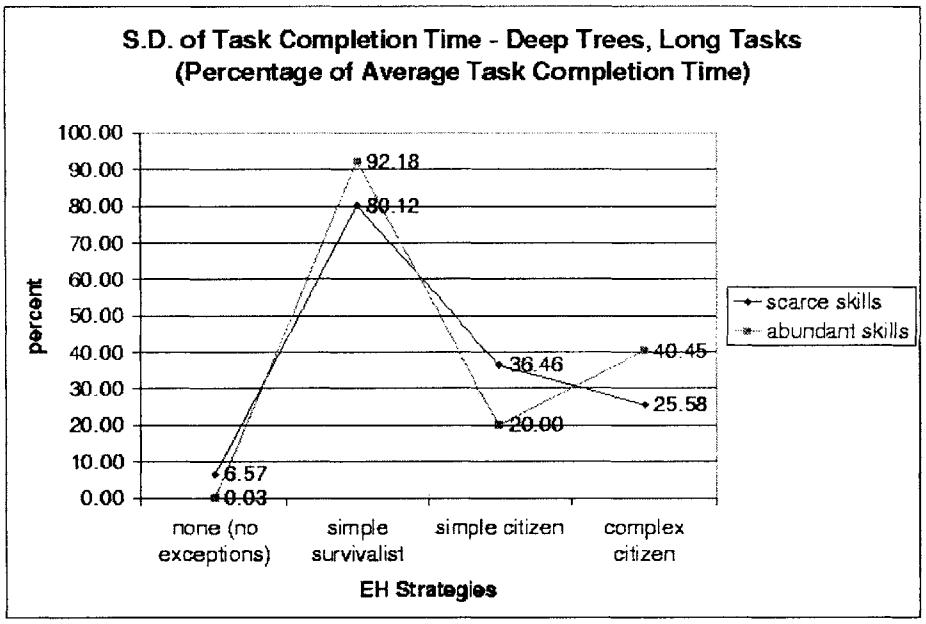


Figure 3-5: S.D. of Task Completion Time for Deep Trees and Long Tasks

The standard deviation (Figure 3-5) and the maximum task completion time (Figure 3-6) follow almost the same pattern as the average task completion time. The significant difference is that the S.D. and the maximum task completion time for complex citizens is higher than that of simple citizens in the abundant agents case, contrary to what happened in the scarce agents case. Citizen agents are more beneficial to scarce agents, because their strategy is to utilize scarce resource (by avoiding the reprocessing of a task in an environment where eligible agents are hard to find). In the abundant case, however, salvaging orphaned tasks means more free agents, which in turn will result in more task announcement and bidding messages. The increase in communication delay generated by additional message passing can contribute to the increase in size of the maximum completion time when an exception occurs. This, in turn, will increase the fluctuation of the task completion time, resulting in a higher standard deviation.

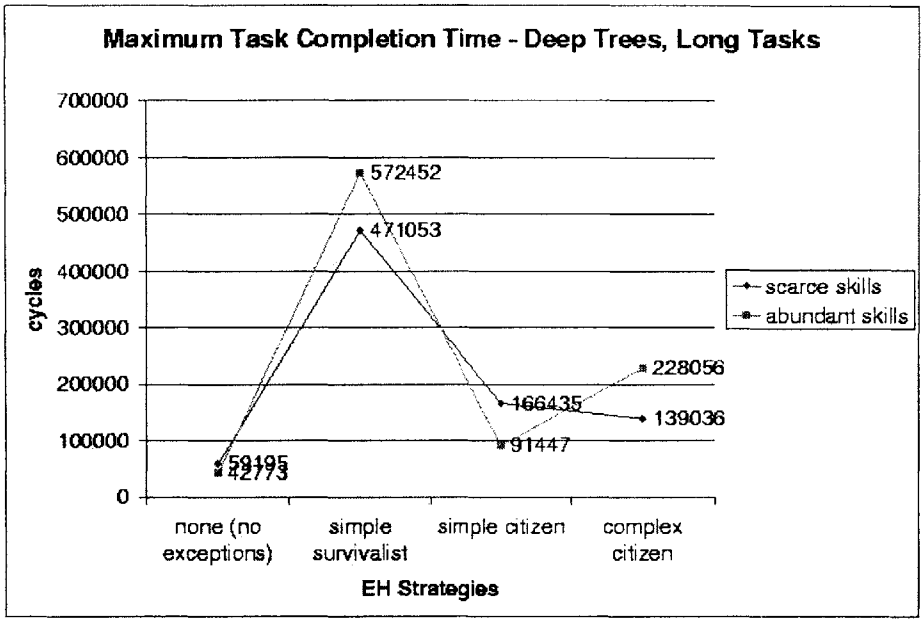


Figure 3-6: Maximum Task Completion Time for Deep Trees and Long Tasks

The reader may notice that in every configuration and every variable measured, scarce agents always outperformed abundant agents under the simple survivalist stategy. This behavior was unexpected. Abundant agents should be able to accomodate the loss of an agent much better than scarce agents due to the high availability of free agents. No plausible explanation exists at the moment. We will have to asses the statistical significance of this behavior.

3.3 Summary

The experiments demonstrated that the exception handling service generally improved system performance. The result was quite remarkable in the case with long task completion time. The average task completion time in the error-prone cases was reduced to within 20 percent of that in the failure-free case. The improvement was

still evident in the short tasks case, though not as significant as the long tasks case.

Consistency of the results is still unsatisfactory. Complex citizens delivered fluctuating results compared to simple citizens, contrary to our expectation. One plausible explanation for this behavior is stated in the standard deviation analysis in Section 3.2.1. Further work is required to investigate the consistency problem.

Chapter 4

Contribution to Related Works

4.1 Related Works

There has been several research projects on the effect of exception handling in multi-agent systems. The following is a summary and comparison of the most relevant efforts against the approach taken in this thesis.

4.1.1 SAM: Socially Attentive Monitoring

Kaminka and Tambe [7] proposed a domain-independent approach to monitoring and diagnosis for multi-agent domains called SAM (Socially Attentive Monitoring). There are three essential parts in this approach: first, its failure detection technique, influenced by social psychology (specifically Social Comparison Theory), utilizes other agents as information sources and detects failures both in agent and in its teammates; second, SAM uses an explicit model of teamwork to reason about the failures in its team (social diagnosis); finally, SAM employs model sharing to lessen the inherent inefficencies associated with representing multiple agent models.

According to Social Comparison theory, differences with other agents are meaningful only to the extent that agents are *socially similar*. If agents are heterogeneous, they may not be able to contribute relevant information towards the monitoring of each

agent's performance. Furthermore, hostile agents may intentionally deceive other agents in order to influence the agents' decision making in order to advance their own agendas. Since SAM uses comparison with peers to detect potential errors, agent homogeneity is assumed within the system. Unfortunately, this assumption does not hold in domains with heterogeneous agents.

The experiments to evaluate the performance of SAM were constructed as to determine whether SAM has succeeded in detecting and/or diagnosing the problem. The results from the experiment helped determine which type of agents should be running SAM in order that the detection and diagnosing be effective. The nature of the experiments is different from the ones in this thesis, which apply the exception handling service to agent domains and observe the performance of the system in terms of task completion time.

4.1.2 Enterprise: A Market-like Task Scheduler for Distributing Computing Environments

Enterprise was a system for scheduling tasks among processors developed by Malone, Fikes, and Howard [11]. The system used a protocol similar to the Contract Net to assign tasks to processors. It adopted the timeout-retry strategy in dealing with late or absent tasks from subcontractors. The authors introduced three assignment policies for assigning tasks to subcontractors: lazy assignment, eager assignment, and random assignment. A series of simulations were conducted to evaluate the efficiency of the assignment policies. Each simulation was composed of a unique combination of job loads, estimation errors, machine configuration, communication delays, and retry policies. The critria used to evaluate the results were based on average flow time, and maximum flow time.

Although Enterprise used the same communication protocol, and measured the performance of the system by the maximum flow time of the tasks, it differed from the

experiments in this thesis through the independent variables of the experiments. The primary goal of the experiments in Enterprise was to measure the performance of the system due to different task assignment policies, while the primary goal of our experiments is to study the effect of different exception handling mechanisms on the task processing time.

4.2 Contributions

Below are the contributions from this thesis to the field of exception handling service in distributed environments:

4.2.1 Domain-independence

Most of the research related to exception handling in agent systems using the Contract Net Protocol are domain-specific [14, 17, 2, 13]. This thesis focuses on domain-independent agent systems. The exception handling service in this system has a knowledge base of rules on the detection and diagnosis of exceptions, which can contain both general information on error diagnosis in agent systems and information specific to errors found in some domains.

4.2.2 Compilation of Agent Domains

In this thesis, I have reviewed several literatures which described agent systems operating under the Contract Net Protocol. In addition, I also looked at real-life applications of multi-agent systems. From those papers and applications, I have indentified four agent domains, according to the shape of the task decomposition trees. Within a domain, there are subdomains defined by task processing time and agent resource-boundedness. The compilation serves as a guideline by which one could model multi-agent system domains.

4.2.3 Evaluation of Exception Handling Strategies

Using the exception handling service described in [8] and [1], the experiments in this thesis measure the performance of two exception handling strategies: the Simple Citizen approach and the Complex Citizen approach (see section 2.2.2). The experimental results are evaluated against those from the classic timeout-retry (Simple Survivalist) approach (see section 2.2.1).

Chapter 5

Future Extensions

There are several possible ways to improve the experiments and their analysis presented in this thesis.

5.1 Variety of Exceptions

Agent death was the only exception considered for the experiments in this thesis. This is clearly unrealistic, but was adopted for ease in implementing the initial prototype of the simulator, and controlling the behavior of the experiments. In the future, more than one type of exception should be introduced to the system. These exceptions include: message lost, resource poaching, message congestion, delay in task executions, etc. Note that all the exceptions mentioned are domain-independent. Domain-specefic exceptions can also be introduced if the focus is on a particular domain.

5.2 Task topology

So far only the "narrow" task topologies are present in the experiments. Our rationale is that the agent death exception will have the most direct effect on tasks that are vertically-related in the task tree. However, the bigger the number of branches in the tree, the more likely that a complete result from one branch of the tree cannot

be used due to an exception that occured in another branch of the tree. Analyzing results accross different number of task tree branches and depths will provide further insights on the effect of exception handling services, though the simulations can take up a huge amount of resource and time.

5.3 Exception Handling Inference Engine

The EH strategies used in the experiments were hard-coded during implementation. Further development of the exception handler to look up the symptoms and resolutions to a problem is needed. A separate effort on the development of the knowledge base of detection-resolution and prevention-avoidance strategies is also valuable.

5.4 Additional Exception Handling Strategy

We have considered the Complex Survivalist exception handling strategy, but did not test it. Including this strategy in future experiments will help provide a more complete picture of system behavior without exception handling service. Below is a brief description of the strategy (see Appendix A for a complete diagram):

Complex Survivalist: Contractors poll the late subcontractors. Alive (but late) subcontractors respond to the poll with new deadlines for task completion time.

5.5 Coordination Protocols

It would be valuable to study the effect of the exception handling service on agent systems operating under communication protocols other than the Contract Net. This can bring up new exceptions that may have not occured under the Contract Net Protocol.

Appendix A

Exception Handling Strategies

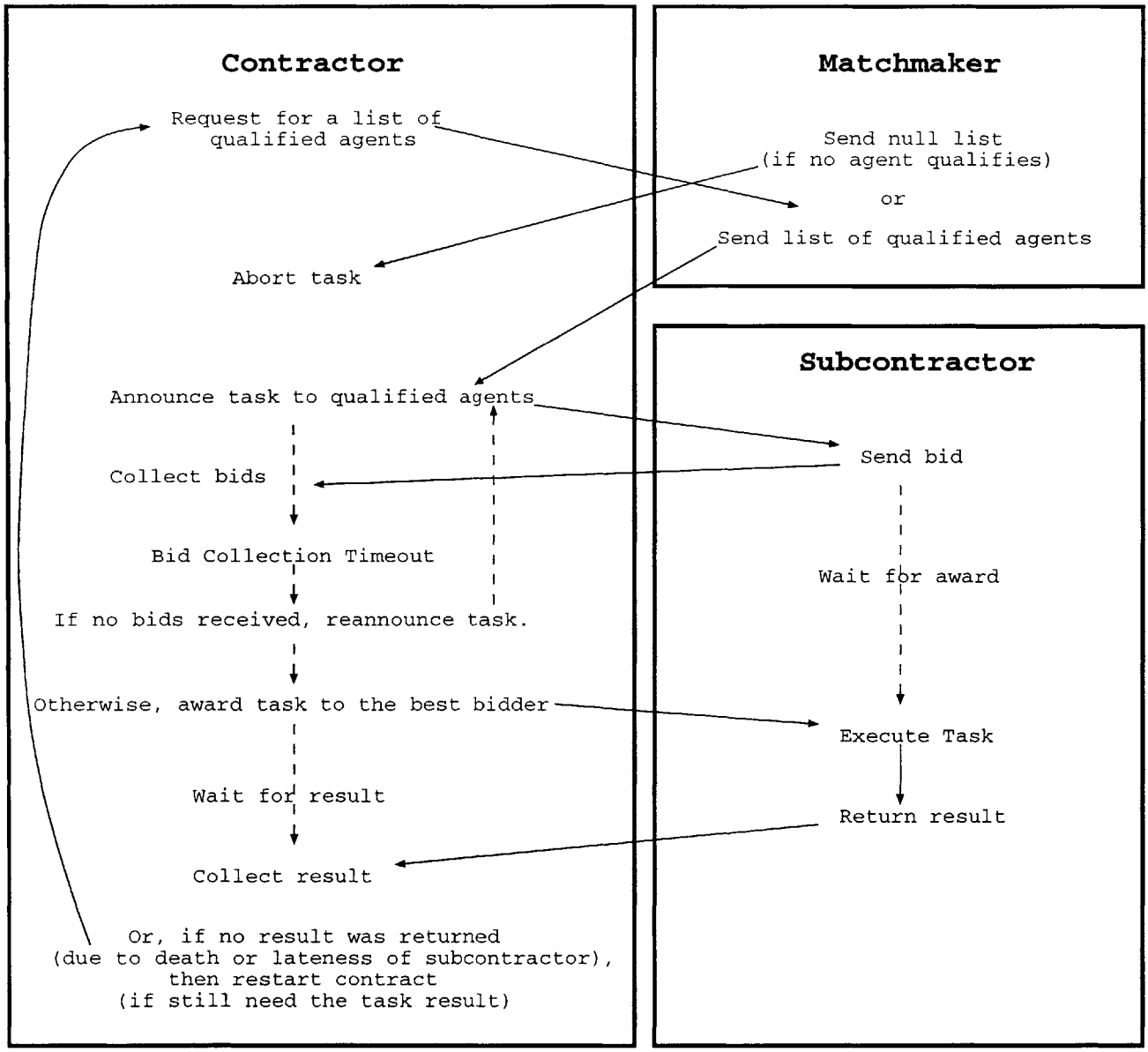


Figure A-1: Simple Survivalist

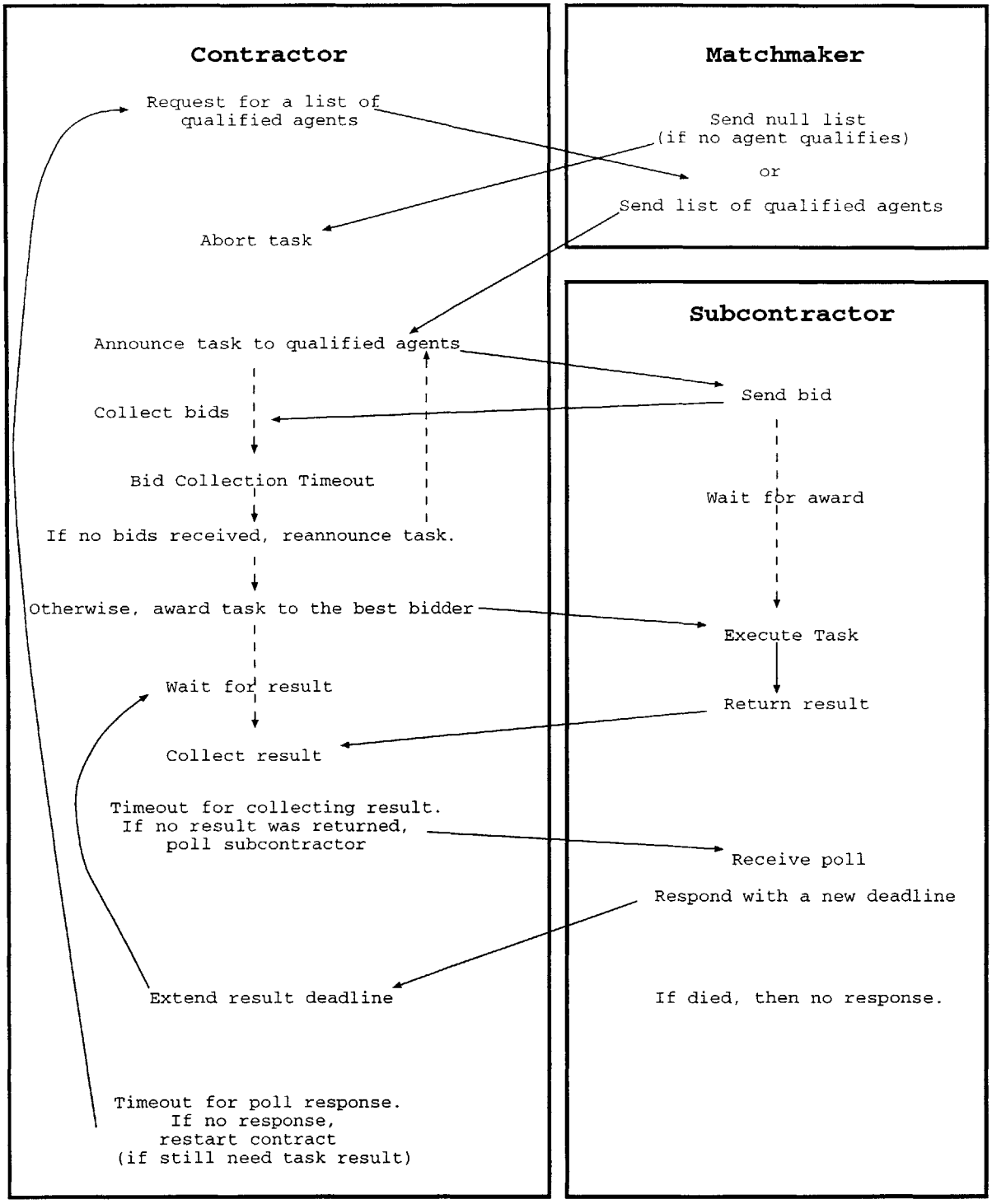


Figure A-2: Complex Survivalist

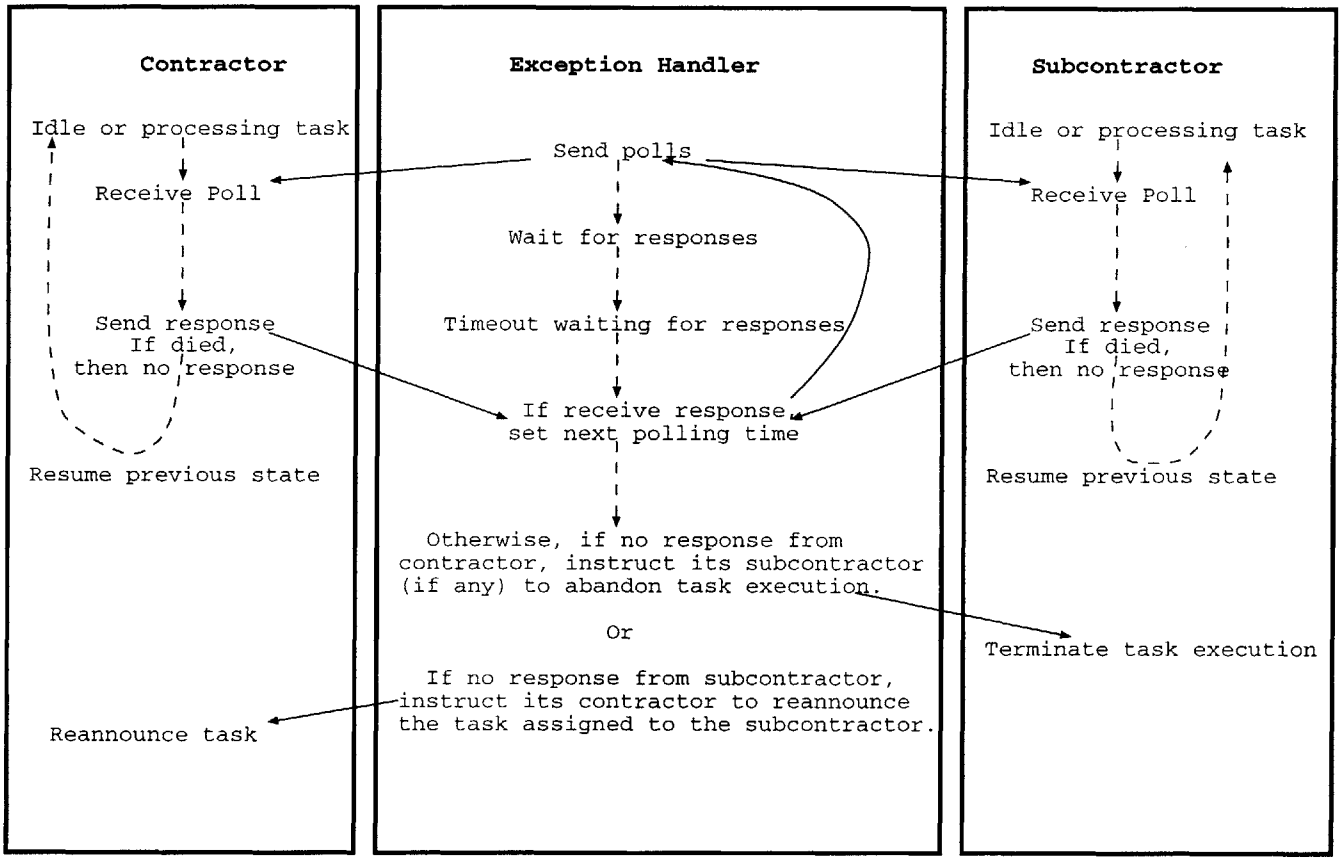


Figure A-3: Simple Citizen

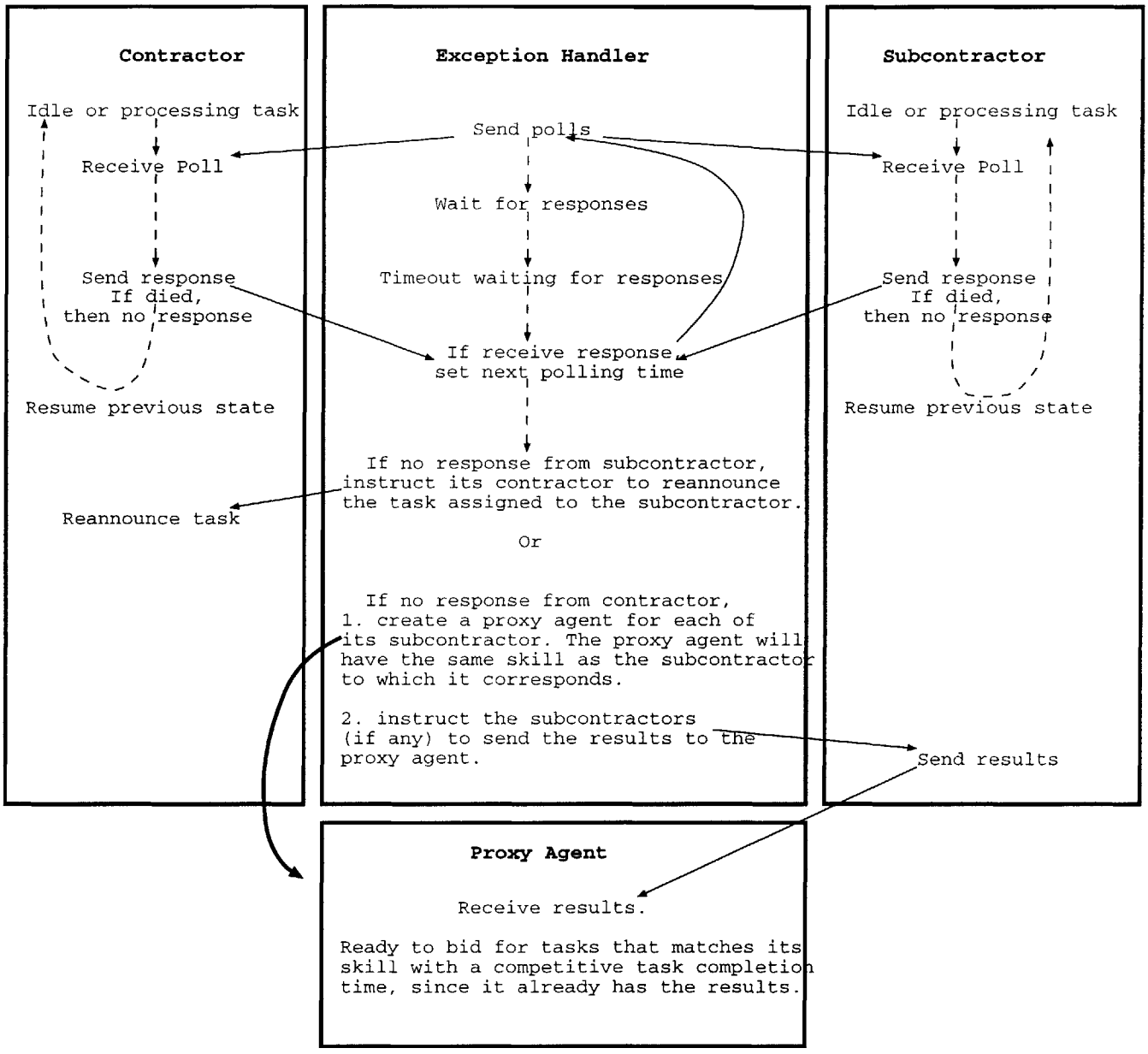


Figure A-4: Complex Citizen

Appendix B

Empirical Results

Experiment Parameters	EH Strategy	Maximum Task Completion Time	Average Task Completion Time	S.D.
deep short scarce	none (no exceptions)	13957	6821.56	2536.86
	simple survivalist	46954	19857.52	13041.53
	simple citizen	29250	10032.02	4173.43
	complex citizen	40345	10330.01	8411.50
deep short abundant	none (no exceptions)	10343	6644.71	1074.19
	simple survivalist	52961	20780.95	15891.33
	simple citizen	25255	9021.85	3324.30
	complex citizen	35025	8324.22	3341.26
deep long scarce	none (no exceptions)	59195	50079.86	3288.58
	simple survivalist	471053	129562.41	103803.28
	simple citizen	166435	63338.14	23090.17
	complex citizen	139036	59251.03	15155.06
deep long abundant	none (no exceptions)	42773	42771.77	11.20
	simple survivalist	572452	142704.23	131551.41
	simple citizen	91447	50944.49	10186.69
	complex citizen	228056	51738.58	20927.55

Table B.1: Numerical Data

Bibliography

[1] Lijin Aryananda. An exception handling service for the contract net protocol family. Master of engineering thesis, Massachussetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 1999.

[2] A. D. Baker. Complete manufacturing control using a contract net: A simulation study. In *1988 International Conference on Computer Integrated Manufacturing*, 1988.

[3] Randall Davis and R. Smith. Negotiation as a methaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983.

[4] S. S. Fatima and G. Uma. An adaptive organizational policy for multi-agent systems-aasman. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 120–7, Los Alamitos, CA, 1998. IEEE Computing Society.

[5] B. Horling, V. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Computer science technical report (99-03), University of Massachussetts, Amherst, 1999.

[6] M. N. Huhns and L. M. Stephens. In G. Weiss, editor, *Multiagent Systems and Societies of Agents*, chapter 2, page 97. The MIT Press, Cambridge, MA, 1999.

[7] G. A. Kaminka and M. Tambe. What is wrong with us? improving robustness through social diagnosis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[8] M. Klein and C. Dellarocas. Exception handling in agent systems. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, 1999.

[9] G. Lin and J. Solberg. Integrated shop floor control using autonomous agents. *IIE Transactions*, 24(3):57–71, 1992.

[10] Pattie Maes. Modeling adaptive autonomous agents. In C. G. Langton, editor, *Artificial Life, An Overview*. MIT Press, Cambridge, MA, 1995.

[11] T. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In B. A. Huberman, editor, *The Ecology of Computation*. Elsevier Science Publishers, B. V. (North Holland), 1988.

[12] Langton Minar, Burkhart and Askenazi. The swarm simulation system. http: www.swarm.org.

[13] A. Saad, K. Kawamura, G. Biswas, M. E. Johnson, and A. Salama. Evaluating a contract net-based heterarchical scheduling approach for flexible manufacturing. In *IEEE International Symposium on Assembly and Task Planning*, pages 147–52, Los Alamitos, CA, 1995. IEEE Computing Society Press.

[14] T. W. Sandholm and V. R. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *ICMAS-95. First International Conference on Multi-Agent Systems*, Menlo Park, CA, 1995. AAAI Press.

[15] David Shue. Simhazard: An agent-world exception simulator. Master of engineering thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 1999.

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

[17] P. Sousa and C. Ramos. A dynamic scheduling holon for manufacturing orders. *Journal of Intelligent Manufacturing*, 9(2):107–12, 1998.