Implementation of Experimental Test beds for Real Time diagnosis

by

Ravi Krishnamurthy

A research paper submitted in partial fulfillment of the degree of
Master of Science

Major Prof	-	Dr. Bruce D'Ambrosio
Minor Prof	-	Dr. Prasad Tadepalli
Committee Member	-	Dr. Timothy Budd

Dept. Of Computer Science
Oregon State University
Corvallis
Oregon - 97331

July 23rd 1999

Implementation of Experimental Test beds for Real Time diagnosis

by

Ravi Krishnamurthy
Oregon State university,
Corvallis, OR – 97330
krishnra@cs.orst.edu

Abstract

Experimental test beds to study Real Time agents are common topic in AI. A principal contribution of this paper is to develop a framework for such experimental test beds. Design and implementation of the test bed and the extension of the framework for different equipment and agents are discussed. Results of an experiment with a simple digital circuit and heuristic agents to test the working of the test bed are reported.

Chapter 1

Introduction

The Problem

A number of test bed systems for studying agent design have been developed. These agents are embedded in the world, which evolves while the agent chooses an action. The utility of an action depends not only on the action selected, but also on the time at which the action is performed, which in turn depends on how long it takes the agent to choose an action[1,2].

Most Research in agent design builds on the classical planning paradigm[3]. The classical planning paradigm assumes an environment that is both controlled and simple. The planning agent is generally assumed to have complete control over the environment. This means that its intended actions are the only events that can change the world's state. Furthermore the effects of its action are fully known both to the agent and to the system designer. The agent is usually assumed to possess complete and errorfree information about the state of the world when it begins planning. Because it knows what the initial state of the world is, what actions it intends to carry out and what effects these actions will have, at least in principle, predict exactly what the state of the world will be when it finishes executing.

Classical planners embody strong simplifying assumptions both in the sense that their capabilities – the class of problems they can solve – tend to be limited and in the sense

that the world in which they operate tend to be small, exhibiting only a few features and a limited physics.

Current work on agent architecture aims towards relaxing these assumptions. Reactive systems for example, deal with the problems in which the world can change unpredictably, between plan time and execution time by deciding what to do at execution time instead of generating a plan prior to execution.

In this paper, we build an experimental test bed using software-engineering concepts to study the effect of heuristic agents on the reactive systems. This paper is organized as follows:

Chapter 2 deals with the requirement analysis for the test bed. Chapter 3 deals with the Architecture design, Chapter 4 with the implementation details, Chapter 5 with the user documentation, and Chapter 6 with the experiment we have done using heuristic agent on a digital equipment,

Chapter 2

Requirements and Analysis of the test bed

2.1 Requirements for the test bed

The equipment under focus is a simple digital system diagnosed in situ by a heuristic agent. After a lot of changes we decided on the following functional requirements for the test bed:

-
- The equipment in between each cycle gives some time(Quanta) for the agent to finish its reasoning.
- Multiple faults can occur (and can continue to occur even after an initial fault is detected).
- Faults can be intermittent.
- The agent senses equipment operation through a set of fixed sensors and one or more movable probes.
- Action alternatives include probing test points and replacing individual components.
- Each action has a corresponding cost.
- The agent can perform only one action at a time.
- The overall task is to minimize total cost over some extended time period during which several failures can be expected to occur.
- A well-defined model of time to calculate the time cost of reasoning and acting.
- The test bed should support experimentation.

2.2 Analysis of the requirement

- The equipment in between each cycle, gives some time(Quanta) for the agent to finish its reasoning.

Between each cycle of the computation, the equipment gives some Quanta for the agent to finish its reasoning. If the agent finishes its reasoning with in this quanta and chooses the correct action to replace a failed component, then the next computation of the equipment will give the desired output provided no other component gets into the faulty state.

- Multiple faults can occur(and can continue to occur even after an initial fault is detected).

Each component of the equipment can be in 2 states- OK and faulty state. There is a probability associated with each component getting into the faulty state. So when a component is in faulty state, some other component can also get into the faulty state. Even after an initial fault is detected, the component can get into a faulty state in the future.

- Faults can be intermittent.

The component can get into the faulty state randomly. It can happen in the first cycle or in the 1000^{th} cycle. But it stays in the faulty state until the agent detects and corrects it.

- The agent senses equipment operation through a set of fixed sensors and one or more movable probes.

The agent can sense the equipment operation through a set of fixed sensors and one or more movable probes. For a heuristic agent, this requirement is not needed. But for a theoretically reasoning agent, this is very useful to make decisions.

- Action alternatives include probing test points and replacing individual components.

Simple heuristic agents can perform only component replacement. Theoretically a reasoning agent can probe test points and also replace the component. However we decide to restrict initial investigation to heuristic reasoning by the agent. Replacement of the component is the only action done by the heuristic agent. Also the agent can replace the component even if it is not faulty.

- Each action has a corresponding cost.

The replacement of the component has a specific cost associated with it. Also there is a cost involved due to the faulty cycles of the component.

- The agent can perform only one action at a time.

The theoretical agent can do 2 kinds of actions – probing test points or replaces individual components. But at one time, it can perform only one action. Since the heuristic agent can only replace the component, it does not matter. Also the agent can replace only one component at a time.

- The overall task is to minimize total cost over some extended time period during which several failures can be expected to occur.

The sum of the cost involved due to the agent replacement and the faulty component vary with the amount of quanta that is given to the agent for reasoning. So the test bed should be able to calculate the cost at each quanta and help to find the minimum cost per failure of the component.

- A well-defined model of time to calculate the time cost of reasoning and acting.

The time taken by the agent to do its reasoning and also how long the equipment stays in the failure state is needed to know the time cost of reasoning and acting.

- The test bed should support experimentation.

The test bed should be able to support experimentation by changing various parameters like the Quantum given to the agent for reasoning, the replacement cost of the component and the cost of component failure per cycle.

Chapter 3 Architectural Design

Our initial objective was to develop an application to see how our test bed works and then convert it into an applet so that it can be accessed from anywhere. Scanning through the requirements gave us the following nouns as classes for our testbed:

Interface, Equipment, Component and Agent.

3.1 Interaction Diagram

The interaction diagram between the above components are shown below in fig 3.1

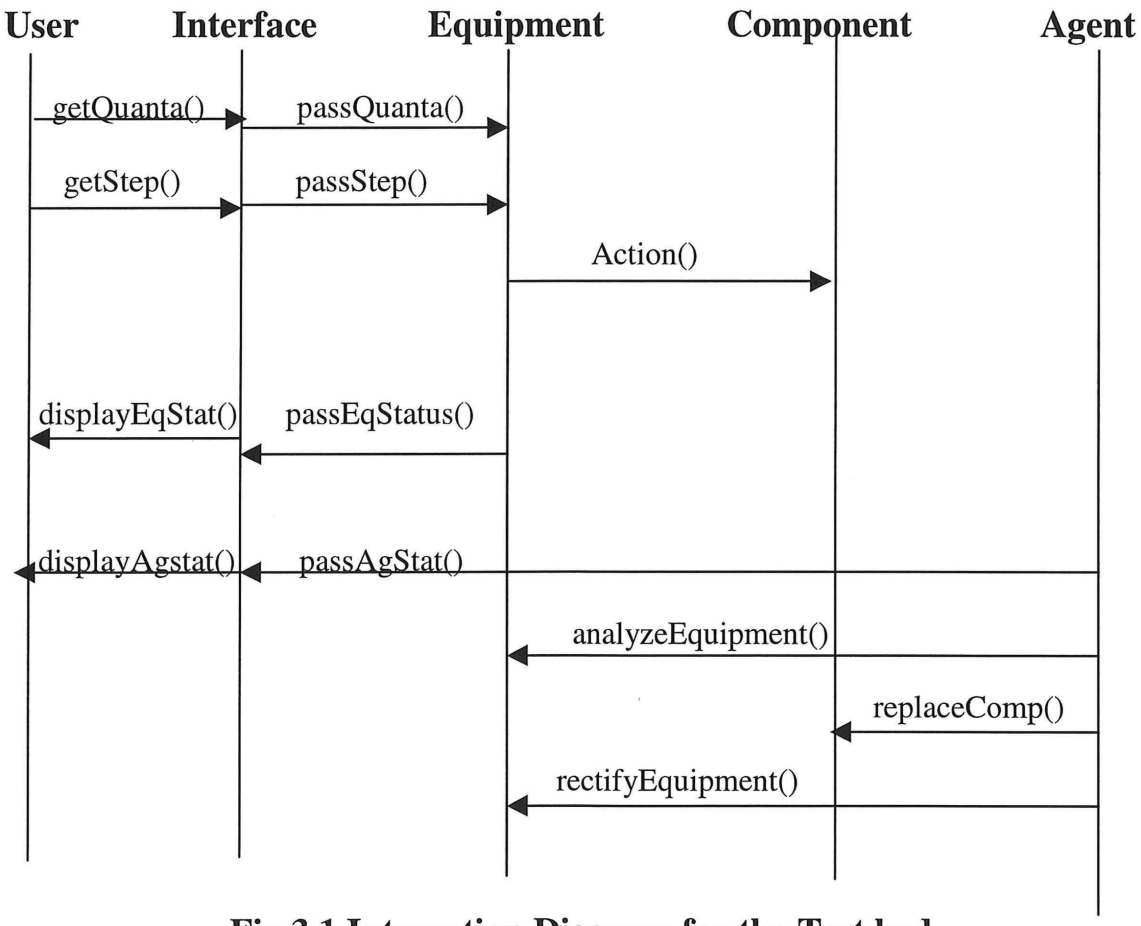


Fig 3.1 Interaction Diagram for the Test bed

3.2 CRC cards

The CRC cards for the test bed are shown in fig 3.2.

Interface	Collaborator
Gets the input from the user	Equipment
Feeds the constraint to the agent and the Equipment	Agent
Display the status and the output to the user periodically	

Equipment	Collaborator
perform computation	Component
passes on the datas needed to the agent for reasoning.	Agent
gives status to the interface	Interface

Component	Collaborator
initializes the component	Equipment
perform the action for each component.	
controls the status of the component	

Agent	Collaborator
Does reasoning by heuristics	Equipment
Gives data to the interface after every reasoning	Interface
Replace the component if necessary	

Figure 3.2 CRC cards for the test Bed

The test bed design can be differentiated into 2 parts – front end for the interface design and back end for the actual working of the equipment and the agent.

3.3 Back end

The back end has 2 major functions. One should be the equipment that performs its computation based on the logical components it is composed of. The other will be the agent that does the analysis of the equipment by heuristics method. The equipment will have components (logic gates). But there are different types of logic gates available. So our design for the equipment will have component as a composition, and different types of logic gates will inherit from a base class component as shown in Fig 3.3. The agent will have an object of class Equipment as a composition so that it can know the status of the equipment.

Equipthread extended from the Java Thread class is used to perform computation of the equipment. Similarly AgentThread extended from the Java thread is used to perform the reasoning for the Agent.

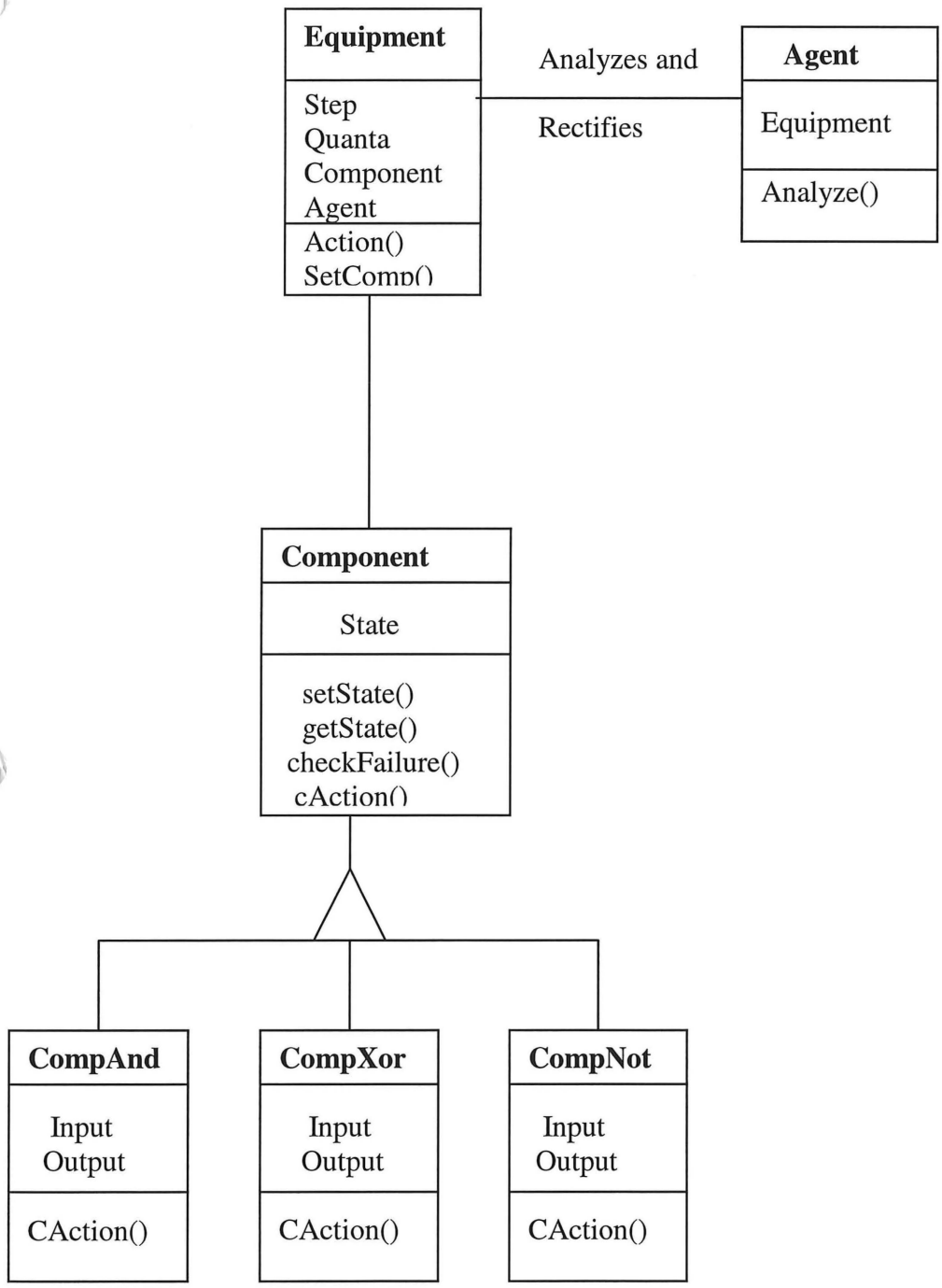


Fig 3.3 Design for the Back end

3.4 Interface Design

The interface should be user friendly. It should be able to get the constraints , such as the Quanta to be allowed between each cycle, the total steps to be run by the equipment etc. The above parameters will be fed in either through a text box or through the edit option in the menu bar. There is also a menu bar to give the save option after each simulation. Also there is a restart control to repeat the experiment under the same or different constraints. The interface is to be developed using JDK 1.1.6.

Chapter 4

Implementation details

The implementation is divided into 2 parts: the interface or front end and the back end comprising of the actual working of the simulation.

4.1 Interface implementation details

Rtime.class: This class extends from applet. It describes briefly on the simulation system and has start control to enter into the actual simulation.

RealTime.class: This class is inherited from Frame class of java and is activated when the start control is desired in the applet. This is the major interface window that takes in the input parameters from the user and gives choices to start the simulation under the desired constraints. It also periodically displays the status of the equipment and the agent.

IntTextField.class: This class is inherited from TextField class of java. Lots of options like how to read a negative number, or what to do when a tab or backspace is entered are dealt in this class. All the text fields objects for the project are instance of this class.

AbtDialog.class: This class is inherited from the Dialog class of java. This class is used to display a dialog box about the project when the about menu item is chosen from the help options.

ChangeDialog.class: This class is also inherited from the java Dialog class. This class is used to display a dialog box to get the input parameter like the Quanta to be set etc. Each input option pops up different dialog boxes and all of them are objects of this class.

RealTimePanel.class: This class is inherited from java Panel class. This class is used to set the location of the panel in the grid bag layout of the RealTime frame class. All other

methods are inherited from the parent Panel.class. All panels used in the project also use grid bag layout.

4.2 Back end implementation details

Equipment.class: This class is the representation of the equipment in operation. Based on the random input, it performs computation using the logic components to produce outputs.

Comps.class: This class is the base class for all the components used in this project. It initializes the component. Each component can be in either an OK state or a faulty state. The faulty state is attained based on the failure probability of the component. But the component stays in the faulty state until the Agent rectifies it. This class has a synchronized method setComponent(). This method is synchronized because both the equipment and the agent can access is at the same time and might lead to a deadlock. This class also verifies whether the component is in a faulty state or not and performs action accordingly.

EquipThread.class: This class is inherited from the Thread class of java. It performs the action for the equipment and sleeps for the Quanta seconds between each cycle so that the agent performs its reasoning in that time. It also updates the status of the equipment on the interface after each cycle.

Agent.class: This class represents the Agent used for reasoning the equipment at all times. The Agent can do its reasoning either by heuristics or by theoretical methods. This Agent can replace the component after its reasoning.

AgentThread.class : This class is inherited from the Thread class of java. It performs the reasoning action of the Agent. It updates the time taken by the Agent for reasoning. It also passes to the interface how may components are replaced periodically.

Other classes like compNot, compXOR, compAND etc: These classes are inherited from the Comps.class. They inherit all the methods of the base class. They have an additional method called Action() which executes the action based on their logical status. Their action is also dependent on whether the component is faulty or not.

Chapter 5 User Manual

Clicking the "start the simulator" button of the Real Time applet pops the simulation frame as shown in Fig 5.1. There is a canvas that describes what is to be done at this stage.

There are 4 input values to be entered by the user. They are:

- Quanta: This is the time that the equipment gives the agent for reasoning. The unit of time is in milliseconds. The value entered should be a positive integer. If a negative number is entered, the value of quanta is taken to be zero.
- Replacement cost: Once the agent finishes its reasoning, it replaces the component. So there is a cost involved per replacement of the component. The input values of the replacement cost should be a positive integer. If a negative value is entered, the replacement cost is taken to be zero.
- Cost of component failure per cycle: There is a cost involved for component failure per cycle. The value to be entered is a positive integer. If a negative value is entered, the cost per failure is taken to be zero.
- Steps: Steps is the number of cycles the Equipment has to run. The minimum value should be atleast 50 for the number of steps.

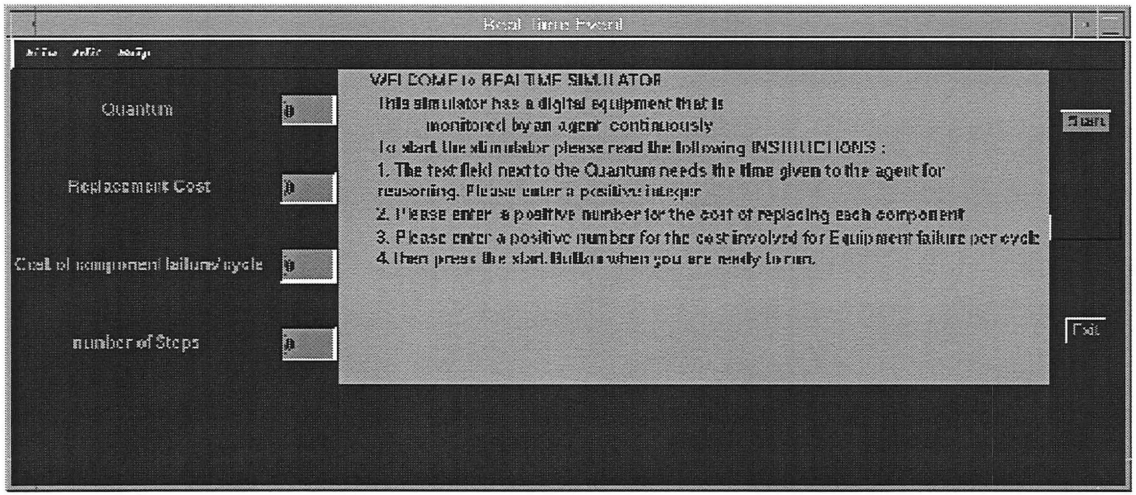
The frame also has a menubar with a File, Edit and Help options.

- File Menu : File menu has 2 options – Save and Exit.

  Save options helps to save the input parameters, the cost per failure of the run and the status of the equipment and agent after the run.

  Exit option helps to exit the simulation.

- Help Menu: As of now, it has only an about menu item which when pressed opens a dialog box to tell about what the software is about.


Real Time Event
Quantum
0
Replacement Cost
0
Cost of component failure/ cycle
0
number of Steps
0
WELCOME to REAL TIME SIMULATOR
This simulator has a digital equipment that is
monitored by an agent continuously
To start the simulator please read the following INSTRUCTIONS :
1. The text field next to the Quantum needs the time given to the agent for
reasoning. Please enter a positive integer
2. Please enter a positive number for the cost of replacing each component
3. Please enter a positive number for the cost involved for Equipment failure per cycle
4. Then press the start Button when you are ready to run.
Start
Exit

- Edit Menu: The Edit menu has the following options:

  Quantum : Choosing this option opens a dialog box to set the Quantum value.

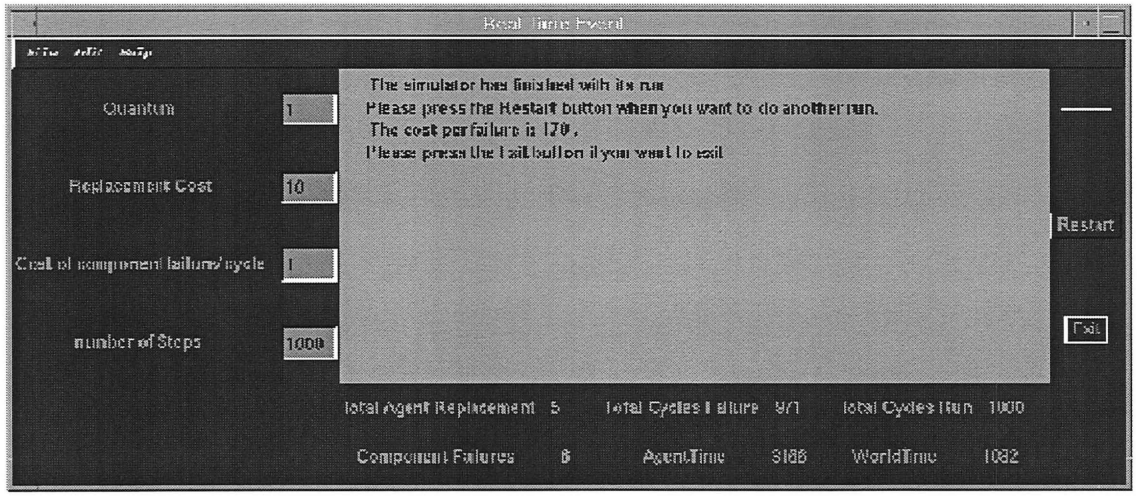
  Replacement cost: choosing this option opens a dialog box to enter the replacement cost of a component.

  Cost of component failure per cycle: choosing this option opens a dialog box to enter the cost of component failure per cycle.

  Steps: choosing this option open up a dialog box to enter the number of cycles the equipment is to run.

Once all the input parameters are set, the user can press the "Start" button to start the run. Once the equipment is running the status of the equipment and the agent are periodically displayed at the bottom as shown in Fig 5.3. The labels have their literal meanings as displayed on the screen.

Once the run is completed, the restart button gets activated as shown in Fig 5.3. If the user wants to rerun, they can press the restart button else press the cancel button to exit. Pressing the restart button will pop up a dialog box for saving the constraints and results of the run. Once the values are saved (or if cancel is chosen), the user can rerun by pressing the start button after changing the input parameters.


Real Time Event
Quantum
1
Replacement Cost
10
Cost of component failure/cycle
1
number of Steps
1000
The simulator has finished with its run
Please press the Restart button when you want to do another run.
The cost per failure is 170 .
Please press the Exit button if you want to exit
Restart
Exit
Total Agent Replacement 5
Total Cycles Failure 971
Total Cycles Run 1000
Component Failures 8
AgentTime 3188
WorldTime 1082

Chapter 6

Experiment

A small experiment using a digital circuit consisting of 2 XOR gates, 1 AND gate and 1 NOT gate was carried out.

6.1 Experimental constraints:

Each component of the digital system can be in one of the following four states:

- OK state.

  For the given input, the component performs computation to give the output based on their logical status.

- ONE state.

  For any given input, the output will always be one.

- ZERO state

  For any given input, the output will always be zero.

- ONE-ZERO state

  For any given input, the output for half of the time will be one and for the rest of the time will be zero.

The failure probability for each component to get into a failure state on each cycle is 0.1%.

Since there are 3 failure states, the total failure probability for each component is 0.3%.

The component stays in the failure state until it is rectified by the agent.

The agent does heuristic reasoning on the equipment. The heuristics employed are:

- Round Robin.

In this heuristic, the Agent replaces the component in order. So each component is replaced in regular intervals.

- Random-Pick

  In this heuristic, the Agent replaces the component in any random order. So a component may not be replaced if it gets into a faulty state but another component may be replaced often irrespective of its working condition.

The total steps(S) the equipment runs is 1000 cycles. The replacement cost per component (R) is 10 and the cost for the component in failure state per cycle (F) is 1. The total cost per failure is then calculated using:

Cost per failure = ((Rc* R) + (Fc*F))/Fr

where

Rc is the number of replacements in 1000 cycles,

Fc is the number of cycles the equipment has run in failure mode for 1000 cycles and

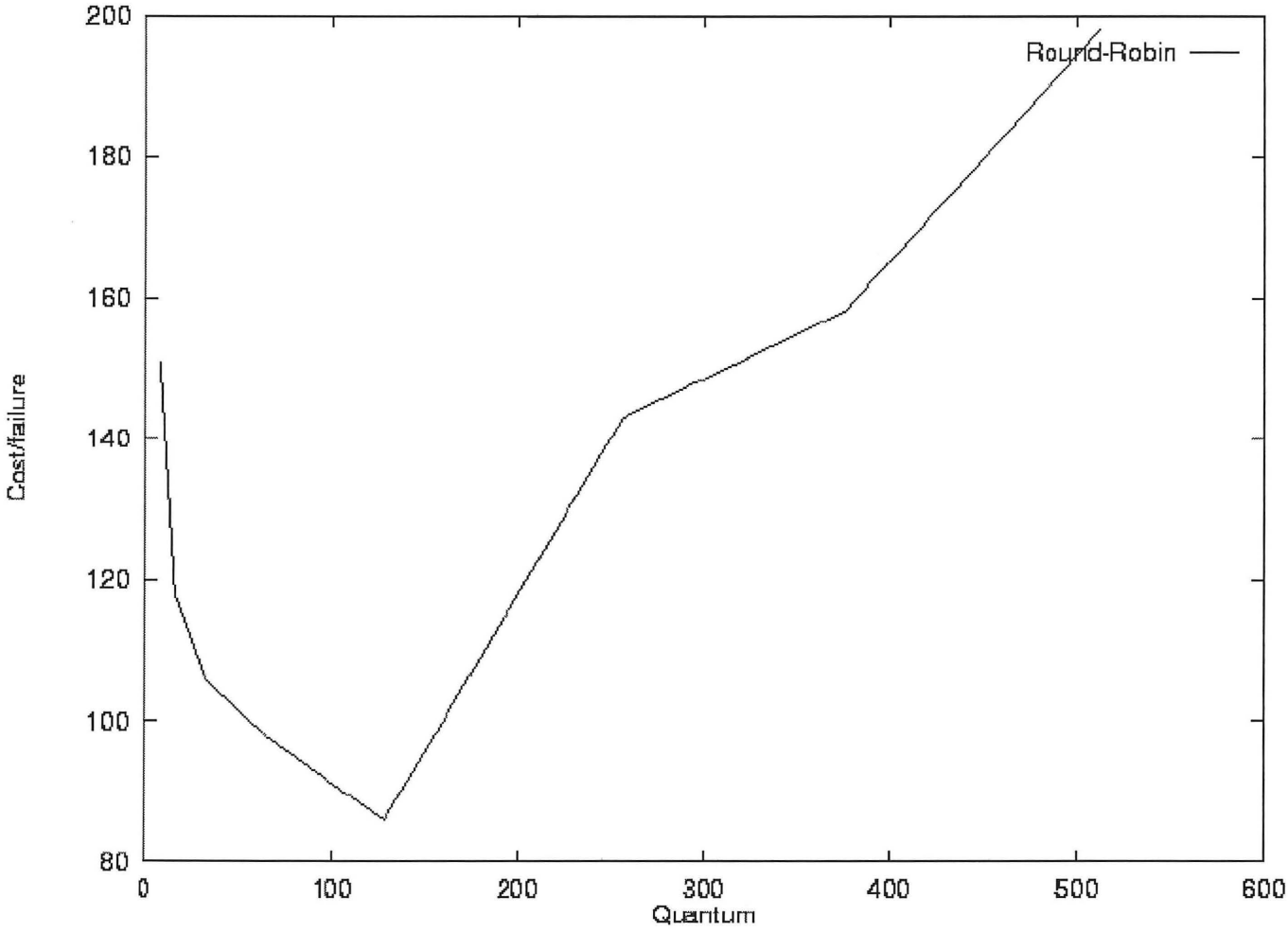
Fr is the total number of failure components for 1000 cycles.

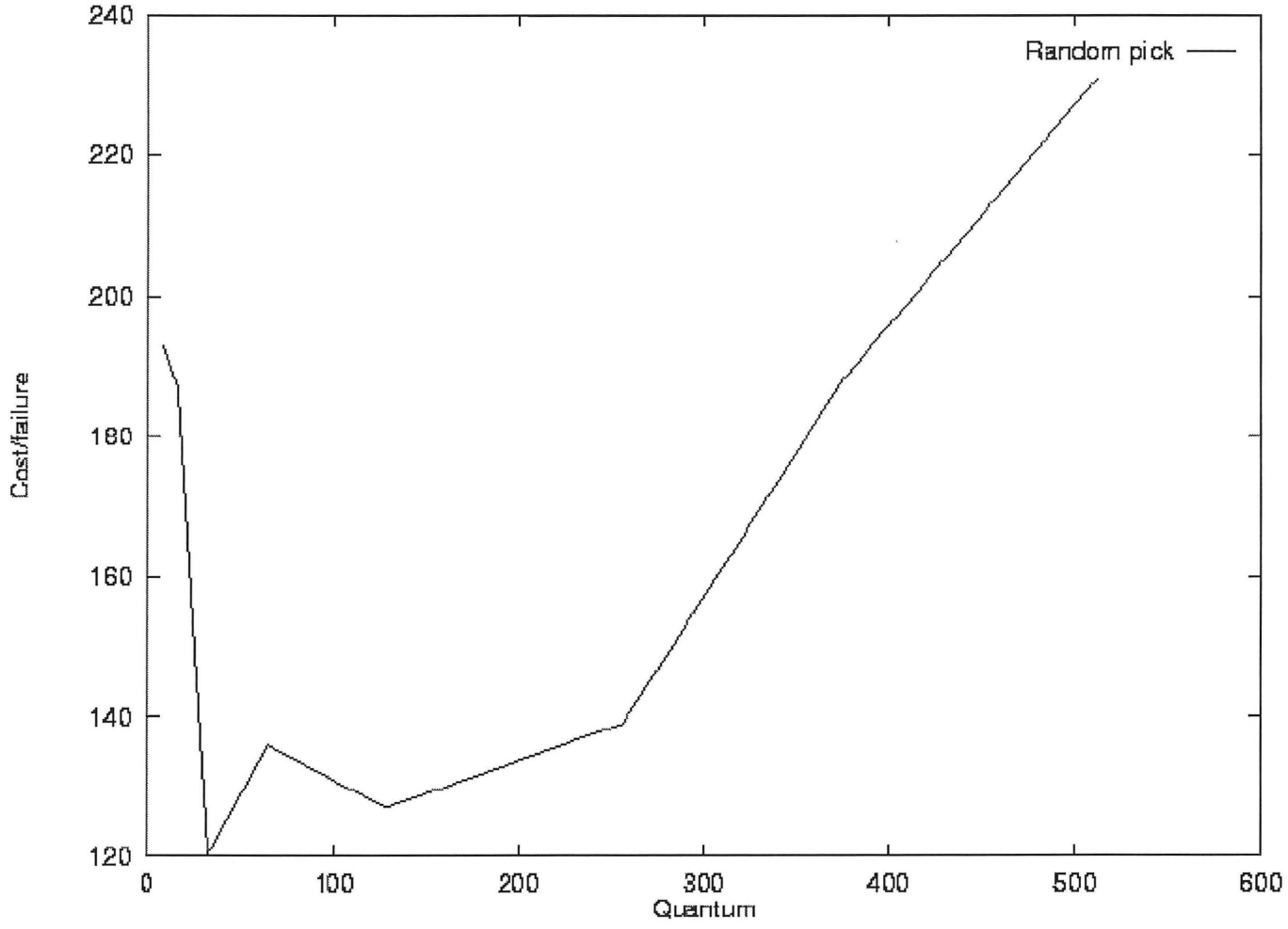
Also at each quanta 4 runs are performed and their average cost per failure is used for analysis.

6.2 Results and discussion

The graphs of cost per failure with different quanta given to Agent for reasoning using Round Robin heuristic is shown in Fig 6.1 and by Random heuristics is shown in Fig 6.2. Initially when the quanta given to the agent is very low, the cost per failure is very high in both Random and Round robin heuristics. This is because, the agent gets very little time to

replace the component if it gets into the faulty state. So the equipment stays in the failure state for many cycles until it is rectified at lower quanta. But as the quantum is increased, the agent gets more time to do its reasoning and replace the component. So the equipment stays in the failure state for less cycles and hence the cost per failure reduces. In Round Robin heuristics, as more time is given to the agent, the cost per failure decreases initially


Round-Robin
Cost/failure
Quantum
200
180
160
140
120
100
80
0
100
200
300
400
500
600


240
220
200
180
160
140
120
Cost/failure
0
100
200
300
400
500
600
Quantum
Random pick

reaches a minimum and again increases with increase in quanta. The increase of the cost per failure with larger quanta can be reasoned as follows:

As more time is given to the agent for reasoning, the number of components replaced increases. Since the cost per failure is proportional to the number of replacement, the cost per failure increases.

In the case of Random pick heuristics, the cost per failure decreases with increase in quantum initially and there is a sharp increase in cost per failure at quantum of 64, and decreases again for quantum of 128. The sudden increase in quantum of 64 is because there is a large variance in the readings of cost per failure observed in Random pick. Since one of the cost per failure observed for quantum of 64 is very large that increased the average cost per failure at that quantum. Further increase of the quanta above 128 increases the cost per failure for the same reasons explained for round robin heuristics.

The quanta can be low as 8 mSecs. Keeping it lower than this limit, gives a big variance in the cost per failure since the equipment does not actually perform its next run after that quanta. It may resume exactly after the chosen quanta or a little later which causes the variance in the cost per failure. Also the higher side for the quanta may be 512 mSecs. Going beyond this limit only increase the agent replacement cost.

Chapter 7 Conclusions and Future directions

7.1 Conclusions

1. A test bed to do experimental study of real time decision making by an agent in an evolving world was developed.
2. An experiment was performed with heuristics agent reasoning on logical equipment.
3.
4. When the agent is given very little time for reasoning, the cost per failure is very high. As the quanta is increased the cost per failure decreases and with further increase in the quanta the cost per failure increases.

7.2 Future Directions

1. Multiple agents reasoning on the evolving world can be developed.
2. The equipment may be made more complex.
3. Theoretical reasoning agents may be implemented and tested.

References

1. Bruce D'Ambrosio, *Some experiments with Real-time decisions Algorithms,* In proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence, pp 194-202, 1996.
2. Bruce D'Ambrosio, *Real-time value driven diagnosis,* In proceedings of the Third International Workshop on the Principles of Diagnosis, October 1992.
3. Steve Hanks; Martha E. Pollack; and Paul R. Cohen, *Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures,* Artificial Intelligence, pp 17-42, 14(4), Winter 1993.
4. Timothy Budd, *An Introduction to Object Oriented Programming (2nd Edition),* Addison Wesley publishers, MA, 1997.
5. Cay S. Horstmann; Gary Cornell, *Core Java 1.1,* Vol 1 & 2, Sun Microsystems Press, 1998.
6. William Stallings, *Operating systems (2nd edition),* Prentice Hall publishers, 1995.