Purdue University
Purdue e-Pubs

ECE Technical Reports | Electrical and Computer Engineering

2-1-1993

XCoHoRT - A Concurrent Hypermedia Reasoning Tool System Documentation

William Hsu
Purdue University School of Electrical Engineering

M. F. Tenorio
Purdue University School of Electrical Engineering

Follow this and additional works at: http://docs.lib.purdue.edu/ecetr

Hsu, William and Tenorio, M. F., "XCoHoRT - A Concurrent Hypermedia Reasoning Tool System Documentation" (1993). *ECE Technical Reports.* Paper 219.
http://docs.lib.purdue.edu/ecetr/219

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

XCoHoRT - A Concurrent Hypermedia Reasoning Tool System Documentation

William Hsu
M. F. Tenorio

TR-EE **93-9**
February 1993

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

XCoHoRT – A Concurrent Hypermedia Reasoning Tool System Documentation

William Hsu M. F. Tenorio

Parallel Distributed Structures Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

January 27, 1993

Contents

1 Introduction 3
- 1.1 Brief History . . . 3
- 1.2 Motivation and Brief System Description . . . 3

2 Our Implementation 5
- 2.1 Overview of Implementation . . . 5
- 2.2 Agent Structure . . . 6
- 2.3 Programming with Agents . . . 6
- 2.4 List of XCoHoRT Files . . . 6
- 2.5 XCoHoRT Oracle . . . 7
- 2.6 Agent Acquaintance List . . . 8
- 2.7 Child Creation . . . 9

3 XCoHoRT Network Communication Mechanisms 10
- 3.1 UNIX Internet Communication Protocols . . . 10
- 3.2 XCoHoRT Communication Mechanisms . . . 10
- 3.3 Agent Queue Structures . . . 10

4 Dynamic Function Linking in XCoHoRT 12
- 4.1 Agent Functions . . . 12
- 4.2 Writing an XCoHoRT function . . . 14

5 Experimental Features in XCoHoRT 16
- 5.1 Passivation/Activation . . . 16
- 5.2 Migration . . . 16

6 XCoHoRT 17
- 6.1 A demonstration in XCoHoRT . . . 18

A Details for System Programmers 21
- A.1 Debugging Aids . . . 21

B Installation Guide 22
- B.1 Directory Structure of XCoHoRT . . . 22

C List of Commands to Agents 23

List of Figures

2.1 The Agent World and Unix Kernel 5

6.1 Agent Creation Template 18
6.2 XCoHoRT User Interface 19
6.3 Learn Function Template 20
6.4 Use Function Template 20

Chapter 1

Introduction

1.1 Brief History

XCoHoRT is an experimental Concurrent Hypermedia Reasoning system developed for

- Demonstration of the feasibility of realizing such a system on a conventional operating system and language.
- Ease of experimentation with a distributed environment for the end users.

The work on *XCoHoRT* began under the direction of Professor M. F. Tenorio. Three students Tony Gibbens, Letticial Villegas and William Hsu have contributed codes and, ideas to this project. Tony Gibbens worked on the initial implementation of CoHoRT which is later modified and enhanced by William Hsu. The current release version 1.0 has a interface developed on X11R4 window system developed primarily by Letticial Villegas.

This document describes the technical details of the internals of the *XCoHoRT* system. The details of the X11R4 system are not reviewed for the lack of space and the interested reader is encouraged to read the X11 documentation and the X Toolkits programming guide.

The *XCoHoRT* documentation is intended to give programmers an idea of the internals of the system for the purpose of modifying the system to his/her own needs and as a reference for building other similar distributed systems. It also contains parameters and files that must be customized at a customer's site (Refer to Appendix B).

1.2 Motivation and Brief System Description

Programming any distributed systems has always been a difficult task. Programmers have to struggle with programming paradigms and concepts that were not designed for distributed programming.

XCoHoRT is a programming environment in which distributed programs can be constructed. *XCoHoRT*'s wintlowing environment allows intuitive distributed programming.

A distributed programming system consists of several separate programs cooperating in some activity. Each of these separate programs is called an Agent. Agents communicate with each other and each Agent has its own bag of tricks (functions).

This paper describes a tool which provides an intuitive environment for the design and implementation of distributed systems with the following characteristics :

1. need for encapsulation, hierarchy and message passing mechanisms.

2. need for mix mode programming : AI and numeric programs.
3. need for dynamic inclusion of new capabilities.
4. need for an easy to use visual programming environment for system instrumentation.
5. need For integration of different language functionalities.

The *XCoHoRT* approach is intuitive, elegant and simple for modeling natural and artificial distributed systems. It permits the design of conibined symbolic (AI), numeric (simulations), and connectionist programming (Neural Networks) in the same system.

Chapter 2

Our Implementation

2.1 Overview of Implementation

Each *XCoHoRT* agent is an independent entity. In our irnplementation we model each *XCoHoRT* agent as a unix process.

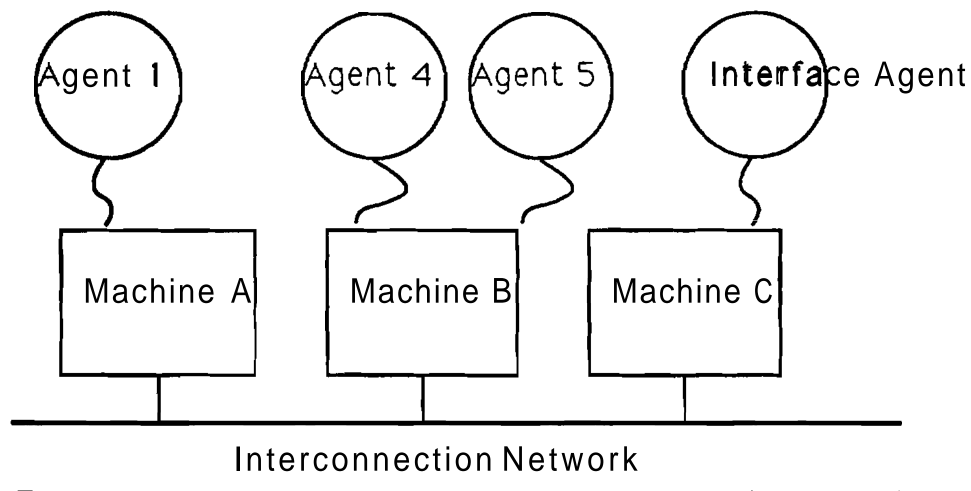

Figure 2.1: The Agent World and Unix Kernel

Local state
Computing power
List of learned functions
Acquaintance list
Knowledge of interface agent
Communication box

Table **2.1:** The Internals of an Agent

2.2 Agent Structure

Each *XCoHoRT* agent (as in Table 2.1) typically has the following components :

1. a local state,
2. local computing power.
3. communication protocols.
4. a collection of learnt functions.
5. knowledge of some other agents (acquaintances).
6. knowledge of the interface agent.

Some *XCoHoRT* agents may be composite agents. Agents organized within a composite agent may view their local community of agents as the world. They may not be able to communicate with agents outside of their local world.

2.3 Programming with Agents

Creating Distributed Programs using Agents :

1. Create basic Agents which have no encapsulated functions.
2. Assign agents functions which are integral parts of the programming tasks.
   (a) Agents can be created on-the-fly by other agents as the task requires.
   (b) Agents can act as a depository of functions needed by other agents.
3. Instruct Agents how to communicate, coordinate, and delegate subtasks to accomplish the objective.

2.4 List of XCoHoRT Files

The Kernel *XCoHoRT* is made up 7 files :

- main.c – the main program and two main loops
- socket.c – unix datagram and stream routines
- oracle.c – manage the oracle functions
- assert.c – for ease of programming
- util.c – utilities needed by other routines
- acq.c –maintain the acquaintance lists
- print.c – keeps all the print functions of all structures

A Makefile is also included to remake the system should any file changes.

Each .c file has a corresponding .h header file that declares the type of the functions that the corresponding .c file contains. File local.h contains the structure definitions of all the data structures used by *CoHoRT*. Each Agent in *CoHoRT* has a structure of type Agent described below.

```
typedef struct Agent_struct{
    int     user-f;          /* user interface flag */
    char    *name;
    char    *hostname;
    int     socket;          /* Socket descriptor */
    int     delay;
    int     rate-sent;       /* rate of incoming */
    int     rate-received;   /* rate of outgoing */
    char    *interface-host;
    int     interface-port;
    acq_list *acqlist;

    /* the following are the variables responsible
      for controlling the input and output queues
    */

    io_q            *ifront;
    io-q            *iback;
    io_q            *ofront;
    io_q            *oback;

    int             numfuncs;
    functions       fcts[10];
    int             numbehavs;
    behaviors      behavs[10];

    struct  sockaddr_in    self;
    struct  hostent *our_host;
} Agent;
```

2.5 XCoHoRT Oracle

The oracle functions are implemented by the interface Agent that keeps a mapping Table 2.2 of logical Agent names to their physical location. This transparent mapping allows for process migration but introcluces an extra level of indirection in the comniunication between Agents.

The oracle maintains the following structures :

```
typedef struct oracle-info {
                int             num_agents;
                acq_list        *pAcqLst[MAX-AGENT] ;
                mapping-type    map[MAX-AGENT] ;
} oracle-type;
```

Entry	Agent Name	Port	Host
0	Interface	4680	monkey.ecn.purdue.edu
1	Monkey	1120	monkey.ecn.purdue.edu
2	Adder	1125	hippo.ecn.purdue.edu
3	Oracle	2125	panther.ecn.purdue.edu

Table 2.2: An Example Mapping

Entry	Agent Name	Port	Host
0	Interface	4680	monkey.ecn.purdue.edu
	Monkey	1120	monkey.ecn.purdue.edu
	Adder	1125	hippo.ecn.purdue.edu
	Oracle	2125	panther.ecn.purdue.edu
1	Monkey	1120	monkey.ecn.purdue.edu
	Oracle	2125	panther.ecn.purdue.edu
2	Adder	1125	hippo.ecn.purdue.edu
3	Oracle	2125	panther.ecn.purdue.edu

Table 2.3: Example Acquaintance Lists Kept by Oracle

```
typedef struct mapping {
                char agent_name[50];
                int port;
                char host[50];
3 mapping-type;
```

The orade keeps track of all the agents in the system. It also maintains the acquaintance list of all the agents in the system. We assume that this structure Table 2.3 is consistent with the acquaintance lists maintained by each individual agent.

```
   Oracle.h
extern void OracleAddMapping(index, name, port, host );
extern void OracleDelMapping( name );
extern int  OracleAgentNumByName( agent-name );
extern void OracleAddAcq( agent-nu, newacq_port, newacq_host,
    type, newacq_name );
extern void OracleDelAcq( agent-nu, newacq_name );
extern int  OracleQuery( name, pport, host );
```

2.6 Agent Acquaintance List

CoHoRT agents can cooperate with a group of other *CoHoRT* agents to accomplish a certain task. This ability is made available by the acquaintance list whereby each agent keeps the name of the other agents that it can talk to on a list. The linked list structure of the acquaintance list is as follows :

```
typedef struct  acq-list-type {
    int                      type;
    /* these are in the mapping : move it later */
    int                      port;
    char                     *host;
    char                     *name;
    struct acq_list_type     *next;
3 acq-list;

extern acq-list *AddAcqLst(/* pAcqLst, port, host, type */);
extern acq-list *DelAcqLst(/* pAcqLst, port, host */);
```

2.7 Child Creation

XCoHoRT agents can give birth to new agents. The mother agent will have the child agent on its acquaintance list. Likewise, the child agent will have his mother on its acquaintance list.

The child agent sends a oracleaddacq to the interface agent to keep its mapping and acquaintance list up to date.

Through the interface any children of any agent can be created.

Chapter 3

XCoHoRT Network Communication Mechanisms

3.1 UNIX Internet Communication Protocols

The UNIX operating system provides a number of C programming language routines for accessing the network using the Internet Protocol (IP). UDP (User Datagram Protocol) and TCP(Transmission Control Protocol) are two commonly used services for network communication.

3.2 XCoHoRT Communication Mechanisms

XCoHoRT agents typically communicate by sending datagrams to each other. Each XCoHoRT agent is uniquely identified by its name and also by its port and host name pair. *CoHoRT* agents can use the stream socket to communicate large amount of data to one another. There is a limit of 17 streams that can be opened at the same one time for a single unix process. Stream sockets are TCP sockets. They are reliable, synchronous and suitable for high volume of data.

We are relying on the network to provide a reliable communication medium which may be acceptable at the local area network scenario on UDP.

The two UNIX communication modes that *CoHoRT* agents use :

1. *STREAM* mode : analogous to our telephone; large bandwidth, on-line, reliable and expensive.

2. DATAGRAM mode : analogous to the US Mail system; small bandwidth, off-line and inexpensive.

3.3 Agent Queue Structures

Each XCoHoRT agent maintains an input and an output queues described as follows :

```
typedef struct io_q_struct {
    info-packet          info;   /* incoming message */
    struct sockaddr_in   addr;   /* address of recipient */
    struct io_q_struct   *next;
3 io_q;
```

The message that is enqueued on these queues are the data that are sent across the network and these packets are described by :

```
typedef struct s_info_packet {
    char alarm;     /* if out of band */
    long time-s;            /* the time */
    char message[MAX-MESSAGE] ;
} info_packet;
```

The procedure to send a mail is to first enqueue the message onto the output queue of the agent and then call the routine **send-mail** to deliver the mail. In this way, we can control how many mails can be delivered within a time period to simulate hosts of different speed and load. The parameter **rate-sent** controls the number of messages we deliver each time.

The procedure to get a mail is to call a procedure **get-mail** which examine the input queue of the agent.. If the input queue is empty then it returns. Otherwise, the message on the input queue is dequeued. The parameter **rate-received** controls the number of messages we dequeue each time.

```
    FILE : util.h
extern void     EnqueueMail(Agent, message, addr,alarm);
extern int      get_mail(pAgent );
extern void     send_mail(pAgent );
extern io_q     *takeletter( pAgent);
```

Chapter 4

Dynamic Function Linking in XCoHoRT

4.1 Agent Functions

XCoHoRT agents differ from other systems in that the agents can "learn" or acquire new C-functions provided in the form of function.o. This process of dynamically linking in new code is the cornerstone of the *XCoHoRT* system. Using this mechanism, any agents may acquire new abilities without having to be restart. In other words, the acquiring of new functionalities by each agent is achieved "on-the-fly". This is a desirable property for a distributed system because it can be partially reconfigured while the system is active.

To achieve this dynamic linking ability we utilized the package dld-3.2.3 developed at UC Davis by Wilson Wong. This dynamic linking ability currently supports VAX (Ultrix), Sun 3 (SunOS 3.4 and 4.0), SPARCstation (SunOS 4.0), Sequent Symmetry (Dynix), and Atari ST.

The dld system has a simple interface.

A sample procedure is provided here for explanation of the dld system :

```
/*
 *  Carry out the user command:
 *  dld object-file.o                -- dynamically link in that file.
 *  ul object_file.o                 -- unlink that file.
 *  function-name arg1 arg2 ...      -- execute that function.
 */
execute (my-argc, my_argv)
```

Operation	Command
Required Initialize,	(void) dldinit (argv[0])
Link a file to the current process	dld_link(filename)
To unlink a linked file	dld_unlink_by_file (my_argv[1], my_argc)

Table 4.1: dld's commands for dynamic linking

```
int my-argc;
char **my_argv;
{
    register int (*func) ();

    if (!my_argc) return;
    if (strcmp (my_argv[0], "dld") == 0)
        while (--my_argc) {
            register int dld_errno;
            extern char *dld_errmesg;

            if (dld_link (*(++my_argv)))
                dld_perror ("Can't link");
        }
    else if (!strcmp (my-argv[0], "ul"))
        dld_unlink_by_file (my_argv[1], my-argc >= 3 ? 1 : 0);
    else if (!strcmp (my_argv[0], "uls"))
        dld_unlink_by_symbol (my_argv[1], my-argc >= 3 ? 1 : 0);
    else {
        func = (int (*) ()) dld_get_func (my_argv[0]);
        if (func) {
            register int i;
            if (dld_function_executable_p (my_argv[0])) {
                i = (*func) (my-argc, my_argv);
                if (i) printf ("%d\n", i);
            } else
                printf ("Function %s not executable!\n", my_argv[0]);
        }
        else printf ("illegal command\n");
    }
}
```

Using this mechanism, *XCoHoRT* agents are able to dynamically learn new functions in respond to the environment.

Previously within each agent, the pointers to the learned functions are kept as :

```
typedef struct  fct_list {  /* To implement Array of functions */
        char    *name;
        int     (*fct)();
        int     size;        /* size in bytes of the addition */
} functions;
```

When a message comes in, it is first checked against the list of built in functions. If it is not one of them then we check the command against the list of learned functions.

However, with the new dynamic linking facilities, the acquired code is stored in the symbol table of the current process so there is no need for an explicit structure to keep track of the functions.

4.2 Writing an XCoHoRT function

A simple example of a *XCoHoRT* function that illustrates the use of this dynamic linking facilities is showned below :

```
::::::::::::::
add1.c
::::::::::::::
extern int x;

add1() {
    x++;
}
::::::::::::::
call_add1.c
::::::::::::::
#include "dld.h"

int x;

/*
 *  Dynamically link in "add1.o", which defines the function "add1".
 *  Invoke "add1" to increment the global variable "x" defined *here*.
 */
main (argc, argv)
int argc;
char *argv[];
{
    register void (*func) ();

    /* required initialization. */
    (void) dld_init (argv[0]);

    x = 1;
    printf ("global variable x = %d\n", x);

    dld_link ("add1.o");

    /* grab the entry point for function "add1" */
    func = (void (*) ()) dld_get_func ("add1");

    /* invoke "add1" */
    (*func) ();
    printf ("global variable x = %d\n", x);
}
```

When this piece of code (call-addl) is run, the following results are obtained :

```
123 panther.ecn /home/jovian3/whsu/dld-3.2.3/test/add1> call\_add1
```

```
global variable x = 1
global variable x = 2
```

Notice that the global variables are modified by the learnt function.

Chapter 5

Experimental Features in XCoHoRT

5.1 Passivation/Activation

XCoHoRT agents achieve persistence through passivation. When a passivate message is sent to a agent from the interface, the agent writes itself to disk. This is done very sparingly as the state of the system may be very uncertain after such an operation. This operation is only done in case of an emergency to preserve the data stru tures of the agents. Only the interface knows how to wake a passivated process up.

5.2 Migration

Traditionally, process migration for the sake of load balancing has not been justified. The overhead with stopping a process and recreating its exact state on the otlier machine is prohibitively high. However in our system, we do not migrate unless the state of the system is stable. We assume that we do migration only very sparingly.

When an agent migrates, its port and host entry changes. Everyone that knows about this agent should be informed. This can be achieved through broadcast. Another alternative is to keep only the agent' logical name on its acquaintance list and requires a lookup at every send operation. A tradeoff can be achieved by keeping a cache of mappings at the local host with timeouts.

In XCoHoRT, migration should only be performed sparingly because there is no guarantee about the state of the system. Only agents who has bi-directional acquaintance list may migrate. During the migration while the agent is packing, all messages sent to it will be lost. After the migrated agent has settled down at a new place it informs the interface as well as everyone on its acquaintance list.

c

Chapter 6

XCoHoRT

XCoHoRT is a X11R3 color version of *CoHoRT. XCoHoRT* makes calls to *XCoHoRT* kernel. *XCoHoRT* also runs on monochrome sun workstations.

In its degraded mode of operation, *XCoHoRT* can provide a menu-driven interface to the basic operations provided by kernel *XCoHoRT.*

XCoHoRT is started by saying "xcohort -11". It can be started remotely from machine A if the local workstation B executes the command "xhost +A" and the remote machine executes the command "setenv DISPLAY B". In this way, the local workstation is giving permission to the remote machine to open up a display on its screen.

The pull down menus in *XCoHoRT* are as follows :

- Agent
  - Create
  - Show
  - Exit
- Messages
  - Add Acquaintance
  - Delete Acquaintance
  - Kill Agent
  - Sleep
  - Awaken
  - Query Agent
  - Send Agent a Message
- Functions
  - Learn Function
  - Forget Function
  - Use Function
- Machine
  - Add New Machine

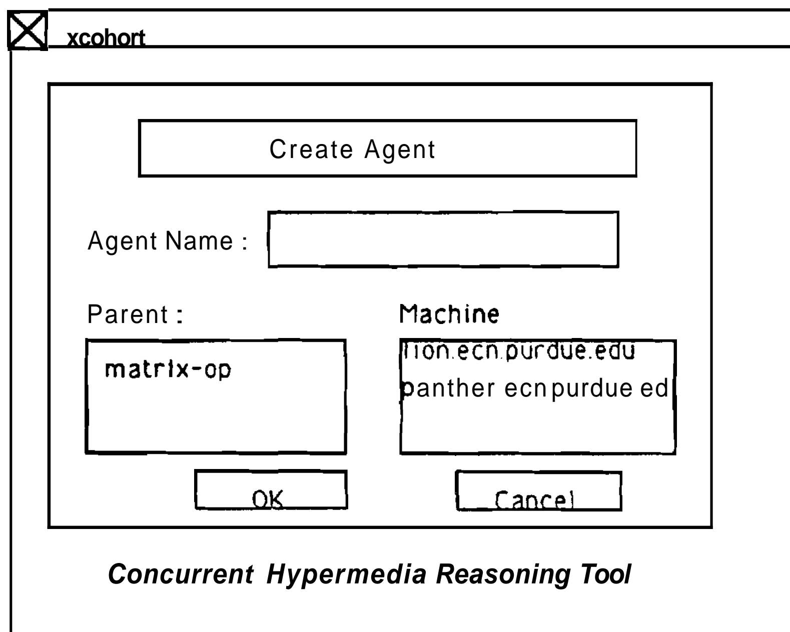

Figure 6.1: Agent Creation Template

- Delete a Machine
- Show current machines

- Help
  - Overview
- Demo
  - Addition

6.1 A demonstration in XCoHoRT

A sample session in *XCoHoRT* are as follows :
Create Agent A :

1. Name of the Agent A
2. Machine name MachineA

The information are entered via a template as shown in Figure 6.1.
Create Agent B :

1. Name of the Agent B
2. Machine name MachineB

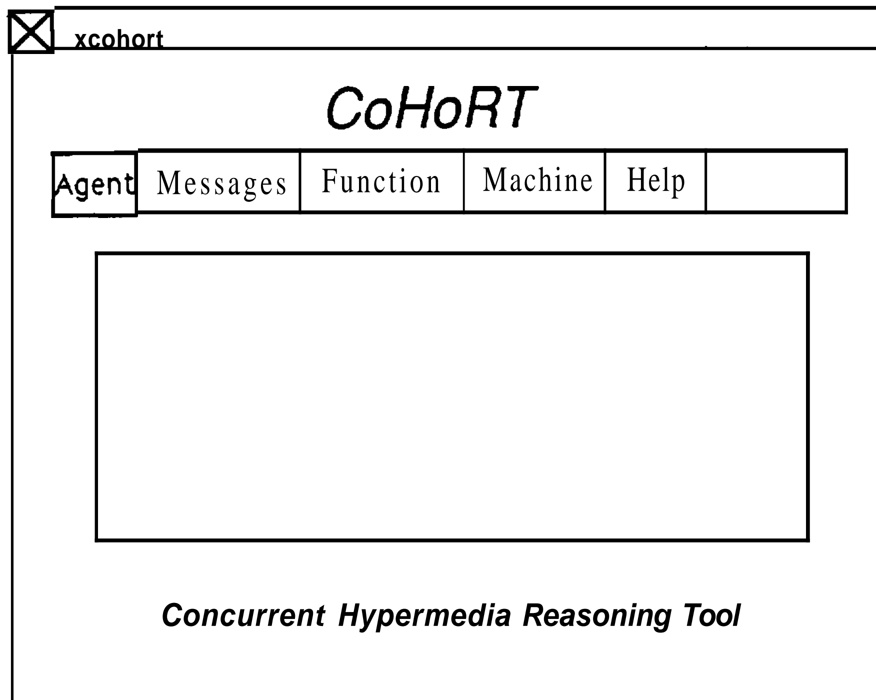

Figure 6.2: XCoHoRT User Interface

There will he a connecting line between Agent A and Agent B to show that **Agent** B is created by Agent A as shown in Figure 6.2.

In our implementation, the interface agent (the one that manages what the user types in) has knowledge of all the agents in the world. When a new agent is created, it sends a message to the interface agent to update its database. User of XCoHoRT then views the wor'ld by browsing this database.

To let Agent A learn a piece of addition function written in C language. First, compile that code using "cc **-c** addition.c". Next cp **addition.o** to your home directory. Now from the menu select **Functions** menu and learn item. Then you will asked to fill in a template as in Figure 6.3.

To use a learned function addition, select the Functions menu and use item. **Fill** in a template as in Figure 6.4. The answer will be return to the interface.

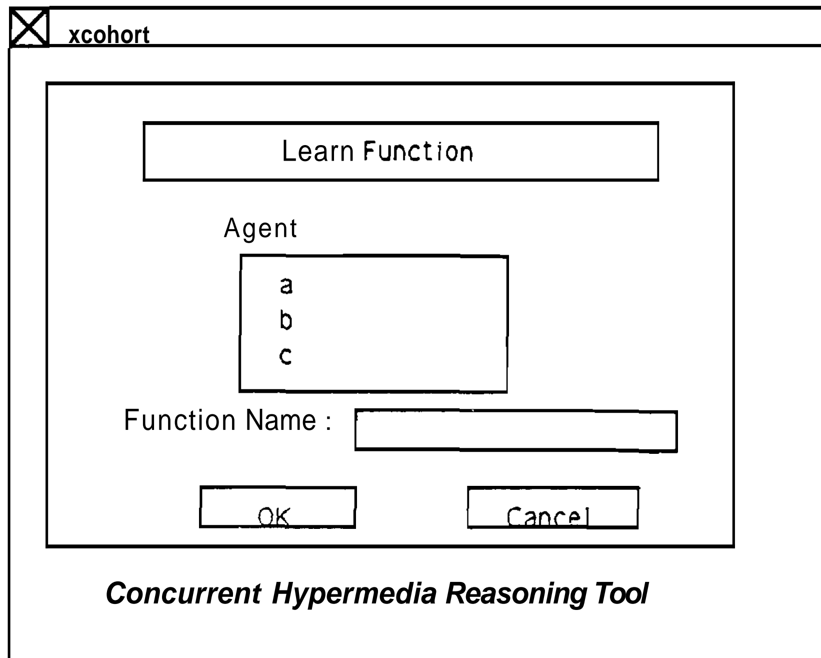

Figure 6.3: Learn Function Template

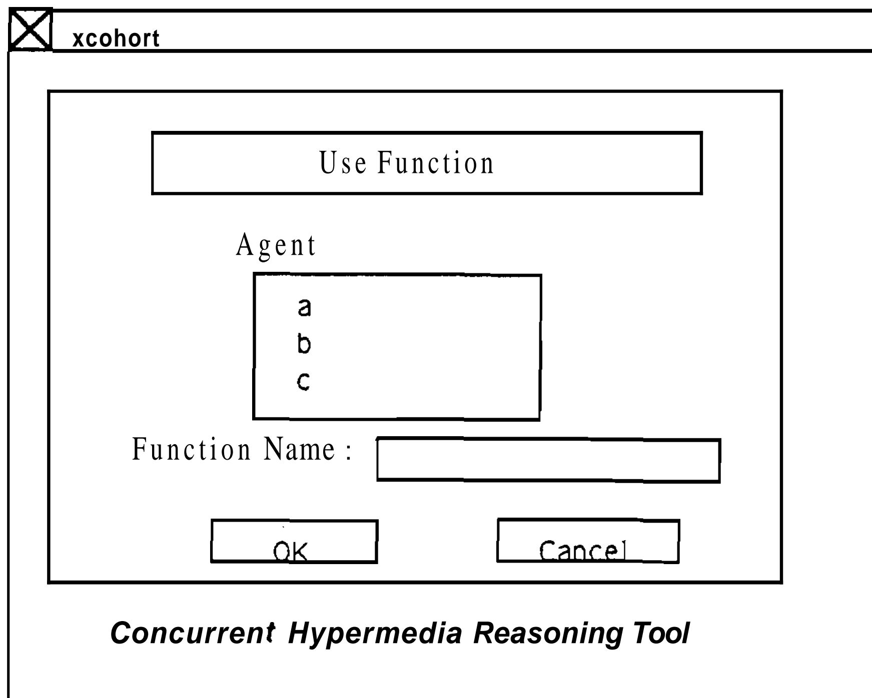

Figure 6.4: Use Function Template

Appendix A

Details for System Programmers

A.1 Debugging Aids

Included in *XCoHoRT* is the ability to place assertions in the code. These are implemented in **assert.c**. **print.c** contains all the print routines necessary to print all the structures defined in the system.

```
FILE : print.c
extern void PrintHostent( pHostent );
extern void PrintSockaddr( self );
extern void PrintAcqLst( pAcqLst  );
extern void PrintInfoPac(  Packet );
extern void PrintIOQ(  pifront, piback );
extern void PrintAgent( agent) ;
```

A log file mechanism is also built in for agents. Since they execute silently outside, it is hard to know if they are working properly. A global compiler directive (LOGFLAG) is used to decide if logging is desired. Agent A will have the log messages captured in a special log directory under the filename A.

Appendix B

Installation Guide

XCoHoRT can be installed on machines listed on page 11 with :

- one or more workstatious.
- monochrome or color monitor. (Color Preferred)
- configured for IP and ethernet network communication.

Things tliat need to be customized includes

- host names must be given in full e.g. monkey.ecn.purdue.edu
- a valid user name and password on the systems on which *XCoHoRT* agents are to run must be entered in util.c. The user name and password are entered on the line that has the command rexec. and change the entry "username" and "password" to the name and password of tlie account that *XCoHoRT* is run.
- Make sure in util.c, tlie entries of sys_path array correspond to the executable of cohort on the entries in sys_name array. e.g.

  /home/jovian3/whsu/cohort/cohort on panther.ecn.purdue.edu
  /home/rainbow/villages/cohort/cohort on brown.ecn.purdue.edu.

- Use the make command

B.1 Directory Structure of XCoHoRT

xcohort and dld tlirectory are on the same level. example directory is under the xcohort directory. These need to be reflected in tlie Makefile, util.h and main.h.

Appendix C

List of Commands to Agents

The list of commands are listed in Table C.1.

Agent Command	Agent Reply	Oracle Command
hello	yes	oracleaddacq
die	dead	oracledelacq
addacq		
delacq		
query	answer	
learnfunc		
forgetfunc		
learnbeh		
forgetbeh		
passivate		
activate		
migrate		
child		

Table C.1: List of Commands

Formats of the commands are listed in Table C.2.

Agent Command	No. Args	Format
hello	1	hello
yes	1	yes
die	1	die
dead	2	dead [agent_name]
addacq	6	addacq [port] [host] [type] [agentname]
delacq	3	delacq [agent_name]
query	2	query [var_name]
answer	>= 3	answer [varname] [type1] [ans1] [type2] [ans2] ...
learnfunc	3	learnfunc [func_name]
use	>= 2	use [func_ name] [list of parameters]
forgetfunc	2	forgetfunc [func_name]
passivate	2	passivate [file-name]
activate	2	activate [file-name]
migrate	3	migrate [port] [host]
child	6	child [childname] [childhost] [momhost] [momport] [momname] [oraclehost] [oracleport]
childreply	4	childreply [name] [port] [host]
oracledelacq	7	oracleaddacq [agentname] [port] [host] [type] [new_agent_name]
oracleaddacq	3	oracledelacq [agentname] [new-agent-port] [new_agent_host] [type] [new_agent_name]

Table C.2: Message Formats for Commands