

LYDIAN

Phuong Hoai Ha

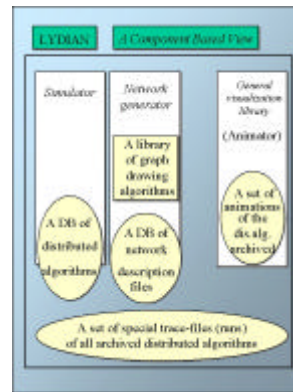
Introduction to Lab. assignments
November 17th, 2005

Schedule

- LYDIAN: Animation Environment for Learning Distributed Algorithms
 - Overview
 - Making new protocols
 - Making new networks

LYDIAN

- Distributed algorithms are complicated
 - Involve a large amount of data describing local state information and complex interactions between elements
- Pseudo-code description given in a book or technical paper is often not sufficient for the understanding of an algorithm
- Lydian is an educational tool for teaching distributed algorithms by allowing users to create their own experiments:
 - Creating/selecting a protocol
 - Describing the network
 - Selecting/adding animations



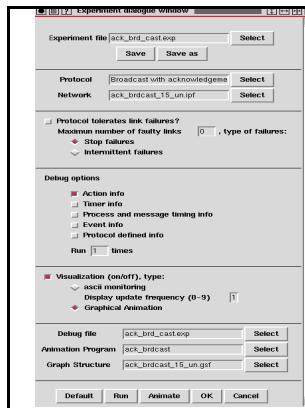
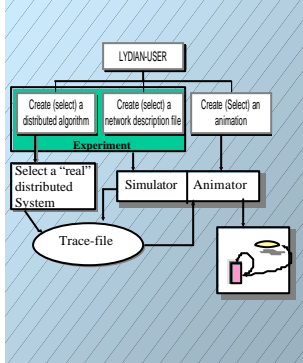
Components

All components are extensible

Accessed via a GUI

Can be replaced by new components

User View on LYDIAN



Demo

The Simulator DIAS

- Simulates any number of processes
- Processes communicate by sending messages
- The behaviour of processes can be defined by defining state transitions
 - State x Event -> function()
- States, messages and functions can be defined in pseudocode
- Simulator supports user defined timing behaviour for message transmission and local computation
- Supports simulation of communication failures for fault tolerance tests

Animator

- For many of the provided protocols LYDIAN provides animations
- Each animation uses the output of the simulator (the tracefile) or alternatively online events of a real distributed system
- The main window always shows the communication graph as it was created by the user
- Different animation aspects are supported by different animation windows which all evolve at the same time and selected by the user
- Further aspects concentrate on causalities, traffic, events and processes occupation in a computation

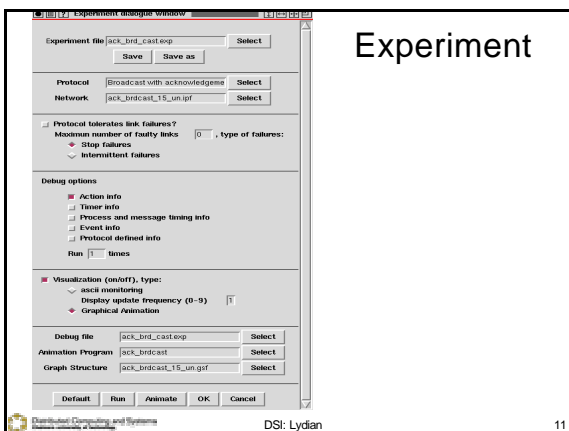
Getting started

- Login to a Sun machine (e.g. persephone.tekno.chalmers.se)
 - Right now Lydian runs only on Sun machines.
- `setenv LYDIANROOT /users/mdstud/phuon/LYDIAN`
- Installation: `$LYDIANROOT/GUI/userinst ./`
- Look at some useful demonstrations
 - Broadcast with acknowledgement algorithm (file `ack_brd_cast.exp`)
 - Ricart and Agrawala's Resource Allocation algorithm (file `ResAlloc_RA_1.exp`)
 - ... (algorithm descriptions are stored at `$LYDIANROOT/Html`)

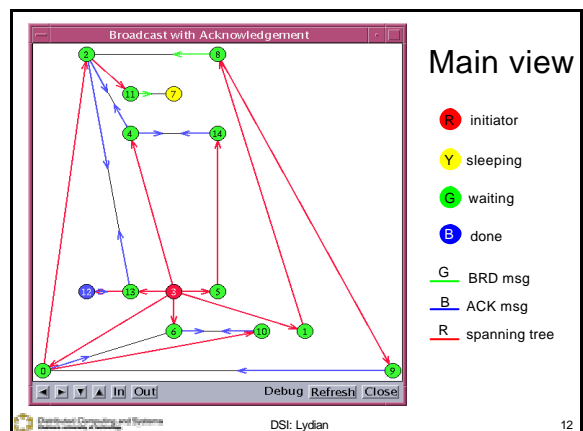
Broadcast with ACK algorithm

- An initiator process sends a broadcast message to all its neighbors
- If receiving the message for the first time, each process sends the message to all its adjacent processes except for its *parent*.
- If receiving the message again, processes reply with ACK.
- Each process waits for ACKs from its receivers. When receiving all the ACKs, it sends a ACK to its parent and stops.
- Algorithm is finished when the initiator received all ACKs

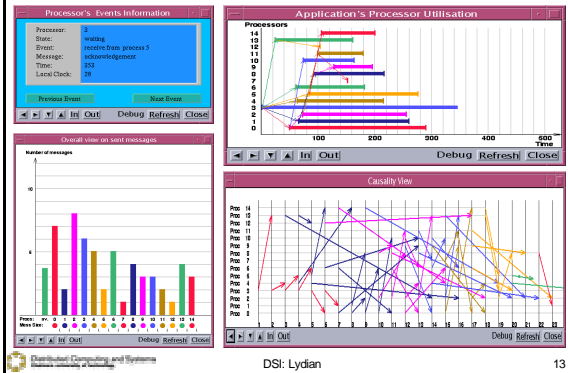
Experiment



Main view



Other animation aspects



Making new protocol

- Design

	$state_1$	$state_2$...	$state_n$
$event_1$				
$event_2$				
...				
$event_m$				

An example: Broadcast with ACK

	SLEEPING	WAITING	DONE
BRD	<ul style="list-style-type: none"> if (leaf) {reply with ACK; new_state==DONE;} <ul style="list-style-type: none"> parent = sender; send BRD to other adjacent processes; store receivers in a set ack; new_state = WAITING; 	reply with ACK;	reply with ACK;
ACK	Never	<ul style="list-style-type: none"> delete sender from ack; if (ack == empty) and (parent != nil) {send ACK to parent; new_state = DONE} if (ack == empty) and (parent == nil) finished; 	Never

Protocol framework

- Implementation

```

typedef struct {
    /* all local variables must be declared here */
    ...
} REGISTERS;
REGISTERS = REG;

reg_alloc() {
    REG = (REGISTERS *) malloc(processes * sizeof(REGISTERS));
}

init() {
    /* all local variables must be initialized here */
    ...
}

/* Below are your own procedures */
    
```

Useful references

- Message structure


```
typedef struct mess { ...
int kind, value[10], from, port; } MESSAGES;
```
- Process Control Block (PCB)


```
typedef struct { ...
int state, id, adjacents; } PCBS;
```
- Global variables


```
me: id of current process
CURMESS: current received message
new_state: the new state of current process
processes: the number of processes
```
- Message routines


```
MESSAGE* create_message()
send_to(MESSAGE *m, int port)
```
- List handling


```
init_list( LIST l)
insert_list( int e, LIST l)
delete_list( int e, LIST l)
int empty_list( LIST l)
int in_list( int e, LIST l)
```
- Timer manipulation


```
start_time( TIMER1[2,3,4,5], int delay)
stop_timer( TIMER1[2,3,4,5])
```
- Miscellaneous


```
simul_end() /*stop simulation*/
init_event( int pid) /*send INITPROTOCOL event to pid*/
debug("p", "%d %s ...", _value, _string, ...)
int uniform( int lower, int upper) /*random*/
```

Ex: Broadcast with ACK (cont.)

```

typedef struct {
    int parent;
    LIST ack;
} REGISTERS;
REGISTERS = REG;

reg_alloc() {
    REG = (REGISTERS *) malloc(processes * sizeof(REGISTERS));
}

init() {
    for(i = 0; i < processes; i++) {
        REG[i].parent = -1;
        init_list(REG[i].ack);
    }
}
    
```

Ex: Broadcast with ACK (cont.)

```

sleeping_brd() {
  if( PCB[me].adjacents == 1) {
    mess = create_message();
    mess->kind = ACK;
    send_to(mess, CURMESS->port);
    new_state = DONE;
  }
  else {
    REG[me].parent = CURMESS->port;
    for( i=0; i < PCB[me].adjacents; i++)
      if( i != REG[me].parent )
        insert_list( i, REG[me].ack);
    mess = create_message();
    mess->kind = BRD;
    send_to( mess, i);
    new_state = WAITING;
  }
}

waiting_brd() { /*done_brd*/
  mess = create_message();
  mess->kind = ACK;
  send_to( mess, CURMESS->port);
}

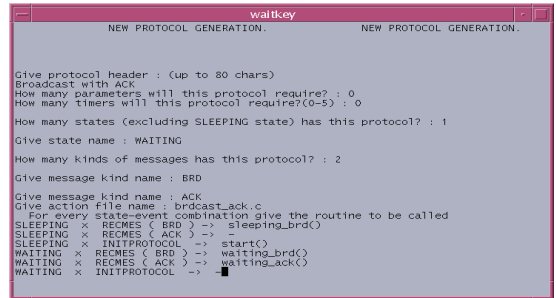
waiting_ack() {
  delete_list( CURMESS->port, REG[me].ack);
  if( empty_list( REG[me].ack) ) {
    if( REG[me].parent == -1) { /* initiator */
      simul_end(); return;
    }
    else { /* send ACK to its parent */
      mess = create_message();
      mess->kind = ACK;
      send_to( mess, REG[me].parent);
      new_state = DONE;
    }
  }
}

start() { /* initiators start simulation*/
  if( PCB[me].adjacents == 0) {
    simul_end(); return;
  }
  for( i=0; i < PCB[me].adjacents; i++)
    insert_list( i, REG[me].ack);
  mess = create_message();
  mess->kind = BRD;
  send_to( mess, i);
  new_state = WAITING;
}

```

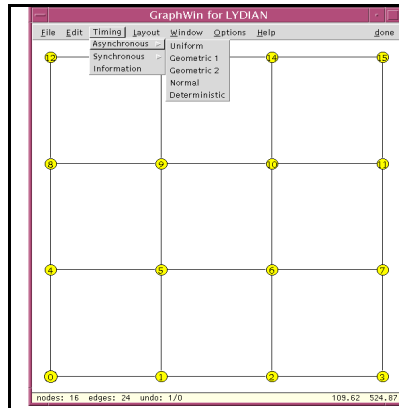
Adding new protocols to LYDIAN

- Copy source file to directory your_installed_LYDIAN/DIAS



Making new networks on GraphWin

- Supports the creation of network description files
- Interconnection of processes is represented by a graph (nodes=processes; edges=links)
- Supports an easy creation of networks as well as an easy specification of timing parameters
- Supports a set of graph drawing algorithms for designing the layout of the network
- The created structure of the network can be used by an animation to create the same layout as it was defined by the user
- Notes:
 - GraphWin menu was changed. All functionalities for creating Lydian networks will appear by pressing *Network* button.
 - Use tool *your_installed_LYDIAN/DIAS/nwconvert* to convert files *NWfiledir*.ipf* generated by GraphWin to the correct format.



GraphWin demo

ASCII monitoring

```

waitkey
-----
Time: 136      Tot. Mess. Sent: 25      Cur. Proc.: 9      Id: 9
Proc.: 9      ( WAITING ) x (RECME: BROADCAST) -> ( WAITING ) x (SEND TO: 8) 3
-----
Process: 2      Process: 7      Process: 8      Process: 9
ID number: 2      ID number: 7      ID number: 8      ID number: 9
WAITING          SLEEPING          SLEEPING          WAITING
-----
RECME Port: 4      RECME Port: 0      RECME Port: 1      RECME Port: 1
BROADCAST        BROADCAST        BROADCAST        BROADCAST
Value: .3         Value: .3         Value: .3         Value: .3
From node: 0      From node: 11      From node: 1      From node: 8
-----
WAITING          SEND TO 11        WAITING          WAITING
SEND TO 0        ACKNOW           SEND TO 2        SEND TO 8
Val. sent: 3      Val. sent: 3      Val. sent: 3      Val. sent: 3
-----
Process: 0      Process: 11       Process: 14      Process: 13
ID number: 0      ID number: 11     ID number: 14     ID number: 13
WAITING          SLEEPING          SLEEPING          WAITING
-----
RECME Port: 3      RECME Port: 0      RECME Port: 0      RECME Port: 2
BROADCAST        BROADCAST        BROADCAST        ACKNOW
Value: .3         Value: .3         Value: .3         Value: .3
From node: 6      From node: 2      From node: 5      From node: 12
-----
WAITING          WAITING          WAITING          WAITING
SEND TO 6        SEND TO 7        SEND TO 4        SEND TO 4
ACKNOW          BROADCAST        BROADCAST        BROADCAST
Val. sent: 3      Val. sent: 3      Val. sent: 3      Val. sent: -

```

More information

- <http://www.cs.chalmers.se/~lydian>
- LYDIAN Lab. in the Distributed Systems I course
 - <http://www.cs.chalmers.se/~phuong/lydianLab.html>