ACE Framework
ADAPTIVE Communication Environment

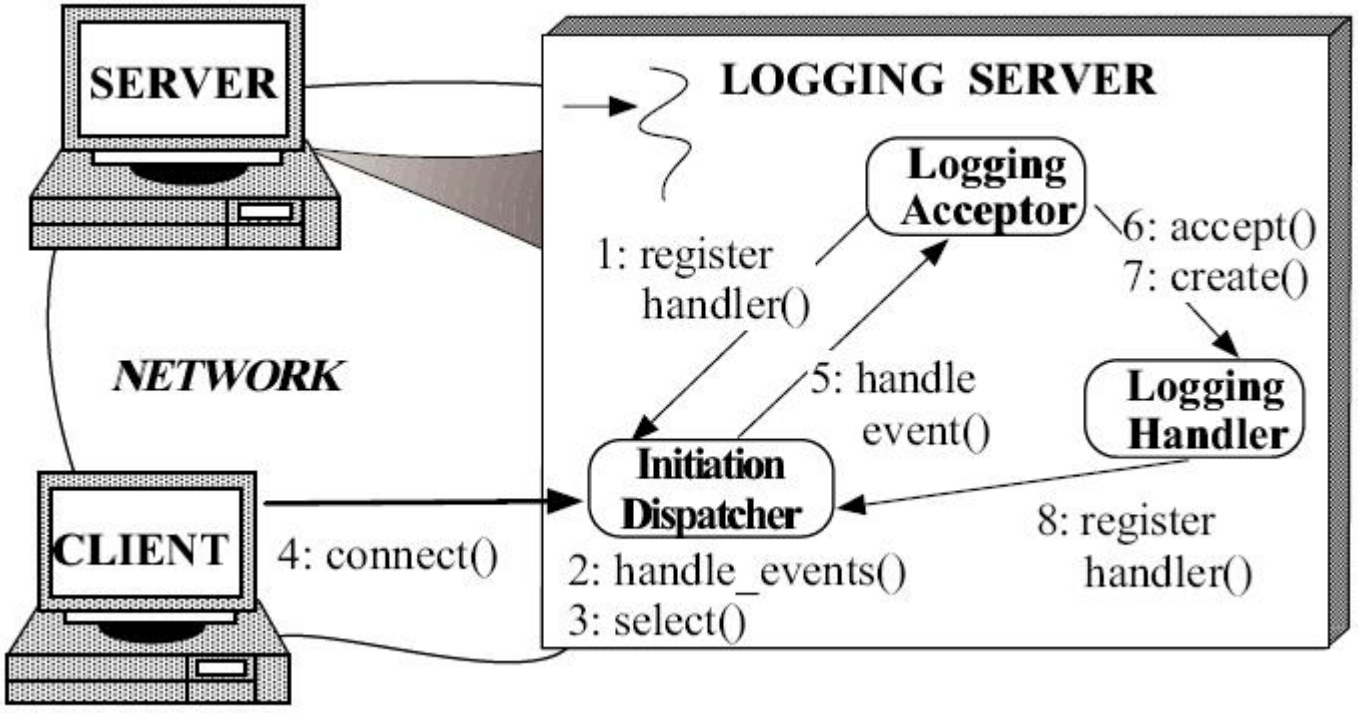
Kyoungho An
Dept. of EECS, Vanderbilt University
July 26, 2012

Presentation Roadmap

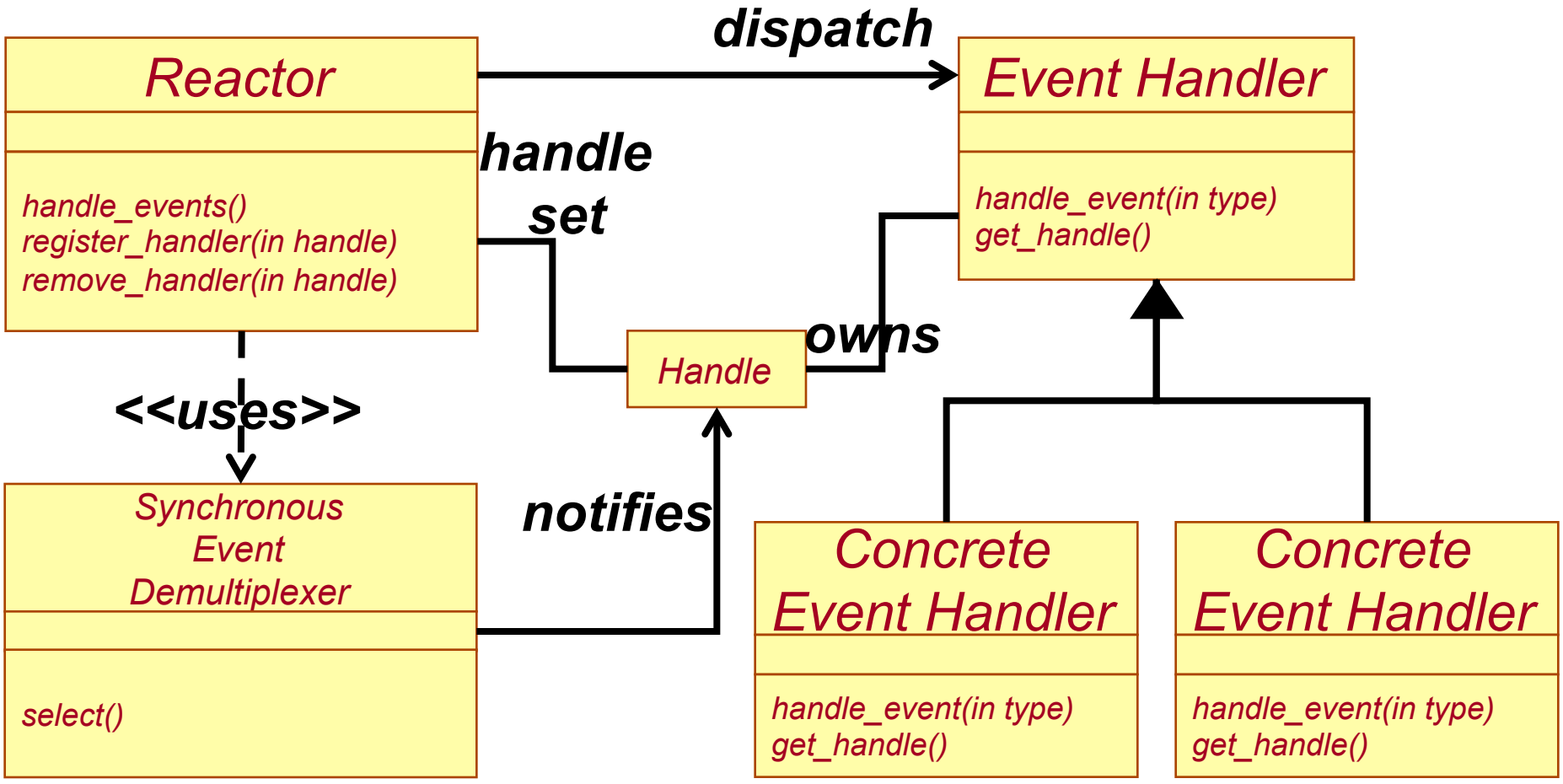
- Reactor Framework
- Acceptor-Connector Framework
- Proactor Framework
- Service Configurator Framework

Reactor Framework Overview

- Implementation of Reactor pattern
- Handling various events occurred by input, output, timer, or signal

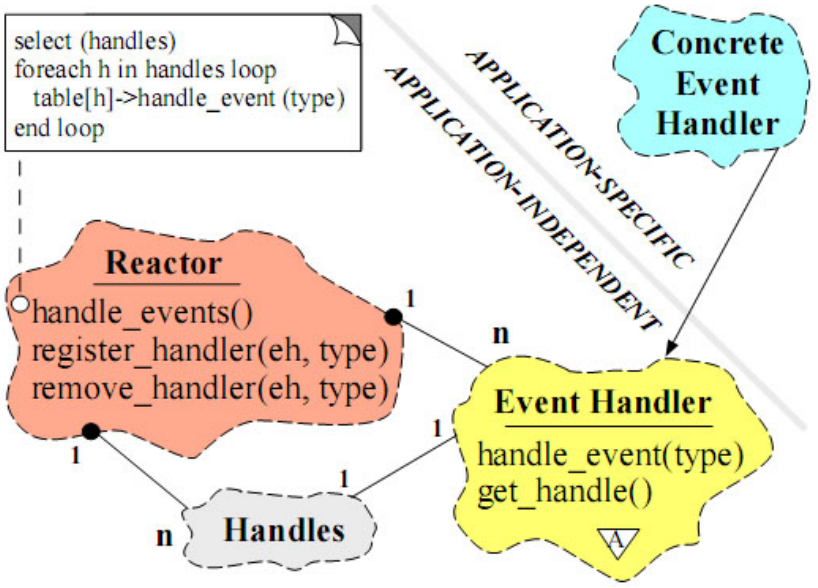


Reactor Pattern



Reactor Framework Overview

- 3 steps to use ACE Reactor framework
  1. Inherit from the ACE_Event_Handler class and implement event-handling behavior in call-back method
  2. Register the implemented event-handler object to ACE_Reactor object
  3. Execute event loop of the ACE_Reactor object



ClientAcceptor Class (1/2)

```
class ClientAcceptor : public ACE_Event_Handler
{
public:
  virtual ~ClientAcceptor ();

  int open (const ACE_INET_Addr &listen_addr);

  // Get this handler's I/O handle.
  virtual ACE_HANDLE get_handle (void) const
    { return this->acceptor_.get_handle (); }

  // Called when a connection is ready to accept.
  virtual int handle_input (ACE_HANDLE fd = ACE_INVALID_HANDLE);
```

ClientAcceptor Class (2/2)

```
  // Called when this handler is removed from the ACE_Reactor.
  virtual int handle_close (ACE_HANDLE handle,
                            ACE_Reactor_Mask close_mask);

protected:
  ACE_SOCK_Acceptor acceptor_;
};
```

ClientAcceptor::open ()

```
int
ClientAcceptor::open (const ACE_INET_Addr &listen_addr)
{
  if (this->acceptor_.open (listen_addr, 1) == -1)
    ACE_ERROR_RETURN ((LM_ERROR,
                       ACE_TEXT ("%p\n"),
                       ACE_TEXT ("acceptor.open")),
                      -1);
  return this->reactor ()->register_handler
    (this, ACE_Event_Handler::ACCEPT_MASK);
}
```

ClientAcceptor Main Function

```
int ACE_TMAIN (int, ACE_TCHAR *[])
{
  ACE_INET_Addr port_to_listen (12345);
  ClientAcceptor acceptor;
  acceptor.reactor (ACE_Reactor::instance ());
  if (acceptor.open (port_to_listen) == -1)
    return 1;

  ACE_Reactor::instance ()->run_reactor_event_loop ();

  return (0);
}
```

ClientAcceptor::handle_input () (1/2)

```
int
ClientAcceptor::handle_input (ACE_HANDLE)
{
  ClientService *client;
  ACE_NEW_RETURN (client, ClientService, -1);
  auto_ptr<ClientService> p (client);

  if (this->acceptor_.accept (client->peer ()) == -1)
    ACE_ERROR_RETURN ((LM_ERROR,
                       ACE_TEXT ("(%P|%t) %p\n"),
                       ACE_TEXT ("Failed to accept ")
                       ACE_TEXT ("client connection")),
                      -1);
```

ClientAcceptor::handle_input () (2/2)

```
p.release ();
  client->reactor (this->reactor ());
  if (client->open () == -1)
    client->handle_close (ACE_INVALID_HANDLE, 0);
  return 0;
}
```

ClientService::open ()

```
int
ClientService::open (void)
{
  ACE_TCHAR peer_name[MAXHOSTNAMELEN];
  ACE_INET_Addr peer_addr;
  if (this->sock_.get_remote_addr (peer_addr) == 0 &&
      peer_addr.addr_to_string (peer_name, MAXHOSTNAMELEN) == 0)
    ACE_DEBUG ((LM_DEBUG,
                ACE_TEXT ("(%P|%t) Connection from %s\n"),
                peer_name));
  return this->reactor ()->register_handler
    (this, ACE_Event_Handler::READ_MASK);
}
```

ClientAcceptor::handle_close ()

```
int
ClientAcceptor::handle_close (ACE_HANDLE, ACE_Reactor_Mask)
{
  if (this->acceptor_.get_handle () != ACE_INVALID_HANDLE)
    {
      ACE_Reactor_Mask m = ACE_Event_Handler::ACCEPT_MASK |
                           ACE_Event_Handler::DONT_CALL;
      this->reactor ()->remove_handler (this, m);
      this->acceptor_.close ();
    }
  return 0;
}
```

ClientService Class (1/2)

```
class ClientService : public ACE_Event_Handler
{
public:
  ACE_SOCK_Stream &peer (void) { return this->sock_; }

  int open (void);

  // Get this handler's I/O handle.
  virtual ACE_HANDLE get_handle (void) const
    { return this->sock_.get_handle (); }
```

ClientService Class (2/2)

```
// Called when input is available from the client.
virtual int handle_input (ACE_HANDLE fd = ACE_INVALID_HANDLE);

// Called when output is possible.
virtual int handle_output (ACE_HANDLE fd = ACE_INVALID_HANDLE);

// Called when this handler is removed from the ACE_Reactor.
virtual int handle_close (ACE_HANDLE handle,
                          ACE_Reactor_Mask close_mask);

protected:
  ACE_SOCK_Stream sock_;
  ACE_Message_Queue<ACE_NULL_SYNCH> output_queue_;
};
```

ClientService::handle_input () (1/4)

```
int
ClientService::handle_input (ACE_HANDLE)
{
  const size_t INPUT_SIZE = 4096;
  char buffer[INPUT_SIZE];
  ssize_t recv_cnt, send_cnt;

  if ((recv_cnt = this->sock_.recv (buffer, sizeof(buffer)) ) <= 0)
    {
      ACE_DEBUG ((LM_DEBUG,
                  ACE_TEXT ("(%P|%t) Connection closed\n")));
      return -1;
    }
```

ClientService::handle_input () (2/4)

```
send_cnt =
  this->sock_.send (buffer, static_cast<size_t> (recv_cnt));
if (send_cnt == recv_cnt)
  return 0;
if (send_cnt == -1 && ACE_OS::last_error () != EWOULDBLOCK)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("(%P|%t) %p\n"),
                     ACE_TEXT ("send")),
                    0);
if (send_cnt == -1)
  send_cnt = 0;
ACE_Message_Block *mb = 0;
size_t remaining =
  static_cast<size_t> ((recv_cnt - send_cnt));
```

ClientService::handle_input () (3/4)

```
ACE_NEW_RETURN (mb, ACE_Message_Block (remaining), -1);
  mb->copy (&buffer[send_cnt], remaining);
  int output_off = this->output_queue_.is_empty ();
  ACE_Time_Value nowait (ACE_OS::gettimeofday ());
  if (this->output_queue_.enqueue_tail (mb, &nowait) == -1)
    {
      ACE_ERROR ((LM_ERROR,
                  ACE_TEXT ("(%P|%t) %p; discarding data\n"),
                  ACE_TEXT ("enqueue failed")));
      mb->release ();
      return 0;
    }
```

ClientService::handle_input () (4/4)

```
if (output_off)
    return this->reactor ()->register_handler
      (this, ACE_Event_Handler::WRITE_MASK);
  return 0;
}
```

ClientService::handle_output () (1/2)

```
int
ClientService::handle_output (ACE_HANDLE)
{
  ACE_Message_Block *mb = 0;
  ACE_Time_Value nowait (ACE_OS::gettimeofday ());
  while (0 <= this->output_queue_.dequeue_head
                              (mb, &nowait))
    {
      ssize_t send_cnt =
        this->sock_.send (mb->rd_ptr (), mb->length ());
      if (send_cnt == -1)
        ACE_ERROR ((LM_ERROR,
                    ACE_TEXT ("(%P|%t) %p\n"),
                    ACE_TEXT ("send")));
```

ClientService::handle_output () (2/2)

```
    else
      mb->rd_ptr (static_cast<size_t> (send_cnt));
    if (mb->length () > 0)
      {
        this->output_queue_.enqueue_head (mb);
        break;
      }
    mb->release ();
  }
 return (this->output_queue_.is_empty ()) ? -1 : 0;
}
```

ClientService::handle_close ()

```
int
ClientService::handle_close (ACE_HANDLE, ACE_Reactor_Mask mask)
{
  if (mask == ACE_Event_Handler::WRITE_MASK)
    return 0;
  mask = ACE_Event_Handler::ALL_EVENTS_MASK |
         ACE_Event_Handler::DONT_CALL;
  this->reactor ()->remove_handler (this, mask);
  this->sock_.close ();
  this->output_queue_.flush ();
  delete this;
  return 0;
}
```

Acceptor-Connector Structure

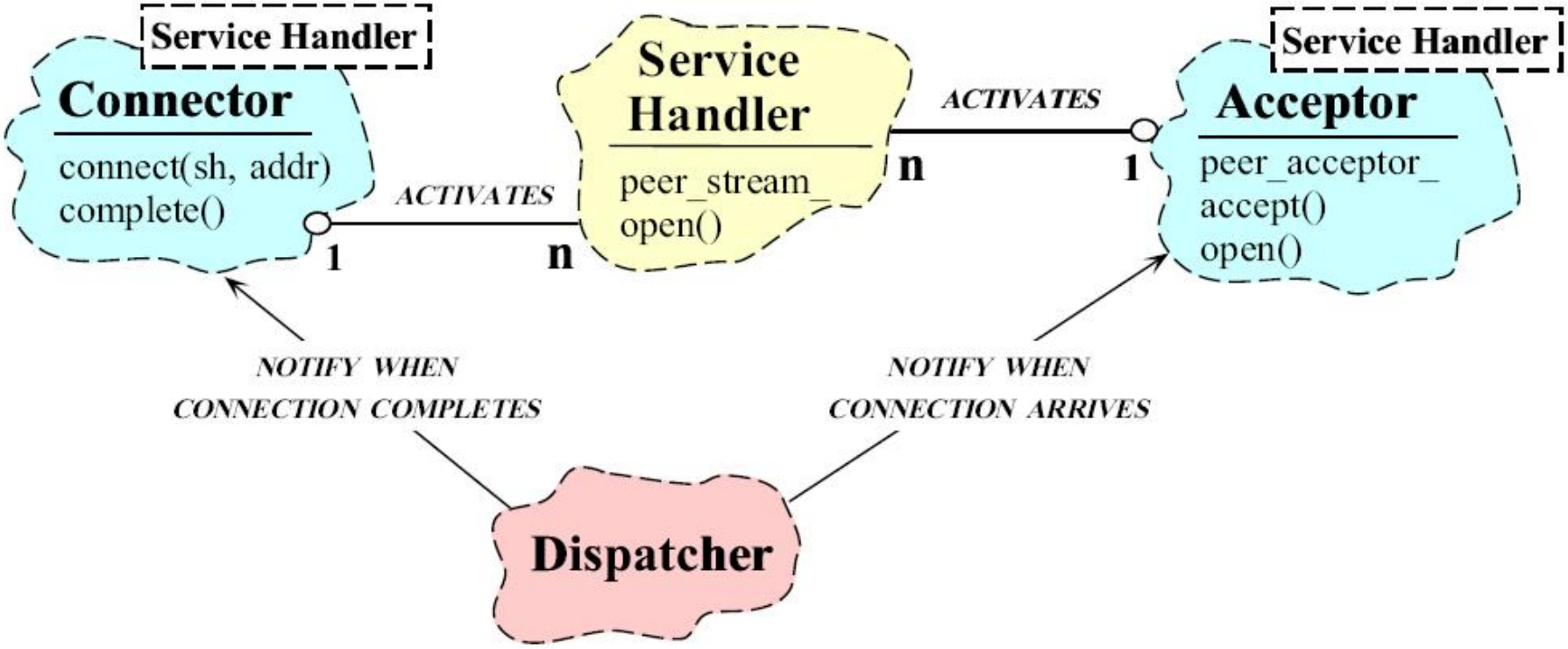


Figure 2: Structure of Participants in the Acceptor-Connector Pattern

Client Acceptor Object

```
typedef ACE_Acceptor<ClientService,
ACE_SOCK_ACCEPTOR>
  ClientAcceptor;
```

ClientAcceptor Main Function

```
int ACE_TMAIN (int, ACE_TCHAR *[])
{
  ACE_INET_Addr port_to_listen (12345);
  ClientAcceptor acceptor;
  if (acceptor.open (port_to_listen,
                     ACE_Reactor::instance (),
                     ACE_NONBLOCK) == -1)
    return 1;

  ACE_Reactor::instance ()->run_reactor_event_loop ();

  return (0);
}
```

ClientService Class (1/2)

```
class ClientService :
  public ACE_Svc_Handler<ACE_SOCK_STREAM, ACE_NULL_SYNCH>
{
  typedef ACE_Svc_Handler<ACE_SOCK_STREAM, ACE_NULL_SYNCH>
super;

public:
  int open (void * = 0);

  // Called when input is available from the client.
  virtual int handle_input (ACE_HANDLE fd = ACE_INVALID_HANDLE);
```

ClientService Class (2/2)

```
  // Called when output is possible.
  virtual int handle_output (ACE_HANDLE fd = ACE_INVALID_HANDLE);

  // Called when this handler is removed from the ACE_Reactor.
  virtual int handle_close (ACE_HANDLE handle,
                            ACE_Reactor_Mask close_mask);
};
```

ClientService::open ()

```
int
ClientService::open (void *p)
{
  if (super::open (p) == -1)
    return -1;

  ACE_TCHAR peer_name[MAXHOSTNAMELEN];
  ACE_INET_Addr peer_addr;
  if (this->peer ().get_remote_addr (peer_addr) == 0 &&
      peer_addr.addr_to_string (peer_name, MAXHOSTNAMELEN) == 0)
    ACE_DEBUG ((LM_DEBUG,
                ACE_TEXT ("(%P|%t) Connection from %s\n"),
                peer_name));
  return 0;
}
```

ClientService::handle_input () (1/4)

```
int
ClientService::handle_input (ACE_HANDLE)
{
  const size_t INPUT_SIZE = 4096;
  char buffer[INPUT_SIZE];
  ssize_t recv_cnt, send_cnt;

  recv_cnt = this->peer ().recv (buffer, sizeof(buffer));
  if (recv_cnt <= 0)
    {
      ACE_DEBUG ((LM_DEBUG,
                  ACE_TEXT ("(%P|%t) Connection closed\n")));
      return -1;
    }
```

ClientService::handle_input () (2/4)

```
send_cnt =
  this->peer ().send (buffer,
                      static_cast<size_t> (recv_cnt));
if (send_cnt == recv_cnt)
  return 0;
if (send_cnt == -1 && ACE_OS::last_error () != EWOULDBLOCK)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("(%P|%t) %p\n"),
                     ACE_TEXT ("send")),
                    0);
if (send_cnt == -1)
  send_cnt = 0;
ACE_Message_Block *mb = 0;
```

ClientService::handle_input () (3/4)

```
size_t remaining =
  static_cast<size_t> ((recv_cnt - send_cnt));
ACE_NEW_RETURN (mb, ACE_Message_Block (remaining), -1);
mb->copy (&buffer[send_cnt], remaining);
int output_off = this->msg_queue ()->is_empty ();
ACE_Time_Value nowait (ACE_OS::gettimeofday ());
if (this->putq (mb, &nowait) == -1)
  {
    ACE_ERROR ((LM_ERROR,
                ACE_TEXT ("(%P|%t) %p; discarding data\n"),
                ACE_TEXT ("enqueue failed")));
    mb->release ();
    return 0;
  }
```

ClientService::handle_input () (4/4)

```
if (output_off)
    return this->reactor ()->register_handler
      (this, ACE_Event_Handler::WRITE_MASK);
  return 0;
}
```

ClientService::handle_output () (1/2)

```
int
ClientService::handle_output (ACE_HANDLE)
{
  ACE_Message_Block *mb = 0;
  ACE_Time_Value nowait (ACE_OS::gettimeofday ());
  while (-1 != this->getq (mb, &nowait))
    {
      ssize_t send_cnt =
        this->peer ().send (mb->rd_ptr (), mb->length ());
      if (send_cnt == -1)
        ACE_ERROR ((LM_ERROR,
                    ACE_TEXT ("(%P|%t) %p\n"),
                    ACE_TEXT ("send")));
```

ClientService::handle_output () (2/2)

```
else
        mb->rd_ptr (static_cast<size_t> (send_cnt));
      if (mb->length () > 0)
        {
          this->ungetq (mb);
          break;
        }
      mb->release ();
    }
  return (this->msg_queue ()->is_empty ()) ? -1 : 0;
}
```

ClientService::handle_close ()

```
int
ClientService::handle_close (ACE_HANDLE h, ACE_Reactor_Mask mask)
{
  if (mask == ACE_Event_Handler::WRITE_MASK)
    return 0;
  return super::handle_close (h, mask);
}
```

Client Connector Class (1/2)

```
class Client :
    public ACE_Svc_Handler<ACE_SOCK_STREAM, ACE_NULL_SYNCH>
{
  typedef ACE_Svc_Handler<ACE_SOCK_STREAM, ACE_NULL_SYNCH>
super;

public:
  Client () : notifier_ (0, this, ACE_Event_Handler::WRITE_MASK)
      {}

  virtual int open (void * = 0);

  // Called when input is available from the client.
  virtual int handle_input (ACE_HANDLE fd = ACE_INVALID_HANDLE);
```

Client Connector Class (2/2)

```
  // Called when output is possible.
  virtual int handle_output (ACE_HANDLE fd = ACE_INVALID_HANDLE);

  // Called when a timer expires.
  virtual int handle_timeout (const ACE_Time_Value &current_time,
                              const void *act = 0);

private:
  enum { ITERATIONS = 5 };
  int iterations_;
  ACE_Reactor_Notification_Strategy notifier_;
};
```

Client::open ()

```
int Client::open (void *p)
{
  ACE_Time_Value iter_delay (2);   // Two seconds
  if (super::open (p) == -1)
    return -1;
  this->notifier_.reactor (this->reactor ());
  this->msg_queue ()->notification_strategy (&this->notifier_);
  this->iterations_ = 0;
  return this->reactor ()->schedule_timer
    (this, 0, ACE_Time_Value::zero, iter_delay);
}
```

Client::handle_input () (1/2)

```
int Client::handle_input (ACE_HANDLE)
{
  char buf[64];
  ssize_t recv_cnt = this->peer ().recv (buf, sizeof (buf) - 1);
  if (recv_cnt > 0)
    {
      ACE_DEBUG ((LM_DEBUG, ACE_TEXT ("%.*C"),
                  static_cast<int> (recv_cnt),
                  buf));
      return 0;
    }
```

Client::handle_input () (2/2)

```
  if (recv_cnt == 0 || ACE_OS::last_error () != EWOULDBLOCK)
    {
      this->reactor ()->end_reactor_event_loop ();
      return -1;
    }
  return 0;
}
```

Client::handle_timeout () (1/2)

```
int Client::handle_timeout(const ACE_Time_Value &, const void *)
{
  if (++this->iterations_ >= ITERATIONS)
    {
      this->peer ().close_writer ();
      return 0;
    }
```

Client::handle_timeout () (2/2)

```
  ACE_Message_Block *mb = 0;
  ACE_NEW_RETURN (mb, ACE_Message_Block (128), -1);
  int nbytes = ACE_OS::sprintf
    (mb->wr_ptr (), "Iteration %d\n", this->iterations_);
  ACE_ASSERT (nbytes > 0);
  mb->wr_ptr (static_cast<size_t> (nbytes));
  this->putq (mb);
  return 0;
}
```

Client::handle_output () (1/3)

```
int Client::handle_output (ACE_HANDLE)
{
  ACE_Message_Block *mb = 0;
  ACE_Time_Value nowait (ACE_OS::gettimeofday ());
  while (-1 != this->getq (mb, &nowait))
    {
      ssize_t send_cnt =
        this->peer ().send (mb->rd_ptr (), mb->length ());
      if (send_cnt == -1)
        ACE_ERROR ((LM_ERROR,
                    ACE_TEXT ("(%P|%t) %p\n"),
                    ACE_TEXT ("send")));
```

Client::handle_output () (2/3)

```
  else
    mb->rd_ptr (static_cast<size_t> (send_cnt));
  if (mb->length () > 0)
    {
      this->ungetq (mb);
      break;
    }
  mb->release ();
}
```

Client::handle_output () (3/3)

```
  if (this->msg_queue ()->is_empty ())
    this->reactor ()->cancel_wakeup
      (this, ACE_Event_Handler::WRITE_MASK);
  else
    this->reactor ()->schedule_wakeup
      (this, ACE_Event_Handler::WRITE_MASK);
  return 0;
}
```

Client Connect Main Function

```
int ACE_TMAIN (int, ACE_TCHAR *[])
{
  ACE_INET_Addr port_to_connect (ACE_TEXT (12345),
ACE_LOCALHOST);
  ACE_Connector<Client, ACE_SOCK_CONNECTOR> connector;
  Client client;
  Client *pc = &client;
  if (connector.connect (pc, port_to_connect) == -1)
    ACE_ERROR_RETURN ((LM_ERROR, ACE_TEXT ("%p\n"),
                       ACE_TEXT ("connect")), 1);

  ACE_Reactor::instance ()->run_reactor_event_loop ();
  return (0);
}
```

Proactor Framework Overview

- Asynchronous I/O

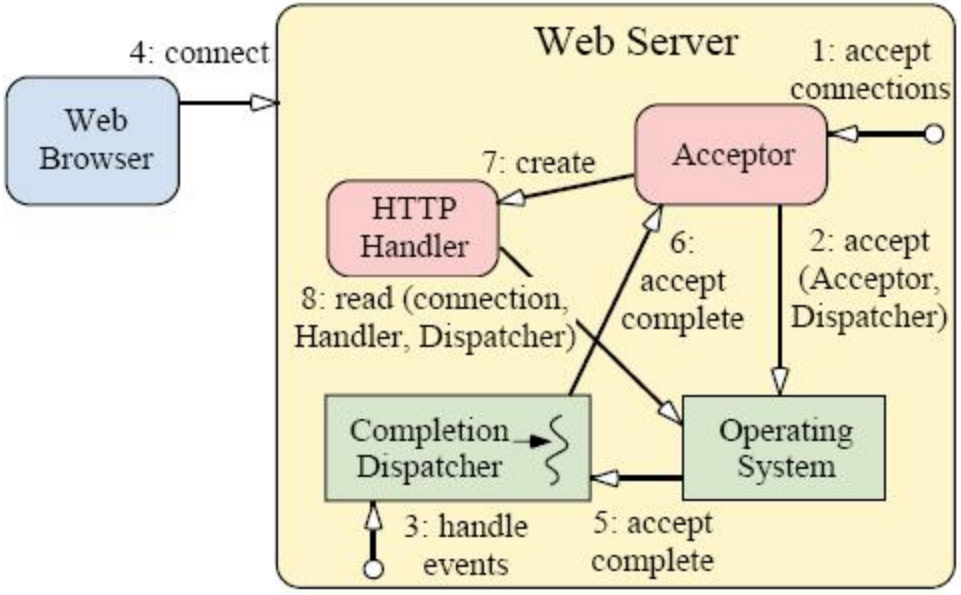
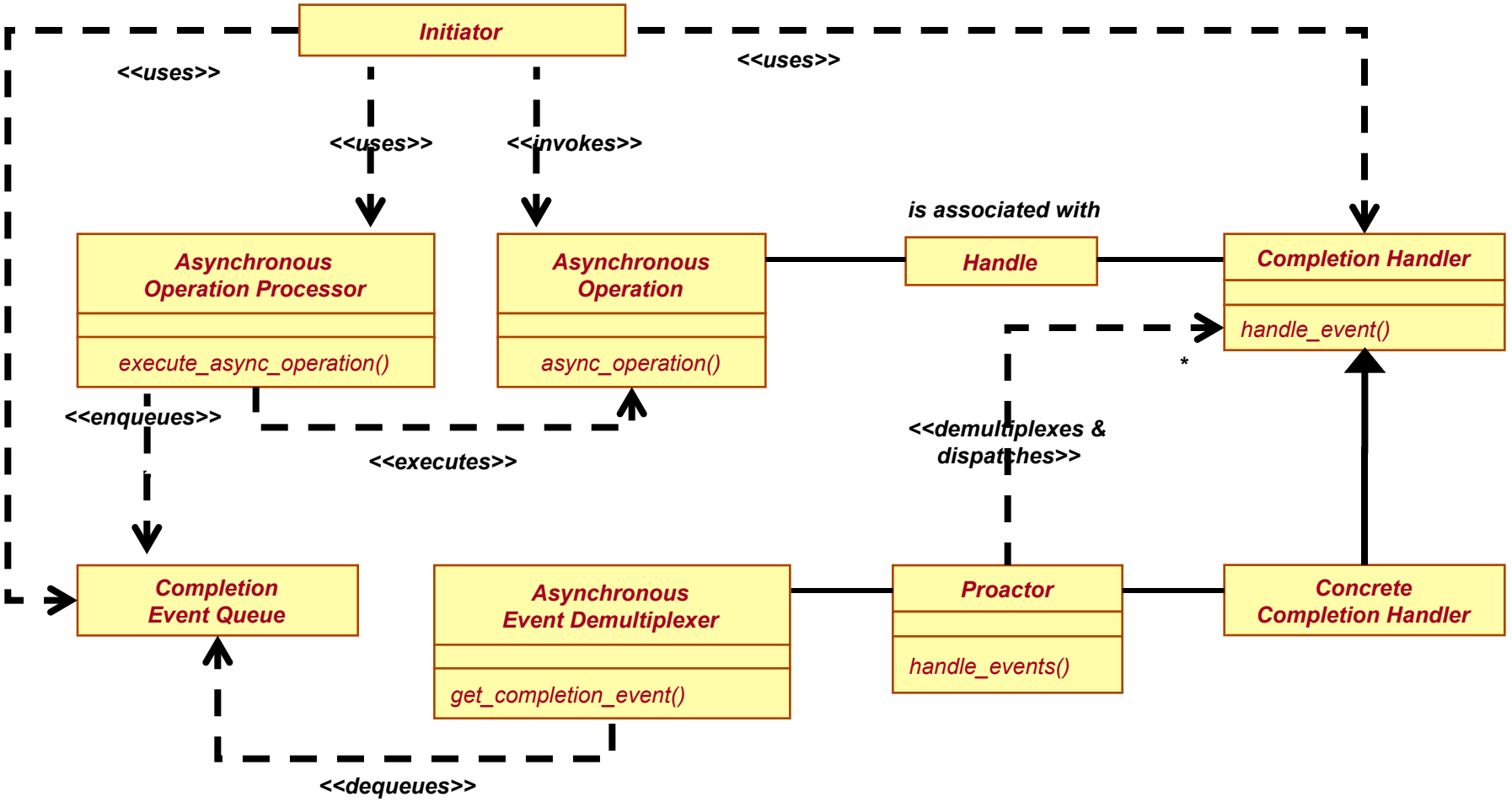
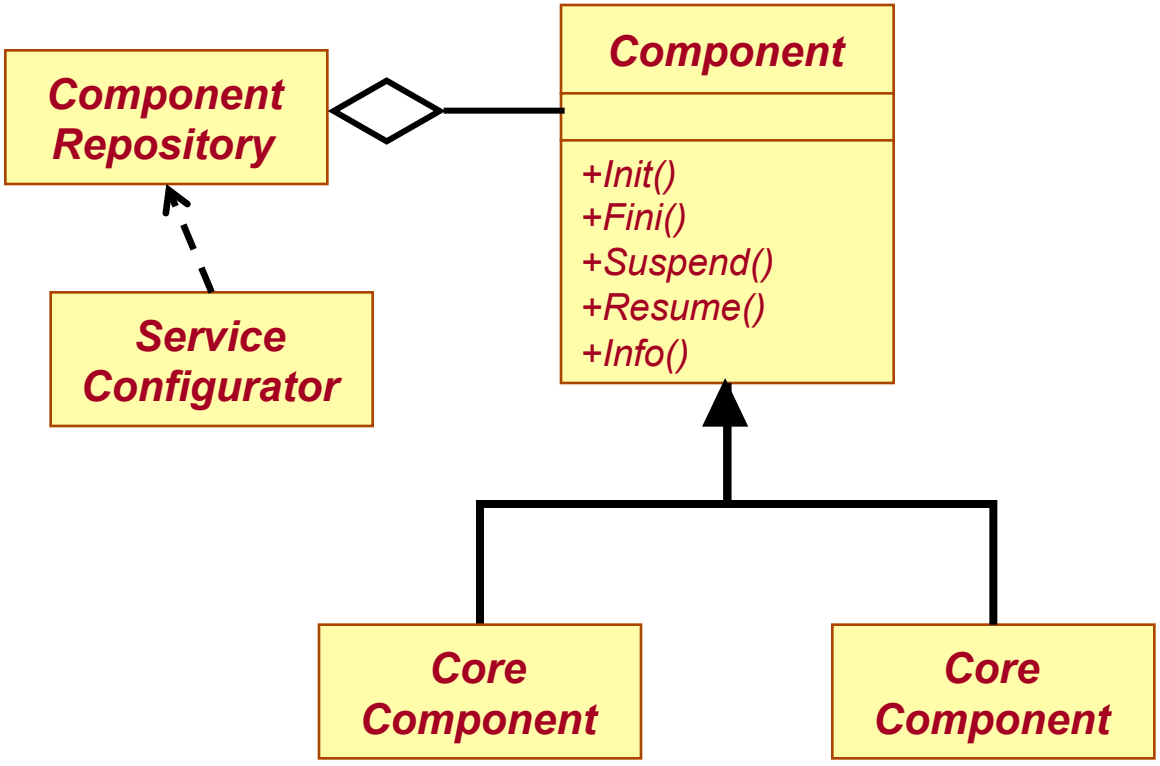


Figure 5: Client connects to a Proactor-based Web Server

Proactor Pattern



Service Configurator Pattern



HA_Status (Static) Class

```
class HA_Status : public ACE_Service_Object
{
  public:
    virtual int init (int argc, ACE_TCHAR *argv[]);
    virtual int fini (void);
    virtual int info (ACE_TCHAR **str, size_t len) const;

  private:
    ACE_Acceptor<ClientHandler, ACE_SOCK_ACCEPTOR> acceptor_;
    ACE_INET_Addr listen_addr_;
};
```

HA_Status::init () (1/5)

```
int
HA_Status::init (int argc, ACE_TCHAR *argv[])
{
  static const ACE_TCHAR options[] = ACE_TEXT (":f:");
  ACE_Get_Opt cmd_opts (argc, argv, options, 0);
  if (cmd_opts.long_option
      (ACE_TEXT ("config"), 'f', ACE_Get_Opt::ARG_REQUIRED) == -1)
    return -1;
  int option;
  ACE_TCHAR config_file[MAXPATHLEN];
  ACE_OS::strcpy (config_file, ACE_TEXT ("HAStatus.conf"));
```

HA_Status::init () (2/5)

```
while ((option = cmd_opts ()) != EOF)
    switch (option)
     {
     case 'f':
       ACE_OS::strncpy (config_file,
                        cmd_opts.opt_arg (),
                        MAXPATHLEN);
       break;
     case ':':
       ACE_ERROR_RETURN
         ((LM_ERROR, ACE_TEXT ("-%c requires an argument\n"),
           cmd_opts.opt_opt ()),
          -1);
```

HA_Status::init () (3/5)

```
    default:
      ACE_ERROR_RETURN
        ((LM_ERROR, ACE_TEXT ("Parse error.\n")), -1);
    }

ACE_Configuration_Heap config;
config.open ();
ACE_Registry_ImpExp config_importer (config);
if (config_importer.import_config (config_file) == -1)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("%p\n"),
                     config_file),
                    -1);
```

HA_Status::init () (4/5)

```
ACE_Configuration_Section_Key status_section;
if (config.open_section (config.root_section (),
                         ACE_TEXT ("HAStatus"),
                         0,
                         status_section) == -1)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("%p\n"),
                     ACE_TEXT ("Can't open HAStatus section")),
                    -1);

u_int status_port;
```

HA_Status::init () (5/5)

```
if (config.get_integer_value (status_section,
                              ACE_TEXT ("ListenPort"),
                              status_port) == -1)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("HAStatus ListenPort does ")
                     ACE_TEXT ("not exist\n")),
                    -1);
this->listen_addr_.set (static_cast<u_short> (status_port));

if (this->acceptor_.open (this->listen_addr_) != 0)
  ACE_ERROR_RETURN ((LM_ERROR,
                     ACE_TEXT ("HAStatus %p\n"),
                     ACE_TEXT ("accept")),
                    -1); return 0;
```


return 0;

HA_Status::fini ()

```
int
HA_Status::fini (void)
{
  this->acceptor_.close ();
  return 0;
}
```

HA_Status (Static) Macros

```
ACE_FACTORY_DEFINE (ACE_Local_Service, HA_Status)

ACE_STATIC_SVC_DEFINE (HA_Status_Descriptor,
                       ACE_TEXT ("HA_Status_Static_Service"),
                       ACE_SVC_OBJ_T,
                       &ACE_SVC_NAME (HA_Status),
                       ACE_Service_Type::DELETE_THIS |
                       ACE_Service_Type::DELETE_OBJ,
                       0)  // Service not initially active

ACE_STATIC_SVC_REQUIRE (HA_Status_Descriptor)
```

svc.conf (Static)

static HA_Status_Static_Service “-f status.ini”

HA_Status (Static) Main Function

```
int ACE_TMAIN (int argc, ACE_TCHAR *argv[])
{
  ACE_Service_Config::open (argc,
                            argv,
                            ACE_DEFAULT_LOGGER_KEY,
                            0);
  ACE_Reactor::instance ()->run_reactor_event_loop ();
  return 0;
}
```

HA_Status (Dynamic) Class

```
#include "HASTATUS_export.h"
class HASTATUS_Export HA_Status : public ACE_Service_Object
{
  public:
    virtual int init (int argc, ACE_TCHAR *argv[]);

    virtual int fini (void);

    virtual int info (ACE_TCHAR **str, size_t len) const;

  private:
    ACE_Acceptor<ClientHandler, ACE_SOCK_ACCEPTOR> acceptor_;
    ACE_INET_Addr listen_addr_;
};
```

HA_Status (Dynamic) Main Function

```
int ACE_TMAIN (int argc, ACE_TCHAR *argv[])
{
  ACE_Service_Config::open (argc, argv)
  ACE_Reactor::instance ()->run_reactor_event_loop ();
  return 0;
}
```

svc.conf (Dynamic)

dynamic HA_Status_Dynamic_Service Service_Object *
HA_Status:_make_HA_Status () “-f status.ini”

remove HA_Status_Dynamic_Service

suspend HA_Status_Dynamic_Service

resume HA_Status_Dynamic_Service

Questions?

Reference

- http://www.cs.wustl.edu/~schmidt/ACE-overview.html