A Framework for Environment Aware Mobile Applications

Authors: Girish Welling and B. R. Badrinath
Department of computer Science
Rutgers University

CIS 640 Mobile Computing
presented by: Suming Chen
March 4, 1998

Presentation Outline

- Introduction
- Design Objectives
- Architectural Overview
- Event Delivery Framework
  - Entities
- Network Aware Subsystem
  - Illustration
- Performance
- Related and Future Work

Introduction

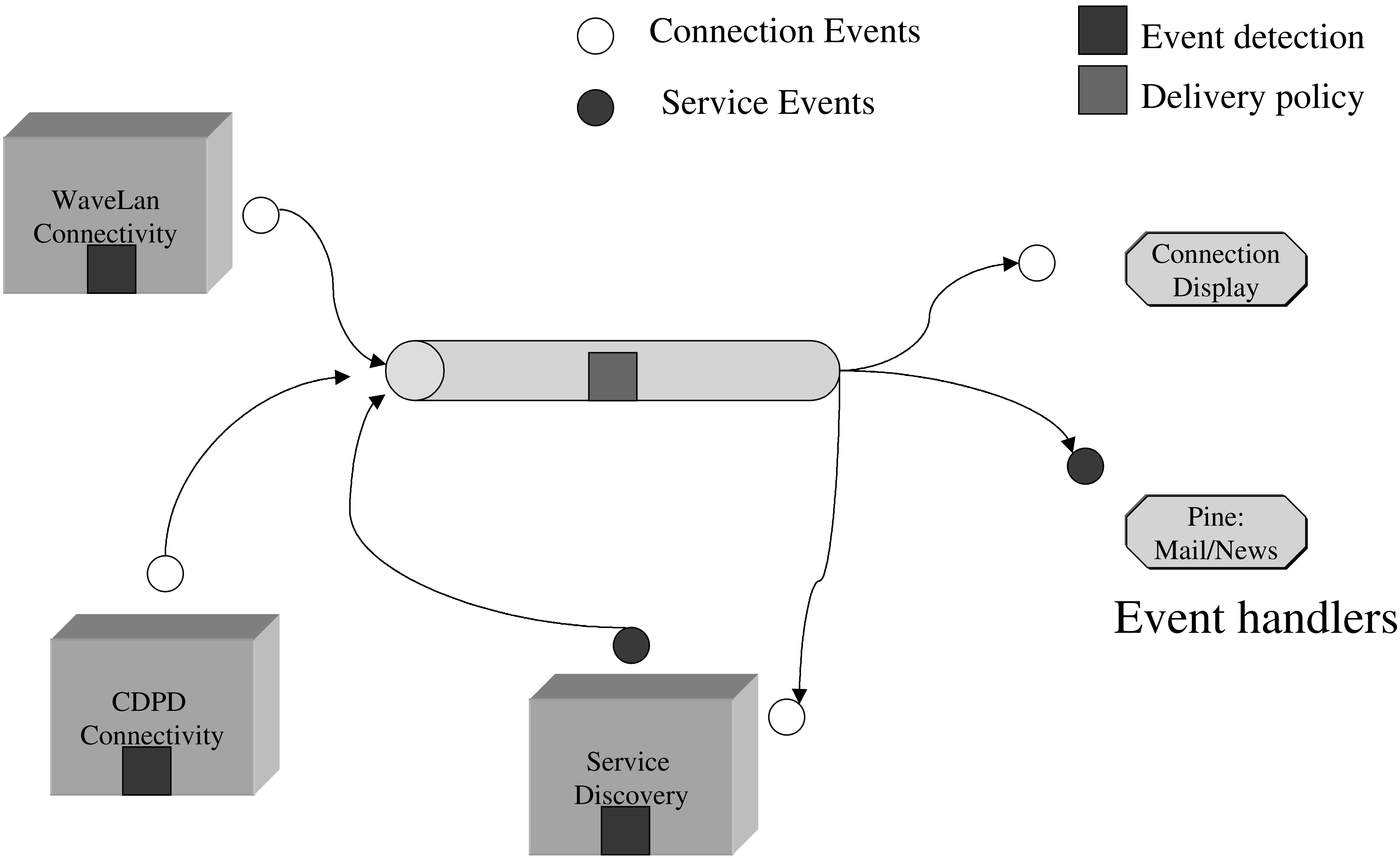
- On current system, operating system allocate resources among applications
  - fixed resources
  - boot-time configuration
- With a mobile computer, resources can be utilized more efficient if application is aware of changes.
  - React to changes voluntarily during execution

Introduction *continue...*

- The need for environment aware mobile applications:
  - resource change
  - dynamic utilization
  - adaptation
  - example: network application
    - need to deal with heterogeneity of different network
    - intermittent connectivity and changing bandwidth
    - usually ignored above transport layer
    - need to export link information to application
      - alternate network services and take advantage of changing link characteristics

Introduction *continue...*

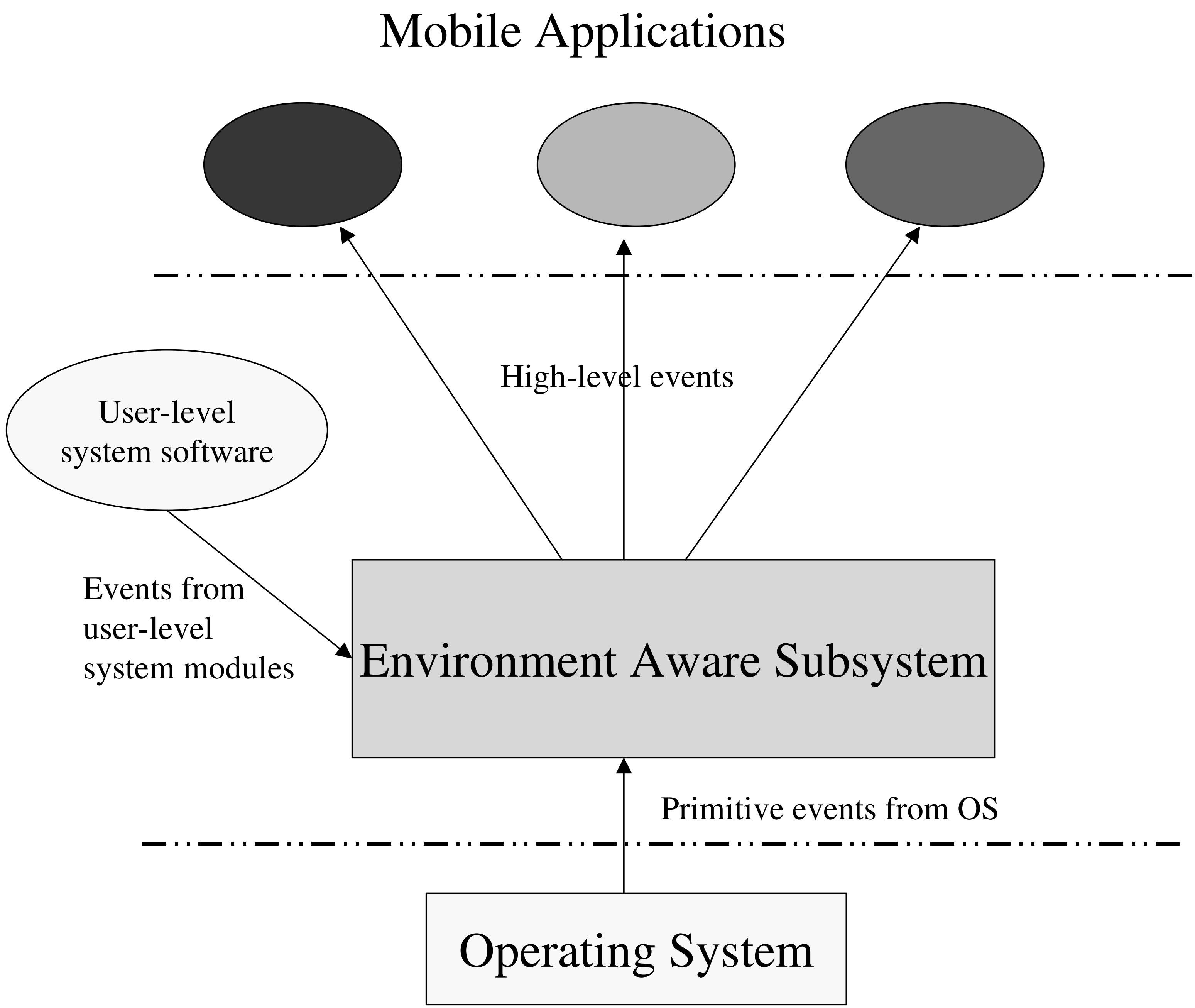
- So the model need to be
  - flexible and extensible for mobile environment
  - aware of resource availability
  - structured to react to changes efficiently
- Approach
  - user-level event detection
  - separation of detection and event delivery
  - preserve state of environment


Connection Events
Service Events
Event detection
Delivery policy
WaveLan Connectivity
Connection Display
Pine: Mail/News
Event handlers
CDPD Connectivity
Service Discovery

Design Objectives

- user-level detection of events
- flexible, event specific delivery policies
  - separation of event detection and delivery
- integrate event notification and transfer of information associated with event
- maintain current state of environment
- reduce event delivery latency by thread scheduling

Architectural Overview



Architectural Overview *continue ...*

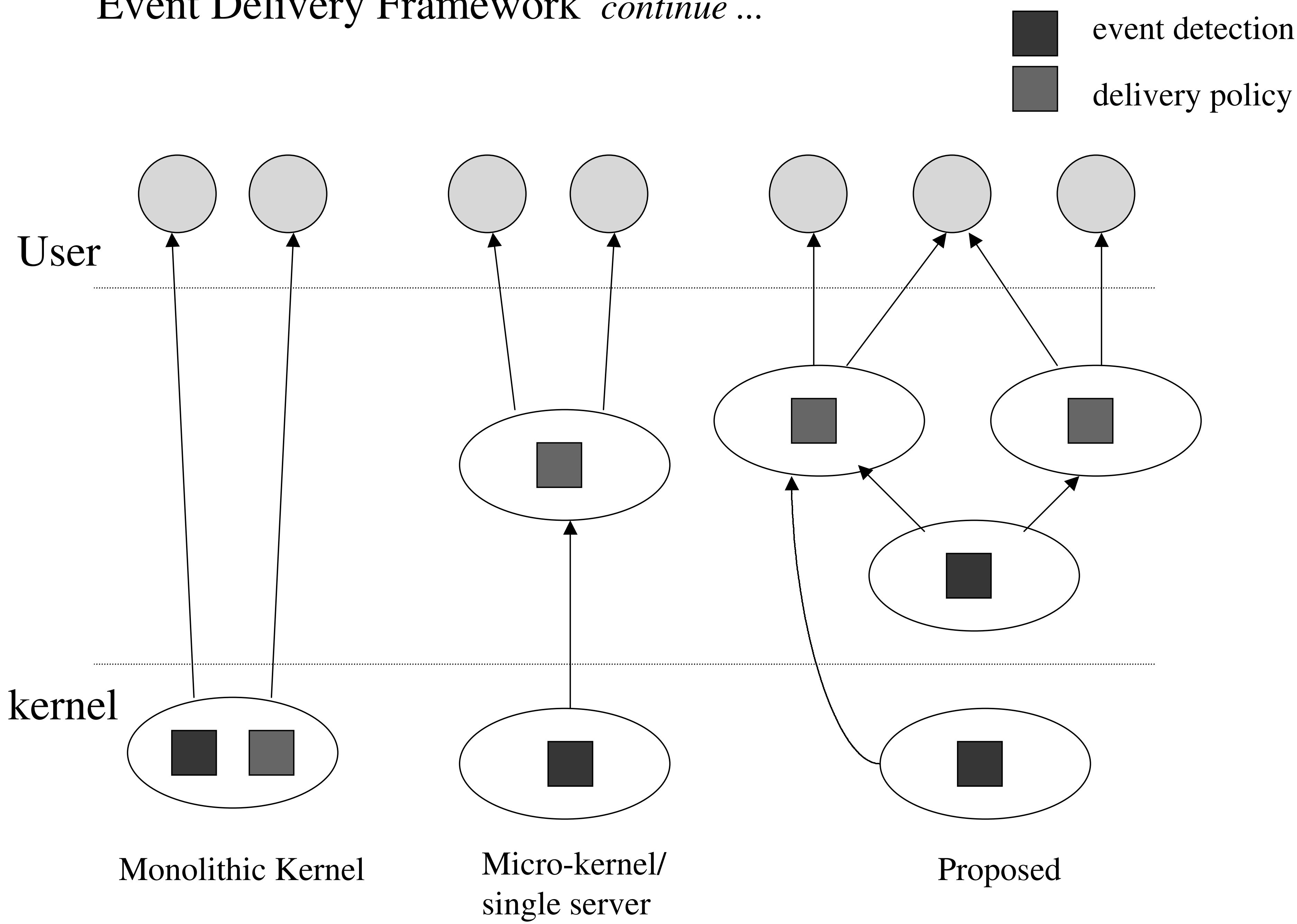
❧ Impact of the architecture

- Allows arbitrary types of events
  - such as processes communication event, monitored event, and remote event.
- Flexible event delivery policy
- Event log
- Reduce delivery latency by giving hints to task scheduler combine with thread-based environment

Event Delivery Framework

- Consider the trend in operating systems
  - monolithic kernel
  - micro-kernel/single server
- Proposed: extension of micro-kernel

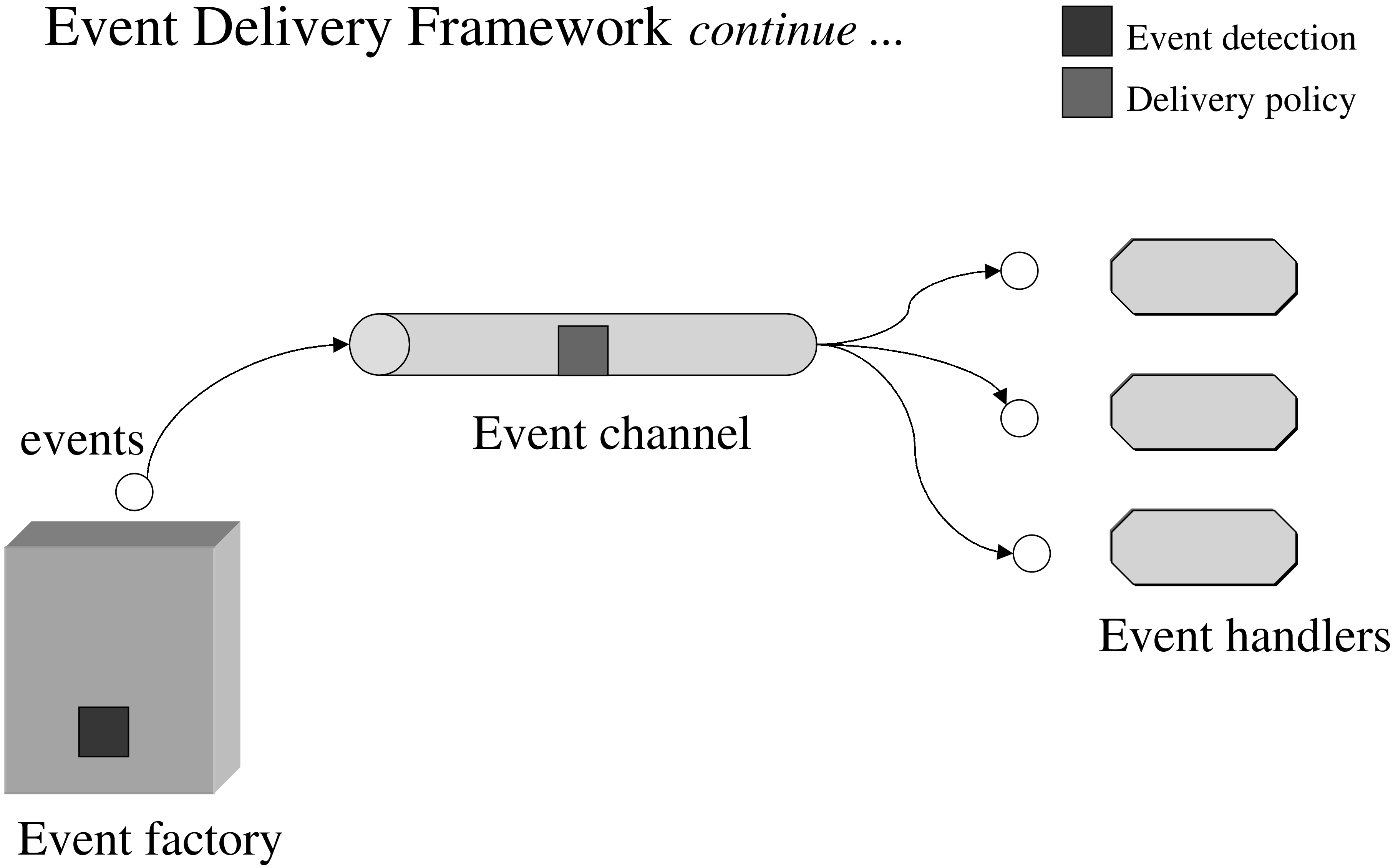
Event Delivery Framework *continue ...*



Event Delivery Framework continue ...

- Four classes of entities that constitute the architecture
  - Channels
  - Factories
  - Handlers
  - Event Object
- In addition, an independent Registrar provides a name-service for the Channel name space

Event Delivery Framework *continue ...*



Event Channel

- Binding event producers(*Factories*) and consumers(*Handlers*)
  - *Handlers* need to be registered
  - network channel, hardware channel, power channel
- Implements delivery policy and preserves environment state
  - determines the delivery order of *Event Objects* to registered *Handlers* and specifies termination condition
- Delivers *Event Objects* base on delivery policy
- Hard to implement delivery policy with large number of *Event Object* types delivered on the same *Channel*
- But large number of *Channels* could increase task overhead on the system
- Preserves current state of mobile environment -- event log
  - when a handler first registers with channel, *Event Objects* maintained in the log are delivered.
- Flexible

```
class Channel {
  Policy* policy;
  Event* State[MAX];
public:
  Channel(char* name);
  Channel(char* name, Policy* p);
  void Body();  //???
  . . .
};

class Policy {
  Handler* List;
  . . .
public:
  void Deliver (Event*);
  virtual void reset();
  virtual Handler* NextHandler();
  virtual void AddHandler(Handler*);
  . . .
};
```

Event Factory

- Detect Events
- Produce *Event Objects*
  - packages information associated with the event
- Post *Event Objects* through *Channel*
- Reference of the *Channel* is obtained by the *Registrar*

```
Class Factory {
  Channel* chan;
  . . .
public:
  Factory(char* channelName);
  void PostEvent(Event*)//call Channel::Delivery(Event*)
  . . .
};
```

Event Handler

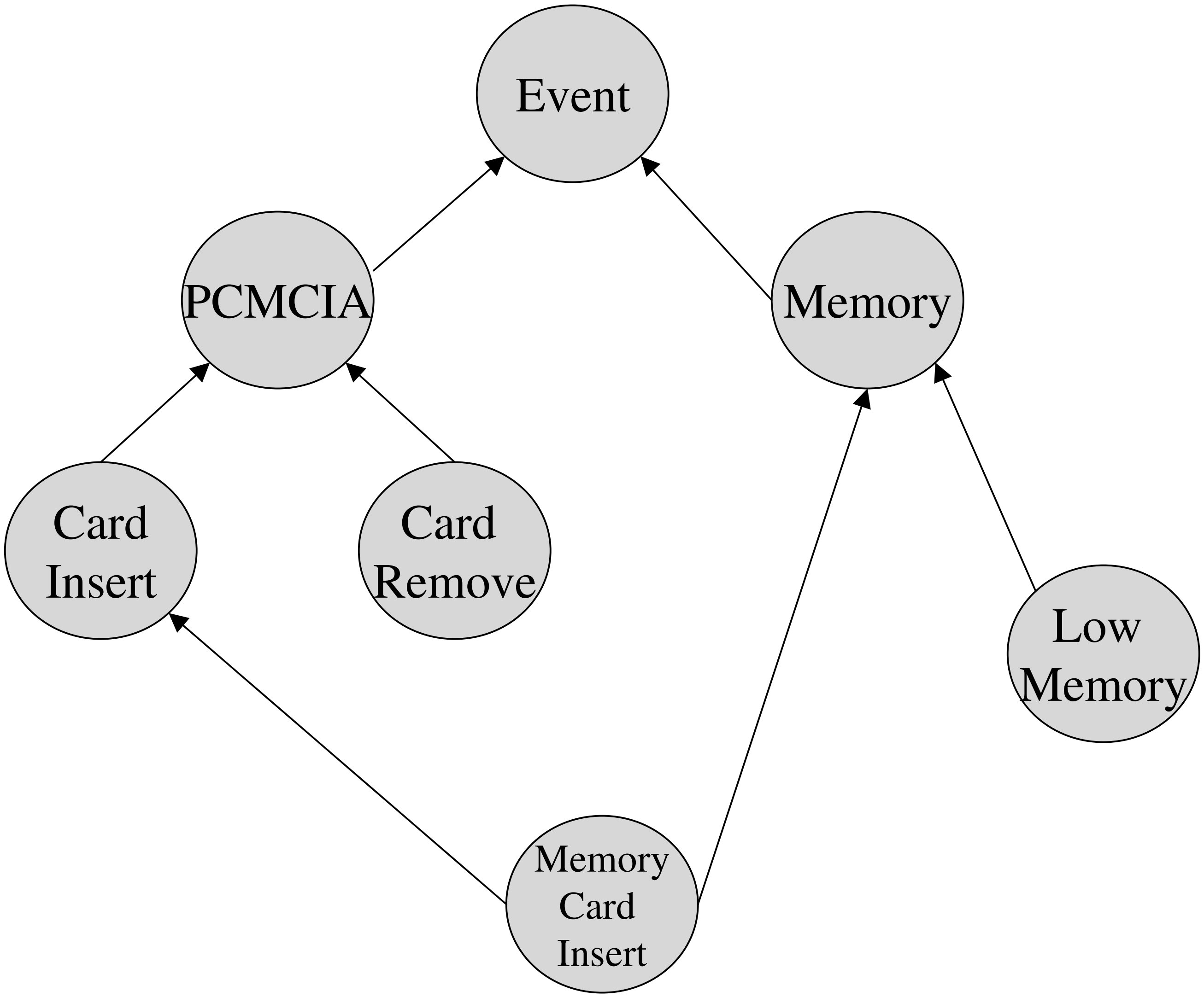
- An environment aware application creates a *Handler* bound to appropriate *Channel*
- Reference of the *Channel* is obtained by the *Registrar*
- *Event Objects* posted onto the *Channel* are delivered to the *Handler*
- Execution of the Handler can be handled using separate thread

```
Class Handler {
  Channel* channel;
  . . .
Public:
  Handler (char* channelName);
  void Body();
  . . .
};
```

Event Object Types

- *Event Object* types are organized as an extensible type hierarchy
- Every *Event Object* type is a subtype of a *root* event
- Each type is associated with a response specified in `<Event_Type>::Handle()` method
- By default, chains the response to the immediate super-type

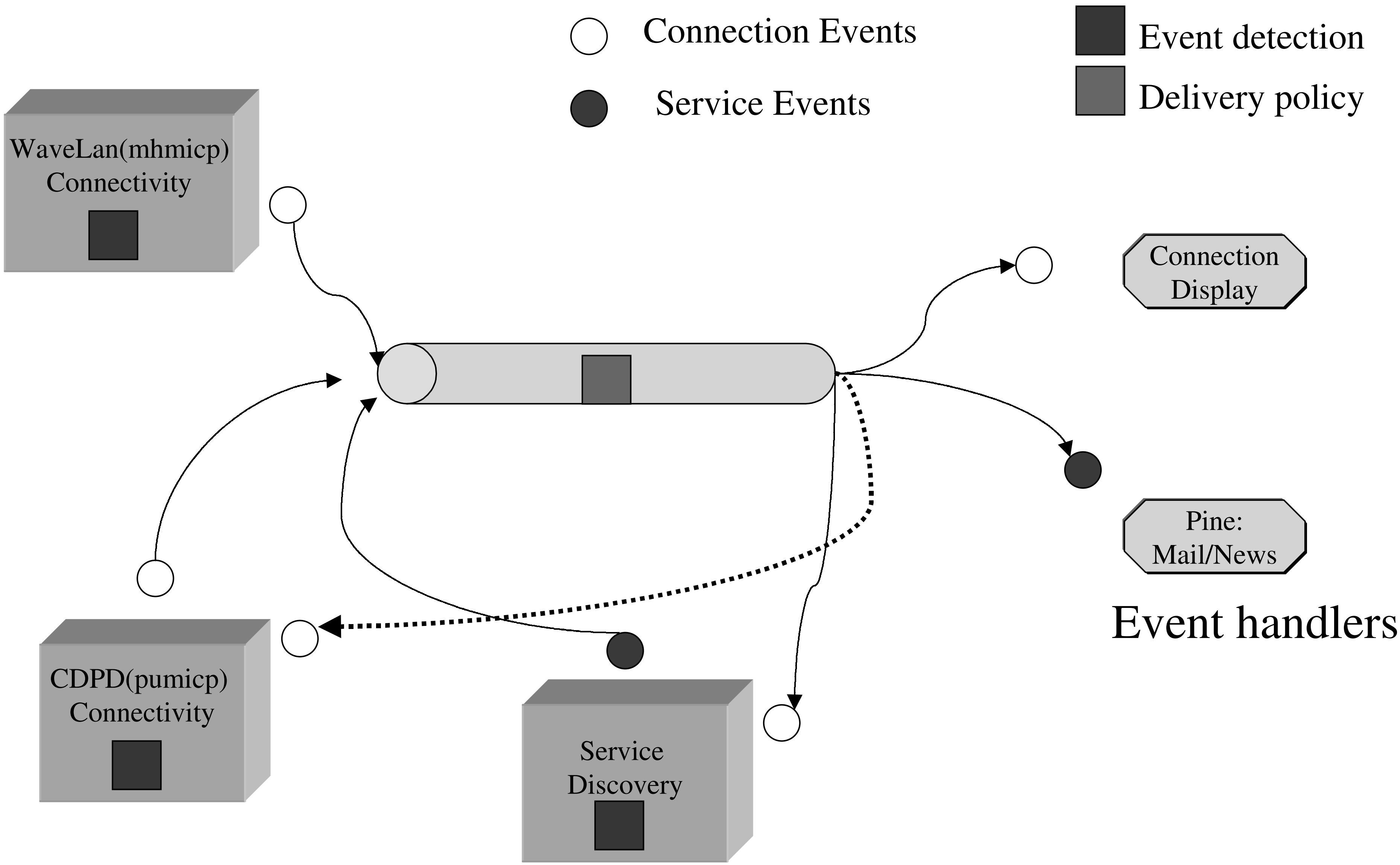
Event Object Types continue ...



```
Class Event {
  . . .
Public:
  virtual size_t Marshal(void*, size_t);
  virtual Event* UnMarshal(void*, size_t);
    . . .
  Virtual void Handle();
};
```

Network Aware Subsystem

- Build a prototype implementation on i486 based laptops running MACH3.0 microkernel
- C++


Connection Events
Service Events
Event detection
Delivery policy
WaveLan(mhmicp) Connectivity
Connection Display
Pine: Mail/News
Event handlers
CDPD(pumicp) Connectivity
Service Discovery

Definition of Events

```
Typedef enum{up, down} LinkState;

class connection: public Event {
  Address LocalIP;
  Address Gateway;
  LinkState State;
public:
  Connection(Address local, Address gateway, LinkState s);
};

class Services: public Event {
  Address Printer;
  Address SMTP;
  Address NNTP;
public:
  Services(Address printer, Address smtp, address nntp);
};
```

Network Channel

```
Channel* c;
main() {
  c = new Channel(“network”);
  c->Body();
}
```

Connection Event Generation (mhmicp)

```
Address localAddr;
Factory* fact;

void initialize() {
  /* create “network” factory */
  fact = new Factory(“network”);
  /* initialize local data */
  LocalAddr = GetMyIpAddress();
}

void inCell(Address remote) {
  Event* e = new Connection(LocalAddr, remote, up);
  fact->PostEvent(e);
  delete e;
}

void lost() {
  Event* e = new Connection(LocalAddr, NULL, down);
  fact->PostEvent(e);
  delete e;
}
```

Service Discovery

```
Handler* eh;
Factory* sf;
Services* NoService;
Services* getServices(Address);

void Connection::Handle() { //the connection event response
  if (State == up)
    sf->PostEvent(getServices(Gateway));
  else if (State == down)
    sf->PostEvent(NoService);
}

main() {
  NoService = new Services(NULL, NULL, NULL);
  eh = new Handler(“network”);
  sf = new Factory(“network”);
  eh->Body();
}
```

Handler Code in Pine

```
Handler* eh;
Thread* et;

Connection::Handle() { //connection event response
  //if CDPD link
  //  Disable full remote mailbox access, buffer outgoing mail
  //if WaveLan link
  //  Enable full remote mailbox access, flush outgoing mail buffer
}

Services::Handle() { //service event response
  // change servers to those indicated in this event;
}

void initialize() {
  et = new Thread(eh = new Handler("network");
}
```

Performance

CPU	Operation system	Scheme	Latency
I486	Mach 3.0	Proposed	1.118 ms
I486	Mach 3.0/ UX42	Signal	0.873 ms
I486	Linux 1.3	Signal	0.140 ms
SPARC	Solaris SunOS 5.4	Signal	0.174 ms

Performance continue ...

- I486 based PC running at 33 MHz vs. 110Mhz SPARC5
- SPARC5 < Linux
  - large context switching time on RISC processors
- UX42 < Linux
  - overhead of user-level UX server
- Ours < Mach/UX
  - extra context switch

Performance *continue* ...

- Delivery latency is not significantly worse
- Feasible, if enhanced
  - reduce context switch overhead by provide channel abstraction as kernel primitive
- Claims that will perform better than signaling scheme when number of processors increases
- Event is handled as soon as it is observed by handler thread

Related Work

- OS provide signal mechanism
  - limited by no information related to the event can be easily delivered
  - delivery policy is fixed and restrictive
- OO PL
  - delivers synchronous events
  - require knowledge about where exceptions occur
- Upcalls (Swift)
  - upward flow by procedure calls or asynchronous signals

Related Work *continue* ...

❧ Reactor (ACE)

- decouple event handler from event demultiplexing vs.. event consumer from event producer

❧ CORBA Event Services

- push and pull vs.. pull driven by factory

Future Work

- To consolidate current implementation
- to develop tools to aid in the primitive utilization
- to investigate how a mobile application should be structured