Aspect Oriented Development Of P2P File Sharing Application

Eren Algan
Doğan Kaya Berktaş
Rıdvan Tekdoğan

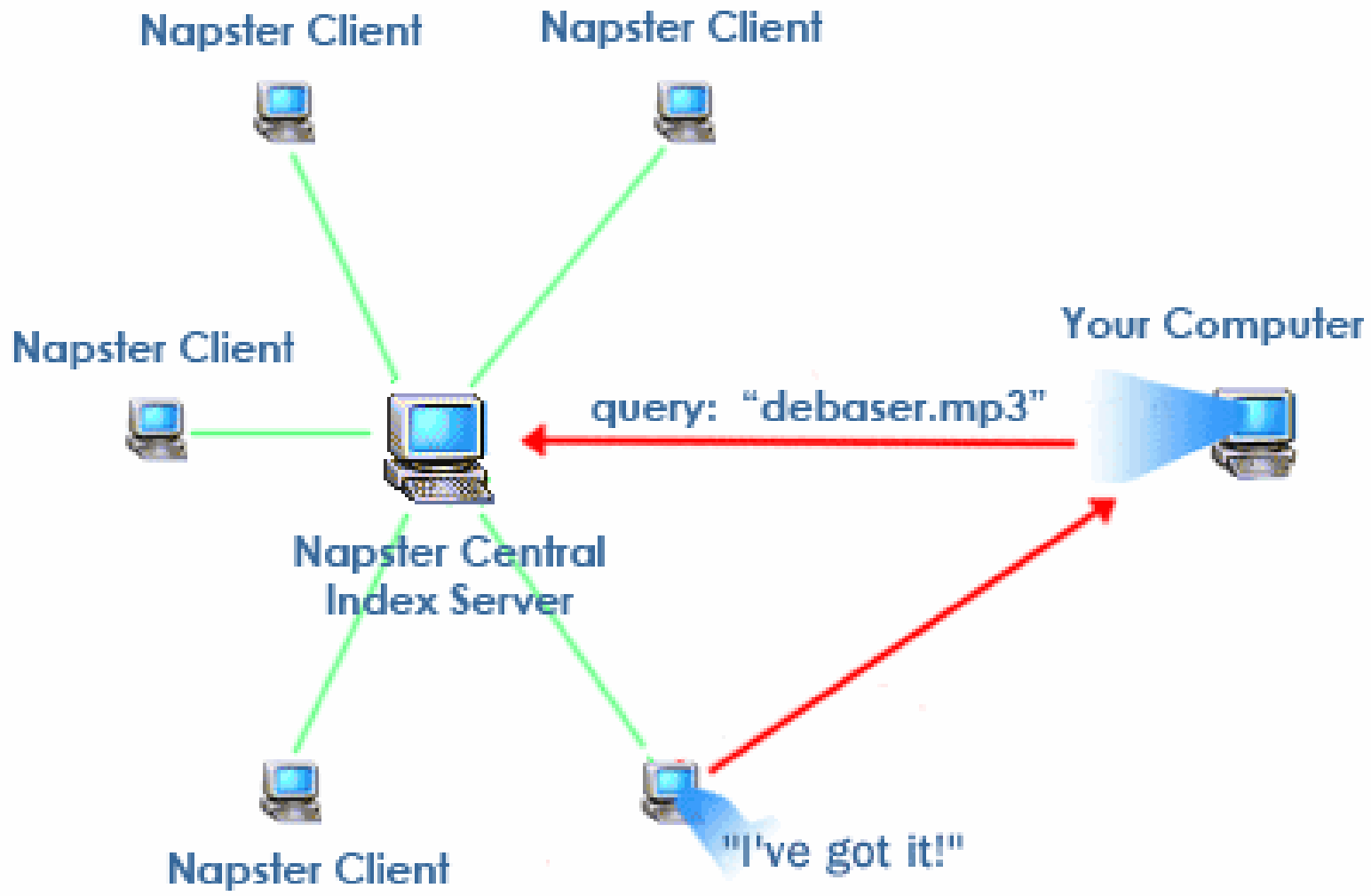
Outline

- Introduction
- Case Study
  - Example P2P Application
  - Applying Design Patterns
- Identifying Crosscutting Concerns
- Aspect Oriented Implementation
- Conclusion
- DEMO

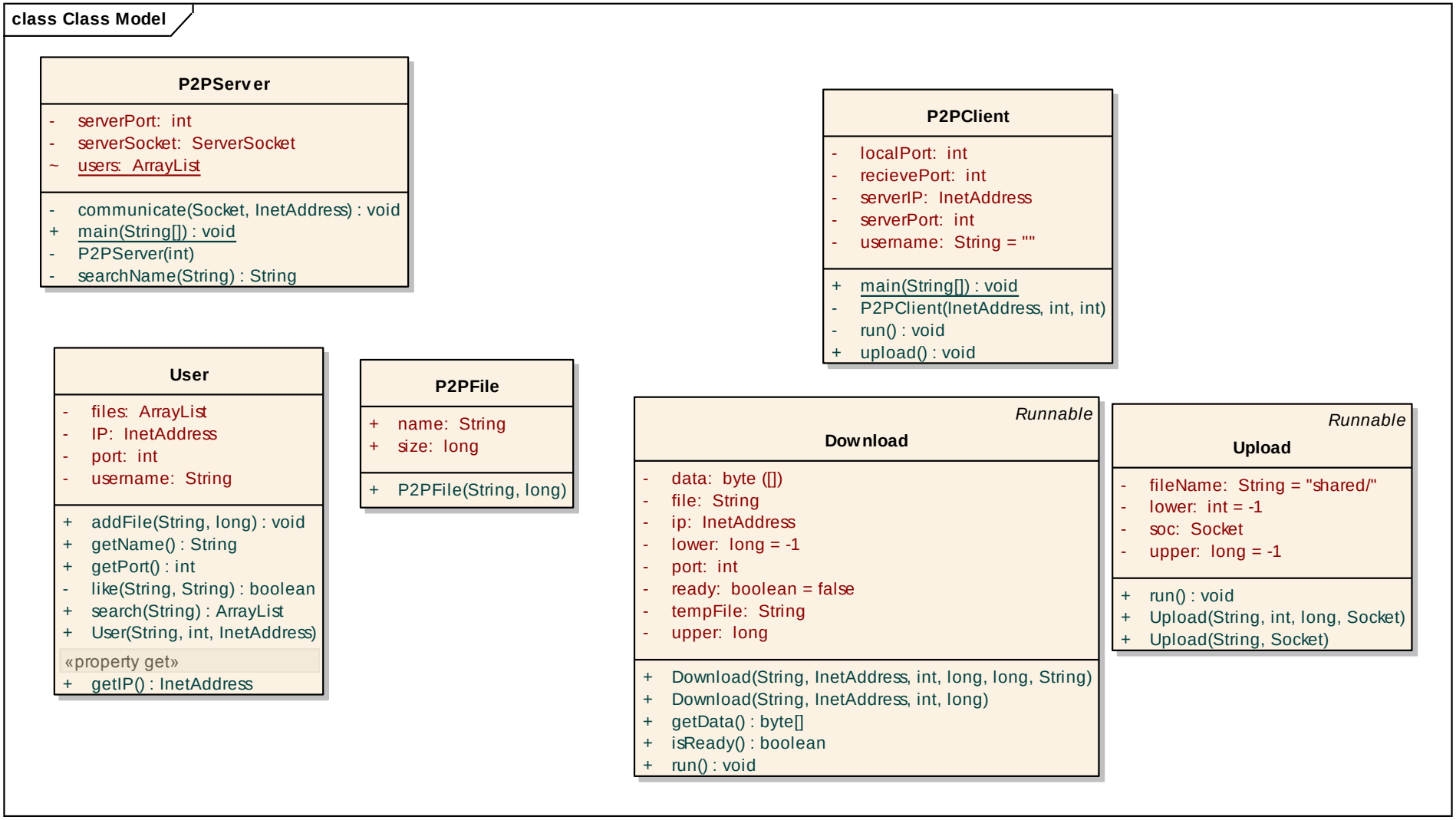
Introduction

- Increase of Internet usage makes P2P file sharing applications populer
- Many aspects in P2P File Sharing Applications
  - Concurreny, fault tolerance, quality of service and adaptibility….
- Separation of Concerns with
  - OOD
  - AOD

Case Study: P2P Application



Object Oriented Design



Applying Patterns

- Observer Pattern
- Active Object Pattern

What is Observer Pattern

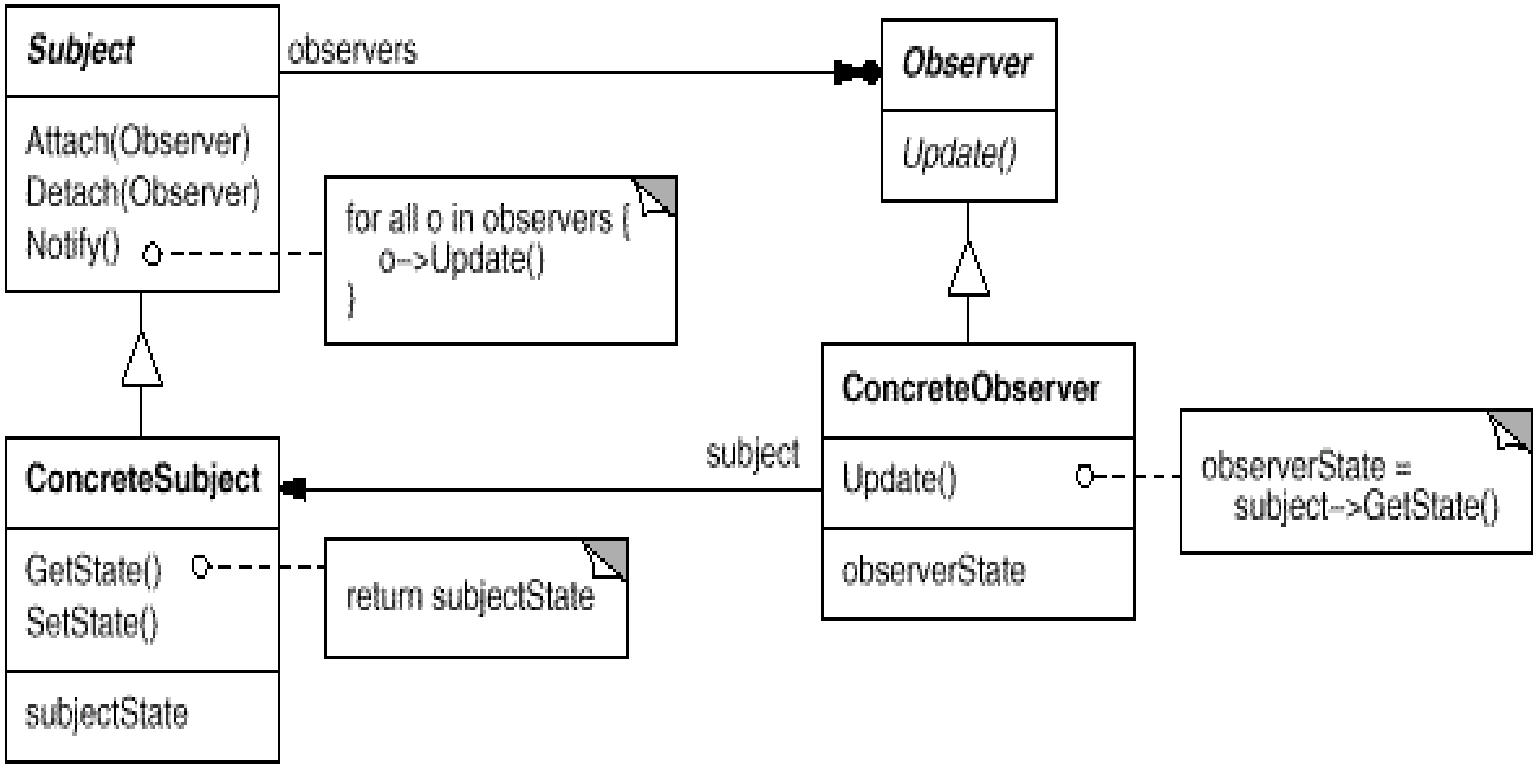
Don’t Call Us, We’ll Call You!

Subject

- This is basically an interface that enables observers to attach and detach themselves
- Holds a list of observers that are subscribed
  - Attach - Adds a new observer
  - Detach - Removes an existing observer
  - Notify - Notifies each

Observer

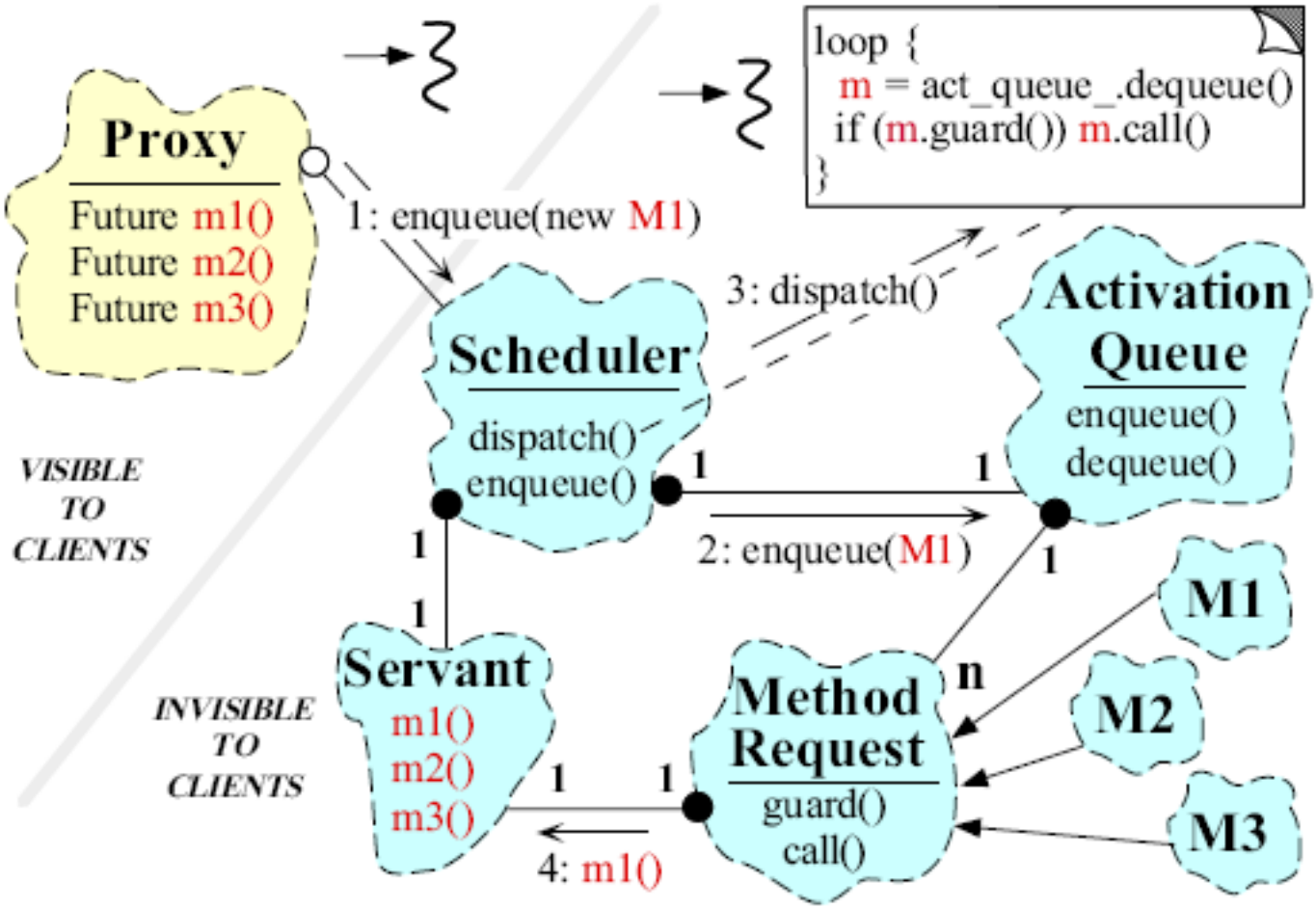
- Subscribed to Subject

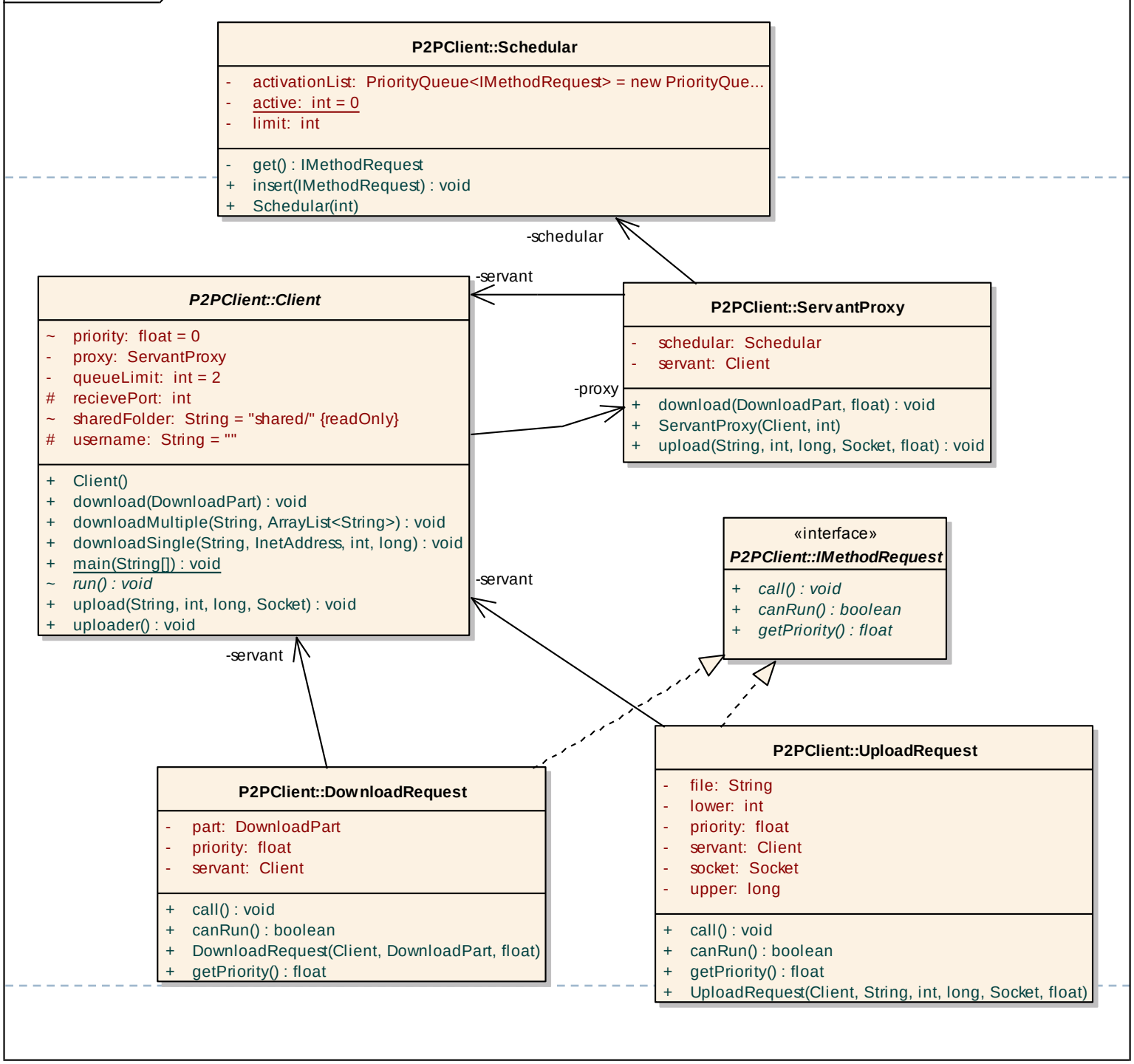


Active Object Pattern

- Object behavioral pattern
- decouples method execution from method invocation in order to simplify synchronized access to and object that resides in its own thread of control.
- The Active Object pattern allows one or more independent threads of execution to interleave their access to data modeled as a single object.
- Commonly used in distributed systems requiring multi-threaded servers.

Active Object Pattern



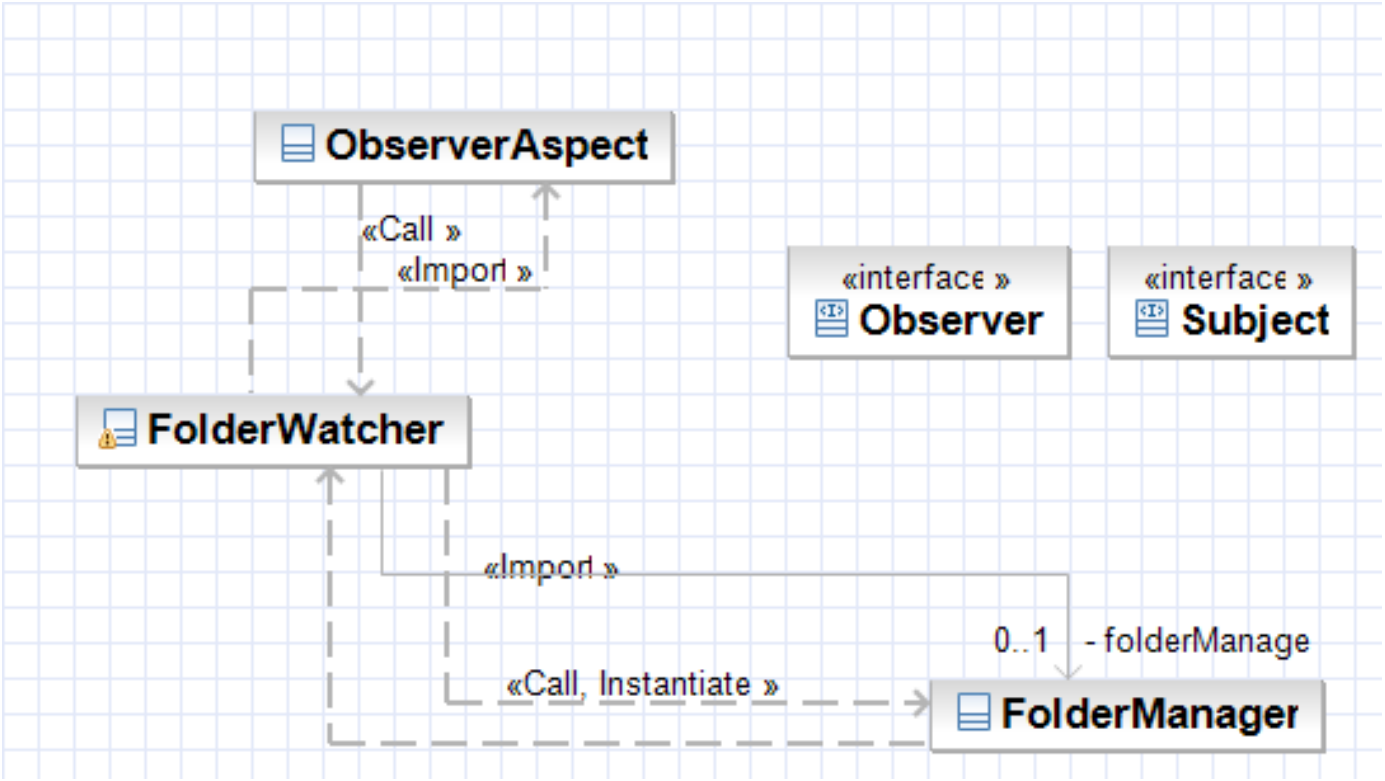

P2PClient::Schedular
- activationList: PriorityQueue<IMethodRequest> = new PriorityQue...
- active: int = 0
- limit: int
- get() : IMethodRequest
+ insert(IMethodRequest) : void
+ Schedular(int)
-schedular
-servant
P2PClient::Client
~ priority: float = 0
- proxy: ServantProxy
- queueLimit: int = 2
recievePort: int
~ sharedFolder: String = "shared/" {readOnly}
# username: String = ""
+ Client()
+ download(DownloadPart) : void
+ downloadMultiple(String, ArrayList<String>) : void
+ downloadSingle(String, InetAddress, int, long) : void
+ main(String[]) : void
~ run() : void
+ upload(String, int, long, Socket) : void
+ uploader() : void
P2PClient::ServantProxy
- schedular: Schedular
- servant: Client
-proxy
+ download(DownloadPart, float) : void
+ ServantProxy(Client, int)
+ upload(String, int, long, Socket, float) : void
«interface»
P2PClient::IMethodRequest
+ call() : void
+ canRun() : boolean
+ getPriority() : float
-servant
-servant
P2PClient::DownloadRequest
- part: DownloadPart
- priority: float
- servant: Client
+ call() : void
+ canRun() : boolean
+ DownloadRequest(Client, DownloadPart, float)
+ getPriority() : float
P2PClient::UploadRequest
- file: String
- lower: int
- priority: float
- servant: Client
- socket: Socket
- upper: long
+ call() : void
+ canRun() : boolean
+ getPriority() : float
+ UploadRequest(Client, String, int, long, Socket, float)

Identfying Crosscutting Concerns

- Observing Object State Changes
- Creation of Worker Objects
- Monitoring

Aspect Oriented Version

- Empty Subject & Observer interfaces
- Assigned according to their roles



Tracking of Observers

- Normally, the participants keeps track of the observers that listening to a particular subject.
- Instead centralized via ObserverProtocol aspect

```
protected List getObservers(Subject subject)
{
    if (perSubjectObservers == null)
   {
        perSubjectObservers = new WeakHashMap();
    }
    List observers = (List)perSubjectObservers.get(subject);
    if ( observers == null )
   {
       observers = new LinkedList();
       perSubjectObservers.put(subject, observers);
    }
     return observers;
  }
```

```
void addObserver(Subject subject, Observer observer)
{
    getObservers(subject).add(observer);
}

void removeObserver(Subject subject, Observer observer)
{
    getObservers(subject).remove(observer);
}
```

Notifying the Observers

- The OO way of doing is via subject.
- Subject calls every observers update method.
- Aspect oriented version is also using a loop to call observers update methods, but this time from the aspect.

```
protected abstract pointcut subjectChange(Subject s);

after(Subject subject) returning : subjectChange(subject)
{
  for (Observer observer : getObservers(subject))
  {
        updateObserver(subject, observer);
  }
}
```

Role Assignments

- Handled via aspect
- Without any interference with modifying classes directly.

```
declare parents : FolderWatcher extends Subject;
declare parents : DownloadManager implements
   Observer;
```

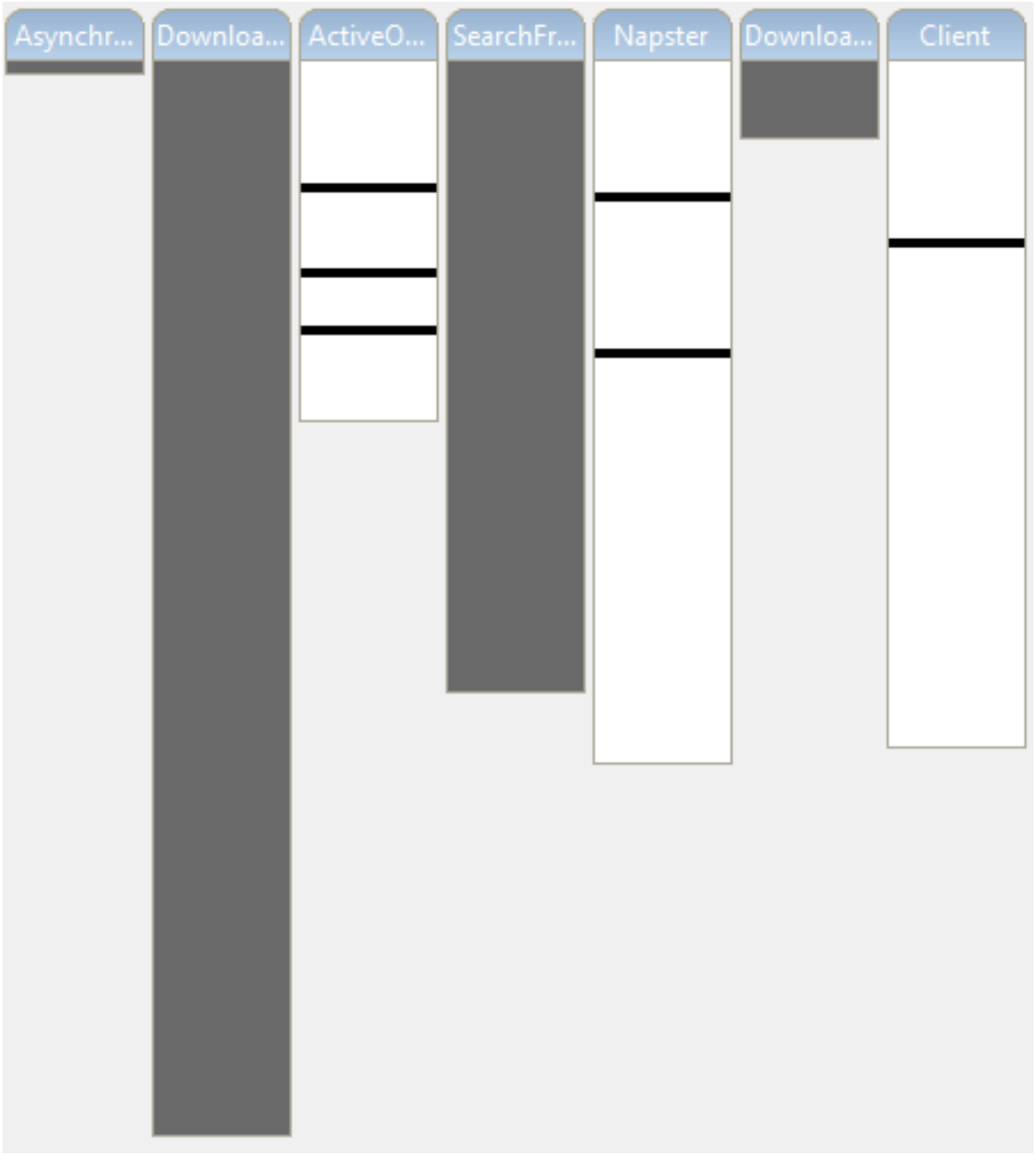
Starting the Observation Relationship

```
// used for injection
private MainFrame defaultObserver =
   MainFrame.newContentPane;

pointcut folderWatcherCreation(Subject s) :
   execution(public FolderWatcher+.new(..))
   && this(s);

after(Subject s) returning : folderWatcherCreation(s)
{
   addObserver(s, defaultObserver);
}
```

Creation of Worker Objects

```
public class Download
   extends Runnable
{
   public Download(..)
   {
         ..
   }
   public void run()
   {
         // Core aspect
   }
}
```

```
public void download(..)
{
    Thread t = new Thread(){
        public void run(){
            // Core aspect
        }
    };
    t.start();
}
```

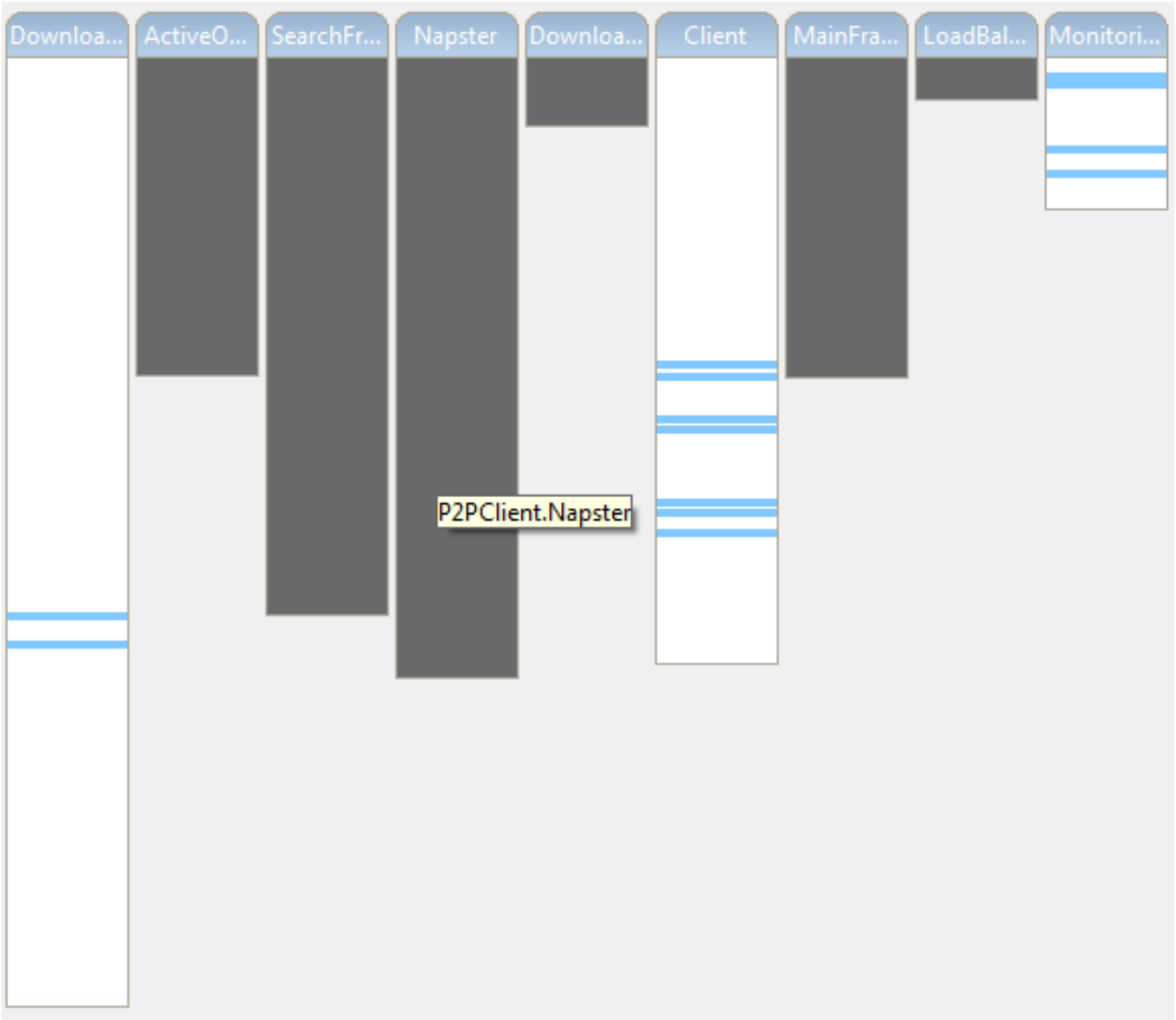

Asynchr...
Downloa...
ActiveO...
SearchFr...
Napster
Downloa...
Client

AO Solution: Creation of Worker Object

```
public abstract aspect
    AsynchronousExecutionAspect {
    abstract pointcut asyncOperations();

    Object around():asyncOperations(){
            Runnable worker = new Runnable() {
                    public void run() {
                            proceed();
                    }
            };

            new Thread(worker).start();
            return null;
    }
}
public aspect Concurrency extends
AsynchronousExecutionAspect{
        pointcut asyncOperations(): call(@Asynchronous * *(..));
}
```

Monitoring

- Some actions trigger state changes in objects
- When a file download completed, GUI must change status of corresponding Download object to completed.
- Total data uploaded and downloaded used for determining priority


Downloa...
ActiveO...
SearchFr...
Napster
Downloa...
Client
MainFra...
LoadBal...
Monitori...
P2PClient.Napster

AO Solution: Monitoring

```
aspect Monitoring {
    long Client.totalUpload = -1;
    long Client.totalDownload = 1;
    after(Client c,byte[] data, int off, int len): this(c)&&call(void
    java.io.DataOutputStream.write(byte[],int,int)) && args(data,off,len){
            c.totalUpload += len;
    }
    after(Client p) returning(int retval) : this(p) && call(int java.io.DataInputStream.read(..)){
            p.totalDownload += retval;
    }
    after(Client p):target(p) &&(set(long Client.totalUpload) || set(long Client.totalDownload)){
            if (p.totalDownload> 0) {
                    p.priority= p.totalUpload/ p.totalDownload;
            }
    }
    after(Download d) : target(d) && set(long Download.downloaded){
            d.downloadFlagChanged = true;
    }
}
```

Conclusion

- Enhancing existing application by applying Design Patterns
- Crosscutting Concerns
  - Observing Object State Changes
  - Creation Object Worker Objects
  - Monitoring
- OOD not suitable for crosscutting concerns
- AO solutions for crosscutting concerns

DEMO