1 Acceptor

Decouples accepting role from the service initialization once a connection is established. The pattern consists of *reactor*, *service handler* and *acceptor*. Reactor handles connections as described previously. Service handler encapsulates service logic and communicates with a peer through the event handle. Acceptor handles accept events from the reactor.

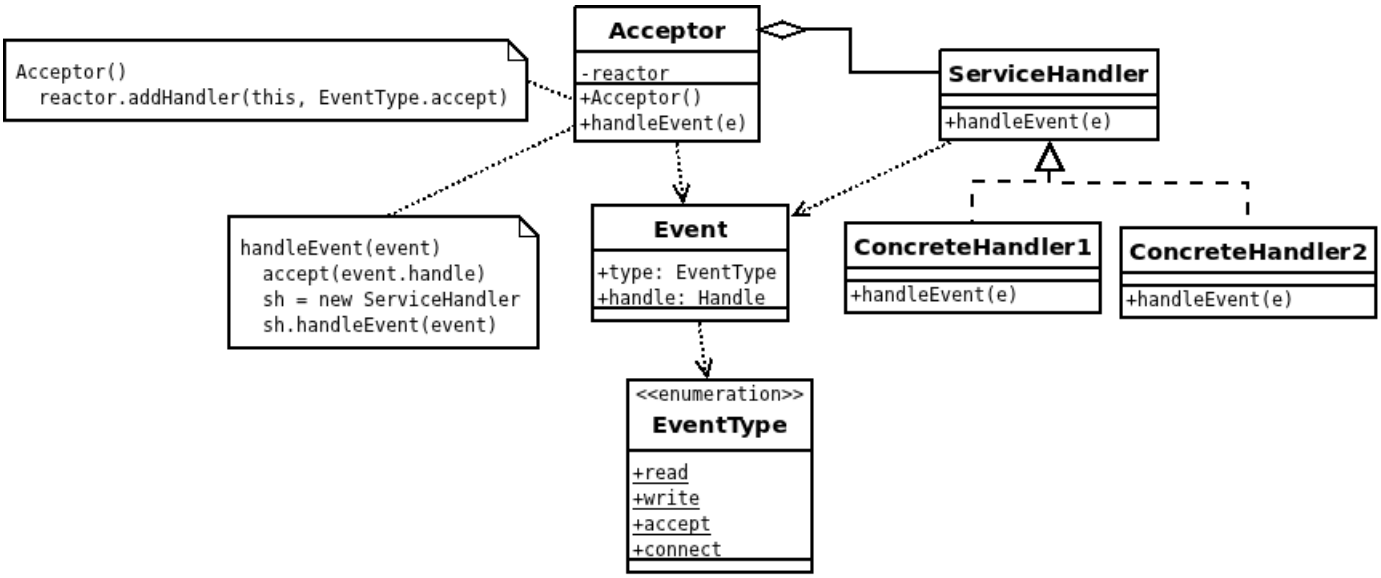

Figure 1: Acceptor structure

Application creates the reactor and registers acceptor at the reactor for accepting events. When the reactor notifies the acceptor, a handle can be accepted, the service is initialized and it starts processing that event/handle.

1.1 C++ Poco

SocketAcceptor is the acceptor. It requires service initializer and handler. When the acceptor gets a connection, it uses the initializer to create service handler.

Service handler deals with read/write operations with the client connection. Once the connection is closed, it destroys event handlers, so the reactor will not use them anymore.

Listing 1: Service handler

```
class ServiceHandler
{
public:

    void onReadable(const AutoPtr<ReadableNotification>& notification)
    {
        char buf[100];
        int no = _socket.receiveBytes(buf, 99);
        if (no > 0)
        {
            buf[no] = '\0';
            cout << "ServiceHandler::onReadable(): buf=" << buf << endl;
            _request = buf;
        }
```

```
        else
        {
            // destroy handlers
            cout << "ServiceHandler::onReadable(): destroying handlers" << endl;
            _reactor.removeEventHandler(_socket, NObserver<ServiceHandler,
               ReadableNotification>(*this, &ServiceHandler::onReadable));
            _reactor.removeEventHandler(_socket, NObserver<ServiceHandler,
               WritableNotification>(*this, &ServiceHandler::onWritable));
            _socket.close();
            delete this;
        }
    }

    void onWritable(const AutoPtr<WritableNotification>& notification)
    {
        if (!_request.empty())
        {
            cout << "ServiceHandler::onWritable(): _request=" << _request << endl
               ;
            _socket.sendBytes(_request.c_str(), _request.length());
            _request.clear();
        }
    }
};
```

Service initializer is notified on accepting events. When new connection is available, the service will be created.

Listing 2: Service initializer

```
class ServiceInitializer
{
public:

    ServiceInitializer(StreamSocket& socket, SocketReactor& reactor) : _socket(
       socket), _reactor(reactor)
    {
        cout << "ServiceInitializer::ServiceInitializer(): client accepted" <<
           endl;
        ServiceHandler* handler = new ServiceHandler(_socket, _reactor);
        _reactor.addEventHandler(_socket, NObserver<ServiceHandler,
           ReadableNotification>(*handler, &ServiceHandler::onReadable));
        _reactor.addEventHandler(_socket, NObserver<ServiceHandler,
           WritableNotification>(*handler, &ServiceHandler::onWritable));
    }
};
```

Reactor and acceptor are created, reactor is passed around to add/remove event handlers.

Listing 3: Create acceptor

```
int main()
{
    ServerSocket socket(9000);
    SocketReactor reactor(Timespan(0, 0, 0, 1, 0));
    SocketAcceptor<ServiceInitializer> acceptor(socket, reactor);

    Thread thread;
    thread.start(reactor);
    thread.join();
```

```
    return EXIT_SUCCESS;
}
```

The source code is within `Acceptor.cpp` file.

2 Connector

Decouples connection role from the service initialization once a connection is established. The pattern consists of *reactor*, *service handler* and *connector*. Reactor handles connections as described previously. Service handler encapsulates service logic and communicates with a peer through the event handle. Connector handles connect events from the reactor.

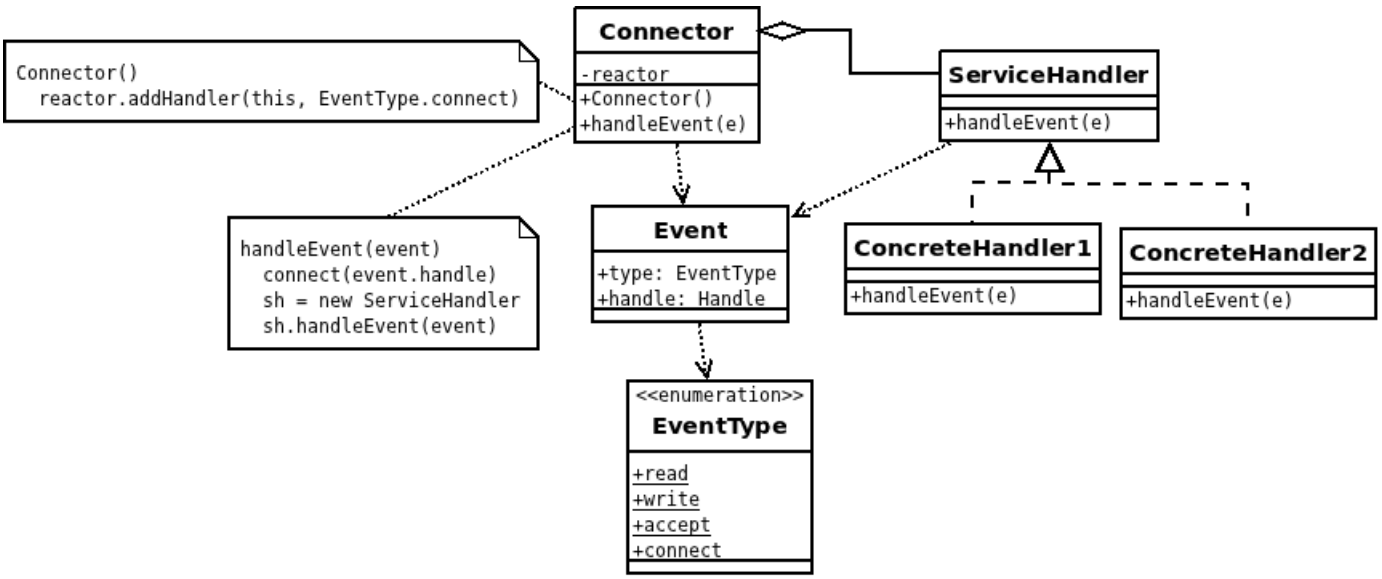

Figure 2: Connector structure

Application creates the reactor and registers connector at the reactor for connecting events. When reactor notifies connector, the handle can connect, service is initialized and it starts processing that event/handle.

2.1 C++ Poco

SocketConnector is the connector. It requires service initializer and handler. When the connector connects to a server, it used the initializer to create service handler.

Service handler deals with read/write operations with the client connection. Once the connection is closed, it destroys event handlers, so the reactor will not use them anymore.

Listing 4: Service handler

```
class ServiceHandler
{
public:

    void onWritable(const AutoPtr<WritableNotification>& notification)
    {
        string request = "Hello, World!";
        _socket.sendBytes(request.c_str(), request.length());
    }

    void onReadable(const AutoPtr<ReadableNotification>& notification)
    {
        char buf[100];
        int no = _socket.receiveBytes(buf, 99);
```

```
        if (no > 0)
        {
            buf[no] = '\0';
            cout << "ServiceHandler::onReadable(): buf=" << buf << endl;
        }
        else
        {
            // destroy handlers
            cout << "ServiceHandler::onReadable(): destroying handlers" << endl;
            _reactor.removeEventHandler(_socket, NObserver<ServiceHandler,
               ReadableNotification>(*this, &ServiceHandler::onReadable));
            _reactor.removeEventHandler(_socket, NObserver<ServiceHandler,
               WritableNotification>(*this, &ServiceHandler::onWritable));
            _socket.close();
            delete this;
        }
    }
};
```

Service initializer is notified on connecting events. When connected, the service will be created.

Listing 5: Service initializer

```
class ServiceInitializer
{
public:

    ServiceInitializer(StreamSocket socket, SocketReactor& reactor) : _socket(
       socket), _reactor(reactor)
    {
        cout << "ServiceInitializer::ServiceInitializer():" << endl;
        ServiceHandler* handler = new ServiceHandler(_socket, _reactor);
        _reactor.addEventHandler(_socket, NObserver<ServiceHandler,
           WritableNotification>(*handler, &ServiceHandler::onWritable));
        _reactor.addEventHandler(_socket, NObserver<ServiceHandler,
           ReadableNotification>(*handler, &ServiceHandler::onReadable));
    }
};
```

Reactor and connector are created, reactor is passed around to add/remove event handlers.

Listing 6: Create connector

```
int main()
{
    SocketAddress address("localhost:9000");
    SocketReactor reactor(Timespan(0, 0, 0, 1, 0));
    SocketConnector<ServiceInitializer> connector(address, reactor);

    Thread thread;
    thread.start(reactor);
    thread.join();

    return EXIT_SUCCESS;
}
```

The source code is within `Connector.cpp` file.