OVOPS++

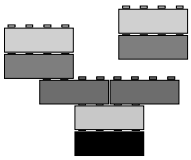
An Object-Oriented Virtual Operating System ++
based on OVOPS by Lappeenranta University of Technology
and Conduits+ by University of Illinois

Vesa-Matti Puro

Open Environment Software Inc.
Object-Oriented Network Protocol Software

P.O. Box 6, 53851 Lappeenranta, Finland
Tel. +358 208 300 200
Fax +358 207 300 200
Email: vmp@oes.fi

Q: How to Build Protocol Software?



"A: Using a black box framework..."

Introduction

- ***"Writing network protocol software is important and difficult ..."***
- ***"... more and more complex ..."***
- ***"... better tools ..."***

Reliable

Flexible

Efficient

These slides show how object-oriented OVOPS++ framework and design patterns can help writing network protocol software.

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Agenda

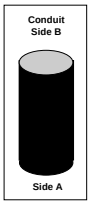
- Protocol Software
- Introduction
- **Agenda**
- OVOPS++ Overview
- Important Words
- Messenger
- Message Transporter
- Visitors
- Conduit
- Multiplexer
- Conduit Factory
- Adapter
- Protocol
- State
- Timer
- Example: ATM Signalling
- Summary
- Where to Get More Info

OVOPS++ Architecture Model

- conduits (adapter, protocol, mux, factory) are parts that process information
- information chunks (transporter, messenger) represent information being processed

Conduit

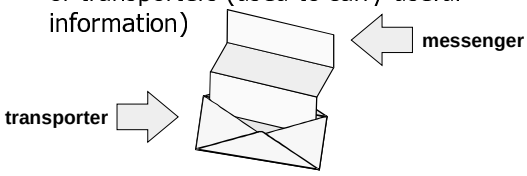
- conduit, pipe, …, something that is used to transfer something suitable from one side to another
- usually has two sides, some conduits can have only one side (lid) or can have multiple sides (junction)
- we use the names side A and side B for the differenct sides conduits can have



Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Information Chunks

- represent the information being processed
- can be messengers (useful information) or transporters (used to carry useful information)



Basic Interfaces of Conduits

- interface for connecting conduits to neighbors
- interface for handling information chunks

Interface for Connecting Conduits

- side A or side B of a conduit can be connected to any other conduit

```
class pfConduit
    {
    public:
    void connectToA (const pfConduit &that_);
*)  void connectToB (const pfConduit &that_);
    };
```

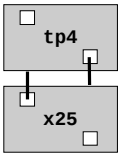
*) An adapter only supports A side and connect operation on that side. A mux can have multiple conduits on B side.

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Example: Connecting Conduits

```
// create tp4 and x25 protocol instances
pfConduit tp4 = … ;
pfConduit x25 = … ;

// connect protocols together
tp4.connectToA (x25);
x25.connectToB (tp4);
```



Interface for Handling Information Chunks

- for sending an information chunk
- is available inside a conduit implementation

```
class pfConduitImplementation
    {
    public:
    void toA (pfMessenger *messenger_);
*)  void toB (pfMessenger *messenger_);
    };
```

*) An adapter only supports A side and send operation on that side. A mux can have multiple conduits on B side.

Example: Sending an Information Chunk

```
// send SETUP messenger in q2931
// protocol implementation
//
void q2931Protocol :: sendSETUP (void)
    {
    SETUP *messenger = new SETUP;
    toA (messenger);
    return;
    }
```

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Interface for Handling Information Chunks

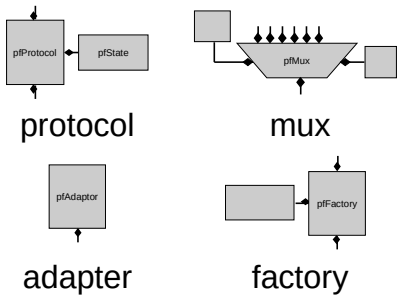
- … and for receiving an information chunk
- is available in a conduit handle

```
class pfConduit
    {
    public:
    void accept (pfMessageTransporter
                 *transporter);
    };
```

Important Words

- Object-Oriented (Analysis, Design, Programming)
- Object-Oriented Design Pattern
- Framework (Black Box/White Box)
- Component (Software)
- Middleware (e.g. OVOPS++, CORBA)

Conduits

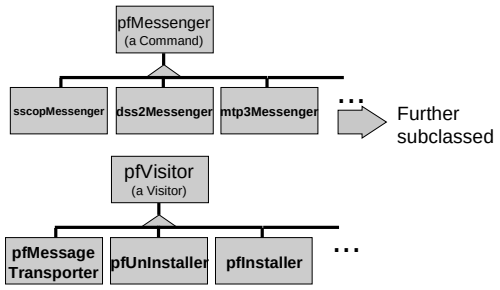


protocol mux

adapter factory

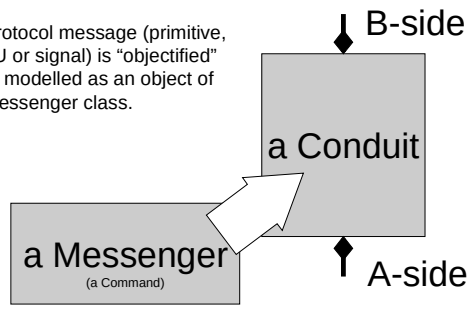
Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

OVOPS++ Overview: Information Chunks



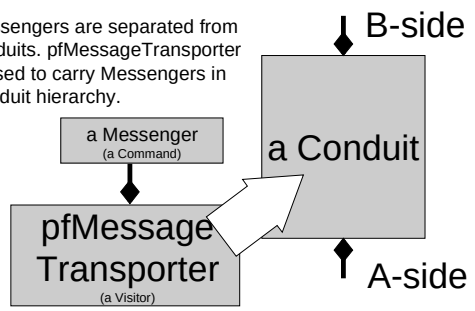
Messenger and Conduit

A protocol message (primitive, PDU or signal) is "objectified" and modelled as an object of pfMessenger class.



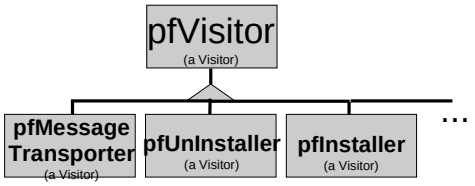
Messenger, Message Transporter and Conduit

Messengers are separated from conduits. pfMessageTransporter is used to carry Messengers in Conduit hierarchy.



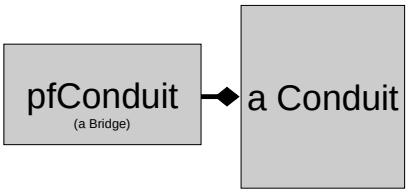
Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Visitors



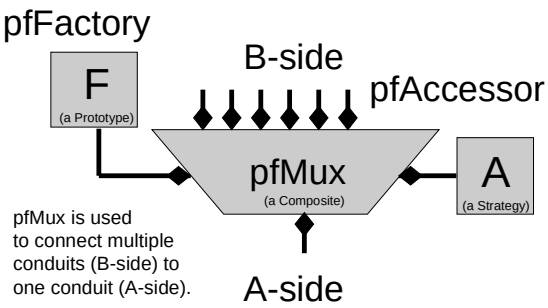
The pfVisitor class is further subclassed to implement different kinds of visitors.

pfConduit and Conduit



Various conduits are accessed through pfConduit proxy. pfConduit (Proxy, Smart Pointer) provides reference counting to avoid problem with dangling pointers.

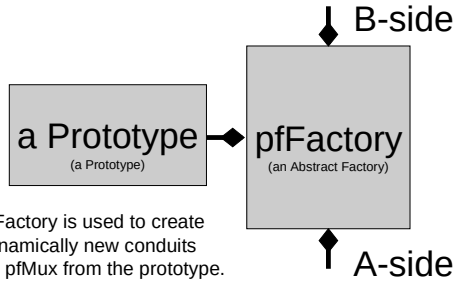
Multiplexer



pfMux is used to connect multiple conduits (B-side) to one conduit (A-side).

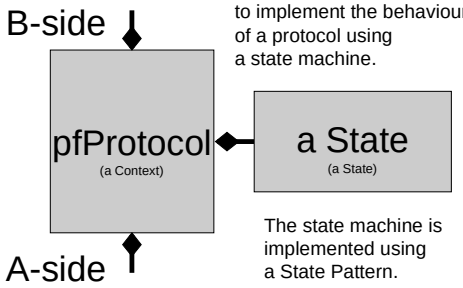
Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Conduit Factory



pfFactory is used to create dynamically new conduits for pfMux from the prototype.

Protocol

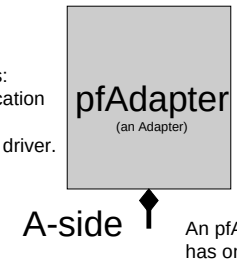


pfProtocol is used to implement the behaviour of a protocol using a state machine.

The state machine is implemented using a State Pattern.

Adapter

A pfAdapter conduit is used for two purposes:
- implementing an application programming interface
- implementing a device driver.

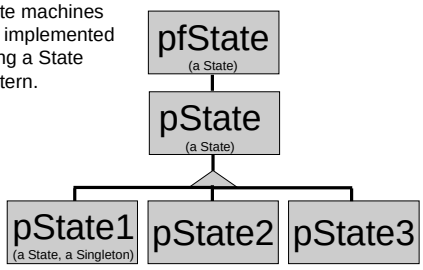


An pfAdapter has one side (A-side).

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

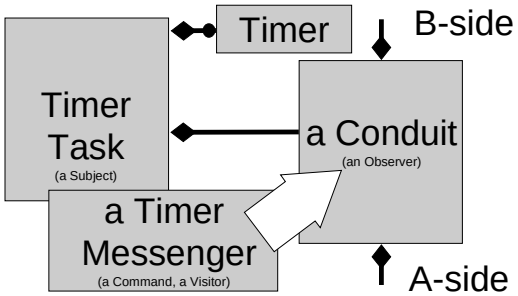
State

State machines are implemented using a State Pattern.



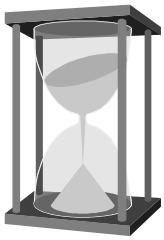
States are made singletons using a Singleton Pattern.

Timer



Timers

- a pfTimer models a timer
- user supplies a timeout messenger for each timer



Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Public pfTImer Interface

- contains a **`setTimeout`** method to set default timeout value
- contains a **`start`** method to start a timer
- contains a **`stop`** method to stop a timer
- contains an **`isActive`** method to check if timer is running or not

Public pfTImer Interface

```
class pfTimer
    {
    public:
    void setTimeout (unsigned long secs_,
                     unsigned long usecs_ = 0);
    void start (void);
    void stop (void);
    pfBoolean isActive (void);
    };
```

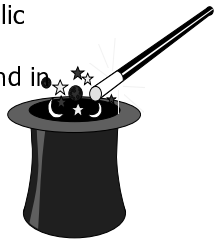
Public pfTImer Interface

```
class pfTimer
    {
    public:
    void setHost (pfConduitImplementation
                  *host_);
    void setMessenger (pfTimerMessenger
                       *messenger_);
    };
```

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

New Additions to Conduits+ model

- Additions to allow symbolic handling of variables and timers in the protocols and in the messengers
  - pfStorage
  - pfTimers
  - use of proxies



pfStorage

- Various data types: boolean, frame, integer, …
- Also known as property

```
class pfStorage
  {
  public:
  void defineInteger(string name_);
  void setInteger(string name, pfUlong value);
  pfUlong getInteger(string name);
  …
  };
```

pfTimers

- Allows symbolic handling of timers

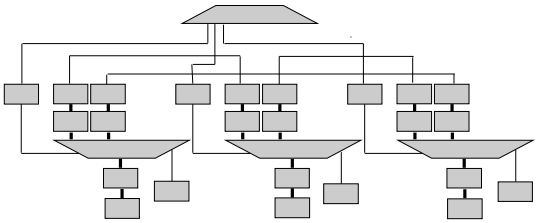
```
class pfTimers
  {
  public:
  void defineTimer(string name_,
                   pfTimerMessenger *messenger_,
                   pfUlong timeout_in_msec);
  void startTimer(string name_);
  void stopTimer(string name_);
  };
```

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Example: createQ2931Protocol Method

```
// create protocol implementation
//
pfConduit q2931Protocol
    :: createQ2931Protocol(void)
    {
    pfConduit result(new q2931Protocol);
    return result;
    }
```

ATM Protocol Architecture Using OVOPS++ Elements



An example of object instances in ATM protocol software (three links, point-to-point connections).

Summary

- Writing network protocol software is a challenging task
- The OVOPS++ framework was presented as a one solution to this challenge
- The use of OVOPS++ can be learned only by using the framework and acquiring O-O and C++ experience

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.

Where to get more information

- OVOPS++
- Gamma: Object-Oriented Design Patterns
- Hüni: A Framework for Network Protocol Software

Copyright 1996-1997 Open Environment Software Oy.
All rights reserved.