Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

A Generic Communication Framework (TC 28/2/95 slides)

Guangxing Li

Abstract

Increasing multimedia application demands together with the modern network technology towards gigabit speeds require new adequate transport system. The new Nucleus communication subsystem is based on a modular protocol architecture with a high degree of configurability and extensibility.

The mechanism allows protocol composition and creation. It is independent of multiplexing, presentation, and concurrency models. The mechanism is also designed to suit the ANSA binding model, support streams and allow QoS processing.

APM.1416.00.01 **Draft** 23rd February 1995

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

A Generic Communication Framework

(Work In Progress)

Guangxing Li

gxl@ansa.co.uk

Drives

- **Today Cool Protocol (TCP) has well known performance problems**
  - **many performance constrained applications use UDP plus an application dependent reliable transportation protocol**
- **Multimedia applications**
  - **QoS controlled transportation**
  - **new protocols (RSVP, IETF's H.261 video stream protocol, current ATM Forum and ITU's effort on QoS class support)**
- **high-speed networks**
  - **efficient protocols - make sure end system can eat the amount of data (e.g it is estimated 780 MIPS is required for a 1 Gb/s TCP/IP link)**
  - **more flexibility - partial ordered protocols**
  - **QoS processing**

Challenges (in the context of DIMMA)

- a generic communication framework
  - function composition
  - configuration
  - independent concurrency
  - independent multiplex
  - independent presentation
  - QoS processing
  - explicit binding
  - stream

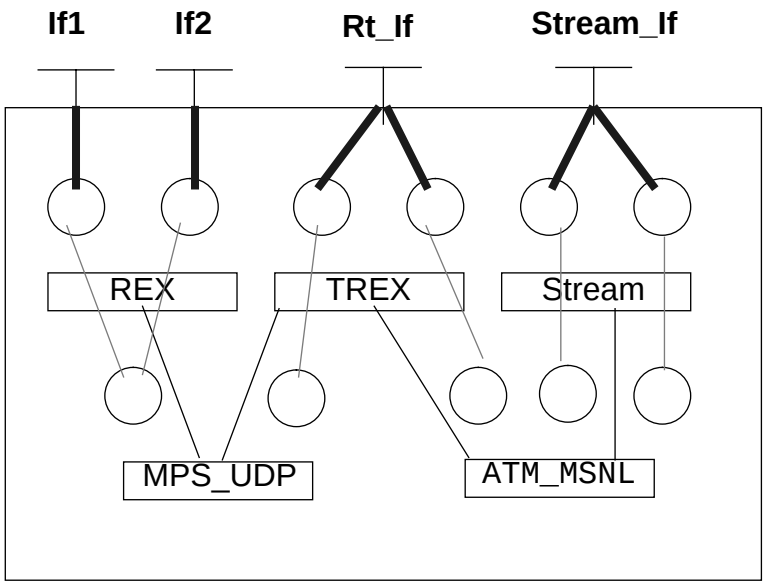
The x-kernel story

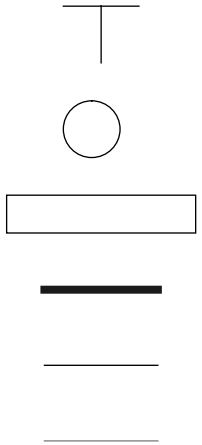
- first protocol composition and confuguration idea: UofA, 1988
- very successful - more than 25 standard protocols implemented plus hundreds of non-standard protocols
- shortcomes
  - object-oriented programming in C
  - one protocol object for everything (memory inefficient)
  - static protocol configuration
  - no resource control
  - no QoS
  - concurrency may lead to deadlock

Concepts

- module - a layer of protocol function
- protocol - a stack of module
- channel - an instance of a module
- binding - associate an interface to a set of channels
- binder - create binding

Example





Interface

Channel

Module

Binding

Module Link (Protocol)

Channel multiplex

Protocol: REX_MPS_UDP, TREX_MPS_UDP, TREX_ATM_MSNL, Stream_ATM_MSNL
Binder_op: REX_MPS_UDP
Binder_rt: TREX_MPS_UDP, TREX_ATM_MSNL
Binder_stream: Stream_ATM_MSNL

Module

- a layer of a protocol stack
- generate (*open*) and release (*close*) channel
  - inteprete address for a binder at client side
- demultiplex (*demux*) messages to channels
- management functions (*header, trailer, startup, module_id* etc.)

```
Class Module { ...
public:
    virtual Channel *open(Dispatch &upcall, QoS_C &qc, ModuleLink &link)  = 0;
    virtual int close(Channel *ch) = 0;
    void startup(ModuleLink &link);
    Dispatch demux;
    ...
};
```

Channel

- **provide the real communication functions (*read, write, call*)**
- **dispatch to upper layer Module**
- **generate address for server (used to form an ifRef by a binder)**
- **channel *control***

```
class Channel { ...
private: Dispatch &upcall;
public:
    virtual int control(int opcode, void *arg);   // control

    // channel functions
    virtual int call(Buffer &buf, QoS_I* qi=0);
    virtual int write(Buffer &buf, QoS_I* qi=0);     // push msg
    virtual int read(Buf &buf, QoS_I* qi=0);         // pop msg

    // address construction
    virtual addressRecord *gerAddress();
};
```

Protocol

- **a stack of Module**
- ***push* and *pop* of Modules**
- **protocol control (*startup, terminate*)**
- **channel creation (*open*) release (*close*)**

```
class Protocol { ...
private:   ModuleLink *stack;
public:
    // build a protocol
    int         push(Module *m);
    Module*     pop();

    // protocol control
    int         startup();  // set up buffers and call module startup
    int         terminate();

    // channel creation
    Channel     *open(Dispatch &upcall, QoS_C &qc);
    int         close(Channel *ch);
};
```

Binder

- **has a set of Protocols**
- **create binding**
- **management: *addProtocol* and *removeProtocol***
- **bind operation: *svrBind* and *cltBind***

```
class Binder { ...
private:
    ProtocolLink     *plink;
public:
    int int addProtocol(Protocol *ptl);
    int removeProtocol(Protocol *ptl);

    // bind
    virtual Binding*    svrBind(Dispatch upcall, QoS_C &qc = 0) = 0;
    virtual Binding*    cltBind(IfRef &ifref, QoS_C &qc = 0) = 0;
};
```

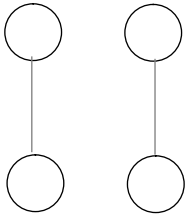
Binding

- **has a set of channel**
- **generate ifRef**
- **index to channel mapping**

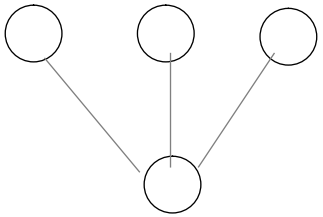
```
class Binding { ...
private:
    Channel   *list;
public:
    IfRef     *getIfRef();

    Channel *channel();
    Channel  *channel (int index);
};
```

Example: multiplex

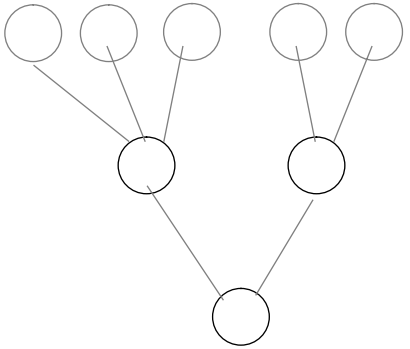
no multiplex



maximum multiplex

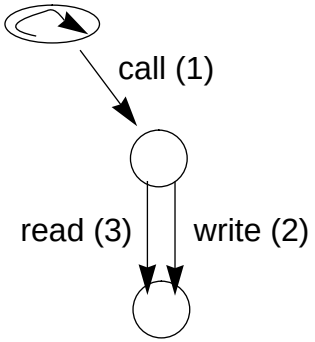


maximum multiplex with cocurrency

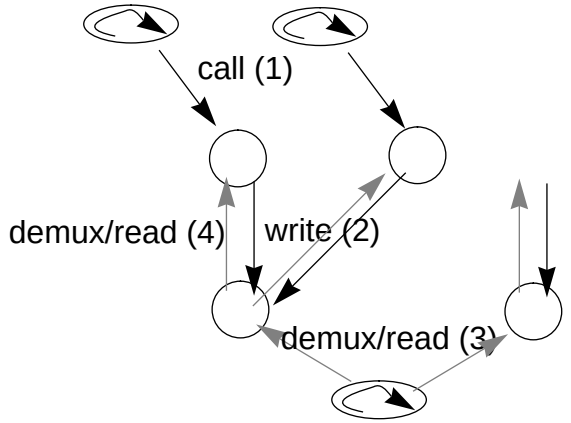


Example: client concurrency

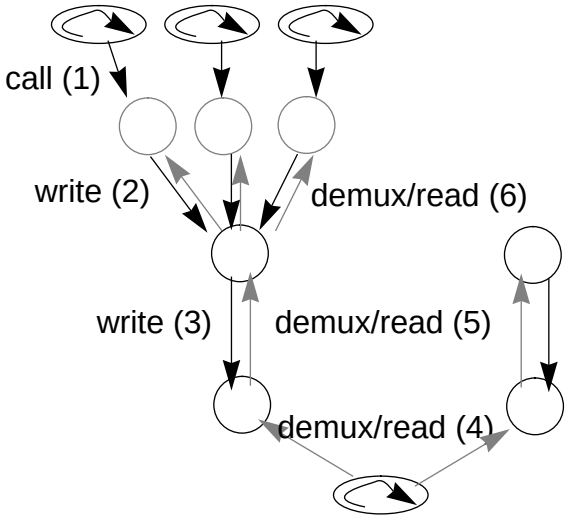
fast



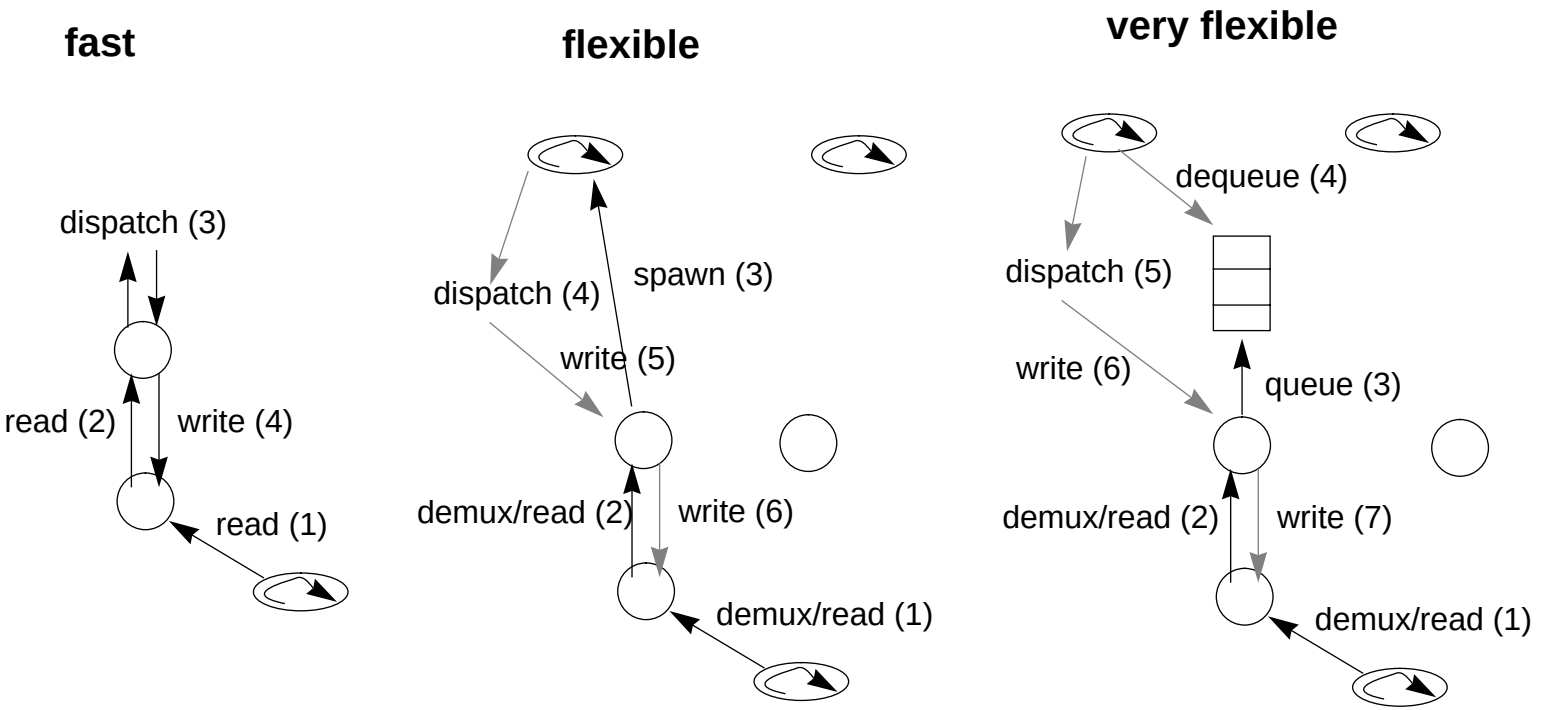
flexible



very flexible



Example: server concurrency



Comparison with x-kernel

- structured (real object-oriented)
- dynamic protocol configuration vs static configuration
- many concurrency models vs one concurrency model
- many multiplex models vs many multiplex models
- qos processing vs no qos processing
- fit ANSA binding model and allow streams
- no passive channel objects (more experimentation required)

Current status

- C++ classes for the framework
- a scheme for scalable memory management (UNIX inode like)
- REX implementation in progress