



---

**Poseidon House  
Castle Park  
Cambridge CB3 0RD  
United Kingdom**

TELEPHONE:  
INTERNATIONAL:  
FAX:  
E-MAIL:

**Cambridge (01223) 515010  
+44 1223 515010  
+44 1223 359779  
apm@ansa.co.uk**

---

## **ANSA Phase III**

# **A Generic Communication Framework (TC 28/2/94 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.1423.02

**Approved**

27th February 1995

Project Management (confidential to ANSA consortium for 2 years)

---

**Distribution:**

**Supersedes:**

**Superseded by:**





# **A Generic Communication Framework (Work In Progress)**

**Guangxing Li**

**gxl@ansa.co.uk**



## Drivers

- **Today Cool Protocol (TCP) has well known performance problems**
  - many performance constrained applications use UDP plus an application dependent (reliable) transportation protocol
- **Multimedia applications**
  - new protocols (RSVP, IETF's protocol for H.261 video streams, 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 (functionality/performance tradeoff)
  - QoS processing



## Challenges to the new Nucleus

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



## The x-kernel story

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

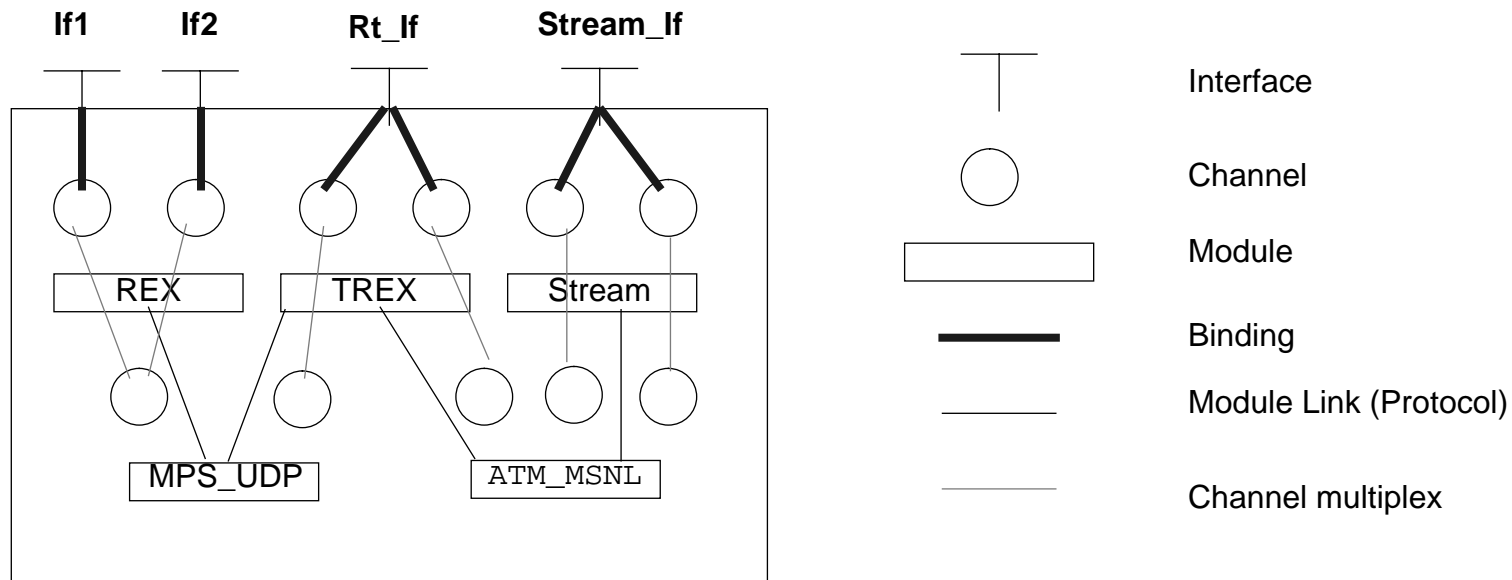


---

## The Nucleus communication framework: concepts

- **module** - a layer of protocol function (eg. TCP, IP, UDP, MPS\_UDP, REX, GEX)
- **protocol** - a stack of module (eg. TCP/IP, UDP/IP, MPS\_UDP/REX)
- **channel** - an instance of a module (eg. UDP socket, REX channel)
- **binding** - associate an interface to a set of channels
- **binder** - create binding (eg. ANSAware binder, ANSAware/RT binder)

# Example



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 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;
};
```



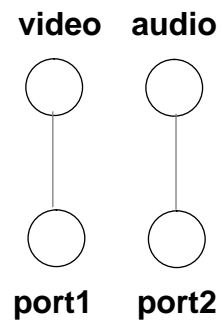
## Binding

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

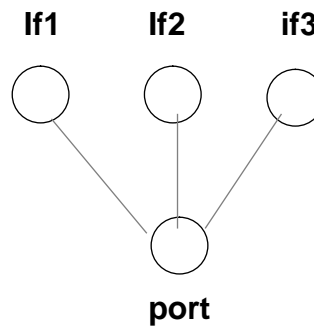
```
class Binding { ...  
private:  
    Channel *list;  
public:  
    IfRef *getIfRef();  
  
    Channel *channel();  
    Channel *channel (int index);  
};
```

# Channel multiplex

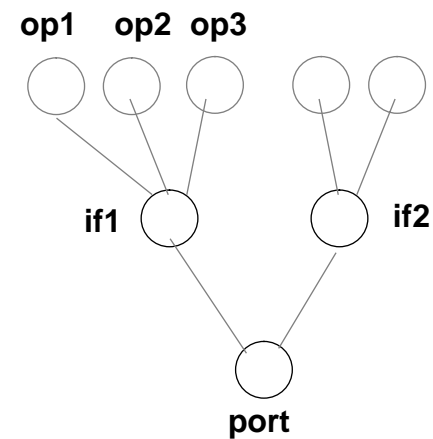
**no multiplex**



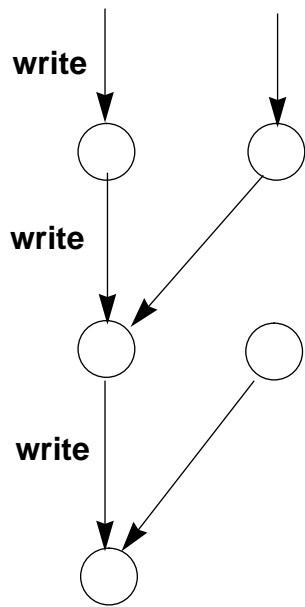
**maximum multiplex**



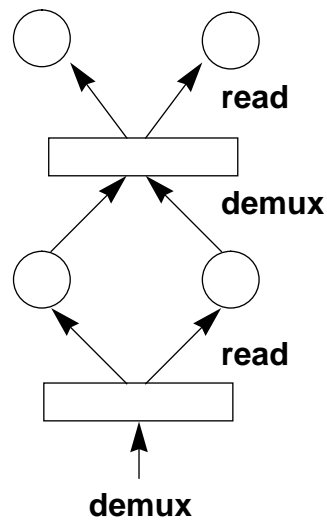
**maximum multiplex with concurrency**



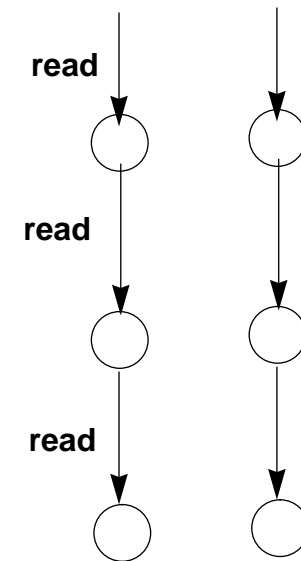
# Message processing



**send**



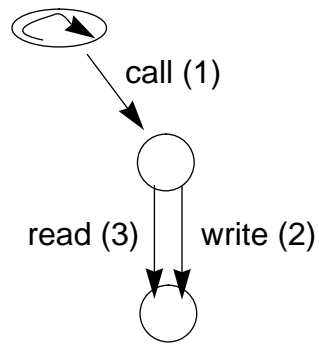
**receive**



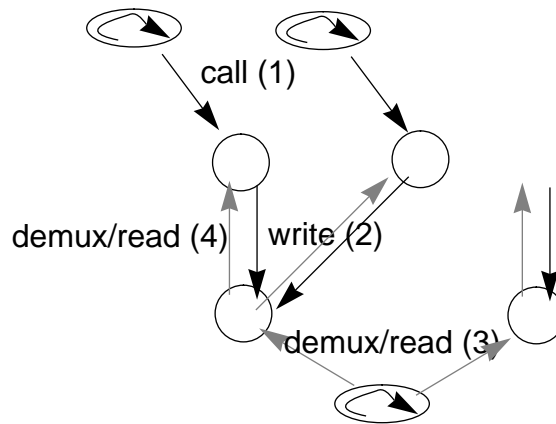
**receive**

# Client concurrency in channels

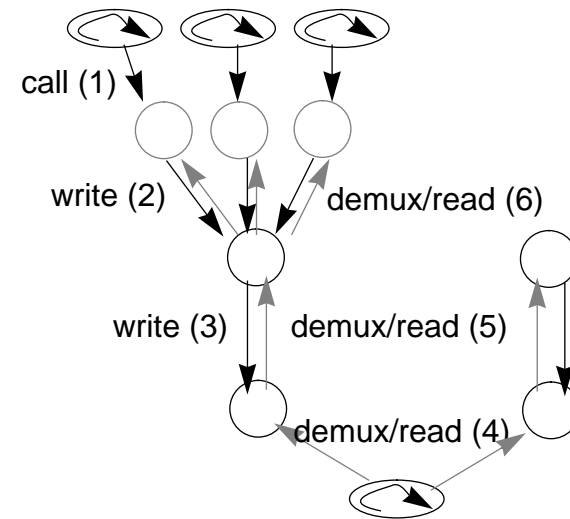
**fast/expensive**



**flexible/slow**



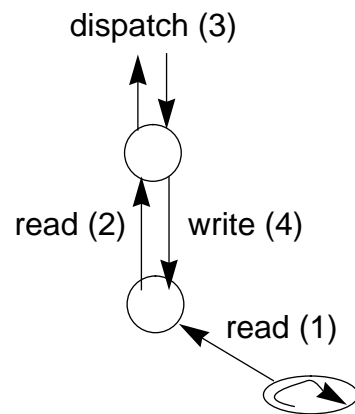
**very flexible/slower**



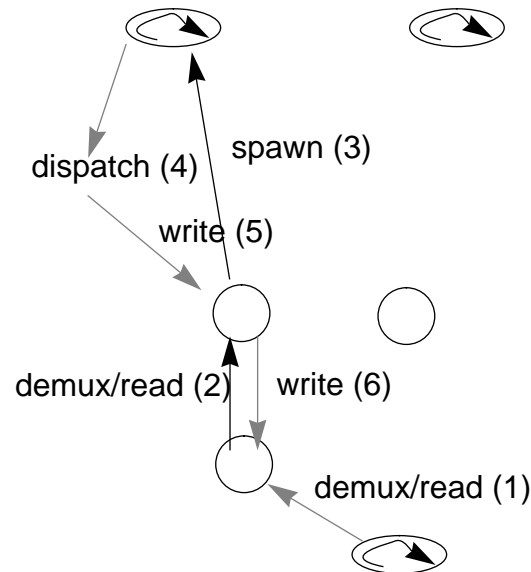


# Server concurrency in channels

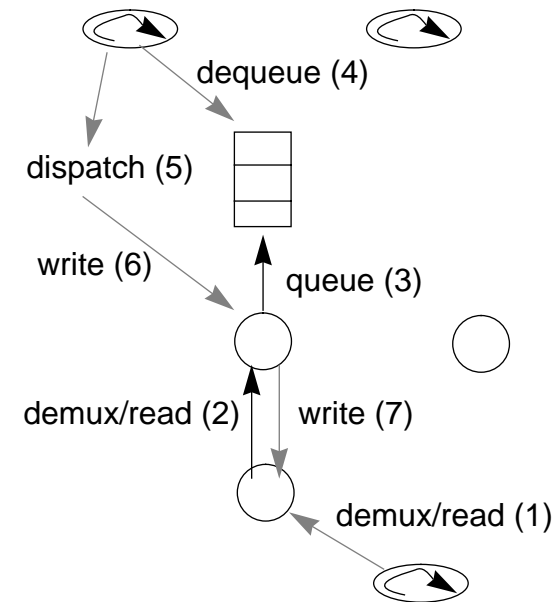
**fast/expensive**



**flexible/slow**



**very flexible/slower**





## Summary and comparison with x-kernel

- structured (real object-oriented)
- dynamic protocol configuration vs static configuration
- many concurrency models vs one concurrency model
- both allow many multiplexing 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**