Infrastructure Communication Manager Introduction

Qinghua Jin

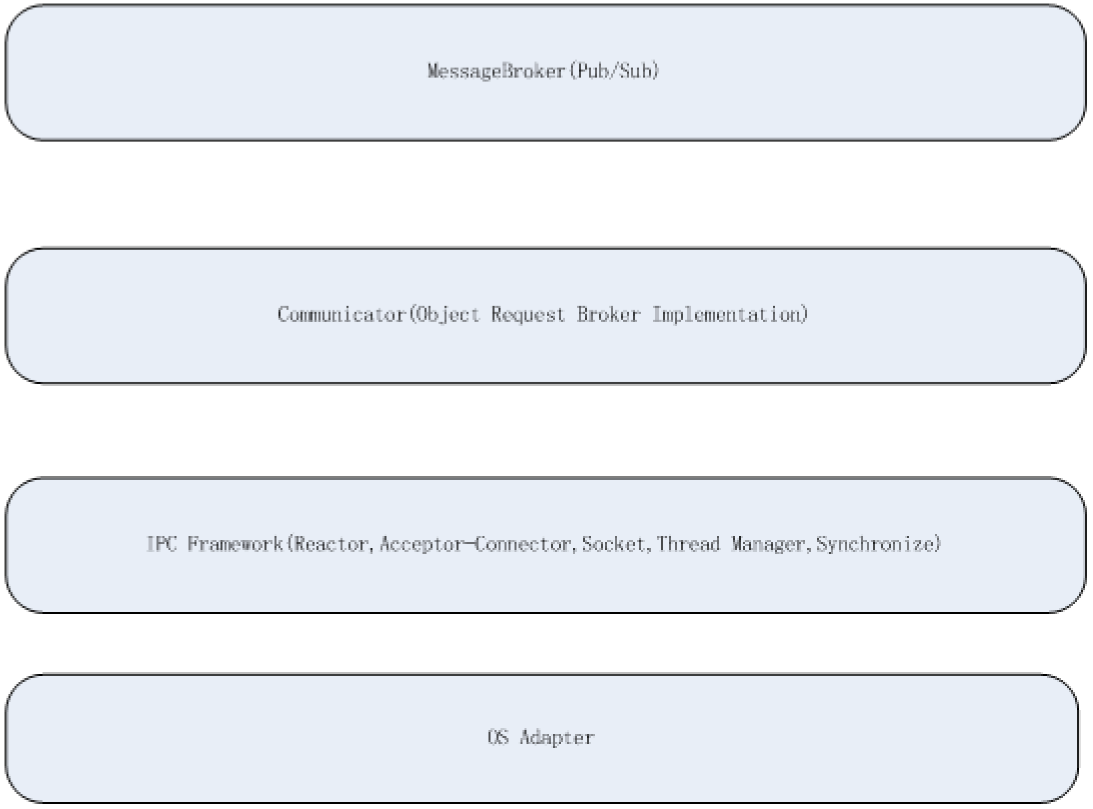
Topic

Feature
Architecture
Usage

Feature

High performance, low latency, real-time
Distributed object computing platform
Pattern oriented/Object oriented software architecture
Client side sync/async method invocation
Server side sync/async method dispatch
DDS style, topic based publish/subsriber message broker
Efficient protocol marshal/demarshal
Pluggable transport protocol
Pluggable message protocol
Proxy/Server side idl code generation

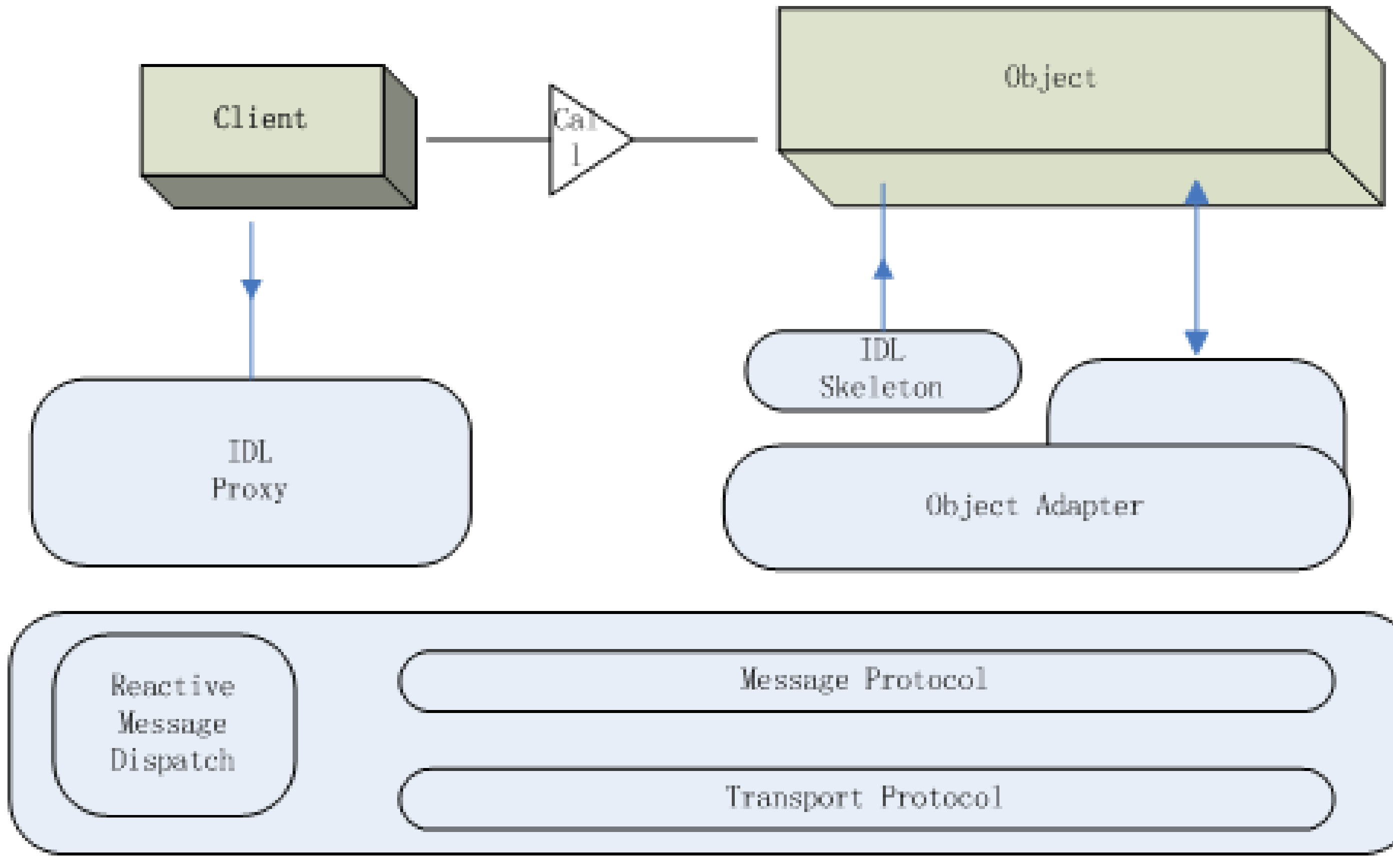
ICM Compoment


MessageBroker(Pub/Sub)
Communicator(Object Request Broker Implementation)
IPC Framework(Reactor,Acceptor-Connector,Socket,Thread Manager,Synchronize)
OS Adapter

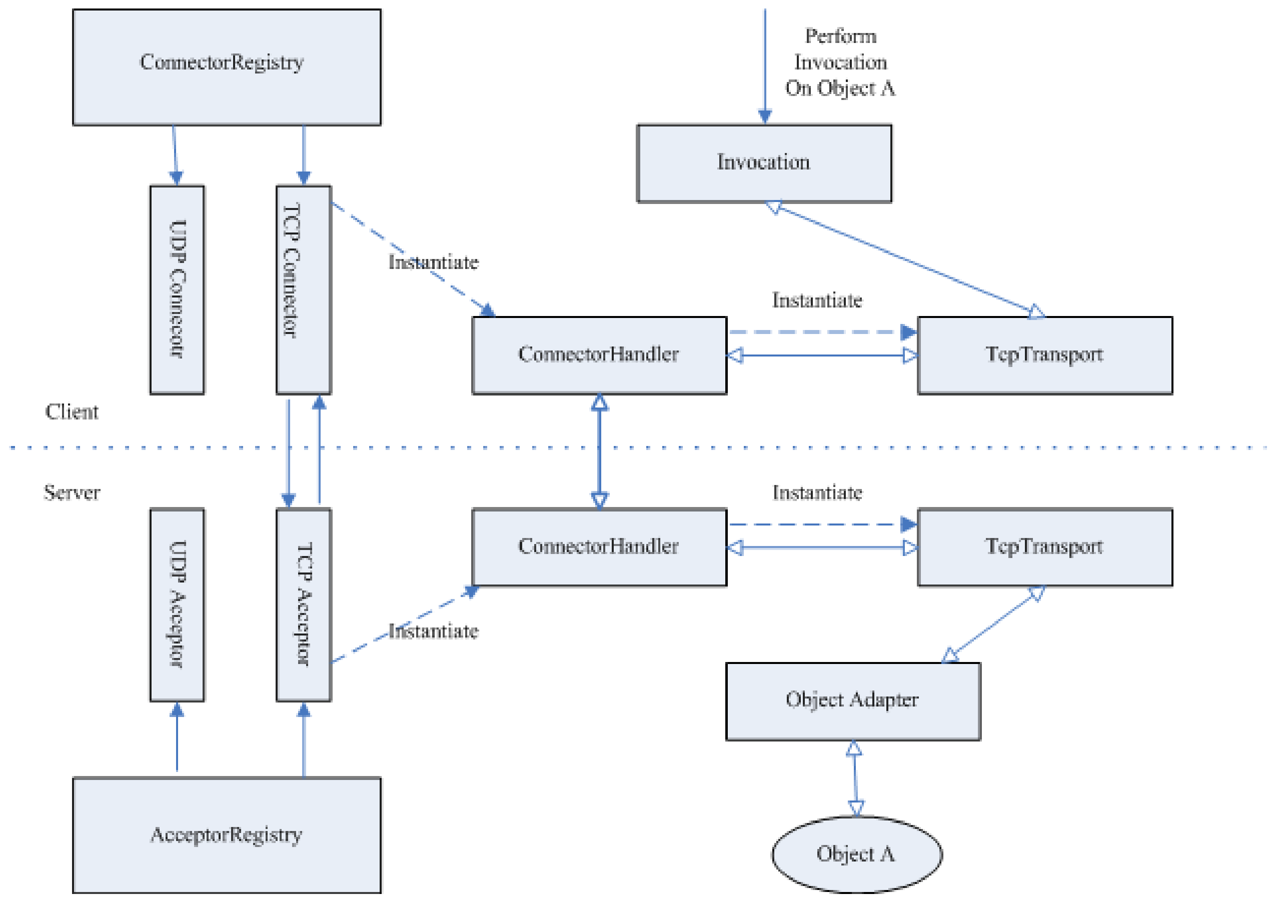
Internet Controller(ICC)

- Support different OS(Linux,Windows)
- Reactor framework(select/epoll based)
- Connection manage framework
- Thread manage and synchronize

Internet Communicator Architecture



Pluggable Transport Layer



Message Protocol

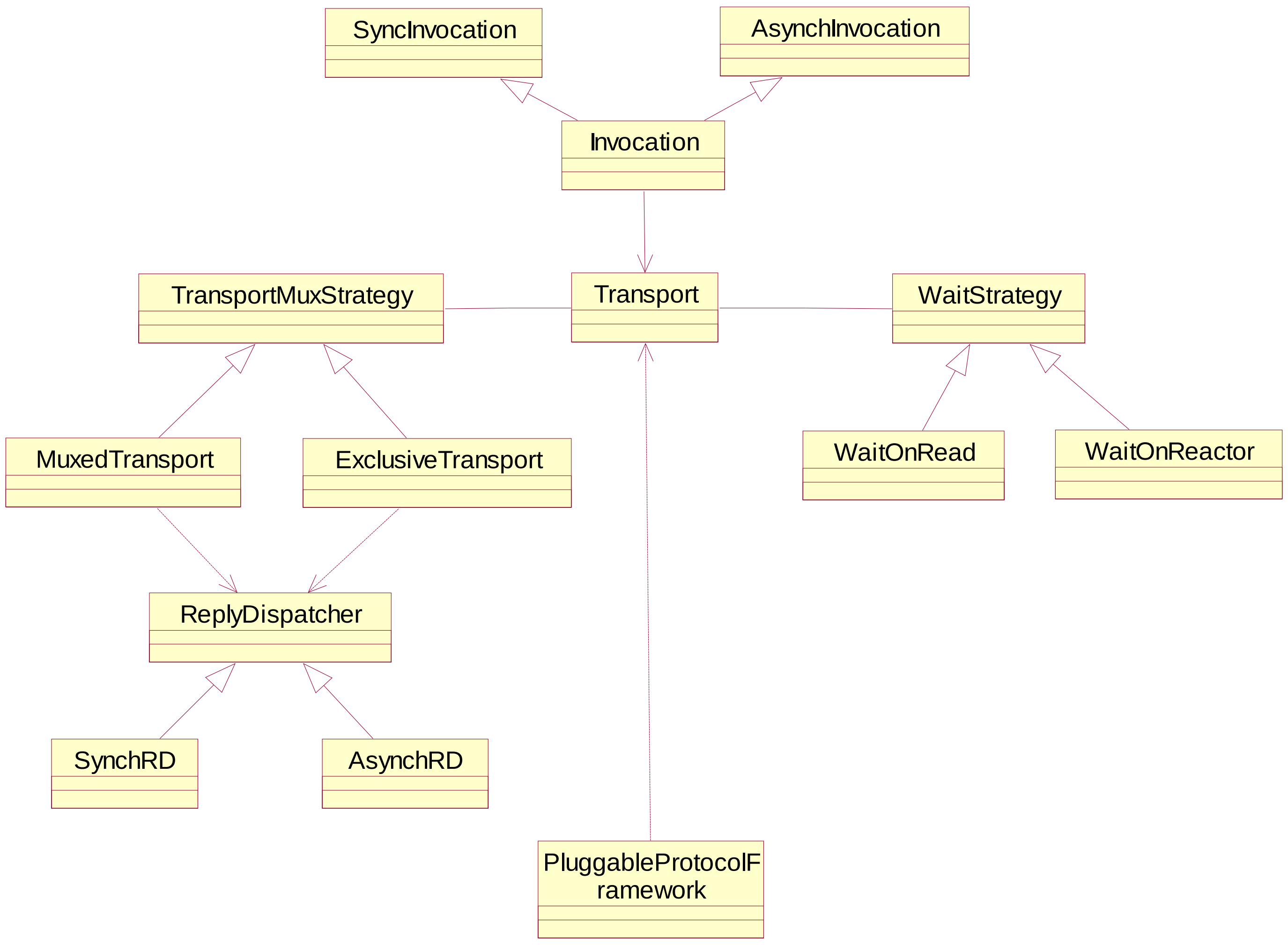
Request message

Response message

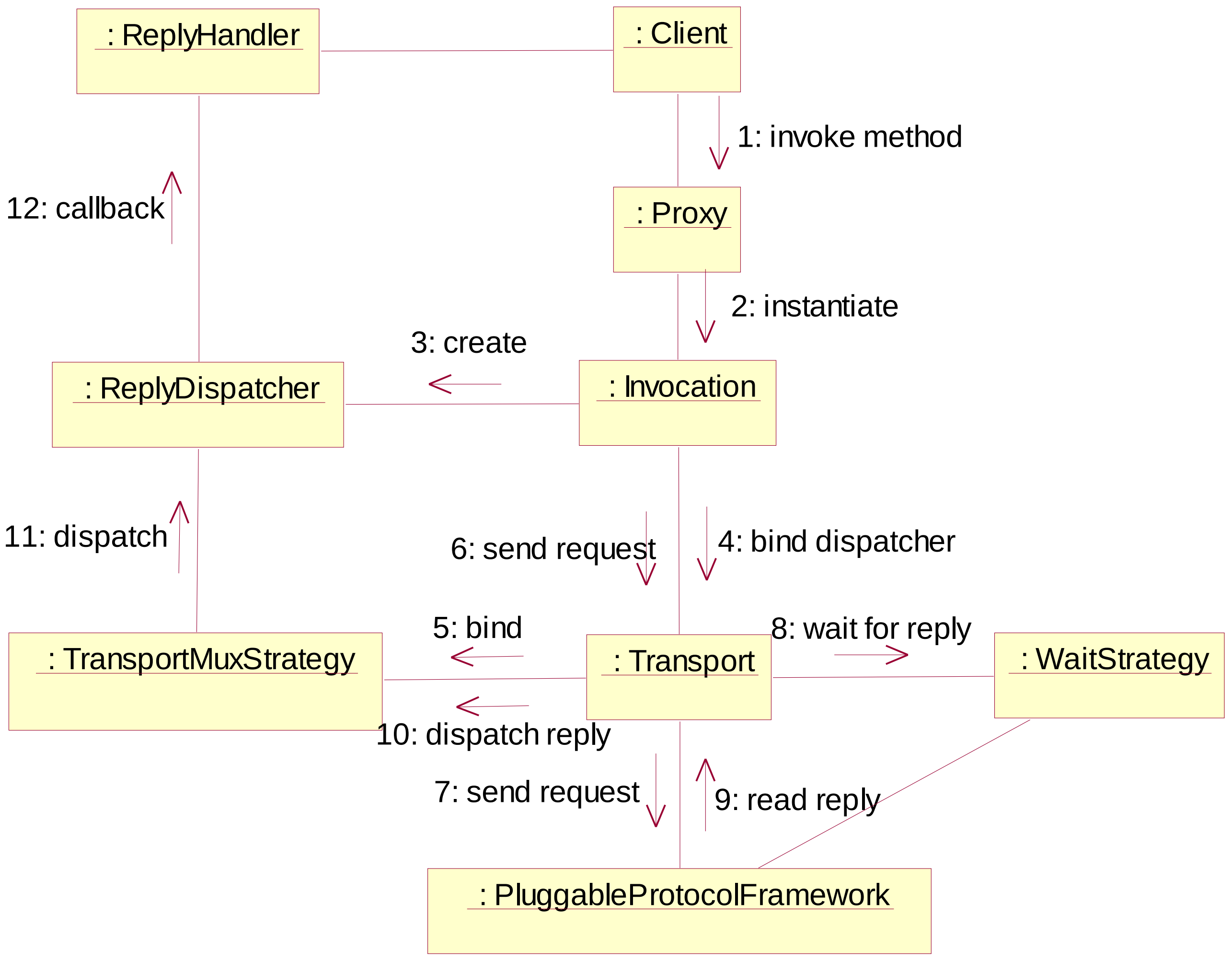
Concurrent Strategy

Single thread reactor
Mutiple thread reactor
Half sync/half async

SMI & AMI


SyncInvocation
AsynchInvocation
Invocation
TransportMuxStrategy
Transport
WaitStrategy
MuxedTransport
ExclusiveTransport
WaitOnRead
WaitOnReactor
ReplyDispatcher
SynchRD
AsynchRD
PluggableProtocolFramework

Asynch Method Invocation(AMI)



Usage

IDL definition and compile
Server side impl
Compile and link
Execution

IDL

Support type:
Integer(short,int,long)
String
Struct
Sequence(list)
Dictionary(map) (not tested)
Compile

```
#s2cpp my.idl  => generate: my.h my.cpp
```

my.idl

```
module demo
{

class MyHello
{
  string sayHello(string msg, short u, out long v);
};

};
```

my.h

```
namespace demo
{

class MyHello : virtual public Object
{
public:

  MyHello() {}

  virtual ::std::string sayHello(const ::std::string&, Short, Long&) = 0;
  DispatchStatus ___sayHello(ServerRequest&);

  virtual DispatchStatus __dispatch(ServerRequest& in);
};

}

namespace IcmProxy
{

namespace demo
{

class MyHello : virtual public IcmProxy::Object
{
public:

  ::std::string sayHello(const ::std::string& msg, Short u, Long& v);
};

}

}
```

my.cpp

```
::std::string
IcmProxy::demo::MyHello::sayHello(const ::std::string& msg, Short u, Long& v)
{
  static const char* __operation("sayHello");
  Reference* ref = this->getReference ();
  TwowayInvocation _invocation (ref, __operation, ref->communicator ());
  int ok = _invocation.start (this->transport ());
  if (ok != 0)
    return "";
  ok = _invocation.prepareHeader (1);
  if (ok != 0)
    return "";
  OutputStream* __os = _invocation.outStream();
  __os->write_string(msg);
  __os->write_short(u);
  ok = _invocation.invoke();
  if (ok != 0)
    return "";
  InputStream* __is = _invocation.inpStream();
  ::std::string __ret;
  __is->read_long(v);
  __is->read_string(__ret);
  return __ret;
}
```

Client side

```
int
main (int argc, char* argv[])
{
  Communicator* comm = Communicator::instance();
  if (comm->init (true) == -1)
    return -1;

  Reference ref (comm, Identity("MyHello"), Endpoint("TCP", "127.0.0.1", 3000));
  IcmProxy::demo::MyHello myHello;
  myHello.setReference (&ref);

  for (int i = 0; i < 10; i++) {
    Short u = 10 + i;
    Long v = 1000 + i;
    std::ostringstream ss;
    ss << "hello, world from " << i;
    string ret = myHello.sayHello (ss.str(), u, v);
    if (ret != "") {
      std::cout << "ret:" << ret << std::endl;
    } else {
      //err process
    }
  }

  return 0;
}
```

Server impl

```
namespace demo
{

class MyHelloI : public demo::MyHello
{
public:
  virtual std::string sayHello(const ::std::string&, Short, Long&);

};

}
```

```
std::string demo::MyHelloI::sayHello(const ::std::string& msg, Short u, Long& v)
{
  ostringstream oss;
  oss << "receive:" << " msg:" << msg << " u:" << u << " v:" << v ;
  std::string tmp = oss.str();
  cout << tmp;

  v = 0x1234 + v;

  return tmp;
}
```

Server side

```
int
main (int argc, char* argv[])
{
  Communicator* comm = Communicator::instance();
  if (comm->init () == -1)
    return -1;

  Endpoint endpoint ("TCP", "", 3000);
  ObjectAdapter* oa = comm->createObjectAdapterWithEndpoint ("MyHello", &endpoint);
  Object* object = new demo::MyHelloI;
  oa->add (object, "MyHello");

  comm->run ();

  return 0;
}
```

Message Broker

module demo
{

```
struct NetEvent
{
  string ip;
  short port;
  string event;
};

class Network
{
  void reportEvent(NetEvent event);
};

};
```

IDL Impl

```
class NetworkI : public demo::Network {
public:
  virtual void reportEvent(const ::demo::NetEvent&);
};
```

void

```
NetworkI::reportEvent(const ::demo::NetEvent& netEvent)
{
  cout << "receive network event:" << endl;
  cout << "ip:" << netEvent.ip << " port:" << netEvent.port << " event:" << netEvent.event << endl;
}
```

Subscriber

```
int
Subscriber::run(int argc, char* argv[]) {
  Communicator* comm = Communicator::instance();
  if (comm->init (true) == -1)
    return -1;

  Reference ref (comm, Identity("TopicManager"), Endpoint("TCP", "127.0.0.1", 5555));
  IcmProxy::IcmMsg::TopicManager topicManager;
  topicManager.setReference (&ref);

  ObjectAdapter* adapter = comm->createObjectAdapterWithEndpoint("Subscriber", "127.0.0.1 8888");
  IcmProxy::Object* networkProxy = adapter->add(new NetworkI(), "NetworkTopic");

  ::IcmProxy::IcmMsg::Topic* topic = topicManager.retrieve("NetworkTopic");
  if(topic == 0)
  topic = topicManager.create("NetworkTopic");
  if (topic == 0)
    return -1;
  topic->subscribe(networkProxy);

  comm->run();

  return 0;
}
```

Publisher

```
int Publisher::run(int argc, char* argv[]) {
  Communicator* comm = Communicator::instance();
  if (comm->init (true) == -1)
    return -1;

  Reference ref (comm, Identity("TopicManager"), Endpoint("TCP", "127.0.0.1", 5555));
  IcmProxy::IcmMsg::TopicManager topicManager;
  topicManager.setReference (&ref);

  ::IcmProxy::IcmMsg::Topic* topic = topicManager.retrieve("NetworkTopic");
  if(topic == 0)
  topic = topicManager.create("NetworkTopic");
  if(topic == 0) {
  cout << "err create topic " << endl;
  return -1;
  }

  ::IcmProxy::Object* pubObj = topic->getPublisher();
  if(pubObj == 0) {
  cout << "err get publisher " << endl;
  return -1;
  }
  IcmProxy::demo::Network network;
  network.setReference(pubObj->getReference());

  cout << "publishing network events:" << endl;
  demo::NetEvent event;
  event.ip = "172.16.10.190";
  event.port = 6789;
  for(int i=0; i< 10; i++) {
    ostringstream oss;
    oss << "evt:" << i;
    event.event = oss.str();
    network.reportEvent(event);
  }

  return 0;
```