CMSG – A PUBLISH/SUBSCRIBE INTERPROCESS COMMUNICATION PACKAGE

E. Wolin, D. Abbott, V. Gyurjyan, G. Heyes, E. Jastrzembski, D. Lawrence, C. Timmer, Jefferson Lab, Newport News, VA 23606, U.S.A.

Abstract

Publish/subscribe message passing is an extremely simple, flexible, and powerful interprocess communication (IPC) paradigm. It is widely used in industry, but not nearly so in High Energy and Nuclear Physics (HENP), perhaps due to the cost of commercial implementations. cMsg, developed at Jefferson Lab in the US, contains a full-featured pub/sub interprocess communication package that is simple to install and use. It is very efficient, and implements both point-to-point and pub/sub communications, server redundancy, hot server failover, and a server discovery service. In addition, for developers cMsg provides a framework within which one can deploy multiple underlying communication packages that do not necessarily need to implement the full pub/sub paradigm. This allows for unification of all communication in a control system under a single API, shortens development time, and allows for simple upgrade or replacement of underlying communication packages and protocols.

INTRODUCTION

Interprocess communication packages based on the publish/subscribe paradigm are not widely used in particle and nuclear physics experiments and accelerator systems. The LHC uses SoniqMQ in its LASER alarm system, CDF at Fermilab uses Tibco SmartSockets to control front-end single-board computers in its DAQ system, and at Jefferson Lab for over a decade the CLAS experiment has also used SmartSockets in its online control system.

Based on its highly successful use in CLAS, the JLab DAQ group decided to develop its own pub/sub IPC package as the foundation for all interprocess communication in the next generation of JLab DAQ software. An additional requirement was that the package provide a framework and proxy service under which legacy JLab IPC packages could be deployed, a requirement not satisfied by any commercial or public domain package, including recent Java Message Service (JMS) implementations. Licensing and distribution issues further led us to develop our own package.

In the following we first describe the pub/sub paradigm and make a clear distinction between it and others, such as the client/server paradigm (note that there is much confusion about this distinction in the HENP community). Next we describe the pub/sub implementation in the cMsg package from a client or user point of view. After this we discuss the cMsg package from the developer point of view, especially its framework and proxy server features. Next we briefly discuss cMsg performance, then conclude with a summary.

WHAT IS PUBLISH/SUBSCRIBE

The publish/subscribe paradigm is deceptively simple, but quite powerful and general [1]. Producers publish messages asynchronously to abstract "subjects" or "topics" with no knowledge of whom, if anyone, might receive the messages, nor how or when the messages are delivered. Subjects are arbitrary strings that can be created at will, and multiple producers can publish to the same subject. Partitioning of the messaging space is accomplished via suitable naming conventions in the subject space.

Consumers subscribe to subjects with no knowledge of who may publish to them. A single process can be both a producer and a consumer, and may subscribe to the same subjects it publishes to. Messages are typically processed asynchronously via callbacks.

Key features are that many producers can publish to the same subject, producers can publish messages at will and not just as a response to a request, and that producer and consumer processes are completely decoupled. In contrast, many IPC systems require coordination between processes that communicate with each other, and the disappearance of one has effects on the other.

Comparing Pub/Sub to Client/Server

In client/server systems the client initiates contact with a server facility requesting the server to perform some action on its behalf. Usually, but not always, the server sends a response back to the client, sometimes more than one response over an extended period.

This model can easily be implemented in a pub/sub system. "Client" processes need only publish request messages to unique subjects that other "server" processes subscribe to, and provide a unique subject that the client subscribes to. The server processes send responses to the unique client subjects.

Note that the reverse, implementing a pub/sub model using a client/server system, is quite complicated.

Many other IPC paradigms can be implemented within a pub/sub system. Some of these will be discussed in the following sections.

WHAT IS CMSG – CLIENT PERSPECTIVE

The cMsg package includes a full-featured pub/sub facility that implements compound message payloads and automatic endian swapping. Payloads can contain arbitrary combinations of primitive types, arrays of primitive types, cMsg messages, and arrays of cMsg

messages. API's are provided for C, C++ and Java, and cMsg is supported on many flavors of Unix and VxWorks. Unlike other pub/sub implementations, cMsg subscriptions and message routing are based on a pair of fields, subject and type. Both can be arbitrary strings and are treated identically. Wildcard characters are supported in subscriptions.

Message routing is performed by one or more cooperating cMsg server processes that can be deployed as needed to equalize cpu and network loads. Server features include hot failover, a server discovery service, and a system monitoring facility. The cMsg server is written completely in Java.

The client API is designed to be as simple as possible. No interface definition files, stub generators, or similar things are required.

cMsg includes many more features than just this facility, and these will be discussed in the section aimed at developers below. It is not necessary to understand the concepts in the developer's section to successfully use the cMsg pub/sub facility, though.

The following C++ code fragments demonstrate how to connect to the cMsg system and how to send and receive messages. Note that a single process can have multiple connections to the cMsg system, if needed.

Sending a Message

```
#include <cMsg.hxx>

// connect to cMsg system, UDL specifies messaging space
cMsg c(UDL, "myName", "My description");
c.connect();

// create and fill message object
cMsgMessage msg;
msg.setSubject("mySubject");
msg.setType("myType");
msg.setText("This is my text");

// send message
c.send(msg);
```

Receiving a message

```
#include <cMsg.hxx>

// connect to cMsg system
cMsg c(UDL, "myName", "My description");
c.connect();

// subscribe and start receiving
c.subscribe("mySubject", "myType", new myCallback(), NULL);
c.start();

    // do something else…
```

where the callback class is:

```
class myCallback : public cMsgCallback {

  // see user manual for description of userObject
  void callback(cMsgMessage* msg,  void* userObject) {
      cout << "Message subject is:   " << msg->getSubject() << endl;
  }
};
```

Universal Domain Locator (UDL)

The UDL, a runtime parameter, specifies how to find and connect to a cMsg message server or broker, and which logical messaging space should be used. UDL server information includes a host name and port, or "multicast" in which the first server to respond is chosen. The logical messaging space name is just an arbitrary string, and messages in one space will not be delivered to another space. A simple example UDL using default ports is:

```
cMsg://ollie.jlab.org/cMsg/myMessageSpace
```

See the cMsg manual for a complete discussion of UDL syntax and semantics [2].

WHAT IS CMSG – DEVELOPER PERSPECTIVE

Note that it is not necessary to understand the concepts in this section to use the full cMsg pub/sub facility.

Some important cMsg design requirements were to provide a framework under which legacy communication protocols could be unified under a single API, and to provide a mechanism to allow processes running on any operating system to access legacy protocols, even if no legacy library existed on that system. For example, some of our legacy protocols are not supported on VxWorks.

This led to two key concepts: "domains" implemented in a client-side framework, and "subdomains" implemented in a proxy-server facility.

Framework

Connection to an underlying communication system in the cMsg package is controlled by the UDL, a string that specifies the kind of communication desired and how to connect to the system implementing it. Which system or "domain" to connect to is determined at runtime in the client when it parses the UDL.

Many domains are provided, and new ones can easily be added to the client framework. Included are the cMsg domain (main focus of this article), FILE domain, EPICS Channel Access [3] domain, and others described in the user's manual found on the FTP site [2]. Technically, specifying the cMsg domain only results in connection to the proxy server described in the next section.

Note that the messaging API is independent of which domain is selected at runtime. Domains do not necessarily implement the full publish/subscribe paradigm.

Proxy Server

If the cMsg domain is specified in the UDL the client connects to a cMsg domain server process. The server parses additional fields in the UDL to determine which "subdomain" handler class should be used to implement

the communications. The subdomain code may well employ a communication library unavailable on the client node. A private protocol is used to transport message data from the client to the cMsg server.

Many subdomains are available, and new ones can easily be added (in Java only). The cMsg subdomain implements the full pub/sub system that is the main focus of this article. Other subdomains include the Queue subdomain, FileQueue subdomain, LogFile subdomain, EPICS Channel Access subdomain, and others described in the user's manual [2].

To be precise, the full pub/sub system described in this article entails communicating via the cMsg domain handler in the client-side framework to the proxy server, and use of the cMsg subdomain handler in the proxy server itself [4].

Note that although the underlying implementation involves use of the client/server model, due to the nature of the networking libraries, from the user point of view the cMsg system implements a pure publish/subscribe model.

CMSG PERFORMANCE

cMsg performance requirements were modest, and were mainly aimed at controls applications. But modern cpu's and Java are so fast that cMsg performance exceeded our requirements by orders of magnitude. This allowed us to extend the range of cMsg applications to include all but the highest speed data acquisition applications. For the latter we use a high-speed shared-memory based data transfer system [5].

Briefly, on modern systems and for message sizes above 1 kByte, data transfer rates are limited by network performance (i.e. over 800 Mbits/sec using GBit Ethernet) with cpu loading less than 20%. For smaller messages, up to 1 kByte, overhead in the network driver begins to dominate, the network does not saturate, and messaging rates top out at around 25,000 messages/sec.

Messaging and data rates are up to three times higher for local messaging, i.e. where producer, server, and consumer reside on the same node and the network is not accessed.

SUMMARY AND CONCLUSIONS

The publish/subscribe messaging paradigm is powerful and flexible, and can satisfy almost all messaging requirements in HENP control and DAQ applications. The cMsg package includes a full-featured asynchronous publish/subscribe messaging component that is simple to use, robust, and has very high performance. It provides C, C++, and Java API's, and runs on many flavors of Unix.

For developers, cMsg can additionally be used as a framework to unify all communications in a control or DAQ system, and provides a Java proxy server that can allow processes running on a node to access communication protocols not supported on that node.

It is unfortunate that the publish/subscribe paradigm is not particularly well known or understood in the HENP community, particularly compared to the client/server paradigm, as our experience shows it to be an excellent match to our requirements. In the past this may have been partly due to the cost of commercial implementations.

The cMsg package is freely available to those working in HENP and other research fields [2].

REFERENCES

[1] See the Wikipedia entry on pub/sub messaging: http://en.wikipedia.org/wiki/Publish/subscribe.

[2] The cMsg manual and package can be downloaded from ftp://ftp.jlab.org/pub/coda/cMsg.

[3] See http://www.aps.anl.gov/epics.

[4] Thus the cMsg domain handler class and the cMsg subdomain handler class are distinct, and perform completely different tasks. Only the names are the same, perhaps a source of mild confusion. This is why "cMsg" is repeated twice in the example UDL, once to specify the cMsg domain, and again to specify the cMsg subdomain.

[5] C. Timmer, D.J. Abbott, W.G. Heyes, E. Jastrzembski, R.W. MacLeod and E. Wolin, "Fast Transfer of Shared Data", CHEP 2000, Padova, Italy, p. 585-588 (2000).