Decoupling Control System Components Using Asynchronous Publish/Subscribe Middleware

Elliott Wolin, Carl Timmer, D Abbott, W Gu, V Gyurjyan, G Heyes, E Jastrzembski, D Lawrence, and B Moffitt

Thomas Jefferson National Accelerator Facility, Newport News, VA 23606

Introduction

cMsg is a full-featured message-based publish/subscribe IPC system. Using this publish-subscribe software, control system components can be completely decoupled from each other. When each component does NOT depend on any other process (either its presence or behavior) then fully decoupling is achieved.

What is Publish/Subscribe ?

Message producers:

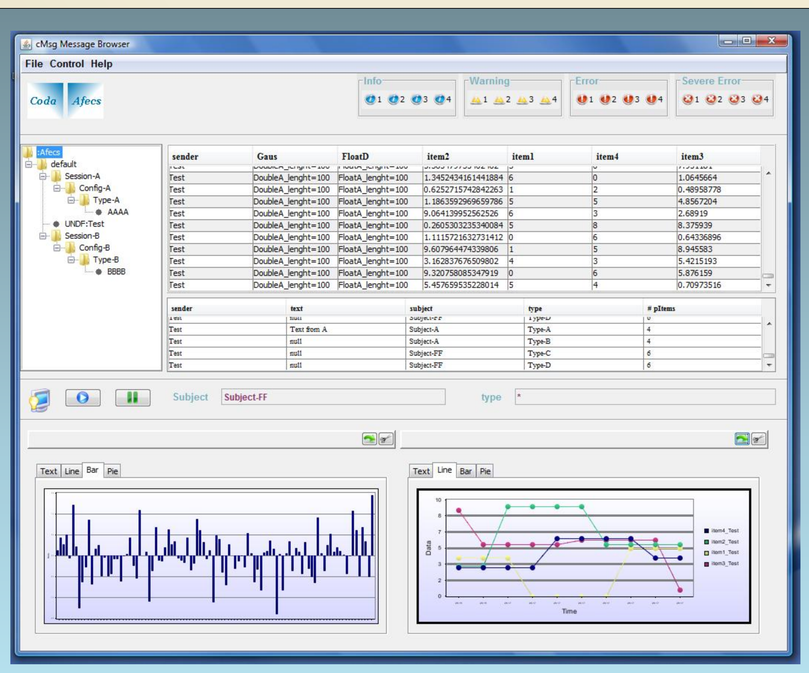
- **Publish or send messages to abstract subjects (strings)**
- **Publish to any subject at any time, independent of others**
- **No knowledge of consumers and their subscriptions necessary**
- **May publish to a subject no consumer subscribes to**
- **No prior registration of subjects required**
- **Subjects can be created dynamically, at will**
- **No connection or "coupling" of a subject to a producer process**
- **Publish messages at will in a "publish-and-forget" mode**

Message consumers:

- **Subscribe to subjects (strings), wildcards often supported**
- **No knowledge of producers and the subjects they publish to required**
- **May subscribe to a subject that no producer ever publishes to**
- **Operate in a "subscribe-and-forget" mode**

Asynchronous communication:

- **Producers do not block when a message is published**
- **Producers do not have to wait for some process to receive it**
- **Consumers receive messages via an asynchronous callback mechanism, running in a separate thread**
- **Consumers do not block when the subscription is made**



CAPTION HERE

What is Decoupling ?

Communication Type:

- **Uses communication which is asynchronous in nature, eliminating needless waits and timeouts**
- **The ability to asynchronously publish and subscribe independent of the existence of other producers and consumers is key to implementing the decoupling**

Control System Modification:

- **Changes to one part of a control system have no effect on other parts of the system**
- **Functionality can be added incrementally, with no disruption to existing systems**
- **System designers can implement basic interprocess communications between processes, then at a later date transparently add more consumers implementing new functionality, with no disturbance to the original system**
- **For example, at a later date a logger process could be activated that subscribes to the same subjects used by other processes and logs all the communications between them to disk or database, with no disruption to the original system).**

What is cMsg and how does it decouple?

What is cMsg?:

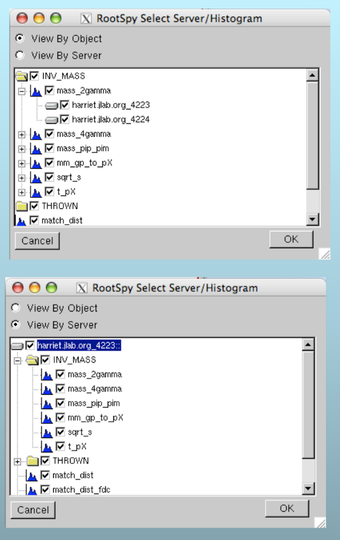
- **Software implementing a sophisticated version of the publish/subscribe model and thus automatically decouples users**

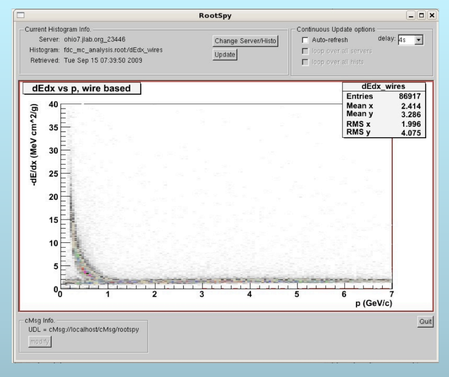
Some cMsg Features:

- **2 subject fields (subject & type) used in publishing & subscribing**
- **Messages hold all fundamental data types, their arrays, as well as cMsg messages and their arrays – as many components as desired**
- **Endian conversions are handled automatically (except for binary)**
- **Message routing is performed by high-performance background servers(written in Java). Servers can be grouped together into "clouds" which implement hot server failover and least-hop routing.**
- **Runs on Linux, Solaris, other flavours of Unix, and VxWorks**
- **The underlying transport mechanism could be replaced or modified transparently, with no modifications to user code needed**
- **Monitoring capabilities exist to supply complete information on all servers, producers, and consumers.**

cMsg API:

- **Available in C, C++ and Java**
- **Simple as possible; no IDL or stub generators needed**
- **The API is narrow in that only basic messaging functionality is provided, i.e. there is only one type of message and one way to fill, publish, subscribe, and receive messages**
- **Additional useful synchronous capabilities are provided for convenience.**





CAPTION HERE

Narrow Interface, API

cMsg unifies communication under a single, message-passing API (Java, C, C++). This table contains a simplified list of the major client functions in Java. The messages can contain unlimited user-settable fields including strings, primitive types, cMsg messages, binary and arrays of each.

Function	Description
connect(UDL, myName)	Connect to a cMsg system specified by the UDL for client myName
disconnect()	Disconnect from the cMsg system
send(msg)	Send a message asynchronously
flush(timeout)	Flush messages to send from client
syncSend(msg, timeout)	Send a message and wait for server response
sendAndGet(msg, timeout)	Send a message and wait for receiving client to send a response
subscribe(subject, type, callback)	Subscribe to messages of a given subject & type, registering a callback for incoming messages
unsubscribe()	Remove a subscription
subscribeAndGet(subject, type, timeout)	Subscribe to a subject & type and wait for one response
start()	Start receiving messages
stop()	Stop receiving messages
monitor(command)	Synchronous call to request monitoring information

Conclusions

The asynchronous publish/subscribe model is ideal for implementing a decoupled interprocess communication system. Producers can publish messages to any subject with no regard for the existence of other producers or consumers. Consumers can subscribe to any subject with no regard for the existence of other consumers or producers. New consumers can be added to implement additional functionality with no change needed to the existing system.

The cMsg package implements a narrow interface that has changed hardly at all over five years. It provides basic messaging functionality, and all additional customization must be done by developers via conventions in the controls system.

We have often created simple systems to implement some basic functionality, then added new functionality via new processes that listen in on the existing messaging and perform some new task (e.g. archiving or display), with no change to the original system needed. In this way functionality can be built up incrementally and transparently, and modified as needed with no effect on existing systems.

Download at ftp://ftp.jlab.org/pub/coda/cMsg