

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

At ICALEPCS 2007 Stephen Lewis advocated use of “decoupled” communications in controls systems. The asynchronous publish/subscribe model is ideal for doing this. The JLab cMsg package implements a version of this model, and has been in use at JLab for many years. Below we describe “decoupling” and the pub/sub model, then give examples of how we have used cMsg to implement decoupled communications at JLab.

What is Decoupling ?

Goal: Ability to add/modify functionality without disrupting working systems.

Communication Type:

- Message-based.
- Asynchronous, eliminate needless waits and timeouts.
- Messaging independent of the existence of other message producers or consumers.

Control System Modification:

- Changes to one part of control system have no effect on other parts of the system.
- Functionality can be added incrementally, with no disruption to existing systems.
- Example: logger/archiver process could be activated that subscribes to the same subjects used by other processes and logs all communications between them to disk or database. Decoupling ensures that logger can be added at a later date without modification to existing processes.

What is Publish/Subscribe ?

Message producers:

- Publish or send messages to abstract subjects (strings).
- Publish to any subject at any time, independent of other publishers.
- No knowledge of consumers and their subscriptions.
- 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 particular producer process.
- Operate 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.
- 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 consumer to acknowledge receipt of message.
- Consumers receive messages via asynchronous callbacks, usually run in a separate thread.
- Consumers do not block when subscription is made.

What is cMsg and how does it support decoupling?

What is cMsg?:

- Software that implements a sophisticated version of the publish/subscribe model, and thus automatically supports decoupling.

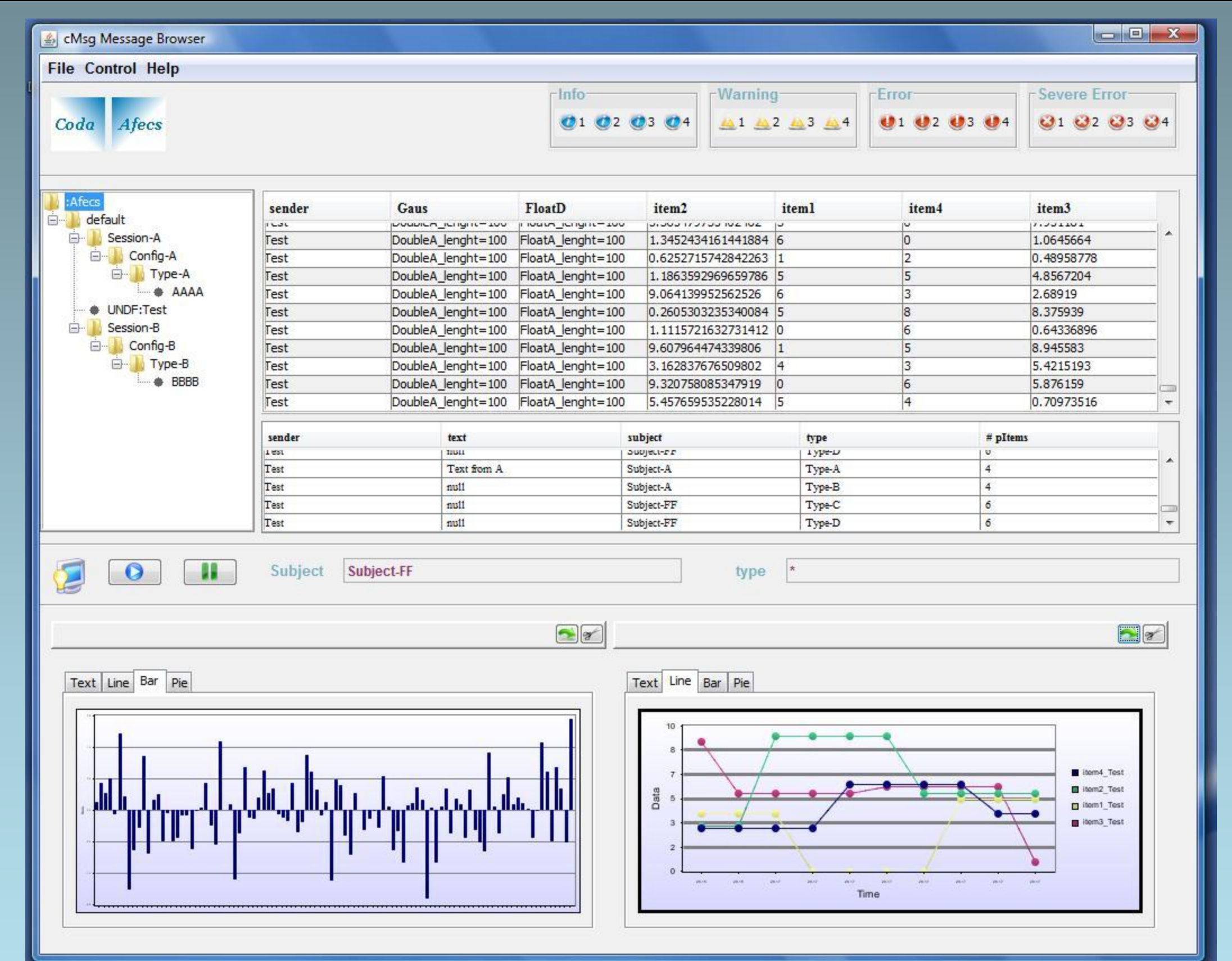
Some cMsg Features:

- Two subject fields (subject & type) used in publishing and subscribing.
- Messages hold all fundamental data types, arrays of these types, as well as cMsg messages and arrays of cMsg messages – 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:

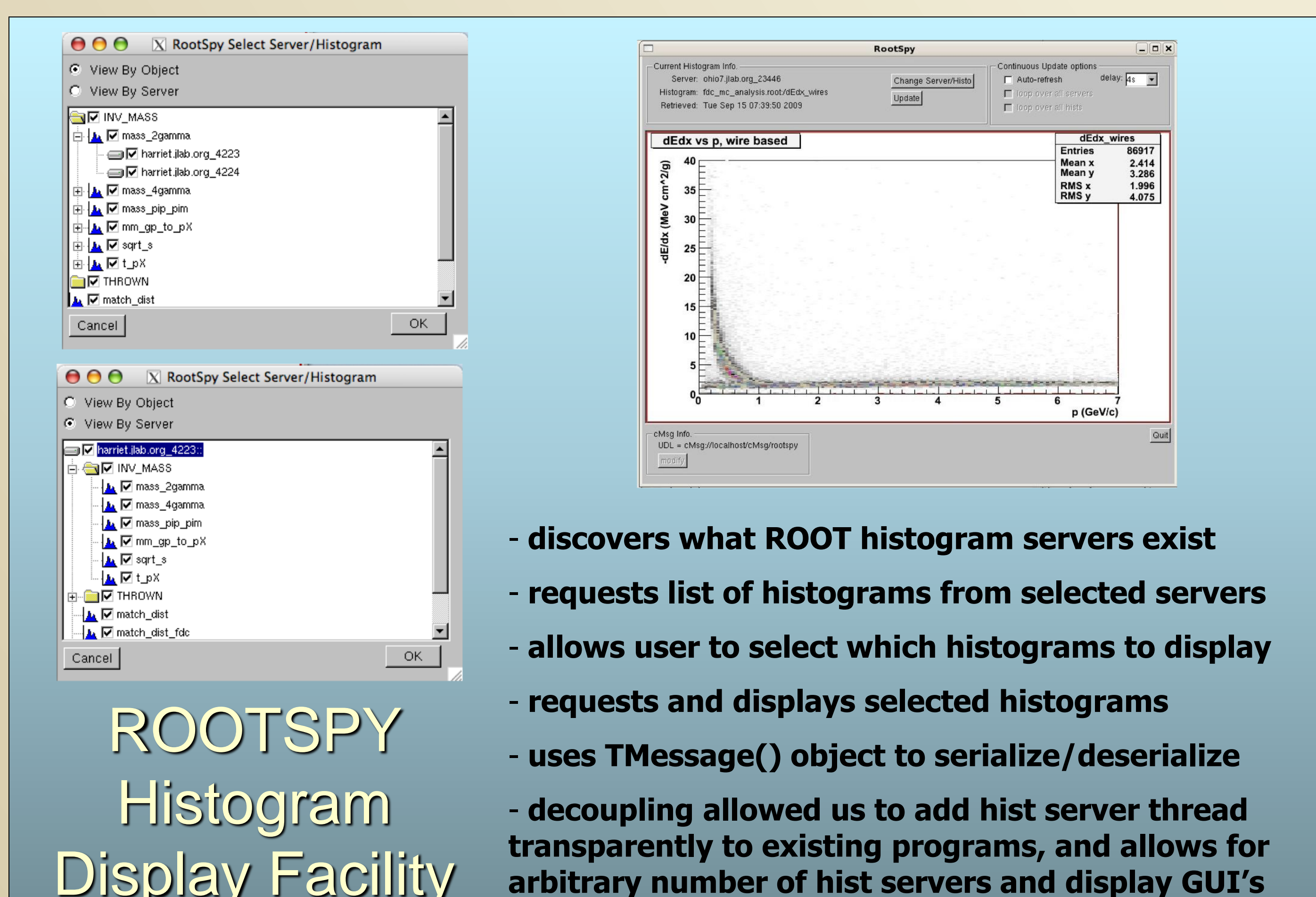
- Available in C, C++ and Java.
- Simple as possible; no IDL or stub generators needed.
- The API is narrow in that only basic messaging functionality is provided, i.e. there is only one type of message and one way to fill, publish, subscribe, and receive messages.
- Additional useful synchronous capabilities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

JLab DAQ Status Message Browser



daLogMsg browser receives status messages from components of a JLab DAQ system, stores them in a circular buffer, and displays messages based on filter criteria.

Decoupling enabled us to transparently add status message generation capability to existing components. and allows running of arbitrary number of independent browsers.



ROOTSPY Histogram Display Facility

- discovers what ROOT histogram servers exist
- requests list of histograms from selected servers
- allows user to select which histograms to display
- requests and displays selected histograms
- uses TMessage() object to serialize/deserialize
- decoupling allowed us to add hist server thread transparently to existing programs, and allows for arbitrary number of hist servers and display GUI's

Simple, Narrow Interface

-Java, C, C++

-There is only one type of message, and one way to create, fill, send, and receive messages asynchronously.

- Note that cMsg includes some synchronous capabilities.

Simplified List of cMsg API Functions

Function	Description
connect(UDL, myName)	Connect to a cMsg system (specified by the UDL parameter) for client myName
disconnect()	Disconnect from the cMsg system
send(msg)	Send a message asynchronously
flush(timeout)	Flush messages in output queue
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.