Accelerator Modelling and Message Logging with ZeroMQ

J. Chrin, M. Aibe, A. Rawat, Z. Wang
Paul Scherrer Institut
ZeroMQ @ ICALEPCS

- W. Sliwinski et al., “Middleware Proxy: A Request-driven Messaging Broker for High Volume Data Distribution”, in Proc. ICALEPCS’13
- A. Götz et al., “TANGO - Can ZeroMQ Replace CORBA?”, in Proc. ICALEPCS’13
- Y. Le Goc et al., “Prototype of a Simple ZeroMQ-based RPC in Replacement of CORBA in NOMAD”, in Proc. ICALEPCS’13
- T. Matsumoto et al., “Next-generation MADOCA for the SPRing-8 Control Framework?”, in Proc. ICALEPCS’13
- K. Rehlich, “Recent Hardware and Software Achievements for the European XFEL”, presented at ICALEPCS’13
- S.G. Ebner et al., “Data Streaming – Efficient Handling of Large Small (Detector) Data at the Paul Scherrer Institute”, presented at ICALEPCS’15, paper WED3O06
- S.G. Ebner et al., “SwissFEL Beam Synchronous Data Acquisition – A Sneek Peek under the Hood”, presented at ICALEPCS’15, paper MOPGF058
A. Dworak et al., “Middleware Trends and Market Leaders 2011”, in Proc. ICALEPCS’11

W. Sliwinski et al., “Middleware Proxy: A Request-driven Messaging Broker for High Volume Data Distribution”, in Proc. ICALEPCS’13

A. Götz et al., “TANGO - Can ZeroMQ Replace CORBA?”, in Proc. ICALEPCS’13

Y. Le Goc et al., “Prototype of a Simple ZeroMQ-based RPC in Replacement of CORBA in NOMAD”, in Proc. ICALEPCS’13

T. Matsumoto et al., “Next-generation MADOCA for the SPRing-8 Control Framework?”, in Proc. ICALEPCS’13

A. Yamashita and M. Kago, “A New Message-based Data Acquisition Sysyem for Accelerator Control?”, in Proc. ICALEPCS’13

K. Rehlich, “Recent Hardware and Software Achievements for the European XFEL”, presented at ICALEPCS’13

S.G. Ebner et al., “Data Streaming – Efficient Handling of Large Small (Detector) Data at the Paul Scherrer Institute”, presented at ICALEPCS’15, paper WED3O06

S.G. Ebner et al., “SwissFEL Beam Synchronous Data Acquisition – A Sneek Peek under the Hood”, presented at ICALEPCS’15, paper MOPGF058
ZeroMQ @ ICALEPCS

- W. Sliwinski et al., “Middleware Proxy: A Request-driven Messaging Broker for High Volume Data Distribution”, in *Proc. ICALEPCS’13*
- A. Götz et al., “TANGO - Can ZeroMQ Replace CORBA?”, in *Proc. ICALEPCS’13*
- Y. Le Goc et al., “Prototype of a Simple ZeroMQ-based RPC in Replacement of CORBA in NOMAD”, in *Proc. ICALEPCS’13*
- T. Matsumoto et al., “Next-generation MADOCA for the SPRing-8 Control Framework?”, in *Proc. ICALEPCS’13*
- A. Yamashita and M. Kago, “A New Message-based Data Acquisition System for Accelerator Control?”, in *Proc. ICALEPCS’13*
- K. Rehlich, “Recent Hardware and Software Achievements for the European XFEL”, presented at *ICALEPCS’13*
- S.G. Ebner et al., “Data Streaming – Efficient Handling of Large Small (Detector) Data at the Paul Scherrer Institute”, presented at *ICALEPCS’15*, paper WED3O06
- S.G. Ebner et al., “SwissFEL Beam Synchronous Data Acquisition - A Sneek Peek under the Hood”, presented at *ICALEPCS’15*, paper MOPGF058
ZeroMQ @ ICALEPCS

- W. Sliwinski et al., “Middleware Proxy: A Request-driven Messaging Broker for High Volume Data Distribution”, in Proc. ICALEPCS’13
- A. Götz et al., “TANGO - Can ZeroMQ Replace CORBA?”, in Proc. ICALEPCS’13
- Y. Le Goc et al., “Prototype of a Simple ZeroMQ-based RPC in Replacement of CORBA in NOMAD”, in Proc. ICALEPCS’13
- T. Matsumoto et al., “Next-generation MADOCA for the SPRing-8 Control Framework?”, in Proc. ICALEPCS’13
- K. Rehlich, “Recent Hardware and Software Achievements for the European XFEL”, presented at ICALEPCS’13
- S.G. Ebner et al., “Data Streaming – Efficient Handling of Large Small (Detector) Data at the Paul Scherrer Institute”, presented at ICALEPCS’15, paper WED3O06
- S.G. Ebner et al., “SwissFEL Beam Synchronous Data Acquisition – A Sneak Peek under the Hood”, presented at ICALEPCS’15, paper MOPGF058
ZeroMQ as middleware in HLA?

SLS 1999

SwissFEL 2016

SLS 1999: CORBA SOFTWARE BUS

SwissFEL 2016: ZeroMQ

Accelerator Model
Event Logger

cf. A. Götz et al., “TANGO - Can ZeroMQ Replace CORBA?”, in Proc. ICALEPCS’13
Distributed Computing with ZeroMQ
Distributed Computing with ZeroMQ

- ZeroMQ:
  - Lightweight
  - Socket-like
  - Asynchronous messaging library

- Provides for the transport of “raw message buffers” in a scalable computing environment

- The *Zero* in *ZeroMQ*:
  - Maximize performance by minimizing:
  - Latency, copying, necessity for brokers

  \[ \text{Lim} = 0 \]
✓ Several messaging patterns
Distributed Computing with ZeroMQ

- Several messaging patterns
- Unicast (tcp) and multicast transport layers
Distributed Computing with ZeroMQ

- Several messaging patterns
- Unicast (tcp) and multicast transport layers
- Patterns and transports can be used as building blocks to establish connections between processes, with/without brokers

Diagram showing various messaging patterns and transports.
Several messaging patterns
Unicast (tcp) and multicast transport layers
Patterns and transports can be used as building blocks to establish connections between processes, with/without brokers
Support for multipart messages
Several messaging patterns

✓ Unicast (tcp) and multicast transport layers

✓ Patterns and transports can be used as building blocks to establish connections between processes, with/without brokers

✓ Support for multipart messages

✓ Large user community provides support for several languages/platforms
Distributed Computing with ZeroMQ

- Several messaging patterns
- Unicast (tcp) and multicast transport layers
- Patterns and transports can be used as building blocks to establish connections between processes, with/without brokers
- Support for multipart messages
- Large user community provides support for several languages/platforms

😊 All available in a single library
Distributed Computing with ZeroMQ, but…

- No Name Service
  - translates logical addresses into bind/connect endpoints

- No Implementation Repository
  - For the activation and reactivation of servers

- No Support for Object Serialization
Distributed Computing with ZeroMQ, but…

- No Name Service
  - translates logical addresses into bind/connect endpoints

- No Implementation Repository
  - For the activation and reactivation of servers

- No Support for Object Serialization

What then are the remedies?
Name Service

- ØMQns:
  ZeroMQ Name Service in the pipeline

- May be developed from among ZeroMQ’s architectural patterns

- JSON Configuration File:
  "bind":
  "tcp://129.129.145.206:5559"
  "connect":
  "tcp://129.129.145.206:5560"
The Majordomo Protocol (MDP) defines a reliable service-oriented request-reply dialog between a set of client applications, a broker and a set of worker applications. MDP covers presence, heartbeating, and service-oriented request-reply processing.
Object Serialization

- Google Protocol Buffers
  - Binary encoding format
- MessagePack
  - Binary encoding in JSON
- ...
- Custom made
Interfacing Accelerator Models with ZeroMQ
Accelerator Models

- Allow developers to manipulate variables that determine particle beam dynamics in a simulated framework

- Most models were originally intended for use in isolation:
  - ASCII Input Files, lattice information and a set of directives to compute desired quantities
  - ASCII Output Files, post-processing analysis

- Certain models, code compiled into a shared object => accessibility from HLA can be anticipated
Advantages of the ZeroMQ Approach

- Lengthy and computer intensive initialization step need only be done once - where the long list of sequences that define the model are interpreted and loaded into memory.
- Numerous iterations (e.g. fitting procedures) can be undertaken without having to re-initialize with the same given model definition.
- Successive single tasks necessitate a newly created address space, a server-client configuration further gives confidence that the accelerator model is properly initialized on re-activation.
- Procedures verified offline can be engaged online.
- Incorporation into ZeroMQ => language neutrality.
  - Resulting data interfaces are structured, data exchange requires serialization.
Google Protocol Buffers: PyLiTrack

pylitrack.proto

```protobuf
syntax = "proto3";
package plt;
message pltInput {
  repeated string fileName=1;
  message ArrayElement {
    repeated float floatList=1;
    repeated int32 int32List=2;
    string strTag=3;
  }
} message pltOutput {
  message ArrayElement {
    repeated float floatList=1;
  }
  repeated ArrayElement BL=1;
  repeated floatList zpos=2;
  repeated floatList dE_E=3;
  repeated float Ebar=4;
  ...
  repeated float fcut=13;
}
```

$protoc pylitrack.proto

---cpp_out

```
pylitrack.pb.h

```class pltInput { ... }
class pltOutput { ... }
```
Advantages:

- IDL allows structured data schemas to be specified
- Backward compatibility, validation and extensibility
- Implemented in several languages

proto2 => proto3

- Simpler IDL structure, more accessible to a wider range of languages
- Support for programming idioms, any, map, oneof
- proto3 not backward compatible with proto2
- Migration from proto2 to proto3 was straightforward
PyLiTrack and MAD-X

- **PyLiTrack**
  - Python computations of LiTrack provides fast, two-dimensional, longitudinal single-bunch tracking

- **MAD-X**
  - Defacto standard for the computation of beam-optics parameters for a given accelerator lattice

- **Applied to SwissFEL Virtual Accelerator**
Message Logging
with ZeroMQ

ZeroMQ’s multipart message frames and the extended publish-subscribe pattern, respectively form the message envelope and communication layer.
Specific details that comprise the message content require careful consideration.

For each message a consistent set of data should be evident:
- Syslog protocol acts as a basis for deciding on the mandatory fields (TS: seconds + nanoseconds)
- Supplemented by a number of optional fields that are filled at the discretion of the user (e.g. error codes, and solutions!)
- The message content finalized in consultation with machine operation leaders.

Each message field is housed within a multipart message frame:
- Resulting “multipart message” effectively adds a coarsely formed structure to the single message that is delivered to the network.
- *No need to marshal/unmarshal the data (!)*
- ZeroMQ’s low-latency performance is not compromised.
DB writer maps multipart message frames to database columns on a near one-to-one basis.

Framework profits from ZeroMQ’s “zero-copy” capability in that buffers created by the publisher can be sent directly by the message.
zmsglog: A Library for Publishers

Inherited Class (Severity=Error)  Base Class

Python

```python
errorMsg = MsgLog.CyErrorMsg(
    " OrbitDisplay", "SFBD Group")
...
errorMsg.setMsg("freq. out-of-range")
errorMsg.send(__file__,__LINE__())
```

MATLAB

```matlab
msglog('setAppName','RF_Feedback')
...
msglog('setMsg','Invalid state')
msglog('send','fatal',dbstack())
```

- **zmsglog:**
  - Composed messages follow a predefined format, required entries being filled automatically by the provided API
  - Message ‘bursts’ are cached by the API on the publisher-side and only a summary of their occurrence need be sent over the network
Graphical User Interface

ICALEPCS’15, Melbourne, Australia, Oct. 2015, paper WEB3004
Graphical User Interface

Message retrieval from database
A number of properties of the message logger are kept separate from the application code and are managed through a JSON configuration file, enhancing flexibility and simplifying code maintenance.

- Setting ZeroMQ bind/connect endpoints
- High Water Mark Limit: Size of cached buffer for message queuing
- Declaring priority applications
- Displaying the display colours for the various message severity levels
Extended Publish Subscribe + ELK Stack

APP
PUB

APP
PUB

APP
PUB

Low-level FB
PUB

XSUB
Proxy
XPUB

SUB
GUI

SUB
DB Writer

SUB
Console

SUB
Logstash

Query + Data Visualization

Kibana

Elasticsearch (Apache Lucene)

DB

Thanks:
SG. Ebner
Aspects of ZeroMQ explored and usefulness within high-level applications recognized

A ZeroMQ Framework + Google Protocol Buffer has been implemented for accessing accelerator models from different languages

PUB-SUB pattern + multipart messaging framework → message logging and monitoring facility that displays live data in real time

The relative ease with which to employ the various patterns (ZeroMQ does all the hard work!) releases time and effort to focus on the specific goals at hand
THANK YOU FOR YOUR ATTENTION