ACCELERATOR MODELLING AND MESSAGE LOGGING WITH ZeroMQ

J. Chrin, M. Aiba, A. Rawat, Z. Wang, Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

ZeroMQ is an emerging message-oriented architecture that is being increasingly adopted in the software engineering of distributed control and data acquisition systems within the accelerator community. The rich array of built-in core messaging patterns may, however, be equally applied to within the domain of high-level applications where a seamless integration of accelerator models and message logging capabilities, respectively, serve to extend the effectiveness of beam dynamics applications and allow for their monitoring. Various advanced patterns that include intermediaries and proxies further provide for reliable service-oriented brokers, as may be required in real-world operations. A report on an investigation into ZeroMQ's suitability for integrating key distributed components into high-level applications, and the experience gained, are presented.

MOTIVATION

ZeroMQ [1] is an emerging message-oriented architecture that has already made a profound impact within the wider accelerator and experimental physics community. A favourable evaluation from among several contemporaries [2] has since seen it target existing CORBA (Common Object Request Broker Architecture) [3] systems at several facilities [4–6]. It is also being adopted in message-based data acquisition systems [7–10] and has provided the means by which beam synchronous data is retrieved in high-frequency pulsed accelerators [9, 11]. Its apparent identification as today's middleware of choice has triggered an interest for its applicability within the domain of high-level applications at SwissFEL, Switzerland's X-ray Free-Electron Laser Facility [12, 13]; where CORBA may once have been used at a previous facility for incorporating distributed components into beam dynamics applications [14, 15], ZeroMQ presents itself as a viable, state-of-the-art, alternative that deserves consideration. Of particular interest is the integration of accelerator models and message logging capabilities which respectively serve to extend the effectiveness of beam dynamics applications and allow for their monitoring.

DISTRIBUTED COMPUTING WITH ZeroMQ

ZeroMQ is a lightweight, socket-like, asynchronous messaging library that provides for the transport of raw message buffers in a flexible and scalable distributed computing environment. The idiosyncratic name lends itself to the project's ambition to reach maximal performance by minimizing latency, copying, and the necessity for brokers (i.e. their numbers approach the limit of *Zero*). A first investigation into the ZeroMQ library already reveals a number of compelling features:

- A rich array of messaging patterns, including the familiar request-reply, publish-subscribe and push-pull (pipeline) patterns, each of which defines a distinct network topology.
- The availability of both unicast (inproc, ipc, tcp) and multicast (pgm, epgm) transport layers.
- The ability to use these patterns and transports as building blocks to establish connections between processes, with or without intermediate brokers/proxies.
- Support for multipart messages, which allow multiple frames to be concatenated into a single message to be sent over the network.

That these features are all available in a *single* library is positively favourable. (By comparison, CORBA requires separate libraries for their event driven and other services.) Furthermore, an active ZeroMQ community provides support for numerous platforms and an increasing multitude of programming languages.

Despite these benefits, some important components, that are outside ZeroMQ's stated interest, still need to be catered for to attain a fully fledged distributed infrastructure:

- A Name Service that translates logical addresses into bind/connect endpoints.
- An Implementation Repository for the activation and re-activation of servers.
- Support for object serialization.

With these shortcomings identified, what then are the remedies? Although a name resolution service may be developed from among ZeroMQ's architectural patterns, for the present time, the use of configuration files for publicising tcp/ip addresses is manageable. Most CORBA developers will have become accustomed to an Implementation Repository that interacted with the server's Object Adapter Mediator to handle the administrative aspects of server (re-)activations [14]. A similar setup for ZeroMQ would ensure that applications are never starved of the services that they require. The lack of an interface to serialize structured data may, at first, appear as a glaring omission given that most use cases would require it. Fortunately, this situation is redeemed through third-party solutions which vary in form, complexity and performance, endowing developers with the prerogative to choose that which best suits their needs.

Having now gained an insight into ZeroMQ's capabilities, its applicability to within the beam dynamics environment is readily recognized. The request-reply messaging pattern,

ISBN 978-3-95450-148-9

coupled with a suitable protocol buffer for the serialization of complex data types, provide the ingredients for the implementation of a platform independent and language neutral, client-server framework, enabling the exchange of data between online accelerator models and distributed applications. The reactive publish-subscribe pattern, in preference, delivers an appropriate event-driven paradigm for constructing a software framework for the propagation of diagnostic messages to a central logging and monitoring facility. Here, ZeroMQ's multipart message protocol allows a single message type to be composed from several frames. The various advanced patterns that include intermediaries and proxies further provide for reliable service-oriented brokers, as may be required in real-world operations.

ACCELERATOR MODELLING

The use of accelerator models is an essential feature in both accelerator design and emulation, allowing developers to manipulate the variables that determine the dynamics of the particle beam in a simulated framework. For the most part, however, these models were originally intended for use in isolation. Typically, an ASCII input file, in the model-specific format, containing lattice information and a set of directives to compute the desired quantities is provided, and the resulting output is directed to files for post-processing analysis. For certain models, however, where the compilation of the code into a shared object is a workable prospect, then their accessibility from a high-level language may be anticipated. Procedures that have been verified *offline* can then be effectively engaged for the optimization of the accelerator *online*. Methods may be executed to retrieve beam dynamics (e.g. linear optics) parameters for a given setting of the accelerator, for example, and model based corrections subsequently applied [16, 17].

The feasibility to expose such accelerator models to beam dynamics applications in a language-neutral manner in itself provides strong motivation for their incorporation into the ZeroMQ messaging architecture. The resulting data interfaces are invariably structured and any meaningful data exchange requires serialization, however.

Serialization with Google Protocol Buffers

While a number of options exist for data serialization, the Google Protocol Buffers [18] provide a binary encoding format that has a number of recognized advantages. An interface definition language allows structured data schemas to be specified from numbered fields affixed with well-defined keywords and data types. These ensure backward compatibility, validation and extensibility. The Protocol Buffers are implemented in several languages easing interoperability between applications from different domains. In the course of this work, Google introduced Protocol Buffers language version 3, `proto3`. The new version has a simplified interface definition language structure, making it more accessible to a broader range of programming languages. Several new features have also been added to support nomenclatures such as the `Any` type, the associative `map` and `oneof` keyword. A future release of `proto3` is to provide a well-defined encoding in JSON (JavaScript Notation Object) [19] as an alternative to binary proto encoding for data consumption by e.g. web browsers. The migration from `proto2` to `proto3` for our limited set of `proto` files was a straightforward endeavour; `proto3` is not, however, backward compatible with `proto2` although the latter syntax may still be incorporated into the former if so declared.

PyLiTrack

The Python computation of LiTrack [20] provides fast, two-dimensional, longitudinal single-bunch tracking. The relatively uncomplicated interface provides a useful test case for integrating an accelerator model into the ZeroMQ architecture through the Google Protocol Buffers. The data schema (`proto`) file, shown in Fig. 1, is, consequently, comparatively light. The `protoc` binary compiler automatically generates stub class files that are used for data encoding and parsing. Input arguments to the line command determine what languages are provided for. In the present setup, PyLiTrack runs as a server and clients connect using ZeroMQ's request-reply pattern. In this way, the Pythonic tracking code is made available to non-Pythonic applications.

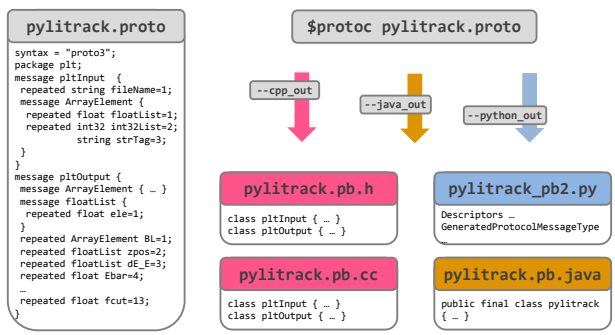

Figure 1: The generation of stub classes, for various languages, from the PyLiTrack proto file.

MAD-X

The MAD-X (Methodical Accelerator Design) simulation code [21] is regarded as a defacto standard for the computation of beam-optics parameters for a given accelerator lattice. In addition to the stand-alone executable, it is available as a C++ library with bindings for Python [22]. An identical procedure for data serialization is applied here. The corresponding proto file exposes the so-called Twiss tables which parameterise the beam ellipse in phase space. One distinct advantage of using the library as opposed to running the stand-alone executable is that the lengthy and computer intensive initialization step (where the long list of sequences that define the model are interpreted) need only be loaded into memory once. Numerous iterations, as required in fitting procedures for instance, can be undertaken without having to continually re-initialize with the same, given model definition. At the same time, in cases where

ISBN 978-3-95450-148-9

successive single tasks necessitate a newly created address space, a server-client configuration further gives confidence that the accelerator model is always properly initialized on (re-)activation. One variant applied sees a Python client communicating with a Python MAD-X server, using ZeroMQ as the transport layer, but preferring Python's own Pickle module for the serialization. The activation of the server is enacted through the use of Python scripts. While this may suffice in the short term, the call for an Implementation Repository for handling the (re-)activation of ZeroMQ servers becomes apparent!

Next Steps

The accelerator models so far considered represent the most use-cases for online modelling at SwissFEL. While a proof-of-principle has been demonstrated, a client-side Application Programming Interface (API) for PyLiTrack and MAD-X that hides the ZeroMQ and serialization implementation details is to be finalized. In the absence of an Implementation Repository, advanced patterns that provide for reliable service-oriented brokers may also be preferred to the present synchronous interactions.

MESSAGE LOGGING AND MONITORING

A messaging logging and monitoring facility, henceforth referred to as the message logger, has been developed to allow applications to report (publish) their diagnostic and information data to interested subscribers. These include a database (Oracle [23]) writer process for the storage of messages and a Graphical User Interface (GUI) for the display of messages both in real-time and offline through database retrieval operations. ZeroMQ's multipart frames and the extended publish-subscribe pattern, respectively form the message envelope and communication layer.

Multipart Messages

ZeroMQ allows messages to be assembled from individual frames arising from different sources. The resulting "multipart message" effectively adds a coarsely formed structure to the single message that is delivered to the network. The need for marshalling/unmarshalling the data is alleviated and ZeroMQ's low-latency performance is not compromised. The leading frame of the multipart message also serves as a "topic" in ZeroMQ's publish/subscribe architecture to set filters that allow only events of interest to propagate to the subscriber.

Extended Publish and Subscribe

The extended publish-subscribe model, shown in Fig. 2, is the pattern adopted for broadcasting messages to interested clients. The proxy (or broker), which lies between the publishers (PUB) and subscribers (SUB), provides the solution to the so-called "dynamic discovery problem" by binding XSUB and XPUB sockets (which expose subscriptions as special messages) to the advertised IP addresses and ports [24]. Publishers and subscribers need only connect to the XSUB and XPUB sockets of the proxy, rather than to one another. The proxy then takes charge of forwarding messages to subscribers. In this way, publishers, i.e. high-level applications, may be easily hooked in to the system and have their messages written to the database, viewed from within a GUI or received by any other subscriber, e.g. console, that can be readily added to the network.

ZeroMQ's response to continuous messages from multiple publishers is for the proxy to collect the messages evenly from among the publishers ("fair-queued"). In the event of message overload, where publishers outpace consumers, messages are queued on the publisher's side with the size of the cached buffer determined by a configurable "high-water mark" limit. The framework also profits from ZeroMQ's "zero-copy" capability in that buffers created by the publisher can be sent directly by the message. The data does still need to be written into the application buffers at the receiving end, however.

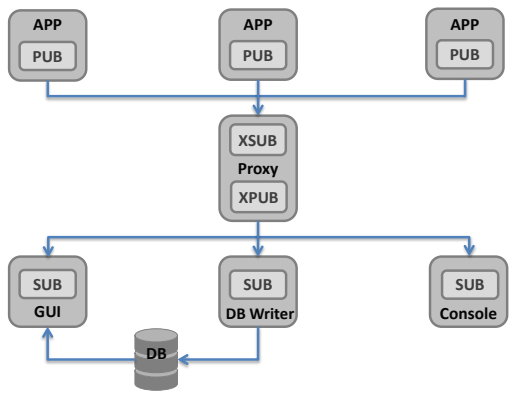

Figure 2: The extended publish-subscribe pattern as applied to the message logger.

Message Content

The specific details that comprise the message content require careful consideration. More information within a message may prove to be synonymous with a better reporting ability, but should nevertheless be balanced against network traffic and storage capacity interests. Naturally, for each message (or event) a consistent set of data should be evident. The established `syslog` protocol [25] acts as a basis for deciding on the mandatory fields. These are supplemented by a number of optional fields that are filled at the discretion of the user. The message content was finalized in consultation with machine operation leaders [26]. It is, however, the provision of the user to supply meaningful and helpful messages, that also propose solutions (for which an action field is also provided) that ultimately hastens a return to normal operation.

Each message field is housed within a multipart message frame, simplifying the data unpacking process. The database writer, for instance, maps frames directly onto a corresponding database schema, as indicated in Fig. 3.

ISBN 978-3-95450-148-9

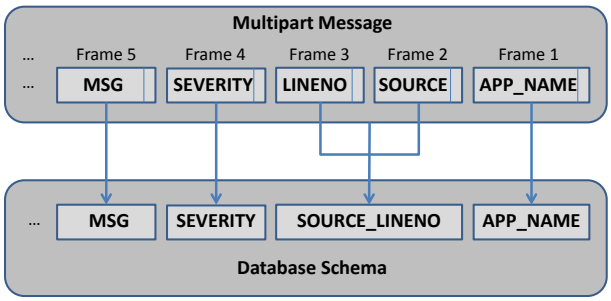

Figure 3: The database writer maps multipart message frames to database columns on a near one-to-one basis.

A Library for Publishers

To facilitate application developers with the reporting of messages, a dedicated library, `zmsglog`, was developed that hides the ZeroMQ implementation details and provides an easy-to-use interface. The composed messages follow a predefined format with required entries being filled automatically by the provided API, with the obvious exception of the user supplied message itself. An important feature of the `zmsglog` is its ability to efficiently handle bursts of repeated error messages. Such message bursts are cached on the publisher's side and only a summary of their occurrence need be sent over the network, thereby minimizing the network traffic. The message logging facility has been successfully put through high-volume data stress tests (without caching of repeated methods). Listings 1 and 2 show a minimal implementation from Python, using an inherited class, and MATLAB, using the base class, respectively.

Listing 1: Python API for `zmsglog`

```
warnMsg = MsgLog.CyWarnMsg("OrbitDisplay")
warnMsg.setMsg("RMS outside operating limits")
warnMsg.send(__file__, __LINE__())
```

Listing 2: MATLAB API for `zmsglog`

```
msglog('setAppName','3DScan')
msglog('setMsg','setVal outside hardware limit')
msglog('send','error',dbstack())
```

A GUI for Subscribers

A GUI to the message logger has been developed in Python/Qt (Fig. 4). It is able to display live messages in real-time, with a dedicated window pane configured to monitor priority, i.e. machine-critical, applications and fatal messages. A third window provides the user with an interface for database retrieval operations, with ample sorting and filtering possibilities. A database polling mechanism may also be activated if preferred. The GUI may also be activated for the purpose of a single, specific publisher by supplying the application name as a command argument. The application name acts as the topic in the multipart message by which the GUI (subscriber) sets a filter in order to select only the given application's messages.

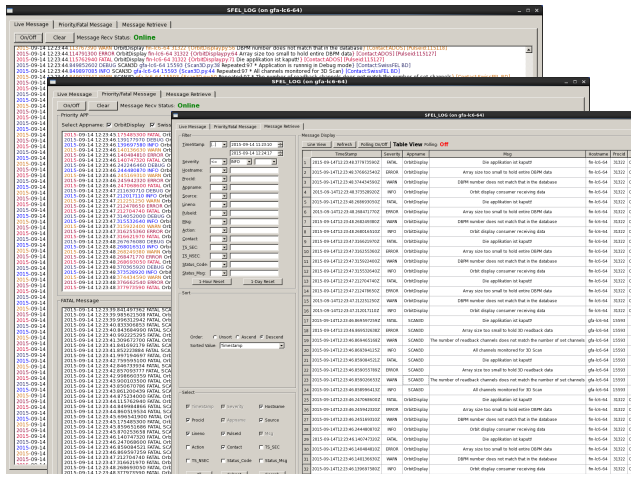

Figure 4: A GUI for viewing messages: in real-time (back, middle), from priority applications (middle top), of highest severity (middle bottom), and from the database (front).

Configuration File

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. The possibilities range from setting the ZeroMQ bind/connect endpoints, the high water mark limit, announcing priority applications, to selecting the display colours for the various message severity levels.

Next Steps

The present framework is well matched for the anticipated demand from high-level applications during the first, two-year, SwissFEL beam commissioning phase [13]. Nevertheless, since interest has surfaced to provide `zmsglog` to low-level (e.g. feedback) systems, and with it the potential for high-volume data flows, the scalability of the logging and data mining components for problem tracing [27] requires further consideration. To this end, a number of open-source solutions, based on Apache Lucene [28], have been identified [29]. Among these is the ELK stack [30] – Elasticsearch, Logstash, and Kirbana – which provide a complete framework for data redirection, storage, analysis and visualization. The Logstash API also provides a data pipeline for receiving ZeroMQ messages that would allow it to plug in effortlessly to the architecture presented in Fig. 2, as a subscriber to the proxy.

SUMMARY

Various facets of the ZeroMQ asynchronous messaging library have been explored and their usefulness to within the domain of high-level applications has been recognized. In particular, a framework based on the request-reply pattern, coupled with the Google Protocol Buffers for the serialization of data, has been implemented for accessing accelerator models from different programming languages. The publish-subscribe pattern together with ZeroMQ's multi-

ISBN 978-3-95450-148-9

part messaging framework have formed the ingredients for a message logging and monitoring facility that displays live data in real-time. A notable feature throughout has been the relative ease with which it has proved to employ the various ZeroMQ messaging patterns, thereby releasing time and effort to focus on the specific goals at hand.

REFERENCES

[1] ZeroMQ, http://zeromq.org/.

[2] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, and M. Sobczak, "Middleware Trends and Market Leaders 2011", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper FRBHMULT05, pp. 1334–1337.

[3] CORBA (Common Object Request Broker Architecrure), http://www.omg.org/.

[4] W. Sliwinski, I. Yastrebov, and A. Dworak, "Middleware Proxy: A Request-driven Messaging Broker for High Volume Data Distribution", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper TUCOCB02, pp. 948–951.

[5] A. Götz, E. Taurel, P. Verdier, and G. Abeille, "TANGO - Can ZMQ Replace CORBA?", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper TUCOCB07, pp. 964–968.

[6] Y. Le Goc *et al.*, "Prototype of a Simple ZeroMQ-based RPC in Replacement of CORBA in NOMAD", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper TUPPC042, pp. 654–657.

[7] T. Matsumoto, Y. Furukawa, and M. Ishii, "Next-generation MADOCA for the SPring-8 Control Framework", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper TUCOCB01, pp. 944–947.

[8] A. Yamashita and M. Kago, "A New Message-based Data Acquisition System for Accelerator Control", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper MOPPC130, pp. 413–416.

[9] K. Rehlich, "Recent Hardware and Software Achievements for the European XFEL", presented at the 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), San Francisco, CA, USA, Oct. 2013, paper THCOBB02.

[10] S.G. Ebner, H.R. Billich, H. Brands, E.H. Panepucci and L. Sala, "Data Streaming - Efficient Handling of Large and Small (Detector) Data at the Paul Scherrer Institute", presented at the 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, Oct. 2015, paper WED3O06, these proceedings.

[11] S.G. Ebner, H. Brands, B. Kalantari, F. Märki, and L. Sala, "SwissFEL Beam Synchronous Data Acquisition - A Sneak Peek under the Hood", presented at the 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, Oct. 2015, paper MOPGF059, these proceedings.

[12] "SwissFEL Conceptual Design Report", R. Ganter, Ed. PSI, Villigen, Switzerland, Rep. 10-04, Version Apr. 2012.

[13] T. Schietinger, "Beam Commissioning Plan for the SwissFEL Hard X-ray Facility", presented at the 37th Int. Free-Electron Laser Conf. (FEL'15), Daejeon, Korea, Aug. 2015, paper MOP017.

[14] M. Böge and J. Chrin, "On the Use of CORBA in High Level Software Applications at the SLS", in *Proc. 8th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'01)*, San Jose, CA, USA, Oct. 2001, paper THAT002, pp. 430–432.

[15] M. Böge and J. Chrin, "Developments to the SLS CORBA Framework for High Level Software Applications", in *Proc. 10th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'05)*, Geneva, Switzerland, Oct. 2005, paper WE4A.1-5O.

[16] M. Böge, J. Chrin, M. Muñoz, and A. Streun, "Commissioning of the SLS using CORBA Based Beam Dynamics Applications", in *Proc. 2001 Particle Accelerator Conf. (PAC'01)*, Chicago, IL, USA, Jun. 2001, paper TOPB012, pp. 292–294.

[17] M. Böge and J. Chrin, "Integrating Control Systems to Beam Dynamics Applications with CORBA", in *Proc. 2003 Particle Accelerator Conf. (PAC'03)*, Portland, OR, USA, May 2003, paper TOPB010, pp. 291–293.

[18] Google Protocol Buffers, https://developers.google.com/protocol-buffers/.

[19] JSON (JavaScript Notation Object), http://json.org/.

[20] K.L.F. Bane and P. Emma, "LiTrack: A Fast Longitudinal Phase Space Tracking Code with Grapghical User Interface ", in *Proc. 2005 Particle Accelerator Conf. (PAC'05)*, Knoxville, TN, USA, May 2005, paper FPAT091, pp. 4266–4268.

[21] MAD-X, http://madx.web.cern.ch/madx/.

[22] K. Fuchsberger and Y. Inntjore Levinsen, "PyMad – Integration of MadX in Python", in *2nd Int. Particle Accelerator Conf. (IPAC'11)*, San Sebastián, Spain, Sep. 2011, paper WEPC119, pp. 2289–2291.

[23] Oracle, http://www.oracle.com/.

[24] P. Hintjens, "The Dynamic Discovery Problem", in *ZeroMQ*, Sebastopol, CA, USA: O'Reilly Media, Inc., 2013, pp. 45–47.

[25] R. Gerhards, "The Syslog Protocol", RFC 5424, Mar. 2009, doi: http://dx.doi.org/10.17487/RFC5424

[26] J. Chrin, A. Lüdeke, and D. Voulot, "Message Logging Specifications for SwissFEL Applications", PSI, Villigen, Switzerland, SwissFEL Document ID. FEL-CJ84-001-01, Sep. 2015.

[27] F. Ehm and A. Dworak, "A Remote Tracing Facility for Distributed Systems", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEPKS024, pp. 840–843.

[28] Apache Lucene, http://lucene.apache.org/.

[29] S.G. Ebner, private communication.

[30] Elasticsearch, https://www.elastic.co/.

ISBN 978-3-95450-148-9