TANGO – CAN ZMQ REPLACE CORBA?

A.Götz, E.Taurel, P.Verdier, ESRF, Grenoble, France
G.Abeille, SOLEIL, Gif sur Yvette, France

Abstract

TANGO [1] is a modern distributed device control system toolkit used to control synchrotrons, lasers and a wide variety of equipment for doing physics experiments. The performance of the network protocol is a key component of TANGO. TANGO is based on the omniORB (for C++) and Jacorb (for Java) implementations of CORBA. CORBA offers an interface definition language with mappings to multiple programming languages, an efficient binary protocol, a data representation layer, and multiple services. In recent years a new series of binary protocols based on AMQP have emerged from the high frequency stock market trading business. At about the same time a protocol called ZMQ [2] was open sourced in 2008. In 2011 the TANGO community decided to take advantage of ZMQ. In 2012 the kernel developers successfully replaced the CORBA Notification Service with ZMQ in TANGO V8. The first part of this paper will present the software design, the issues encountered and the resulting improvements in performance. The second part of this paper will present a study of how ZMQ could replace CORBA completely in TANGO.

TANGO TOOLKIT

TANGO is a toolkit for building simple and complex control systems. TANGO has been designed to manage complexity simply. It does this by implementing all control objects as Devices or hierarchies of Devices. Devices are local or network objects which implement device specific behaviour and follow the same model. All TANGO Devices have state, a state machine, commands, attributes, network access, polling, threading, security, database support, persistence, etc. TANGO implements synchronous, asynchronous and event driven local and network communications as part of its library. The TANGO device model has been successfully tested and used by many programmers to implement device access for hundreds of different devices in tens of institutes.

TANGO comes as a library and with a complete set of tools for monitoring, configuring and managing a TANGO based control system. TANGO supports 3 programming languages fully (C++, Java and Python) and a number of languages are interfaced as clients (Labview, Matlab, and IgorPro)

TANGO is open source software available free of charge. The source code is on stored on two Sourceforge sites [3, 4].

TANGO AND CORBA

CORBA [5] was a de facto standard in the 90s for distributed communications used in many control systems. The CORBA specification is managed by the Object Management Group. A number of commercial and open-source implementations exist of all or a subset of the specifications.

TANGO has used CORBA for its communication layer from the very beginning. The object oriented model of CORBA partially inspired the TANGO Device Model. The availability of open source highly efficient CORBA implementations like omniORB [6] in C++ and JacORB [7] in Java have enabled TANGO to implement a high performance protocol.

TANGO has been designed from the beginning to offer the services a control system needs without relying completely on CORBA. For this reason TANGO uses only a minimum set of essential features which are available in all CORBA implementations. TANGO hides CORBA as much as possible from the users and developers. This makes it possible to replace CORBA in TANGO.

What's Right with CORBA?

CORBA has many useful features which are required by most distributed systems. TANGO uses the following CORBA features:

- *IDL* - Interface Definition Language was used to define all TANGO data types and the Device class interface
- *languages* - bindings to C++ and Java used for writing servers and clients
- *ORB* - Object Request Broker for dispatching network requests to Devices
- *IOR* - to identifiy objects and stringify object references to the database
- *corbaloc* - connect to the database name service of TANGO
- *interceptors* - to intercept all calls to Device and add logging
- *invocation* - synchronous calls with timeout is the basic call for executing commands and reading/writing attributes
- *DII* - asynchronous calls use the Dynamic Invocation Interface
- *collocation* - in same process calls are used for transparently co-locating Devices in the same process
- *CDR* - essential for efficient data marshalling and unmarshalling of TANGO data types
- *threading* - multi-threading model of omniORB is essential for implementing parallel client access efficiently
- *performance* - the binary protocol of CORBA (IIOP) and the efficient implementations were essential for high throughput performance required by a modern control system

ISBN 978-3-95450-139-7

What's Wrong with CORBA?

Despite CORBA's useful features and many success stories it has failed to deliver on its promise of becoming the de facto standard for network computing. The reason for this has been discussed at length on the net and various publications [8] have documented the shortcomings of CORBA in detail. The main problem is that CORBA has been designed by a committee without a reference implementation. This has resulted in a bloated specification. There is a lack of open source implementations of the various CORBA services (Naming, Notification, Trading, Transaction, Persistence). Even if not all of these services are adapted for control systems, those which are (e.g. Notification) lack a well maintained open source implementation.

How TANGO Fixes CORBA

TANGO has worked around the shortcomings of CORBA by implementing the missing features plus some additional features as part of the TANGO library. The following services are implemented by TANGO:

- *naming* - based on a naming convention ($\langle Domain\rangle/\langle Family\rangle/\langle Member\rangle$), a database and a database server
- *versioning* - uses inheritance to implement full compatibility between all versions of TANGO
- *connection* - management so that connections are stateless and reconnect automatically
- *persistence* - memorised attributes are persisted via the database
- *security* - access to devices in read and write can be filtered based on the process UID, HOST and NETWORK
- *startup* - device distribution is configured in the database, automatic startup managed by Starter
- *logging* - built-in distributed logging service
- *polling* - device polling built-in, automatically triggers alarms and events
- *caching* - polling thread caches read values
- *data types* - large number ($<$20) of generic data types implemented

With these features as part of TANGO the shortcomings of CORBA are **NOT** an issue for TANGO. It is important to stress that the choice of *CORBA* as a *protocol* has been **successful** for TANGO. However CORBA's problems are real for systems adopting CORBA directly instead of using a toolkit like TANGO. This has lead to CORBA falling out of favour with software developers. For this reason we think the long term availability of CORBA can become an issue.

Over the last couple of years new network communication libraries have appeared which offer new paradigms for communication. They also implement new features not possible or difficult to solve using CORBA. Since TANGO has been designed from the beginning to be independent of the network protocol it is natural to reflect on what is required to replace CORBA with another communication protocol like ZMQ and what the advantages would be.

WHY ZMQ?

ZMQ is a rising star in the world of network communications. ZMQ is an embeddable networking library that acts like a concurrency framework. It provides sockets that carry whole messages across various transports like inprocess, inter- process, TCP and multicast. Sockets can be connected N-to-N with patterns like fanout, pub-sub, task distribution and request-reply. Its asynchronous I/O model is ideal for scalable multicore applications, built as asynchronous message-processing tasks. It has many language bindings and runs on most operating systems. ZMQ is released under the LGPL open source licence by iMatix.

Due to its efficient implementation of parallelism and batching ZMQ has been benchmarked (cf. [9]) by the authors and a number of studies (see [10] for example) to perform better than other middleware including CORBA.

TANGO EVENTS ON ZMQ

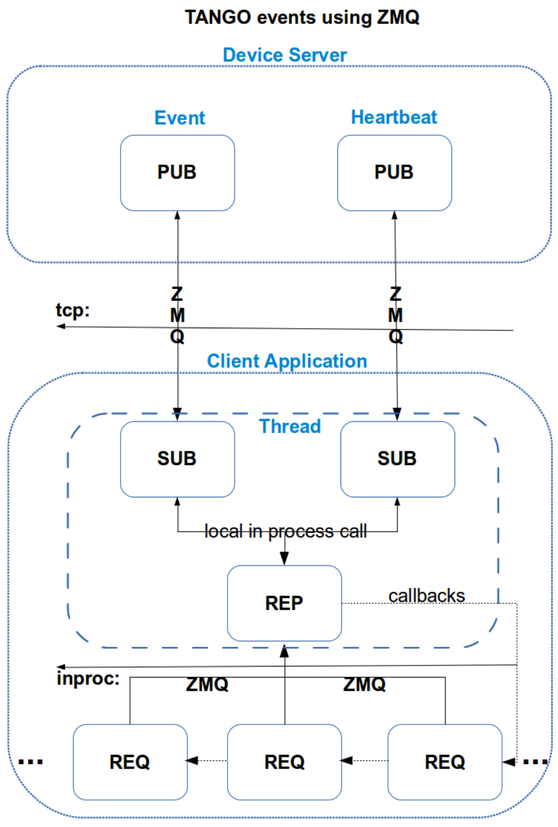

Figure 1: TANGO events with ZMQ.

The TANGO kernel team has successfully ported the TANGO events from CORBA Notification service to ZMQ. The motivations for replacing the CORBA Notification service were - (1) omniNotify (the Notification service

ISBN 978-3-95450-139-7

TANGO uses) is a dead project (bugs are not fixed, packages for new versions of operating systems are not available), and (2) the quest for higher performance, and (3) the decision to simplify the architecture by suppressing the need for a broker between the client and server. The project started in 2011 and have been presented at the last ICALEPCS conference (see [11]). The architecture of the TANGO event service based on ZMQ is presented in Fig. 1.

C++ Implementation

TANGO events use 2 types of ZMQ sockets to send events to clients. The PUB-SUB and REP-REQ socket pairs. The PUB-SUB socket is used to send events from device servers to clients over tcp running on the same or different hosts. Two PUB-SUB sockets are created - one for sending the event itself using CDR for encoding the data, and one for sending a heartbeat to inform the client that the server is still alive. The use of CDR has meant code for marshalling and unmarshalling data types did not have to change. On the client side all event data are received and then dispatched via callbacks in the same thread.

Client threads subscribe to events via an *inproc* socket to the TANGO event thread. Using a single thread to dispatching events in the client means the client could wait behind a slow callback. Experience shows this is not a problem and is even a requirement for graphical applications. If ever this becomes an issue the model can be extended to dispatch events via a thread pool.

Being built on the C++ implementation, the TANGO Python binding (PyTango) profits directly from the TANGO events using ZMQ in C++ by simply recompiling.

Java Implementation

The Java implementation of TANGO events with ZMQ follows the C++ implementation with the main difference being the ZMQ library. Two versions of the library exist - the JNI version which calls the C library version, and the pure Java version (jeromq). Both versions have been tested and are in production. The pure Java version has the advantage that it is easier to use on mobile platforms. This is the first time the TANGO Java servers implement events. The code has profited from the recent upgrade of the TANGO Java device server library [12].

Multicasting

ZMQ supports multicasting as an option. It uses the openpgm [13] implementation of Practical General Multicast (PGM). Multicasting is ideal for sending data to many clients at high speed efficiently. The server sends the event once to the network. The event data is distributed by the network switches. PGM is a reliable protocol therefore ensuring a reliable delivery of events.

The TANGO implementation of multicast events is complicated by the fact that openpgm does not implement multicasting on the same host. TANGO detects if the client is on the same host and if so, it switches to unicast. Multicast events have to be configured via the TANGO database. The rate and size of memory caching buffers have to be specified as properties in the TANGO database. Multicasting is tested in TANGO but not used by any sites yet. Typical use cases for multicast events are (1) a large number of clients interested in the same event e.g. beam current, and (2) a lot of data being sent at high speed e.g. images. Multicasting is only implemented in C++ (and Python) currently.

See Fig. 2 for a comparison between TANGO scalar events with ZMQ versus CORBA.

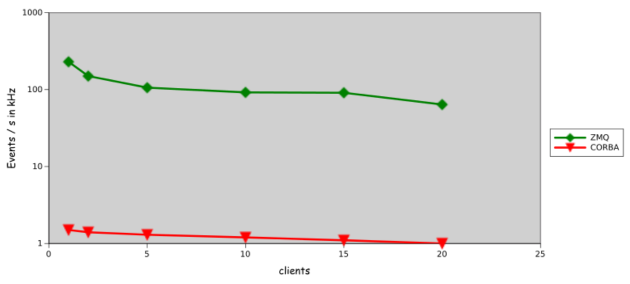

Figure 2: TANGO scalar events with ZMQ vs CORBA.

Table 1: TANGO Events With ZMQ Performance on Xeon at 3 GHz

server to client	**data**	**throughput**
Java to Java	8 bytes	14.2 kHz
C++ to C++	8 bytes	230 kHz
C++ to C++	1 kbyte	122 kHz
C++ to C++	1 Mbyte	1.2 kHz

High Water Mark

The High Water Mark (HWM) is a limit in ZMQ which specifies the number of messages to retain in the local buffer before taking special action like dropping them. The philosophy of ZMQ is to continue working even if the client does not treat messages as fast as they are received. In TANGO this limit can be set via an api call or via an environment variable. Two values are managed - one for the device server and one for the client. The default size is 100. If the HWM limit is reached then messages are dropped and the client or server are informed via an exception. The limit is most critical for clients. For clients that want to ensure receiving a large number of messages it is necessary to set HWM to a large value e.g. 1000000.

REPLACING CORBA COMPLETELY

How to replace CORBA completely in TANGO with ZMQ? The first issue is how to choose a ZMQ socket pattern which ensures high performance for many clients and servers. In Fig. 3 we propose an architecture based on ZMQ ROUTER pattern to replace the current ORB based architecture of TANGO device servers. The main features of this proposal are a synchronous and asynchronous socket which send requests via a ROUTER socket to device

ISBN 978-3-95450-139-7

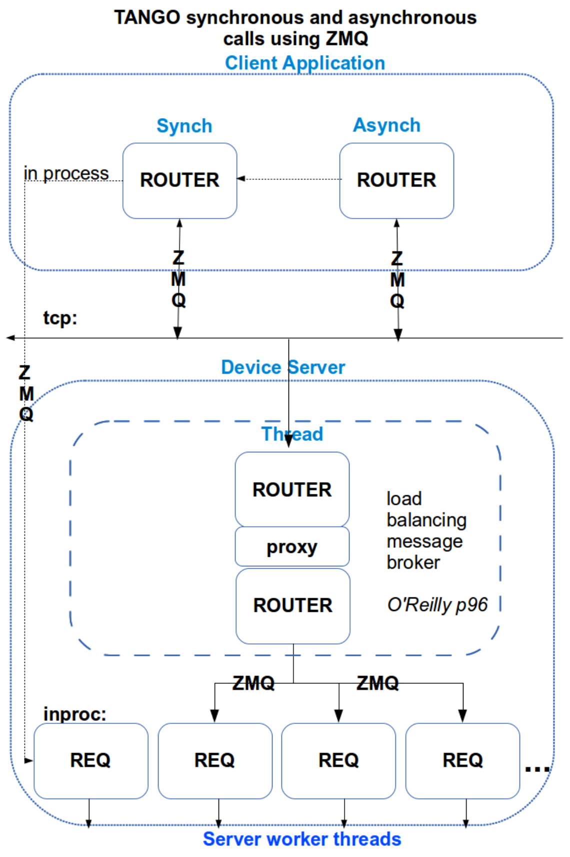

Figure 3: TANGO a/synchronous calls with ZMQ.

servers. Each device server load balances the incoming requests by dispatching them to a pool of REQ sockets. Each REQ socket has a worker thread for executing the request. This pattern resembles the multi-threading model of omniORB in C++ except that the pool of worker threads have to be created and managed by TANGO. Once the request is in a worker thread the execution can be done using the existing code in the TANGO library.

Another issue which has to treated manually is co-located calls i.e. calls to devices in the same process (where a server is also a client of one of its own devices e.g. a serial line). This is a well used pattern in TANGO and needs to be treated in ZMQ by detecting co-location and then issueing a call to an *inproc:* socket. The request is then dispatched to a worker thread as if the request came from the network.

Other major issues are how to replace the CORBA features TANGO relies on. Here is how we propose to replace the CORBA features in TANGO:

- *IDL* - the IDL generated code will be replaced by handwritten code, this is not a major issue because TANGO only has ONE interface defined in 4 versions,
- *languages* - bindings exist to all languages used for writing servers and clients (C++, Java and Python)
- *ORB* - a manually coded broker will dispatch network requests to Devices, this requires the Device id to be sent with each request
- *IOR* - a unique identifier will have to be re-invented e.g. ZOR://Host:Port/Server/Device
- *corbaloc* - the database network address will be replaced by a series of ZMQ tcp address to allow redundancy
- *interceptors* - to be coded if still needed in the TANGO library
- *invocation* - synchronous and asynchronous calls with timeout will be replaced by a ROUTER-to-ROUTER ZMQ pattern (see above)
- *co-location* - needs to be detected manually and will be replaced with an *inproc:* call
- *CDR* - a new library for serialising data types will be needed (see below)
- *threading* - ZMQ provides multi-threading for receiving and dispatching requests, a thread pool of worker threads will be implemented in the server
- *performance* - not a problem, ZMQ is a high throughput protocol which is more efficient than CORBA's IIOP

Serialisation

An important issue is how to serialise and de-serialise TANGO data types to and from the network. Over 20 data types are implemented in TANGO today. CORBA's CDR serialisation is used to encode and decode them. Although this works very well and has been used successfully for sending TANGO events via ZMQ, CDR has to be replaced if we want to remove all dependencies on CORBA.

One possibility is to use and existing library like Google's *protobuf* library or Capn Proto (faster version of protobuf). It has the advantage of providing an IDL and generates code for C++ and Java. There are pros and cons to using an external library.

The alternative is to implement a manually coded solution for serialisation. Even if the number of data types to support is significant they are stable. The code has to be written (or generated) for C++ and Java and optimised by hand. We reckon this will be one of the more time consuming tasks to replace CORBA.

Compatibility

A major requirement of the TANGO community is to maintain compatibility with existing code. This means device servers and clients do not have to be changed, only recompiled. In addition it must be possible to introduce servers and clients using the new protocol gradually in a running TANGO system without recompiling everything. Forwards and backwards compatibility requires having both protocols (CORBA and ZMQ) in the library and detecting at runtime which one to use as was done for the TANGO events on ZMQ. This requirement is the key to enable existing sites to adopt a new version of TANGO based on a new protocol

Implementation

The implementation of a pure ZMQ-based TANGO protocol is a non-trivial task. We estimate the work to be about 12 months of coding and 12 months of testing for C++ and about half as much for Java because it will follow the C++ implementation. Work has started on a prototype on Github. So far the ZMQ socket pattern has been tested successfully. The next steps are to implement simple versions of the request broker and a simplified version of data type serialisation. The community will be invited to test, comment and contribute.

PROS AND CONS

What do we gain or lose by moving from CORBA to ZMQ.

The main advantages moving to ZMQ are: (1) higher performance, (2) simpler to program, (3) lighter, (4) more portable for mobile and embedded platforms, (5) opens the way to supporting other protocols e.g. SCTP, websockets, (6) devices on embedded platforms could publish the TANGO ZMQ protocol (TZMP) using a minimal protocol stack, (7) longer life time for TANGO due to more modern protocol, (8) large active user community.

The main disadvantages of moving away from CORBA are: (1) more code to write because of less services in ZMQ e.g. lack of serialisation.

CONCLUSION

The move to ZMQ for the TANGO event system has been successfully completed. Replacing CORBA completely with ZMQ while still staying compatible with the existing TANGO controls systems is possible. The tasks to replace CORBA completely have been identified and there are no show stoppers. The work involved is estimated to be 24 person months of development to have a first version: 12 for C++, 6 for Java and 6 testing. An additional year of use in a running control system is required for testing and debugging before having a 24/7 production system ready. Replacing CORBA with ZMQ will prove that TANGO can profit from modern protocols while preserving the long term investments of developers and user. The decision to move to ZMQ will depend on the TANGO community and the availability of financial and human resources.

ACKNOWLEDGEMENTS

The authors would like to thank Matias Guijarro and Tiago Coutinho for their comments on the design and proof reading.

REFERENCES

[1] http://www.tango-controls.org

[2] http://zeromq.org/

[3] https://sourceforge.net/projects/tango-cs

[4] https://sourceforge.net/projects/tango-ds

[5] http://en.wikipedia.org/wiki/CORBA

[6] http://omniorb.org

[7] http://www.jacorb.org

[8] Michi Henning, "The Rise and Fall of CORBA", ACM Queue, Vol. 4 No. 5 June 2006, http://queue.acm.org/

[9] http://ZMQ.org/results:10gbe-tests-v031

[10] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, and M. Sobczak, "Middleware trends and market leaders 2011", ICALEPCS2011, Grenoble, October 2011, FRBHMULT05, p. 533, http://www.jacow.org

[11] E. Taurel, "TANGO Collaboration and Kernel Status", ICALEPCS2011, Grenoble, October 2011, TUAAULT02, p. 1334, http://www.jacow.org

[12] J. Meyer, "TANGO V8 - Another Turbo Charged Major Release", ICALEPCS2013, San Francisco, October 2013, TUCOCB10, http://www.jacow.org

[13] http://code.google.com/p/openpgm/