TANGO COLLABORATION AND KERNEL STATUS

E. Taurel, ESRF, Grenoble, France on behalf of the Tango community
ALBA, DESY, ELETTRA, ESRF, FRM-II, MAX-LAB, SOLEIL

Abstract

This paper is divided in two parts. The first part summarizes the main changes done within the Tango[1] collaboration since the last Icalepcs conference. This will cover technical evolutions but also the new way our collaboration is managed. The second part will focus on the evolution of the so-called Tango event system (asynchronous communication between client and server). Since its beginning, within Tango, this type of communication is implemented using a CORBA notification service implementation called omniNotify. This system is being re-written using zeromq as transport layer. The reasons for the zeromq choice as well as a first feedback of the implementation will be given.

WHAT IS TANGO?

Tango is a control system tool kit developed by a community of several institutes. It is object oriented with the notion of devices (objects) for each piece of hardware or software to be controlled. Each device is an instance of a Tango class. Each Tango class is hardware or software specific. Tango classes are merged within an operating system process called a Device Server. Device configuration parameters and network addresses are kept in a database or in a file. Three types of communication between clients and servers are supported: synchronous, asynchronous and event driven.

KERNEL LIBRARIES

Since the Kobe conference, Tango has had 3 kernel library updates. The first release (Tango 7.1.1 in November 2009) was a minor changes and bug fixes release.

The second one was the release 7.2 in October 2010. The main change in this release is the thread safety of the client part of the Tango API. This means that you can share a C/C++ pointer to DeviceProxy instances between different threads. DeviceProxy is the name of the main Tango API client part class. Much faster algorithm when a device server process is shutdown was implemented. Another change is that an application (client) is now able to subscribe to the same event several times.

Then in March 2011, we had release 7.2.6 which was again a minor changes and bug fixes release.

PACKAGING, GUIS AND OTHER

Since several releases, Tango kernel libraries and basic tools are available for Linux via a source code distribution. It is based on classical GNU Autotools and allows a user to build and install the Tango control system with the standard configure / make / make install commands. For Windows, we provide a binary and ready to install distribution. Since several months, we have a binary distribution available for Linux as well. It is based on the Debian packaging system. The classical source distribution has been split into two source packages (for licensing issue related to our Java CORBA Object Request Broker) and 19 binary packages including documentation and debug packages. All these packages are available for Debian and Ubuntu linux flavours. For Ubuntu, a launchpad Personal Package Archive (PPA)[3] has been created making the Tango installation process a matter of a few clicks. The next Ubuntu release available end of October 2011 will natively incorporate these Tango binary packages in the Ubuntu Software Center.

Tango support three languages to write clients and servers. These languages are C++, Java and Python. We also have Graphical layers for these three languages. Since the very beginning of Tango, we have a Java layer called ATK (Application Tool Kit). This layer allows Java Swing application development with widgets (Java beans) interfaced to Tango objects (device, command or attribute). ATK is continuously developed by adding new widgets adapted to requests regarding graphical application development. We now have another Java GUI layer named Comète. It is developed by our Soleil colleagues. This layer opens the data source to something else than Tango objects. Using Comete, it is possible within the same application to get data coming from live Tango devices, but also from the Tango history database (Hdb) or from data files. See mini oral WEMAU012 for more informations on this subject. A C++ graphical layer named Qtango[3] and based on Qt[4] is also available. It is actively developed by Elettra and a online GUI development tool has been added recently. See poster WEPKS022. Finally, a Python layer named Taurus[5] is in active development at the Cells-Alba synchrotron. It is based on PyQt and is fully integrated in the Qt designer tool.

Our code generator named Pogo is since its major release 7 based on a Domain Specific Language (DSL) using the Xtext[6] and Xpand[7] technologies. It is now routinely used to create / update C++ Tango classes. Nevertheless, this tool in its release 7 does not support Python or Java Tango classes. You still have to use the previous Pogo release 6 in these cases.

Since the very beginning of 2011, the Tango security system is routinely used to protect the ESRF machine control system. This allows safer routine operations of the accelerator complex.

The Tango archiving service is actively developed and is now used in several institutes. See poster MOPKN016 for more information on this subject.

COLLABORATION MANAGEMENT

With the increasing number of collaborators using and/or developing Tango, it was becoming difficult to take decisions regarding its evolution. This could become a major problem in the near feature if nothing was done. Therefore, the rules governing our collaboration have been re-discussed and refined. We now have a new release of our Memorandum Of Understanding (MoU)[8]. Three types of collaborators are now defined. The first type are collaborators who do not sign this MoU. Not signing the MoU means that the institute (or individual) is not part of the Tango management board and does not have a right to vote on Tango issues. They are Tango users. The two other types of collaborators are the so-called contributors and committers. The committers contribute resources to the collaboration. The contributors can propose code modifications to the committers for Tango core issues and /or submits Tango device classes to the public repositories. Today (October 2011), we have 4 committers and 3 contributors.

All the strategic decisions about Tango development are now taken by an executive committee. This committee has one member for each institute who has signed the MoU (committers or contributors) plus a "collaboration coordinator". If there is no common agreement between all the committee members on a particular subject, decisions are made by voting. To take a decision, a 2/3 majority is required. Each committee member has at least a weight of 1. An extra vote is acquired for members representing committers institute.

The collaboration manager does not have voting right. His rule is to chair the executive committee meeting, to inform the collaboration of the strategic decisions made during the meetings and to discuss with Tango related project leaders matters like schedule and resources. A Tango executive committee meeting is organized at each classical Tango collaboration meeting. Examples of decision taken by this committee are:

- Tango is no more supported on Solaris platform
- The Source Code Management system used in public repositories related to Tango has to be SVN.
- The new Tango event system will be implemented using the zeromq software
- A list of 9 kernel improvements (extracted from a list of 27) has been selected as having the highest priority.

ON-GOING PROJECTS

On top of the classical evolution of the software already developed around Tango, we have several new projects in their development phase.

On the Java side, our colleagues from Soleil decided to take over the rewriting of the kernel part used when writing Java Tango class. Before this Soleil decision, we did not have the necessary resources within our collaboration to develop and maintain the Tango framework for our three languages at the same level. New features are implemented first in C++. Tango in Python is a layer above the C++ implementation and therefore benefits from the new C++ features with little effort. Java is an independent development and Tango kernel used in Java Tango classes were several major releases late compared to C++ and Python. The new development uses new features added in Java 1.5 like annotations which we hope will made the maintenance of this project less time consuming.

As already explained, the new major release of our code generator is not able to generate Java or Python code. Once the development concerning Java Tango classes will be finished, the code generator will be updated. For Python, more thinking has to be done about the best way to integrate this language in the code generator.

With the always increasing number of features incorporated within Tango, it was more and more difficult to achieve a good level of stability when a new release is introduced. To address this problem, we are now doing Continuous Integration using Jenkins[9]. The tool is configured in a way that as soon as we commit some changes in the repository of Tango C++ kernel code, it generates 20 different flavours of the libraries on 5 Operating Systems (mainly Linux and Windows). Then it compiles 10 Tango classes and finally run our test suite on the 5 operating systems.

Our test suite were composed of two different parts:

- Several test cases developed using a home made test system
- Other test cases using shell script and small C++ software with classical assertions.

A new project is now well on its way to merge these two blocks of test cases in a single one using CxxTest. CxxTest[10] is a xUnit like testing framework for C/C++. By adding new test cases, we will also try to increase our test coverage of the Tango kernel libraries to something close to 75%.

The work needed to implement the 9 kernel improvements selected by the Tango executive committee has also started. Here are some examples of these nine tasks:

- The new Tango event system (detailed below)
- The test suite improvements (shortly explained above)
- The need to have Tango device attribute with enumerated data type
- Implement structures as possible data type for Tango device attributes. This is a limited definition of structure: Only one level (no structure as data member of a structure) and all data members have to be simple Tango data type.

RE-THINKING THE EVENT SYSTEM

The Tango event system is based on the CORBA notification service. When an event is detected (or thrown by the user code), it is sent to the notification service. Then, it is the job of the notification service to forward

the event to all the processes which have subscribed to this event. We are using the CORBA notification service implementation called omniNotify. We now have some experience with this architecture and the following drawbacks have been detected:

- In the case of several clients (event consumers) interested by the same event, the notification service forwards the event to each client using unicast network transfer. This can be a bottleneck in case of a large number of consumers and in case of events carrying large amounts of data (eg images)
- The event data are transferred using CORBA Any objects. This means that in term of performance, it suffers from the unavoidable memory copy due to CORBA Any usage
- In case of event supplier sending events at a high rate with events carrying large amount of data and several subscribed consumers, the notifd has to buffer the event data. This could easily leads to a large memory consumption in the notification service process.
- The omniNotify implementation we have selected is an open source software but it is a "dead" project.

ZEROMQ

After miscellaneous studies, decision has been taken by the Tango executive committee to do a re-factoring of the Tango event system based on zeromq[11] software. The main point leading to this choice are:

- The high level of performance given by zeromq
- Its simplicity in term of infrastructure required (no additional process, no shared memory usage)
- Its availability in many different languages
- The support of a multicast protocol

But what is zeromq? This definition is taken from the zeromq guide available from their web site: *0MQ (ZeroMQ, OMQ, zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry whole messages across various transports like in-process, inter-process, TCP and multicast. You can connect sockets N-to-N with pattern like fanout, pub-sub, task distribution and request-reply. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems. 0MQ is from iMatix and is LGPL open source*.

During the pre-choice studies, the Data Distribution Service (DDS)[12] was also a candidate. Tango is definitively Open Source and this limits the number of DDS implementations available. Even if the level of performance given by DDS is also attractive (but less than 0MQ), it has been judged as less simple for the end-user. The tested implementation requires several additional processes to run on each host where it is used which make the system heavy. It also uses shared memory which could be damaged in case of process using DDS crashes.

THE TANGO USAGE OF ZEROMQ

0MQ provides a way to transport your data but it does not address the problem of data formatting for communication between computers built on different architecture. For synchronous and classical asynchronous communications (not event driven), Tango uses CORBA which has a well defined Common Data Representation (CDR). If the data you want to transport are defined in a CORBA IDL file, all the Object Request Broker (ORB) compilers will generate methods to encode or decode your data to/from this CDR definition. Therefore, the CORBA CDR is the data encoding selected for the new Tango event system while the transport is done using 0MQ.

We also need to define which data has to be transferred between the event supplier and the event consumer(s) to implement a full features event system. These data have been grouped in four parts:

- A string describing the event type: This string is built from the fully qualified Tango device name, the device attribute name and the event type (eg: tango://host:port/the/dev/name/attribute_name.change)
- A single byte encoding the event sender host endianess
- Some call informations allowing the Tango software layer to retrieve which object / method has to be called on the event consumer side. These informations are mainly the receiving object identifier (global information for a whole Tango system) and the method name. As explained previously, these data are encoded using the CORBA CDR
- The event data themselves. These data are already defined in the Tango IDL file for the synchronous communication. These event data are sent on the wire using the CORBA CDR.

Due to this splitting, we are able to use 0MQ multipart messages with one message part for each data group. A 0MQ multipart message is an entity which is fully transferred or not at all. Either you receive all message parts or none of them. Each part of the multipart message is itself managed like a simple 0MQ message.

The event propagation between the event supplier and one or several event consumer(s) is implemented using the 0MQ pub/sub pattern. The event supplier (the Tango device server process) is the publisher while event consumers (the Tango clients) are the subscribers. When you have several subscribers connected to one publisher, it is the 0MQ layer which takes the responsibility to propagate the data to all subscribers. By default this is done using TCP unicast communication. 0MQ also supports a multicast transport using OpenPGM[13] which is an implementation of the Pragmatic General Multicast (PGM) protocol. PGM is a reliable multicast transport protocol. Using multicast to transport Tango event seems a natural way. Nevertheless, it needs to solve the multicast address problem. Every host belonging to a multicast group will receive all the events sent to this group. For

instance, if you have only one multicast address, all the hosts with publishers/subscribers processes will see all the events flying in the system. If some of the events carry large amount of data, it will rapidly become a performance bottleneck. Ideally, one multicast group (address) should be assigned to each event but this will lead to a very high number of addresses. In a more realistic world, the number of multicast addresses available is limited and you have to find an algorithm to spread your events in these multicast group. It's not at all an easy task and depends a lot on which kind of data are generated by the controlled equipments. The decision has been taken to, by default still uses TCP unicast transport for the event propagation. Nevertheless, a Tango control system administrator will have the choice to use multicast transport when tuning the control system. A Tango control system property (configuration data) is defined to specify multicasting usage. This configuration parameter contains the multicast address, the port number and the list of event (device name/attribute name and event type) which should be propagated using this multicast group.

With 0MQ pub/sub pattern, subscriber(s) must set a subscription. This subscription is used by 0MQ as a message filter. Subscription are length-specified blobs. By default, a subscriber filters out all incoming messages. When the subscription is defined, the subscriber receives only messages beginning with the specified subscription buffer. We are using the first part (fully qualified event name) of the multipart message sent by the publisher as the subscription buffer. Thus, a client will receive only the events it is interested in even if on the device server side, the same publisher is used to publish several types of events for several devices.

IMPLEMENTATION

On the server side, the implementation uses two publisher sockets. The first one is used to propagate the heartbeat event. This event regularly sent allows client(s) to know that the device server process is alive, The second socket is used for the real events publishing and is used for all event types for all devices hosted by the device server process.

On the client side, the implementation uses 3 sockets. The first one is a subscriber socket connected to the publisher(s) sending heartbeat events. The second one is the subscriber connected to the real event publisher(s). During Tango event subscription, this socket is connected to the publisher supplying this event and a new subscription blob is associated to this socket. The third socket is a 0MQ Request/Reply socket pair using in-process communication. The 0MQ Request/Reply pattern covers the classical case of one requester asking a service to do something and to send a reply to the requester. 0MQ sockets are not really thread safe. You can use them within different threads only if a full memory fence has been executed before its usage in another thread. The Tango API is thread safe. Therefore, we have to cover cases where several threads required Tango events subscription. As explained above, this requires some actions on the event subscriber socket. Therefore, the Tango event subscription is done via a Request/Reply socket couple with in-process transport.

We have selected release 3 of 0MQ because it implements subscription forwarding. This means that the subscription requests are forwarded to the publisher and the associated filtering is done on the its side. This leads to less network bandwidth usage and less CPU consumption on the subscriber side (client side).

0MQ is written in C/C++ but it's API is C. Nevertheless, a C++ binding is provided and used in the C++ Tango implementation. (thus also covering the Tango Python case). On Java side, 0MQ also provides a binding based on the Java Native Interface (JNI).

Some very preliminary performance tests have been done. The result are summarized in table 1. This is the number of event/sec for events carrying 1 32 bits integer and 1024 integers (32 bits as well) forwarded to 1 and 10 subscribers. Tests have been done using unicast transport. The publisher runs on a Intel core 2 duo at 2.6 Ghz. The subscribers run on a Intel P4 at 2.3 Ghz. The network bandwidth is 100 Mbit/sec.

Table 1: New Event System Preliminary Tests

	1 Long (32 bits)	1 K Long (32 bits)
1 Subscriber	25500	2150
10 Subscribers	2700	270

CONCLUSION

From the first part of this paper, it is clear that Tango is still evolving. The community still wants to improve it and the problem is not a lack of ideas on how it could be improved but rather a lack of resources to improve it. Concerning Tango event re-factoring, it is still too early to draw conclusions on 0MQ usage in a long term. Nevertheless, we now have in labs a Tango event system based on 0MQ. It gives a significant improvement in term of performances and allow Tango to be more user friendly by removing the need of one extra process (notifd) .

REFERENCES

[1] http://www.tango-controls.org
[2] https://launchpad.net/~tango-controls/+archive/core
[3] http://www.elettra.trieste.it/~tango/docs/qtango/doc/html/index.html
[4] http://qt.nokia.com
[5] http://www.tango-controls.org/static/taurus/latest/doc/html/index.html
[6] http://www.eclipse.org/Xtext/
[7] http://wiki.eclipse.org/Xpand
[8] http://www.tango-controls.org/about
[9] http://jenkins-ci.org/
[10] http://cxxtest.tigris.org/
[11] http://www.zeromq.org/
[12] http://portals.omg.org/dds/
[13] http://code.google.com/p/openpgm/