View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CiteSeerX

TANGO KERNEL STATUS AND EVOLUTION

E. Taurel (ESRF) on behalf of the Tango community
ALBA, DESY, ELETTRA, ESRF, SOLEIL

Abstract

This paper has two different parts. After a brief introduction, the changes done within the Tango[1] kernel since the last Icalepcs conference will be described. The second part will focus on the foreseen evolution of the Tango kernel. Special emphasis on the the so-called Tango event system (asynchronous and event driven communication between client and server) will be given. Since its beginning, within Tango, this type of communication is implemented using a CORBA (Common Object Request Broker Architecture) notification service implementation called omniNotify. This will be replaced by a new system based either on a home made design or based on an implementation of the DDS (Data Distribution Service). The final choice is not done at the time of writing this paper.

WHAT IS TANGO?

Tango is a control system tool kit developed within a community of institutes. It is object oriented with the notion of devices (objects) for each piece of hardware or software to be controlled. Tango classes are merged within operating system process called Device Server. Three types of communication between clients and servers are supported (synchronous, asynchronous and event driven).

KERNEL LIBRARIES

Since the last conference, Tango has had 3 kernel libraries updates. The first release (Tango 6.1 in 04/2008) was dedicated to decrease the load on our configuration database in case of massive device server process re-start (after a mains power supply failure for instance).

The second release was the major release 7 in May 2009. It was a major release because the Tango IDL (Interface Definition Language) file has been modified in order to change the data exchanged on the wire. Nevertheless, compatibility between Tango devices is maintained by using IDL inheritance. The main reason of the CORBA interface change was the replacement of CORBA Any's object by union. The CORBA Any object has the drawback of one unavoidable memory copy on the client side which is a performance issue in case of large data transfer. An optimized JPEG encoder/decoder has been written and can be used to transfer compressed images. This encoder/decoder has been optimized using assembler code for the image color encoding/decoding and for the discrete cosine transform used in the JPEG standard. This optimized code is available only on x86 architecture and a classical C++ code is available for the remaining platforms. Queues have been added to the Tango asynchronous communication framework allowing decoupling between event suppliers (device server processes) and consumers (applications). Each Tango device server process has an internal polling mechanism coupled to a small data cache allowing fast response time in case of slow hardware. The polling mechanism has been modified and is now a pool threads allowing the user to select which thread within the pool will be in charge of which device(s). Using Tango, it is very easy to build a hierarchical set of devices. An automatic way to retrieve the device hierarchy has been added allowing faster debugging in case of problem and a graphical display of this hierarchy. A device locking feature has also been added. This allows a client to lock a device. Other clients which are not the lock owner can only do “read” actions on the locked device..

GUIS, PYTHON, ARCHIVING

Tango support three languages to write clients and servers. These languages are C++, Java and Python. New features are implemented in C++. For the Python language, rather than re-writing new features implementation we use the C++ implementation with the help of the Boost[2] python interface library and a Python binding called PyTango. This PyTango binding has been highly optimized during these last two years by Alba. It has been re-structured in order to remove useless data copy. See poster THP 016 and THP 079 for more informations on this subject.

Tango already had two GUI layers: A Java layer called ATK (Application Tool Kit) and a C++ layer called QTango. A new Python Graphical layer called Tau has been developed. It is PyQT[3] based and is fully integrated in the Qt designer tool. The other two GUI layers have also evolved. QTango is now in its major release 3. It is also Qt 4 based and its thread management has been re-written Its internal communication between the Graphical objects and the Tango devices has been re-organized. These changes lead to application less demanding in term of threads and memory consumption. See poster THP096. On the ATK side, several new widgets have been added to its already rich widgets set.

The JDDD[4] (Java Doocs Data Display) tool developed by Desy now supports Tango. JDDD is an interactive panel builder which can use ATK widgets as plug-in. Most of the ATK widgets are available in the tool palette. The link between the ATK widgets that you embed within your panel and the Tango device attributes is done in a graphical way using a Tango devices tree. This allow simple graphical application to be built without requiring any coding from the application

designer. See poster TUP 035 for more informations on this subject.

Our Soleil colleagues have also developed a protocol to access Tango devices from the WEB. It is based on the Java WEB start technology and on a Jboss server to do the link between the HTTP protocol and the Tango protocol (CORBA IIOP). This tool allows an application developed using ATK to run over the WEB without any changes.

ON-GOING PROJECTS

We actually have several new projects in their development phase.

We will soon provide a Debian packaging of the Tango kernel. This will allow a much faster and easier installation of the Tango kernel compared to the source distribution which is provided today. For instance, with a tool like Synaptic which is available in the Ubuntu Linux distribution, installing Tango will simply be a question of a few mouse clicks.

CORBA is multi-platform and multi-languages. Nevertheless, in the control system world, toolkits like Tango do not implement only communication between clients and servers. In the C++ kernel libraries, we can estimate 30 % of code is related to CORBA and network communication. All the remaining code is dedicated to control system specific features implementation. For Java, the Tango community as it is today is only able to maintain the code at the same level than C++ layer on the client side. We do not have the necessary resources to make the Java Tango device server code following the C++ side. To solve this problem, a project has started to split the Java Tango in two parts. The first one will support all features client related and will stay in pure Java. The second part dedicated to server processes will be transformed in a layer above the C++ libraries using a JNI (Java Native Interface) layer. A first prototype as a proof of concept has already been written. Obviously, we will loose portability due to this JNI layer but we will gain in term of features availability and maintenance.

All the Tango classes follow the same skeleton. Therefore, a code generator (Pogo) has been written to generate these skeletons. This tool was available at the very beginning of Tango. Pogo was implemented using hand written parsing techniques. The decision has been taken to re-write this code generator. The new release of this tool will be based on modern techniques using Xtext[5] to create a Tango DSL (Domain Specific Language). This DSL is then used to describe the new Tango class. Using Xpand and a set of templates, the Tango class skeleton is generated. Xtext and Xpand are used through the openArchitectureWare[6] tool. See poster THP 080 for a description of this new code generation system.

RE-THINKING THE EVENT SYSTEM

The Tango event system is based on the CORBA notification service. When an event is detected, it is sent to this notification service. It is the job of this CORBA tool to forward the event to all the processes which have subscribed to this event.

We are using the CORBA notification service implementation called omniNotify. Figure 1 is a drawing of the actual Tango event system.

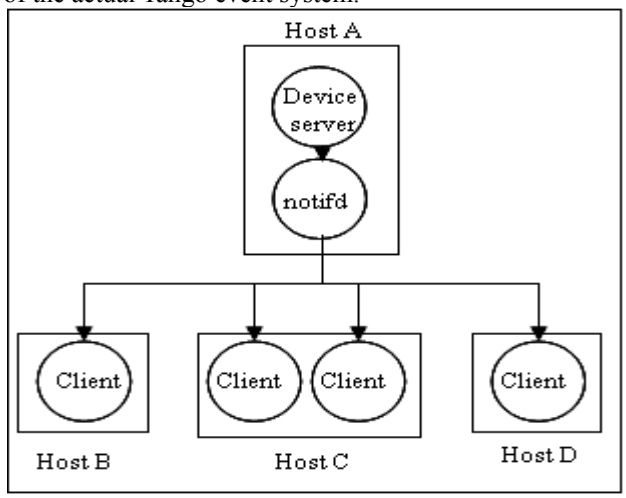

Figure 1: Tango event system.

We now have some experiments with this architecture and the following drawbacks have been detected:

- In case of several clients (event consumers) interested by the same event, the notification service forward the event to each client using unicast network transfer.
- The event data are transferred using CORBA Any objects.
- In some cases, 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 open source but it is a “dead” project. Its mailing list is too quiet and since several years we do not notice any changes in this code

We are actually studying two others solutions to replace the actual system.

The first solution is based on Tango itself. In Tango, we already have what is called Group object. One creates a group in which you add Tango devices. Then, one is able to execute a command or to read/write attributes on all the group members. The group object uses tango asynchronous communication to first send the request to all the devices and then get their replies to finally returned to the caller. The event provider will use an event specialized group object to add the application as a group member. The event propagation will be a call on the event specialized group object. By a proper management of exceptions received by the group object, it will be possible to manage group members in case of application crashes. This solution has the following advantages:

- Re-use the group concept already used within Tango
- Simplicity for Tango control system users. There is no extra process

Nevertheless, it suffers from the following points:

- Still unicast network transfer
- Size of the code to be written

A second solution is the use of the Object Management Group (OMG) Data Distribution Service (DDS)[7]. DDS is a specification of a publisher/subscriber communication system. It is the rule of the DDS middleware to forward data between publishers and subscribers. DDS defines many Quality Of Service (QoS) to tune the way data are exchanged. A standardized network protocol called RTPS (Real Time Publish Subscribe) allowing DDS Interoperability is available from the OMG. This RTPS protocol is designed to be able to run over multicast and connectionless best-effort transports such as UDP/IP. The main features provided by this protocol are

- quality-of-service properties to enable best-effort and reliable publish-subscribe communications for real-time applications over standard IP networks
- Plug-and-play connectivity so that new applications and services are automatically discovered

Several implementations of DDS are nowadays available. We are testing the implementation called OpenSplice[8] which is delivered by the PrismTech company. It is a LGPL licensed software. It supports two network protocols (RTPS and a proprietary protocol), the full set of QoS but only the first level of the DDS specification. It is available for Windows and Linux.

On each host where it is used, OpenSplice is based on a two layers system. The first layer is a network service listening on the network and storing its data in an operating system shared memory segment. The second layer is the set of publishers/subscribers running on this host. The software is configured using an XML configuration file containing parameters like shared memory size, protocol used (RTPS or proprietary).

DDS defines a large set of QoS. This could be seen as an advantage because it gives you the power to fine tune your system but this also means that you have to learn about all of them making the learning curve steep. During our tests, in order to get the same features than what we actually have with our present system, we had to tune 6 different QoS out of the many available. These QoS are:

- Reliability to use the reliable protocol
- Deadline
- Liveliness to be informed of process shut-down/restart
- History in order not to loose events
- Destination order to keep event order (the publisher order)
- Partition

Using multicasting to propagate events seems to be a good solution. Nevertheless, it needs to solve the multicast address problem. With IP V4, multicast address are class D addresses between 224.0.0.1 and 238.255.255.254. 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 be 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 to manage. OpenSplice DDS allows the management of different multicast addresses. Publishers and subscribers registers in a specific partition using the partition QoS. With DDS configured with several partitions, the problem is to assign the event to one partition. It has to be noticed that the OpenSplice DDS implementation of the RTPS protocol today support only one partition making it not really usable in our environment.

Some very preliminary performance tests have been done using prototypes or simulating its usage. The results are summarized in table 1. The number are the increase in number of events/sec we could expect compared to the figures we have actually. From this table, it is clear that DDS gives the best performance. This is particularly true when the number of event subscribers increase.

Table 1: Event System Preliminary Tests

	1 Long (32 bits)		1 K Long (32 bits)	
	Group	DDS	Group	DDS
1 Sub	72%	350%	23%	460%
10 Sub	100%	2500%	50%	2000%

Advantages and drawback of both systems are summarized in table 2.

Table 2: Event Systems Advantages and Weakness

	Group	DDS (OpenSplice)
Advantages	Simplicity No dependency	Performance QoS
Drawbacks	Code to be written	Multicast address QoS Extra processes RTPS not usable

CONCLUSION

From 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.

REFERENCES

[1] http://www.tango-controls.org
[2] http://www.boost.org
[3] http://qt.nokia.com
[4] http://jddd.desy.de
[5] http://www.eclipse.org/Xtext/
[6] http://www.openarchitectureware.org/
[7] http://www.omg.org/tlogy/documents/dds_spec_catalog.htm
[8] http://www.opensplice.org