# AN IMPLEMENTATION OF THE VIRTUAL ACCELERATOR IN THE TANGO CONTROL SYSTEM*

P. Goryl#, A. I. Wawrzyniak, Solaris at the Jagiellonian University, Krakow, Poland
T. Szymocha, ACK Cyfronet AGH, Krakow, Poland
M. Sjöström, MAX IV laboratory, Lund, Sweden

## *Abstract*

Integrating physics codes into the control system gives a possibility to improve machine operation. Providing tools for making computations directly within the control system and letting exchange data between the control system and models is a way of simplifying the whole process of calculating and applying machine's operational parameters as well as keeping track of them. In addition, having a so-called on-line model could be useful for the system diagnostic and faults detection, especially when the objective approach is considered. The concept of the Virtual Accelerator as well as its implementation for the Tango control system will be presented as it is planned to be used for both facilities: the Solaris in Krakow, Poland and the MAX IV in Lund, Sweden. This includes the ModelServer tango device, the simplified C/C++ Tango API to be used with physics codes like Tracy 3 and the tango2elegant script providing an easy solution for integrating the `elegant` tool with the Tango Control System.

## INTRODUCTION

The Virtual Accelerator is a concept of integrating and interfacing different simulation codes into the control system (CS). However, an implementation presented here could be used with any kind of computation codes. It has been successfully implemented at the SLS[1] using the CORBA protocol and at the Diamond Light Source for the EPICS control system[2].

The Virtual Accelerator implementation for the Tango CS presented in this paper has been developed with several goals:

- Standardize a way to starting and stopping computations from the control system
- Simplify data exchange between the control system and the simulation code
- Provide a standard interfaces to different kind of equipment (classes of devices in case of the Tango CS)
- Provide a central point for storing calculated machine parameters within the control system architecture thus making these directly available for all control system applications
- Provide a facility to debug control system scripts and high level applications before these will be run for a real machine. It is expected to make the machine commissioning more efficient.

### *The Tango CS*

The Tango control system implements an objective approach to interface controlled devices. It provides an interface with attributes, representing process values (PVs) or state values and commands representing operations one can invoke on a particular device. Each device is in fact represented as an object of a certain class as it is in the Object Oriented Programming (OOP) [3].

The OOP introduce a mechanism called the polymorphism along with a mechanism of the classes inheritance. This provides a way to abstract from what is behind an interface. As an example, all power supplies integrated in a control system (CS) could expose the same interface to users regardless of a fieldbus the CS is using to communicate with a particular equipment. This also means that it is possible to use the same interface to a model of a controlled system as to the real one. With some assumptions actors interacting with the CS will not be able to distinguish if they work with a real or a modelled system. It does not mean that the same could not be achieved with a non-object oriented CS. Nonetheless, the OOP makes it natural.

Integrating physics codes with the CS takes advantage of the existing tools of the control system. The integration is expected to improve the system diagnostic as well as simplify applying to the machine calculated machine's parameters. The computation results could be logged with the Historical and the Temporary Databases (the HDB and the TDB). Computed settings could be remembered as a snapshot with the Tango SNAP tool. Making the model available through the control system could enable online comparison between the real machine and the virtual one. Distributing machine's parameters, simulation results and real machine's signals through the Tango control system will prevent from clutter caused by using different protocols for data exchange.

The Virtual Accelerator for the Tango Project provides the following components:

- The Model Server device. It is a device that standardizes and simplifies invoking of computation from the Tango CS.
- The Simple-Tango (STango) API. This is C/C++ library enabling access to a Tango CS in a 'one line code'. It hides some Tango complexity.
- The tango2elegant. This is a script interfacing the `elegant` tool with the Tango CS.

## THE MODEL SERVER

The Model Server is a tool to run computations inside the Tango control system. It is implemented as a Tango device. Its class name is ModelServer. It is written in the Python language with the PyTango library. The Model-Server class inherits from the DynamicDS class[4].

* Work supported by the European Regional Development Fund within the frame of the Innovative Economy Operational Program: POIG.02.01.00-12-213/09
#piotr.goryl@uj.edu.pl

The Model Server device is able to run any arbitrary program, script or operating system command. All what is supposed to be run is called a Job. The Model Server device is an interface to run and to control the execution of jobs through the Tango CS.

For the security reasons jobs are placed on a disk in subdirectories of a selected directory. This is specified by a device property named *JobsPath* (device properties are the feature of the Tango CS). Those subdirectories names denote Jobs' names. Additional security measure is provided with possibility to execute the jobs with system credentials provided by two another device properties: *SystemUser* and *SystemGroup*. Details on how to deploy and configure the Model Server are found in [5].

### *The Model Server Interface*

The Model Server provides four main commands to control jobs

- On – to initiate a job
- Off – to clean after a job
- Start – to start the initiated job for certain number of iterations or in an infinite loop.
- Stop – to stop execution of the job. It is possible to stop it immediately (kill the process) or wait for the current iteration to finish.

The model server provides reading attributes, to diagnose job progress: number of finished iterations, when the job has been started, how long it takes to do the job, the job's name.

Since the device class is based on the DynamicDS class it is possible to define so-called dynamic attributes. These may be used by a job to set and read values.

### *Jobs*

However, while the Model Server can run any arbitrary script or code. Its primary application is to run a beam physics code using input from the control system. This kind of codes typically run iteratively in steps:

- Loading a model (lattice file)
- Reading values from the control system
- Modifying the model according to the values from the CS
- Computing
- Reading the results
- Writing the above to the CS

It could be that some of the steps are not implemented in a particular job. However, the reading from and the writing to the CS are common. It is not necessary for an operator or a beam physicist to know details of the control system programming. This is where two another components - the Simple Tango and the tango2elegant - comes into play. Those are described later in this paper.

### *The Model Server Limitations*

The main limitations for the current implementation of the Model Server are that it can only be run on *nix operating systems and that it depends on the *sudo* tool. However, future versions will aim to improve flexibility in starting jobs. As an example it will be possible to directly use the Unicore middleware for setting the job's credentials as well as sending it to a remote computing resource like the PL-Grid infrastructure [7]. In addition, it would be of interest to trigger a job based on CS events, such as changes in certain settings or read values.

## THE SIMPLE TANGO (STANGO)

Simple-tango is a C/C++ library that simplifies access to the Tango control system with C/C++ codes. It is a set of functions for reading and writing devices' attributes in a "one line" code. It hides most of the Tango complexity (even though it is not very complex) from a code writer. It is not intended to replace the Tango API. For more complex operations, one will need to use the Tango API anyway.

The STango uses the standard C/C++ data types to get and to set attributes' values. It does a real conversion from the Tango data types to the C numeric types: `double, float, short, unsigned short, int, unsigned int`.

There are functions for reading and for writing values of scalar, spectrum (1D array) and image (2D array) formats. There is also a function for calling argument-less tango commands. For compatibility with the C there are specific functions that deal with each of the supported C data types. Template functions are provided for use in C++ codes. Please refer to the documentation [5] and the STango source code for details.

### *The STango Source Code*

Simple-Tango provides two header files: stango.h, stangoimpl.h and two implementation files: stangolib.cpp that implements functions for the C and stangoimpl.cpp that implements a device proxies management. These are compiled to the library libstango.so, which have to be linked with a program (i.e. the *gcc* option -lstango).

The stango.h is the header file for use with pure C programs. In C++ code one can use the stangoimpl.h. It provides generic functions as well as giving access to some internal library functionalities.

## THE TANGO2ELEGANT

The `elegant` physics code is used at MAX IV and at Solaris to design and investigate the linear accelerators as well as the storage rings [8,9]. `elegant` is a code provided by the Advanced Photon Source [10]. It does a variety of beam dynamics related calculations. The main input file of `elegant` is structured as a series of Fortran namelists. These namelists are commands with their parameters. The commands setup and invoke calculations. The command parameters' values are specified directly or in specified external files. As an example, a lattice file describing a machine lattice is provided as a reference. The commands are defined and described in the elegant documentation [6]. Among the elegant commands there is a command `&alter_elements.` It allows runtime modification of a parameter value for one or more elements [6]. This feature is exploited by the tango2elegant.

ISBN 978-3-95450-116-8

The tango2elegant is written in the Python language. It acts as a pre- and a post-processor of the elegant input and output files. In pre-processing it reads a main input file and sends effects to the standard output or a file. Its output can be used by the elegant as an input. For the post-processing it uses the main input file and the elegant output files and modifies values in the control system.

It searches the elegant main input file for additional commands (not defined in the elegant) and special strings. Two commands has been arbitrarily defined - `&tango_input` and `&tango_output`. Special strings are replaced with respective values see [5].

If `&tango_input` command is found it is altered with a set of proper elegant commands in the output. Values for parameters in these commands come from the control system.

If `&tango_output` is found it is not sent to the output (hence `elegant` will not see it) but according to its parameters the tango2elegant reads elegant output files and set values in the control system respectively.

Other commands are directly passed to the elegant. Comments are omitted. Please refer to the documentation [5] to see details on how to use the commands.

### *Usage of the tango2elegant*

The simplest way of using the tango2elegant is to call it as the following:

```
tango2elegant input.ele | elegant \
-pipe=in,out
```

Providing that *input.ele* is an elegant input file with added `&tango_input` and/or `&tango_output` commands.

The other way is suitable for running with the Pelegant. To assure that there is ready and valid output from the Pelegant, it is worth to wait until it stops before processing the output:

```
# pre-process only &tango_input:
tango2elegant --no-tango-out input.ele tmp.ele
 # run computation:
mpirun --mca btl openib,tcp,self -np 6 \
  Pelegant  tmp.ele
# process only &tango_output:
tango2elegant --no-tango-in input.ele
```

## STATUS

All three modules have been implemented.

There are efforts to deploy the Virtual Accelerator on the PL-Grid infrastructure [7]. A dedicated virtual machine (vm) acting as a Tango host [4] and running several Tango device servers is up and running. The vm is run inside a cloud being a part of the PL-Grid computation infrastructure. It is possible to send jobs from the virtual machine to infrastructure clusters using the Unicore midleware. The Model Server is supposed to use it. A current setup reveal a bug in the *sudo* tool (on Scienific Linux el5), that in the certain condition wait infinitely for a subprocess to finish even if it has already ended. It makes impossible to detect the end of an iteration. Measures has been taken and the Model Server is in redevelopment to be more flexible in the way it starts and waits for iterations.

The `elegant` and the Pelegant have been installed and successfully used for the beam dynamics calculation on the ZEUS cluster at the ACK Cyfronet in Krakow, which is a part of the PL-Grid infrastructure. Using parallel elegant on the Zeus cluster for particles tracking studies, performing frequency maps analysis, etc. benefits in much shorter calculation time (hours instead of days on a PC). The Tango with its libraries and tools has been compiled on the ZEUS, too. It has been proven that it is possible to use the Tango protocol to exchange data between the cluster and the virtual machine. It means that it is possible to have the Virtual Accelerator running on a clusters infrastructure.

It is planned to deploy the Virtual Accelerator at the MAX IV Laboratory, soon. It will be used to tests high level applications and scripts for the MAX IV and the Solaris control system, before real machines will be installed. It is expected to shorten the commissioning time as well as make it smother. Later it will be run in parallel to the real machines also providing an online model.

All the source codes and documentation will be available through a publicly available SVN server, soon.

## ACKNOWLEDGMENT

This research was supported in part by the PL-Grid Infrastructure [7].

Thanks for Johan Bengtsson for bringing the idea of the Virtual Accelerator and his comments on the project.

Thanks for Claudio Scafuri from the Elettra and Filip Lindau from the MAX IV Laboratory for their codes which, although now heavily modified, served as a base for tango2elegant.

## REFERENCES

[1] M. Böge et al., “Commissioning of the SLS using CORBA based beam dynamics applications”, PAC’01, TOPB012 (2001).

[2] M. T. Heron et al., “The DIAMOND Light Source Control System”, EPAC’06, THPCH113 (2006); http://accelconf.web.cern.ch/Accelconf/e06/PAPERS/THPCH113.PDF

[3] The TANGO Team, *The TANGO Control System Manual*, Pink Site (2012); http://ftp.esrf.eu/pub/cs/tango/tango_80.pdf

[4] S. Rubio-Manrique et al., “Dynamic attributes and other functional flexibilities of PyTango”, ICALAPCS’09, THP079 (2009).

[5] P. Goryl, “The Virtual Accelerator for the Tango CS documentation”, Solaris (2012).

[6] M. Borland, *User’s Manual for elegant*, APS (2011).

[7] M. Bubak et al., *Building a National Distributed e-Infrastructure -- PL-Grid,* (Springer, 2012).

[8] A. I. Wawrzyniak, et al., “Injector Layout and Beam Injection into Solaris”, THPC123, Proceedings of IPAC 2011, San Sebastián, Spain.

[9] A. I. Wawrzyniak, et al; “Solaris storage ring lattice optimization with strong insertion devices”; TUPPC025; Proceedings of IPAC’12, New Orleans Louisiana, USA (2012).

[10]http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/software.shtml

ISBN 978-3-95450-116-8