TINE RELEASE 4.1: RESPONDING TO THE USER'S NEEDS

Philip Duval, Piotr Karol Bartkiewicz, Steve Herb, Mark Lomperski, DESY, Hamburg
Stefan Weisse, DESY, Zeuthen.

Abstract

In the period between the shutdown of the HERA collider and the commissioning of the PETRA 3 synchrotron light source the TINE [1] control system was upgraded and modernized to the next major release level, namely 4.0. Many of the new features and capabilities have been reported before [2]. As can be expected, when what was 'designed and planned' is actually put to use, various imperfections and deficiencies begin to surface, the natural 'enemy' of the developer being the 'user'. To this end there has been a slow and iterative progression toward TINE Release 4.1 which will be reported on here. Many of the embellishments involve improving data transfer efficiency (such as enforcing the use of multi-channel arrays even when the user makes single channel calls) or meeting the user's expectations of what should be possible (such as allowing variable-length TINE data types to appear within TINE data structures). In addition, TINE Central services have been more systematically integrated into the protocol.

INTRODUCTION

Originally a spin-off of the ISOLDE control system [3], TINE is now a mature control system. A great del of developmental effort has gone into the control system protocol, resulting in a multi-faceted and flexible API with many alternatives for solving data flow problems. As the standard TINE kernel is written in straight C and based on Berkeley sockets, it has been ported to most commonly used operating systems. Additionally, a native java port of TINE is being used extensively in the control systems of PETRA3 and its pre-accelerator chain and has been integrated into jDDD [4] and Control System Studio [5]. Likewise, interfaces to LabView and MatLab have a veritable 'army' of users. Recently, Python bindings (PyTINE) and a .NET interface have also been made available.

Throughout the commissioning phase of the PETRA3 pre-accelerator chain and the first several months of PETRA3 operations, the TINE release level in use has been in the 4.0.x area. TINE Release 4 itself marks a major upgrade in the TINE protocol and has been described elsewhere [2]. It is not surprising that 'design flaws' and 'missing features' were made evident during this commissioning period, causing a natural evolution to take place. At the same time, as hardware and infrastructure improve (e.g. 64bit operating systems and Gigabit Ethernet are becoming standard), expectations of the users concerning the abilities of the control system also tend to rise. Furthermore, it is becoming common practice to control a facility with a primary control system but with interfaces to elements of other control systems. The users are not concerned to any extent with how difficult this might be but simply expect it all to work.

This systematic progression, following the normal 'give-and-take' interactions with operators and engineers using the new control system tools and applications, has now culminated in TINE Release 4.1 which we describe below, with particular emphasis on satisfying the user's needs.

THE USERS

The primary category of 'users' consists of the control system developers, who are making use of the control system API to accomplish the control system tasks at hand. The next major category consists of the hardware engineers who are not only using the control system tools to test and maintain their hardware but also placing demands on design criteria. Then there are the machine physicists who are commissioning the machine along with the operators who are running the machine. In many cases these categories overlap.

Developers

The control system developers are the most prone to exposing weaknesses and deficiencies in the control system API, their feature requests naturally leading to an *expansion* of the API. On the other hand, resourceful developers who find a way of "getting the job done somehow" can promote systematic changes or new mechanisms that appear *beneath* the API layer.

In the former category, TINE now supports for example, property-specific *access control lists* and property access *signals* simply because these things were wished for by the developers. TINE Access Locks are now easier to deal with and offer the ability to request *exclusive read* access. Users can now register a so-called *cycle trigger function* (a dispatch routine called when a new cycle number is received).

In the latter category (responding to use cases), it was seen that, although certain server properties are duly registered as providing Multi-Channel Arrays (MCAs), some application programs were written to monitor each channel individually. Think of a vacuum server with 300 sputter pumps being interrupted 300 times per second to return the readback value of each pump individually as opposed to being interrupted once per second to return the values of all pumps with a single atomic call. TINE now enforces MCA access by introducing simple handshaking that informs the caller which element of the entire MCA is being addressed. This of course happens underneath the API, so that the caller is unaware that this efficiency improvement is taking place. Another example of innovation beneath the API is the inclusion of *lazy* scheduling versus *eager* scheduling, where property-

scheduling events from a server can either invoke the property dispatch directly (eager) or simply mark the property for scheduling at the next available opportunity (lazy).

Normal bug reports and fixes will not be discussed in any detail here, but we mention that this category of users is that most prone to stumble over unforeseen bugs and use-cases.

Engineers

The hardware engineers are often the first to use the semi-finished control applications; they have their own expectations about what is needed to test their hardware and are among the first to notice what doesn't 'behave'. Most of the examples given below in the section *Central Services* in fact come from the engineers using the central services general applications tools.

We mention here that systematics were introduced in TINE 4.1 which enable device servers to refer archive calls to the appropriate channel in the central archive and vice versa to allow the central archive to reference the appropriate local history channels at the device server.

Machine Physicists and Operators

The machine physicists and operators not only expect things to behave properly but demand accuracy in what is being displayed.

If the alarm viewer is displaying an alarm, it must be 'real' and displayed at the proper severity. And, if there is a real alarm it must be displayed on the alarm viewer! Responding directly to requests from the operators has in TINE 4.1 led to among other things, oscillation-window learning (when is a new alarm a 'new' alarm and when is it just 'flickering'), the ability to place the same alarm in multiple alarm systems, the ability for the operators to suppress alarm categories completely, and so on.

Sometimes expectations based on 'what should be possible' also drive development. For instance, if the on-line analysis of high-resolution video frames could in principal be met with a gigabit Ethernet, then it is expected that the control system should be able to achieve this. Allowing for some overhead concerning contract and connection management, TINE 4.1 has come a long way toward meeting these expectations.

INTEROPERABILITY

In a control system environment using TINE and TINE applications, it is important that any 'foreign' elements appear to the users as if they were also TINE elements. This of course requires seamless interoperability with these elements, which involves not only simple data transfer, i.e. the transaction translation layer, but also the mapping of alarm and archive information as well. In some cases 'name-mapping' is also expedient in helping users locate items of interest. We discuss below recent work concerning TINE Release 4.1 and its interface to other popular control systems.

DOOCS

As TINE is completely embedded in DOOCS [6] and both long- and short-term plans are to run DOOCS via the TINE protocol, much effort has already been expended in fixing and avoiding what one might term *impedance mismatches*, formally assuring that the DOOCS environment *is* the TINE environment and vice versa. This includes making sure the name space and the format space are synchronized, that browsing strategies always work, that the alarm mappings are complete, and that archive data acquisition works as expected. This has by and large been achieved.

EPICS

An EPICS [7] view of TINE or a TINE view of EPICS requires a fully functional mapping between the two systems. This has mostly been accomplished via the epics2tine translation layer [8], which can run embedded on any EPICS IOC as well as on the TINE instantiation of the javaIOC. Tantamount to having seamless operation is understanding the difference between EPICS pvData and device server property access. These are not identical and reflect the differences between a database view of the control system ('get', 'set', 'monitor') and a device instance view of the control system ('property' calls). As long as properties reflect attributes of device instances the mapping is straightforward. Where properties represent commands ('RESET') or methods (calls with distinct input data) the mapping becomes more complex. In addition, one of the innovations of the javaIOC is to allow structured data, which has been a feature of TINE for some time. However the javaIOC requirement of allowing mutable (i.e. non-fixed length) strings within structures has resulted in the requirement that TINE 4.1 also supports variable length formats, such as strings, image types, spectrum types, etc. (where the length of an array of these 'things' does not give number of bytes) within structures.

TANGO

TANGO [9] has generally been seen to be a good fit to the TINE API. Only a few interface routines were added to TINE over the course of the year in order to simplify the tine2tango [10] and tango2tine gateways. One potential stumbling block remains. TANGO has no official name length restrictions for its *family*, *class*, *member*, and *property* names, whereas the equivalents in TINE do (e.g. 64 characters for registered device and property names). For practical purposes, there is only a problem for the TANGO *member* which maps to the TINE *device*. Even here, one seldom encounters such long device names, unless the natural hierarchy is artificially extended at this juncture (i.e. */context/server/device/sub-device/sub-sub-device/etc*)
This is not a problem for most practical purposes as the TINE protocol allows device names to carry up to 1024 characters.

STARS/COACK

STARS [11] or COACK also maps to TINE in a straightforward manner using a STARS 'bridge' [12]. STARS, however, has no restrictions to the name space hierarchy. This leads to the solution alluded to above in the discussion on TANGO, where the hierarchy is extended via the device name. Practically, a hierarchical device name comprising more than 1024 characters should never be seen.

CENTRAL SERVICES

The TINE central services have all experienced extensive feedback over the past year, which has in many cases led to improvements in the TINE kernel itself. We describe below some of more important improvements to central services to be found in TINE 4.1.

Archive System

The TINE archive system for instance, is extremely responsive concerning 'lookup' requests. Time-centric lookups of multi-channel arrays cost on the order of 1 millisecond (including network), largely independent of the size of the MCA (a few micro-seconds per channel). Time-range lookups typically cost ~100 milliseconds for 1000 values (less than 1 millisecond per channel-lookup). The TINE archive viewers are likewise tailored to fast, on-line browsing of archive data, in that a maximum data size on which to raster the stored data is used (zooming on a time region forces a reacquisition of the archived data). Nonetheless, browsing over a week's worth of data distributed over only a few thousand points could jump over important 'glitches'. To prevent *missing* these glitches, TINE 4.1 now marks detected 'jumps' as *interesting* and ensures that archive calls will always contain such *points of interest*. In the same vein, some MCAs tend to be *volatile* with regard to the channel name lists composing the names of the array elements. That is, the list of comprised channels might be different today than it was yesterday. Note that the channel names are stored separately from the array data in order not to impact heavily on the available disk storage. The TINE 4.1 archive server now allows these MCAs to be marked as volatile, causing archive lookups to check and adjust the name list configuration over the time range requested. This will cost a bit more time concerning the lookup, but will ensure that the data for a specific channel is consistent over a time range even if the channel was moved to a different array element during the selected time range.

In analogy with the MCA viewer, general viewers for scope traces and transient recorders have a set of systematically implemented features which encompass a wide variety of servers and allow easy analysis.

Alarm System

Hardware alarms (bus I/O errors) are often of particular use to the hardware engineers, who have a set of questions involving 'how often?', 'which hardware modules?', 'what kind of error?' etc. The TINE 4.1 Alarm Viewer offers the ability to generate alarm 'reports' as an aid in answering these questions.

Naming System

TINE offers plug-and-play server configuration, where server developers do not need to involve control system administers to add new servers to the system. However, removing a server once it has been declared 'defunct' was always a task left to the server developer or administrator as an explicit request to the naming services. Popular demand has now given rise in the TINE 4.1 Equipment Name Server (ENS) to a 'deadweight' checker as part of the normal naming services. Non-responsive servers (no response over a 90-day period) are now automatically removed from ENS database.

CONCLUSIONS

We have given only a summary of some of the issues which drive the evolution of a control system and in particular have driven the progression of TINE 4.0.0 to 4.1.0. Many details have been omitted as has a discussion of the co-evolution of the control system with operating systems ("How does Microsoft's change in Winsock behavior impact the control system?"). We note that TINE 4.1 will continue to evolve into TINE 4.2 and beyond.

REFERENCES

[1] http://tine.desy.de
[2] "TINE Release 4 in Operation," P.Duval et al., PCaPAC 2008.
[3] "A PC Based Control System for the CERN ISOLDE Separators", R. Billnge et al, ICALEPCS '91.
[4] "First Experiences with jddd for PETRA Vacuum Controls", E.Sombrowski, et al., PCaPAC 2008.
[5] "Control System Studio (CSS)", Jan Hatje, et al., ICALEPCS 2007.
[6] http://doocs.desy.de.
[7] http://www.aps.anl.gov/epics
[8] "An EPICS to TINE Translator", Z.Kakucs, et al., ICALEPCS 2001.
[9] http://www.tango-controls.org
[10] see "EPICS to TANGO Translator", R.Stefanic and L.Geoffroy, ICALEPCS 2007.
[11] http://pfwww.kek.jp/stars/
[12] "The interconnection of TINE and STARS", T. Kosuge, PCaPAC 2006.