TINE RELEASE 3.20: STATUS REPORT

Philip Duval, DESY, Hamburg Germany

Abstract

The dominant control system in the HERA accelerator is TINE[1],[2]. TINE is playing an increasingly important role in the pre-accelerators at DESY as well as in Zeuthen. Furthermore, recent efforts have made the EPICS2TINE [3] translator a viable alternative to EPICS over Channel Access. TINE has also been successfully integrated into DOOCS [4] so that DOOCS clients and servers can run happily over TINE as opposed to SUN RPC. TINE plugs for Abeans [5] as well as COACK [6] have also been developed. As one might surmise then, demands on and from control system components have made TINE extremely flexible and adaptable to a wide variety of operating systems and platforms. The most recent release of TINE (Version 3.20) offers many new features, some of which are completely unknown in other control system protocols (such as network subscriptions via multicast). In this report, we shall give a status report on the TINE control system highlighting the latest features available in release 3.20.

1 INTRODUCTION

For the most part, TINE defines a data-exchange protocol, which can be used as the basis for a control system. As a control system proper, it does not specify a hardware layer and it in this aspect “do-it-yourself” oriented. On the other hand, where database driven servers are desired, TINE can run on top of say an EPICS database and in lieu of Channel Access. Likewise, TINE can run on standard DOOCS servers. On the other hand, as the hardware layer is unspecified TINE can provide a “flexible response” to advances in industry where the “latest IO platform” doesn’t need to be first properly incorporated into the control system before it can be used.

TINE follows the traditional “Client-Server” dichotomy, where the control system elements play either the role of (and have the characteristics of) a server or a client. Servers can be attached to hardware (Front End Computers – FECs) or can provide middle layer services. They have unique names and addresses and are known on the net via database or name server (i.e. they do not broadcast their services). Clients are “anonymous” and can exist in multiple instances anywhere on the network. By this, we mean that clients are nowhere entered in a database, and several clients can appear bearing the same name. (Each computer on the ethernet must of course have a unique address). Clients find Servers by querying the equipment name server.

2 THREE-FOLD INTEGRATION

Perhaps the most distinguishing feature about TINE is its integration of client and server components of vastly different networking environments. To begin with, TINE is a multi-platform system, running on legacy platforms such as MS-DOS, Win16 (Windows 3.X), and VAX and ALPHA VMS, as well as Win32 (Windows 95,98, NT, 2000, XP), most UNIX machines including Linux and FreeBSD, and VxWorks. TINE is also a multi-protocol system to the extent that IP and IPX are both supported as data exchange protocols. In this day of the internet, IPX should be regarded as a ‘legacy’ protocol, but is nonetheless supported where an IPX stack exists. Finally, and perhaps most interestingly, TINE is a multi-control system architecture system, allowing client-server, publisher-subscriber, and producer-consumer data exchange in any variation. A recent innovation in the TINE data-exchange package also includes a hybrid between publisher-subscriber and producer-consumer, which might be called producer-subscriber and brings to light the idea of a ‘network subscription’.

This is an important feature of TINE, so we will expound open it in a bit more detail below.

Client-Server: A traditional data exchange mechanism available in most control systems is pure, synchronous client-server data exchange, where a client makes a request and waits for the completion of the request. This is generally used to change hardware settings for instance. However polling front ends to get display data becomes very inefficient when the number of clients is large (> 10 say).

Publisher-Subscriber: For many cases, a much better approach is the publisher-subscriber data exchange. Here a client (the subscriber) communicates its request to a server (the publisher) and does not wait for a response. Instead it expects to receive a notification within the timeout period. This can be a single command, or for regular data acquisition it can be a request for data at periodic intervals or upon change of data contents. In this format, the server maintains a list of the clients it has and what they are interested in. This is much harder to program than simple client-server communication, but the payoff is large. As all data exchange is scheduled by the server instead of the client, the problem of a large number of clients is reduced by an order of magnitude or more. When the number of clients is very large (~100), one could consider yet another mechanism: Producer-Consumer.

Producer-Consumer: A third alternative for data exchange is the Producer-Consumer model. In this case a server is the producer. It transmits its data via broadcast or multicast on the control system network. Clients (i.e. consumers) simply listen for the data. This is frequently the appropriate data transfer mechanism, when the number of clients is large. For most control systems, there are certain parameters of system-wide interest. At HERA for instance, the Electron and Proton beam-energies, beam-currents, beam-lifetimes, etc. are made available via system broadcast at 1 Hz. As soon as multicasting is enabled across all routers at DESY, this will in fact be changed to a multicast. Note that in this case, a server has no knowledge of clients whatsoever. Furthermore, clients no longer get what the want via a contract. Instead they get what has been produced on the network.

Producer-Subscriber: A very attractive alternative to the above data-exchange mechanisms is the Producer-Subscriber mode. On the surface, this looks like the Publisher-Subscriber model, except that the calling subscriber can request that the contract be sent to for instance his entire subnet or his multicast group. Then if such a client program is started on many stations, a server ‘sees’ only one client, namely the network. This reduces the load on the server, and can dramatically reduce the throughput requirements for the server, if say images from it at a high rate on many stations. As always, one should use this mode of operation where it makes sense. Note that if all control system elements are transferred this way, then all applications will see all control system traffic at the application layer (whether the data is to be thrown away or not). This could place an unnecessary load at the client-side.

Full details of the TINE protocol, its systematics, functionality, and operability are given in reference [1] and [2], as are descriptions of available services.

3 NEW IN RELEASE 3.20

MULTICASTING: All globals and multicasting sockets now use the SO_REUSEADDR option, so that daemons are no longer required on any platform. Furthermore, the socket is not opened unless the application makes a request to receive globals or network subscriptions. The means that all applications on the same station making such requests will end up processing the incoming data individually, but turns out to be a much more efficient scheme of handling network data, particularly if java and C clients are running simultaneously.

Property REDIRECTION: A server can export a property which lives on another server. A redirection string can be specified, interpreted as <deviceServer> / <deviceName> [deviceProperty]. Requests to the server for the exported property will then be redirected to the device server given. If [device property] is given, the property name will also be redirected, otherwise the same property name is expected to be found on the redirected server. In any case the <deviceContext> and <deviceName> are assumed to be the same on the redirected server. Note: by specifying the <deviceName> in the redirection tag, you are instructing the server to redirect requests for the given property only for the specified device.

Assigned ERROR VALUES: On the client-side, it is now possible to assign error values to data upon status error. Upon error, the contents of the buffered data array will be filled with the assigned error value. This gives a client programmer an easy way 1) to display faulty data (e.g. give a value out of range so that it catches the eye) and 2) to determine which data are bad inside of grouped calls.

Comprehensive property QUERY: All information for the requested property can be queried with one call. This includes input and output data types and lengths, engineering units, ranges, history information, etc., and for all overloaded properties (for those cases when the same property can be called with different format types and lengths).

DEVICE LISTS per property: It is now possible to register deviceQueryFunctions so that a property can be queried for its associated devices. The returned lists can be completely independent from one another. This is a very useful feature for certain middle layer servers (archive, alarm, gateways, etc.) which might be providing information for both vacuum pumps and beam position monitors. A new 'meta' property '.NAM' was introduced to facilitate the above feature. Thus, each registered property <deviceProperty> implies an associated property <deviceProperty>.NAM which can return a list of device names.

Remote access to LOG FILES: New stock properties are available for acquiring and manipulating the server log files. For servers without a disk (e.g. VxWorks) it is possible to maintain a virtual log file in memory, reflecting transactions since server startup

Better WILDCARD HANDLING: The wildcarding in property and device queries is now more extensive. It's now possible to query "*ABC*". Previously the first wildcard '*' terminated the search string.

More control over BURST PARAMETERS: It is now possible to adjust some of the transmission parameters away from their default settings. This becomes important for extra large data transmissions (hundreds of Kbytes). Initial tests show that over 7 MBytes/sec can be reliably transmitted (e.g. ~450 Kbytes at 20 Hz with a loss of 4 frames per minute). This order of frame loss can be considered reliable for video signals. Note that this benchmark is also valid for a "NETWORK" subscription (i.e. the server can multicast this information) meaning that N clients (with N large!) can get the information simultaneously with no extra drain on the server.

5 CONCLUSIONS

The flexibility of TINE has been invaluable in integrating the HERA front ends into a working system. Just as important, it has demonstrated a transparent way to progressively upgrade existing hardware. Where for practical reasons the latter must remain on "older" platforms and operating systems, TINE servers can nonetheless be installed and maintained. When it becomes practical to "modernize" front-end elements, this can be achieved piecemeal, without any blanket restructuring. The implementation of TINE at DESY is PC-dominated. Although TINE works fine in say a pure UNIX world, the number of GUI tools developed for PC consoles running Win32 make the latter (currently) the most attractive platform on the client-side.

Where are areas of applicability for TINE? Since TINE is based on sockets, and not on something 'modern', such as DCOM or CORBA, one might be inclined to dismiss it as a relic which found its niche in HERA. Nothing is farther from the truth. At the developer's end, the tools offered are as modern as anywhere (ActiveX, Java, MatLab, LabView, etc.). The transport layer is of course invisible to the developer. Furthermore, neither DCOM nor CORBA runs on the number of platforms supported by TINE. A switch to either one would immediately dispense with several legacy platforms such as MSDOS, WIN16, and VMS. Finally, both DCOM and CORBA work in a unicast world. The ability to multicast or to have 'network' subscriptions would be completely missing. When any of these considerations (legacy platform support, multicasting) is important, TINE should be considered as one of the best alternatives around.

REFERENCES

[1] P.Duval, "TINE: An Integrated Control System for HERA", Proceedings, PCaPAC'99, 1999.
[2] http://desyntwww/tine
[3] Z. Kakucs, P. Duval, M. Clausen, "An EPICS to TINE Translator", Proceedings, PCaPAC 2000.
[4] http://tesla.desy.de/doocs.
[5] I.Verstovsek et al., The New Abeans and CosyBeans: Cutting Edge Application and User Interface Framework, PCaPAC02, Frascati 2002.
[6] I.Abe, et al., "Recent status on COACK project", Proceedings, PCaPAC 2000.