Meas. Sci. Technol. **18** (2007) 2379–2386 doi:10.1088/0957-0233/18/8/012

TINE as an accelerator control system at DESY

Piotr Bartkiewicz and Philip Duval

DESY, Hamburg, Germany

E-mail: piotr.bartkiewicz@desy.de and philip.duval@desy.de

Received 30 October 2006, in final form 13 February 2007
Published 6 July 2007
Online at stacks.iop.org/MST/18/2379

Abstract
Large research institutes such as accelerator-based research centres deal with a large number of complicated components, which should be integrated into and controlled as one homogeneous system. The components of such systems have been frequently developed at different times, in different countries and use different standards and technologies. The control system of components or subsystems may use different computer hardware, operating systems, network protocols, hardware interfaces and field buses. This paper describes the three-fold integrated networking environment (TINE) (http://tine.desy.de) control system, which provides control integration of accelerators at Deutsches Elektronen Synchrotron—DESY, Hamburg, Germany.

Keywords: control system, accelerator, computer network

(Some figures in this article are in colour only in the electronic version)

1. Introduction

In this paper we describe the concept and real implementation of a control system, which has been used with great success for many years at Deutsches Elektronen Synchrotron, DESY, Hamburg. In order to save time for someone who might want to learn more about control systems where they are understood as a set of algorithms, implied from the control theory, mathematical modelling or theory of optimization, we now emphasize that this is not the right paper. We present here a complete software solution which enables efficient management of data flow between various process controlling components, such as sensors, programmable logic controllers (PLCs), actuators, data processing computers, central computers and so on. Consider, for the moment, a remote sensor data readout mechanism and remote actuator steering. This would not yet constitute a real control system. Simple data flow mechanisms plus data processing do not create a system. Still missing are central services, such as archiving, alarm processing, name resolution, etc. The control system will also have states, even if nothing else beyond 'on' or 'off'. This means that the control system is apt to offer such tools as finite state machines, sequencing and automation, where the system can be driven from state A to state B without human intervention. The control system will also have to deal with synchronization of the distributed processes. This in turn means that all of the players will have to agree on a clock, and there should be some attention paid to the amount of jitter allowed in the individual process clocks. The control system will also have to maintain a strategy for security. We do not mean simply Internet security, but also 'system' security, where user A (although he is a trusted user) should not be allowed to change setting B because the system is not in the appropriate state. The control system will also likely have to manage databases, meaning the general server-side configuration databases (hardware addresses, names, etc) as well as the controlled process data, the latter being strongly tied to the archive system. In order to trap faults and keep an overview of the system itself, the control system will have logging capabilities and statistics of the control system as a collection of controlled elements as well as the system operation. The control system should also offer presentation and user-interface services as well as maintenance and even development tools.

The control system could be (and frequently is) represented by three tiers: 'the lowest', closest to the controlled process hardware tier, is called here a 'device server layer'. This tier is responsible for data flow from sensors and to actuators. It usually covers connections to field buses or controllers, or to specific hardware connected to the front-end

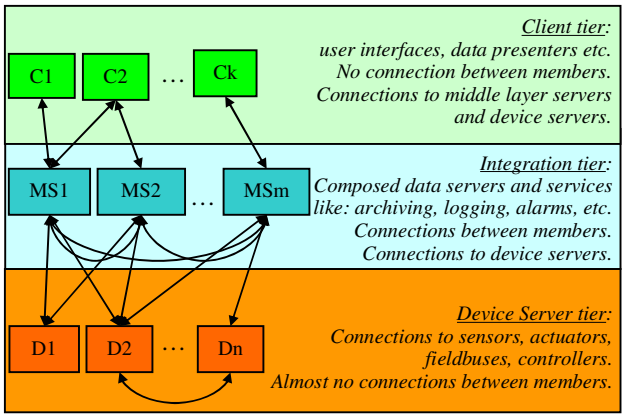

Figure 1. Three tiers of a control system.

computers. The second tier is a 'middle-layer server' playing the role of an 'integration tier'. To this tier belong those servers processing the data from various 'device servers' or other 'middle-layer servers'. These 'integration tier' servers offer composed, integrated and consistent data sets, needed for various control perspectives. Similarly, services such as archive services, alarm services, state and state-permit services belong to this tier. The top layer is a 'user-interface' tier, typically providing a set of graphical client interfaces. (figure 1).

2. Overview of control system tasks at DESY

Before presenting the three-fold integrated networking environment (TINE) control system at DESY, let us look at the 'job specifications' (i.e. those tasks which are expected to be performed by the control system) and the environment and conditions in which the system should work.

2.1. Short overview of our facilities

Deutsches Elektronen Synchrotron, DESY, founded in 1959 in Hamburg, is one of the leading centres for research at particle accelerators in the world. DESY has two locations: Hamburg with 1400 employees and Zeuthen, near Berlin, with 200 employees. Since its inception in Hamburg, DESY has built several accelerators, three of which are currently used for research: DORIS, PETRA and HERA.

DORIS and PETRA are used by geologists, biologists, chemists, physicians and material scientists as a source of synchrotron radiation to investigate atomic details of materials. In addition, PETRA plays the role of a pre-accelerator for HERA.

The hadron-electron ring accelerator (HERA) is a large, 6.3 km long dual ring accelerator, which delivers proton–electron (or proton–positron) collisions for two experiments: ZEUS and H1, both of which are primarily searching for new kinds of matter and investigating the inner structure of the proton and the strong interaction. HERA is also a source of longitudinally polarized electrons (or positrons) for the HERMES experiment, used for the exploration of the spin structure of nucleons.

Every year more than 2700 visiting scientists from 33 countries come to DESY to lead research.

Table 1. Principal devices in HERA and its accelerator chains.

Device type	Number of units (approx.)
Main magnet power supplies	600
Correction coil power supplies	1400
Pulsed magnet power supplies	70
RF systems	230
Vacuum	3000
Beam position monitors	800
Other beam measurement instrumentation	2000
Air conditioning, water cooling	500

2.2. Some facts about the accelerator chains

The accelerators at DESY were of course not built at the same time. The older accelerators are now in use as pre-accelerators for the newer machines. HERA, our largest machine now, uses a chain of three pre-accelerators for the proton ring and four pre-accelerators for the electron ring. The older accelerators and their components as well as the control systems' infrastructures have been repeatedly upgraded throughout their lifetimes and are now representative of various stages of technology. The complexity of the entire chain of pre-accelerators and the main accelerator HERA can be appreciated by a quick scan of table 1 [2], where a rough count of the number of principal elements is shown.

Groups of devices such as RF components, magnets, vacuum pumps and monitors are typically controlled by isolated, 'local' control systems, especially in the initial stages of machine commissioning. Commissioned in 1990, HERA was no exception. One of the tasks at hand, then, was to integrate these local systems into one homogeneous system.

In total, these 'local' systems comprise now, as they did in the early 1990s, more than 500 computers, all involved one way or another in the myriad control processes. Most of these computers are now PCs, running Linux or Windows XP and NT, whereas in the past Windows 3.1 and DOS played a more prominent role. Such legacy operating systems (Windows 3.1 or DOS) are nonetheless still in use in many places (owing primarily to hardware restrictions). To be sure, classic UNIX workstations, such as Sun workstations running Solaris, Hewlett Packard machines running HP-UNIX, etc, have always been in use and still play a prime role in accelerator control at DESY. A number of subsystems make extensive use of VME standard-based computers running VxWorks, and there are also several embedded system solutions, such as PC104 running ELINOS (an embedded Linux distribution) or ALTERA with NIOS II core. At one time, some subsystems were running on a VAX VMS controller. The last such subsystem (HERA reference magnets) was decommissioned in 2001.

Figures 2 and 3 show the structure of the 'local' control systems for two HERA subsystems: orbit control and superconducting magnet quench protection.

2.3. Connection to the experiments

In addition to the control of the accelerator components and subsystems, data exchange with the experiments is needed for safe and proper accelerator steering. Although it was

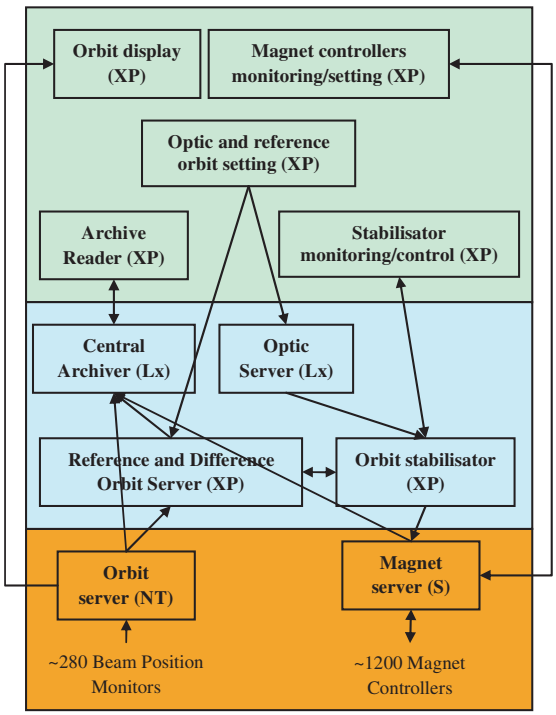

Figure 2. Orbit control system. TINE servers are running on MS Windows NT (NT), XP (XP), Linux (Lx) and Solaris (S) operating systems.

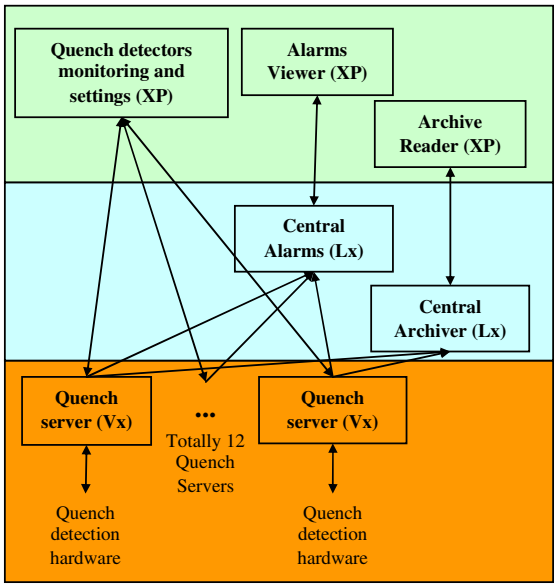

Figure 3. Magnet quench protection system. TINE servers are running on MS Windows XP (XP), Linux (Lx) and VxWorks (Vx) operating systems.

not needed to achieve full integration with the experiments' control networks, it was nonetheless necessary to exchange some amount of data with these systems, which were often developed abroad and which sometimes made use of obsolete technologies. Thus, connection interfaces with machines running different brands of UNIX, VMS or MAC OS were also required.

2.4. Requested features of the control system for the DESY accelerator chain

Now that we have highlighted some of the details and requirements concerning the integration and control of the subsystems and components found in the DESY accelerator chains we are able to list a set of criteria, which must be available in the control system.

- Generally (all tiers)
  - (1) *Multi-platform*. A common library and interface should be available on various hardware platforms and operating systems.
  - (2) *Multi-protocol*. Since network infrastructure has developed with the accelerator chain growth, the system should integrate various Ethernet technologies.
  - (3) *Multi-architecture*. Here we are referring to 'data-transport' or 'data-exchange' architecture. There are some reasons why this feature must be present. One is the need to provide compatibility with different 'local' control systems, which are to be integrated into one system. Another is the data transfer efficiency for certain cases. This will be discussed later.
  - (4) *Capability of transferring various data types*. Any set of primitive data types (simple integer or float arrays, for example) or complex data types (e.g. mixed doublets of two primitives or user-defined structures) between different platforms should work seamlessly, without concern for byte swapping between dissimilar platforms (little-endian versus big-endian).
  - (5) *Clear and easy method of application-specific code integration with the system*. This involves an easy-to-understand interface where it is clear to a developer when he should integrate his own code (if necessary) into the system.
- Device server tier
  - (1) *Capability of interfacing to various field buses*. The most popular field buses at DESY are SEDAC (DESY proprietary field bus), CAN [3], ProfiBus [4], RS232 and GPIB.
  - (2) *Small footprint*. The control system kernel library should have a small footprint in order to fit inside embedded systems, where resources are frequently limited.
- Integration layer
  - (1) *Naming service*. All servers and their services are addressed only by names. The huge number of servers excludes physical addressing.
  - (2) Well-defined integration strategy for other control systems existing at DESY, such as EPICS [5] or DOOCS [6].
  - (3) System for filtering and archiving data, events, alarms, etc.
- Client tier
  - (1) Client browsers for the easy access of data provided by any existing server.

(2) Generic tools offering access to archived data, alarms and event viewers.
(3) Client widgets which can connect to any control system endpoint and can be used in rapid application development in both simple and rich clients.

Among many modern control systems, commercial or open source, we shall see below that only TINE can cope with the above requirements.

3. TINE: three-fold integrated networking environment

The key word in the TINE acronym is 'integrated': this is a multi-platform system, supporting currently MS-DOS, Win16 (Windows 3.X), Win32 (Windows 95, 98, NT, 2K, XP), UNIX (Solaris, HP-UX, OSF, SGI, Linux, FreeBSD), MACOS, VAX and ALPHA VMS, VxWorks and ALTERA-NIOS II. TINE is a multi-protocol system, where data exchange among the participants can occur via any of the UDP, TCP/IP or even IPX protocols. TINE is also multi-architecture, where data transfer can follow any of the strategies listed below.

- *Client–server*. This is a traditional data exchange mechanism, available in most control systems. It is pure, synchronous client-server data exchange, where a client makes a request and waits for the completion of the request. This is also a necessary mechanism for sending commands to a front end, where the next action to take depends on the outcome of the command. The client–server approach has however two disadvantages: if the server for any reasons goes down or network problem occurs, the client application will wait until the timeout mechanism considers a transaction to be aborted. The second disadvantage appears, when several clients want the same information (regular updates of control data, for instance); a server will see each request from each client separately. This can become a burden to the server if many clients (say 50 or more) are getting a kilobyte's worth of data at 1 Hz, as the server will have to acquire the data at 50+ Hz.
- *Publisher–Subscriber*. For many cases, a much better approach is the publisher–subscriber data exchange: the 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. Now if many clients want the same kilobyte's worth of data at 1 Hz, the server must acquire this data set only once per second and notify the clients on its list. This is much more efficient than the client–server model under such circumstances.
- *Producer–Consumer.* A third alternative for data exchange is the producer–consumer model. In this case, a producer transmits his data via broadcast on the control system network or via multicast to a multicast group. Consumers simply listen for the incoming data. For most control systems, there are certain parameters which are of system-wide interest. In the world of accelerators, it might be beam energies, beam currents, beam lifetimes, accelerator states, etc.
- *Producer–Subscriber*. A hybrid between the above two modes is also possible under TINE, in which the subscribers request data to be produced on the network (a 'network subscription'). This is an especially useful method of data transmission for either large-scale machines with many clients needing the same data or large amounts of data sent to multiple clients (such as video). Using the producer–subscriber architecture makes the most efficient possible use of the available network bandwidth.

Each of the above modes of data exchange could be used individually to define the control system architecture, but more likely you will want to use these modes in combination.

For simplicity, in the following sections of this paper, the term '*server*' will mean a data producer in general, and not only a server in a 'client–server' approach context. The same refers to the client and data consumer: the term '*client*' will be used.

4. TINE: security

In an Ethernet-based control system, two security issues should be considered.

The first is, of course, intruder attack. This relates to the general intranet security problem and has more to do with system administration than the control system proper.

The second is to prevent accidental interference with control system operations. TINE allows open read access to all control parameters; namely all data provided by servers can be freely read by middle-layer servers or client applications. Write access, however, can be restricted to a set of users and/or specific networks or even specific network addresses.

5. TINE: no data-type limits

Since TINE data servers and clients can be running on various platforms, the byte ordering of the transferred data must be considered. The proper byte reordering of transferred data is handled by TINE automatically.

All primitive data types such as *byte*, *integer*, *long*, *float*, *double*, etc and their arrays are handled. There is also a large set of frequently used predefined composite types, e.g. CF_FLTINT or CF_FLTINTINT, which are obviously a doublet containing one float and one integer number or a triplet containing one float and two integer numbers, respectively. This set of predefined composite data types is introduced both in order to handle certain systematic requirements (history data are typically transferred as arrays of data plus time-stamp pairs) and as a convenience for the developers. The developers, however, can also make use of an additional feature, usually not offered by other control systems. Namely, one can transfer user-defined types or structures and/or arrays of these structures. There is also no size limit. A good example of a composite data type transferred in our system by TINE is the beam orbit readout from the beam position monitors, containing not only positions of the beam but also

information about each monitor hardware status as well as the measured beam intensity. The 'orbit' is then transferred as an array of triplets (the CF_FLTINTINT type alluded to above). An example of a 'user-defined' data type is a transfer of live pictures from CCD cameras by one single 'read' transaction (containing a video header plus a video frame). Another example is an orbit correction request issued to the 'orbit correction' server, where structures containing the current orbit, machine optics, magnet power supply settings and other beam parameters are sent to the server and the corrected 'orbit' and settings are returned with a single call.

6. TINE: application programmers' support

TINE provides application programmer interfaces (APIs) for Java, Visual Basic, C/C++, LabView, MatLab, ActiveX and scripting tools. An interface for the laboratory automation environment AgilentVee (formerly HPVee) in a Windows environment is readily available via the ActiveX support, while a native interface is being prepared. Also under preparation is a native .NET interface, applicable to all .NET and Mono [7] supported languages.

Programmers building graphical-user-interface (GUI) applications using a graphical programming environment (such as, for example, Microsoft Visual Studio or Borland C++ Builder) will most likely make use of the ActiveX controls (or their .NET equivalents). Developers using LabView will make use of the rich set of VIs which is offered, although the use of ActiveX and .NET controls in a Windows environment is also possible. Java graphical client applications can either use the TINE Java API directly or make use of the advance component oriented programmer (ACOP) components and the ACOP family of graphical beans, which are more or less equivalent to the set of ActiveX controls [8–10].

Furthermore, to make the development of server and client application easier, code-generation wizards are available. By supplying the relevant device server information, the developer launches a code-generation wizard to produce a functional C, Visual Basic or Java code project. The generated project will indeed produce a TINE server, but not be able to offer the finer logic involved in hardware readout, process control, etc. The developer can concentrate on writing his own control algorithms and incorporating them into the server by locating the TODO statements and sections in the generated code and filling them in with his application-specific code. The current TINE Server Wizard addresses only the basic server functionality and not hardware IO. Developers of TINE front-end servers can either access the server hardware on a 'do-it-yourself' basis, where the hardware readout is not in any way coupled to the TINE libraries, or may make use of the common device interface (CDI) [11] layer which can be optionally incorporated into the server project. This will be described in the following section. In any event, simulated data will be transferred from the generated server, until the developer actually modifies the code to interface to the real hardware.

7. TINE at the device tier

7.1. Common device interface

In order to provide the device server's connection to the various field buses or hardware-specific interfaces of the accelerator components, TINE offers a unified, generic CDI library. CDI hides all hardware-specific details and presents the developer with full access to the local hardware devices via the TINE client API. The local hardware registers appear to the developer as named (or numbered) devices. All details concerning hardware addresses, bus IO specifics, read-back calibration, etc are buried inside the CDI database. Different field buses are accessed via CDI through a bus 'plug', which must wrap all of the bus-specific details in a manner understood by CDI. Currently existing bus plugs include generic access methods to device electronics via SEDAC, CAN, RS232, GPIB and TwinCat [12]. The bus plugs for USB and VME buses are under construction. For those cases not yet supported by CDI, developers can either supply a bus plug themselves (a straightforward task) or follow the 'do-it-yourself' method of handling the interface to the hardware by using a hardware-specific IO API in 'C' or Visual Basic code. The current CDI and bus plug libraries exist for both Windows and UNIX. A CDI library for VxWorks is still under development and will be available soon. Servers written in Java access the CDI library via the Java Native Interface (JNI).

7.2. Embedded servers

The small footprint of TINE makes possible the creation of embedded servers. So far our experiences include successful server implementations for ALTERA-NIOS II and on PC104 [13] based system running the embedded Linux, ELINOS [14].

8. TINE at the integration tier

The integration tier hosts system-specific middle-layer servers and several TINE services.

8.1. Application-specific middle-layer servers

TINE servers can also play the role of clients to other servers, collecting data from device servers or other middle-layer servers. The aim of these servers is to provide levels of processing, coordination and management (i.e. business logic) which is either not available or practical to do directly at the front end. Although they can have very complicated processing logic, they utilize the same standard TINE server framework as classic device servers. As mentioned earlier, the source code of the TINE server skeleton can be generated by the TINE Server Wizard.

8.2. TINE connections to experiments and other control systems

Experiments being carried out on DESY accelerators are built and managed by scientists from several countries and institutes, and represent a very eclectic approach to data acquisition and availability. For the purpose of

accessing experiment data, the NETMEX (network-machine-experiment-exchange) server was developed. This is a TINE server with simplified functionality, providing 'read-only' access to experimental data. Since this is a pure TINE code, it can be compiled for all supported platforms.

TINE can also seamlessly integrate to other popular control systems, among them EPICS—using the Tine2Epics translator [15] (which essentially runs EPICS over TINE instead of Channel Access)—and DOOCS, which has embedded the TINE protocol. Since TINE is also used by KEK, Japan, there is also an interface to their STARS [16] control system and via a STARS bridge to a COACK [17] control system.

8.3. TINE services

TINE offers several central services as TINE servers. The most important of these services are given below.

- *Equipment name server.* In TINE as in any control system, user application programs (clients) make use of the services provided by device servers or middle-layer servers. Essential to smooth operation is then the locating of the services offered, i.e. server address resolution. This, in general, amounts to matching a human-readable device server name to its network address. TINE uses an address resolution mechanism, transparent to the server and client developers, which incorporates the TINE equipment name server (ENS). The ENS manages address resolution for all control points. Using the ENS also allows a 'plug and play' mechanism for servers, which, upon start-up, 'plug' themselves into the ENS database, thus obviating the need for a control system administrator to add new server addresses by hand. Similarly, address changes are verified by the ENS when a server seeks to make use of a new network address. First, a query is sent to determine if a server by the same name is still in operation under the old address. If so, an 'address in use' message is returned to the start-up server and the address change is refused. Due to the 'plug and play' feature, the TINE ENS can operate without intervention by a control system administrator. Nonetheless, if obsolete servers are not removed from time to time, the equipment database can contain a lot of unnecessary information. It might also become desirable to add 'alias' names to the equipment database or to otherwise make changes by hand. An administrator is always free to locate the database files read by the ENS and edit them by hand. The ENS checks regularly for external changes to its database and can reread its database on the fly. For the administrative database manipulations, a graphical user-interface tool is also provided.
- *Archive services.* TINE offers three different systematic ways for archiving data.
  (1) *Central archiving.* Central archiving consists of acquiring the specified data, passing them through the relevant filters and writing them to disk if warranted. Here, the 'filtering' can be adjusted to store data based on state and/or tolerance criteria. Central archiving is managed by a pair of TINE servers running on the same machine or otherwise sharing the same file system. The first server is designated as the 'archive server' and is responsible for collecting the data and committing it to disk. The second server is designated as the 'archive reader' and is responsible for handling all requests for archive data. For the archived data manipulation, a set of client viewers is offered.
  (2) *Local archiving.* All TINE servers contain a built-in module of a local history server (LHS), which can be activated by request. This means that local archiving takes place at the server, where the server can be configured to maintain an archive of specified properties both on a short-term basis (in main memory) and on a long-term basis (on disk). In this case, the filtering can only accommodate tolerance specifications. A TINE server can be queried for archived data by all other control system participants. In particular, the same toolset for history analysis can be used, as for centrally archived data.
  (3) *Event-driven archiving.* In contrast to the case of central archiving, where data are collected periodically, applying defined filtering rules, event-based archives are archive snapshots which are based on scripts specifically targeting the event in question. In our accelerator world, the event can be a message such as 'the beam was lost due to a magnet quench' (i.e. a post-mortem event) or 'proton injection complete' (i.e. a normal operational event). In such cases it is desirable to store a snapshot of the relevant machine parameters (like machine optics, current orbit, etc) at that time. A scan of 'injection' events over a time range (say, the past week) would then tell you precisely when the beam was injected and you could easily examine the stored information at those times. Event scripts can also trigger other activities by issuing commands to other servers, transient recorders, etc delivering data sampled with a higher rate.
- *Alarm services.* TINE alarms are processed at two levels. The first level is located directly at the front end and is known as the local alarm server (LAS). The LAS is a standard part of all TINE servers, independent of platform. Here, it is most easily determined whether an alarm is oscillating (coming and going), is persistent, has terminated, etc. The second level of processing occurs at a dedicated middle-layer TINE server known as the central alarm server (CAS). Here, alarms are collected, filtered, sorted and made available for client-side review. The CAS also performs several additional tasks more appropriate to central level processing. It can determine whether 'server-down' alarms need to be issued in the case of a non-responsive server (something an individual server of course cannot do). It can also take 'actions' following the receipt of particular alarms. For instance, it can issue event triggers for archiving, send e-mails, write reports, etc. The TINE alarm viewer receives data primarily directly from the CAS. Specific alarm information (descriptions, references, etc) is always retrieved from the individual servers.
- *Synchronization service.* The TINE time server is a server operating in the producer–consumer mode and produces

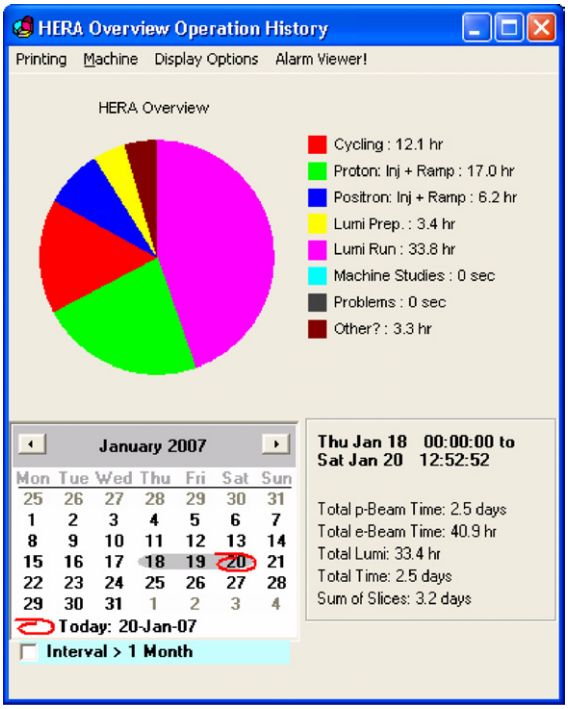

Figure 4. A summary of 3 days of operations at HERA using the archived state counters.

one quantity, namely the current TINE data time stamp. It sends via multicast the current TINE time stamp at 1 s intervals. If a TINE server is configured to intercept these multicasts (the default), it can then apply an offset to its own clock when time-stamping its data, while the local clock is not affected, so log file entries will reflect the time read by the local clock, whereas data time stamps will be synchronized with the TINE time server.

- *State server*. The current implementation of the TINE state server consists of a machine state repository. That is, it accepts transient change-of-state triggers, generates change-of-state events (used by the archiving service, for example), maintains a running count of the duration of any given state and responds to queries as to the current machine state and current machine procedure. The state information is, in turn, archived in the form of the transient state change information as well as in the form of state 'counters' which increase incrementally as long as the accelerator remains in a particular state. Archiving such counters allows an easy method of calculating the amount of time the machine was in a particular state over any particular time interval, lending itself well to producing operation history 'pie-charts' (see figure 4).

9. TINE at the client tier

At the client level TINE offers several general GUI applications, used primarily for system administration, accessing archive data, multi-channel data, analysing alarms, control system browsing, etc. These applications are available as Windows native applications or JAVA applications.

A propos 'control system browsing', the most popular TINE GUI application (known as the 'instant client'), is capable of browsing the control system to locate a device server, browsing the device server itself for exported properties and devices, and obtaining and displaying the data offered by the server. This application is especially useful during server development work, where a new or upgraded server has to be tested. The same browsing capabilities are embedded into the ACOP ActiveX controls and ACOP Java beans, which allow for the rapid application development of 'thin clients', where no programming is required of the applications developer. TINE also supports (and strongly encourages) the development of 'rich clients', where the applications developer can introduce his own coding algorithms, massage data or coordinate data from multiple sources prior to display. It should be noted that ACOP controls are first and foremost 'displayers', which means that they provide both a connection to the TINE control system and advanced data presentation and plotting functionality.

10. Closing note

Although the system was developed at DESY (although modelled to some extent on the ISOLDE [18] system at CERN in the early 1990s) it can be applied everywhere, where the Ethernet is used as a data transmission medium for data acquisition and control. It does not matter if one needs to build a network containing three PCs or several hundreds of machines with different operating systems; TINE can scale perfectly to one's needs. Note that the HERA machine is one of the largest and most complex accelerators in the world. Industrial plants and large research institutes, such as high-energy research centres, astronomical observatories, biology and chemical labs, will especially appreciate the benefits offered by the multi-platform, multi-protocol and multi-architecture nature of TINE. TINE has a small footprint and offers efficient data transfer even in extreme cases (for instance, 500 Kbyte video frames multicast at 10 Hz). For bench mark comparisons with other control systems, see [19].

The TINE source code and makefiles for many platforms can be downloaded free from the DESY website: http://adweb.desy.de/mst/tine/tineDownloads.html.

The documentation, examples and tools are available in the official TINE website: http://tine.desy.de.

For questions and comments, the authors of this paper can be contacted at their e-mail ids.

References

[1] http://tine.desy.de
[2] Schmitz R (DESY) 1999 A control system for the DESY accelerator chains *Proc. PCaPAC 99, KEK (Tsukuba, Japan)*
[3] http://www.can-cia.org
[4] http://www.profibus.com
[5] http://www.aps.anl.gov/epics
[6] http://tesla.desy.de/doocs/doocs.html
[7] http://www.mono-project.com
[8] Deloose I, Duval P and Wu H 1997 The use of ACOP tools in writing control system software *Proc. ICALEPS'97*
[9] Duval P and Wu H 2002 Acop as a Java bean *Proc. PCaPAC 2002*

[10] Duval P, Kriznar I and Wu H 2006 The Acop family of beans *Proc. PCaPAC 2006, (Jefferson Lab, Newport News (VA, USA))*
[11] Duval P 2006 Using the common device interface in TINE *Proc. PCaPAC 2006 (Jefferson Lab, Newport News (VA, USA))*
[12] http://www.beckhoff.de
[13] http://www.pc104.org
[14] http://www.sysgo.com/en/products/elinos
[15] Kakucs Z, Duval P and Clausen M 2001 An EPICS to TINE translator *Proc. ICALEPCS 2001*
[16] http://pfwww.kek.jp/stars
[17] http://coack.kek.jp
[18] Billinge R, Bret A, Deloose I, Pace A and Shering G 1991 A PC based control system for the CERN ISOLDE separators *Proc. ICALEPCS '91 (Tsukuba, Japan)*
[19] Duval P and Plesko M 2002 The Babylonization of control systems *Proc. PCaPAC 2002 (Frascati, Italy)*