ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA046

A NEW SIMULATION TIMING SYSTEM FOR SOFTWARE TESTING IN COLLIDER-ACCELERATOR CONTROL SYSTEMS*

Y. Gao†, T. Robertazzi, Stony Brook University, Stony Brook, USA
K. A. Brown, M. Harvey, J. Morris, R. H. Olsen, Brookhaven National Laboratory, Upton, USA

Abstract

Accelerators need to be operated in a timely way to successfully accelerate the beam from its creation at its source, to the experiments at its destination. Thus, synchronization among accelerator devices is important. The Collider-Accelerator Department (C-AD) of Brookhaven National Laboratory (BNL) develops their own control systems for their accelerator complex[1]. The synchronization in the C-AD control systems is accomplished by a distribution of timing signals, which are sent out along so-called[2] *time lines* [1] in the form of digital codes. Accelerator devices in the complex which require their times synchronized to the acceleration cycles are connected to the time lines. Those devices are also equipped with time line decoders [2], which allow them to extract timing signals appropriately from the time lines. In this work, a new simulation timing system is introduced, which can generate user-specific timing events for software testing in the C-AD control systems.

INTRODUCTION

Accelerator systems must be synchronized for the proper operations of equipment over a wide area. In order for the beam to have the desired properties (momentum, size, intensity, etc.), devices must act in concert, and evolve together in a particular way [2]. Hence the synchronization among accelerators, and devices within each accelerator is crucial.

In the C-AD control systems, the synchronization is accomplished by a distribution of timing signals around the accelerator facility [2]. Devices which need timing signals synchronized to the acceleration cycles are connected to the time lines. Timing signals are sent out along a time line in the form of digital codes, and those codes (representing the timing signals) can be extracted by devices (equipped with time line decoders) in the accelerator complex from the time line as signals to perform certain operations.

To better understand it, consider the following example. The Booster main magnet power supply is programmed to start to follow a reference function when it receives a time line trigger from the time line. A time line trigger is also typically called a time line event, which corresponds to a specific hexadecimal number and is given a 3-letter acronym. In this case, the hexadecimal number is 000*A*, and its acronym is *BT*0, which stands for Booster-Time-zero. This time line event triggers the start of the Booster main magnet cycle, hence this name. Moreover, *BT*0 is distributed to all devices on the time line, therefore any other device who is interested in this event will also be programmed to respond to it.

In the C-AD accelerator complex, there is an elaborate timing system to accomplish the synchronization, which provides a highly reliable, serial timing link to all equipment locations. Events and clocks derived from this link are used to initiate hardware operations including changes in settings, state changes, and data acquisition. Particularly, the synchronization is collectively conducted by three timing systems [3], the event link system, the beam-sync event system, and the Real Time Data Link (RTDL). The event link system provides a reliable serial timing link to equipment locations throughout the RHIC complex. It is a crucial component in the C-AD control systems.

This work mainly focuses on the event link system. Specifically, a new simulation timing system is proposed, which can generate user-defined timing events at specific times on specific event links. Developers can use this simulation system to interact with timing-sensitive applications for testing purposes, hence improving the reliability of the controls timing system.

A more detailed motivation is presented.

Motivation

The occurrences of timing events on the C-AD event links affect the running of controls software in many ways. Some particular timing events trigger the executions of some software methods directly. Other events (such as PPM[3] user codes) establish a context that affects the way software operate.

In the front end level, Front End Computers (FECs) detect events by VME boards with direct connections to an event link. In the console level, ADO managers[4] and other console level processes receive notifications over the network from FECs. FECs use "relMon" ADOs to deliver notifications of events as they happen on the event link. For each event link, a special "evMon" ADO delivers regular reports that summarize all the events that have occurred during a Supercycle (a 4 seconds time period on the RHIC event link, see details in the next section).

In order to test software thoroughly, the software should be run in a variety of timing conditions. For example, we sometimes need to arrange a time to test a piece of software when multiple PPM users are active at the same time in the

* Work supported by Brookhaven Science Associates, LLC under Contract No. DE-SC0012704 with the U.S. Department of Energy.
† ygao@bnl.gov
[1] The complex includes the Linear Accelerator (Linac), the Electron Beam Ion Source (EBIS), the Tandem Van de Graff pre-accelerators, the Booster accelerator, the Alternating Gradient Synchrotron (AGS), and the Relativistic Heavy Ion Collider (RHIC).
[2] These time lines are like timing buses inside a computer.
[3] Pulse to Pulse Modulation, which will be introduced later.
[4] Console level servers which hold ADOs. ADOs stand for Accelerator Device Objects, see next section for details.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA046

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Supercycle or when the Supercycle is particularly long or particularly short. This can be difficult to do in practice because changes made to the Supercycle affect the entire complex. Since console level servers and applications learn about event link activities by notifications over the network, they do not necessarily require changes to the events on the actual event link. A "Virtual Event Link Server (velServer)" could deliver "relMon" notifications and "evMon" reports that simulate the timing conditions desired for software testing.

A virtual event link server would be a special ADO manager. The expected functionalities are:

- Each virtual event link server would support one "evMon" ADO and multiple "relMon" ADOs for some selected sets of events;
- It would accept a definition of all the events in the user-specific event links as an ordered list of events code-time pairs;
- It would repeatedly "play back" all the events in the right order and approximately at the right time (looped fashion);
- When any of the "relMon" events occurs, the virtual event link server would send an asynchronous report to any client with asynchronous requests;
- At the end of the event list, the virtual event link server would send an "evMon" report summarizing all the events that had occurred in the event list.

PRELIMINARIES

In this section, we will introduce some basic conceptions in the C-AD control systems that are related to the proposed simulation timing system.

Accelerator Device Object

One of the most fundamental concepts in the C-AD control systems is the Accelerator Device Object (ADO) [4]. The ADO model is a flexible way to view accelerator equipment. It was introduced during the development of the RHIC control system in the late 1990s. One of the primary goals of the ADO concept is to establish unified standards for controls software development, which automates the integration of device level controls into the overall control systems and simplifies the coding process.

ADOs provide developers ways to control the accelerator complex. They (instances of C++ or Java classes) abstract features from the underlying controls devices into a collection of collider control points (also known as parameters), and provide those parameters to the users of the control systems. Each parameter can possess one or more properties to better describe the characteristics of the devices. ADO designers determine the number and names of parameters based on the needs of the system.

Two of the most important ADO class methods for device control are the *set()* and *get()* methods. They act as the interface to access controls hardware. The accelerator complex is controlled by users or applications which set() and get() parameter values in instances of the ADO classes. ADOs also support event codes, which will execute some specific codes whenever the defined events happen. This is the mechanism of generating simulated timing events in the proposed simulation structure.

Since the crafting of ADOs is so crucial, special development tools are used. The source file of an ADO is typically stored in a file with a ".rad" extension, which stands for RHIC ADO Definition file. A preprocessor called "adogen" is used to transform ".rad" file source codes into C++ codes [5]. It takes care of the necessary details and allows ADO designers to focus on the more important parts, i. e. the controls interfaces, which are the set(), get() methods and event codes.

Controls Name Server

The Controls Name Server (CNS) provides [4] a centralized repository where unique name/value pairs can be efficiently managed and queried[5]. In C-AD control systems, given an object's instance name[6], the CNS will provide enough information so that the associated data can be accessed.

The CNS is session oriented which means several copies of it can be run at the same time as long as each of them has its own host. This feature allows developers to have a "private" CNS. This makes it possible to signal a process to look for an ADO instance in a different place from where it normally resides. The simulation framework introduced in this work leverages this property to redirect applications to interact with simulated ADOs instead of real ADOs.

Supercycle

One of the major concepts in C-AD timing system is called the "Supercycle". It is a very important timing mechanism to the accelerator complex, which enables the machines to act in concert to accelerate the beam from its source to its destination. A Supercycle typically consists of a set of events from which all other events are delayed in one way or another. In this sense, the Supercycle timing is considered the highest level of timing [2].

The events that generally compose the Supercycle are one or more Tandem or Linac cycles, one or more Booster cycles, and one AGS cycle. The granularity of these events is 1/60 second (this time length is also called one "Jiffy"). The length of the Supercycle is usually about 3 to 5 seconds long.

The software application that is used to control the Supercycle is called "SuperMan", which will be introduced in the next section.

Pulse to Pulse Modulation

Another important concept in the timing system is called Pulse to Pulse Modulation (PPM). PPM refers to the fact that

[5] Function likes a Domain Name Server (DNS) [6].

[6] That object can be an ADO parameter, a Complex Logical Device (CLD), a manager's parameter name, or an alias (a name used by developers which is more human-readable), etc.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA046

machine cycles within the same Supercycle, for the same machine[7], can be used differently [2].

The function of an individual PPM user is to control different machine cycles that occur on the Supercycle. A PPM user can be configured to allow a certain number of cycles to occur on one machine before allowing another PPM user to begin on the same machine, or while overlapping with another PPM user on a different machine.

Each machine usually has multiple users, and each user can be identified by a specific color in software used to control PPM devices. Moreover, user settings can always be found in the controls system, which allows quick changes to be made to the machines' setup by switching or adding PPM users as necessary.

To better illustrate the concept, consider the following example. Suppose PPM user 1 has a Booster cycle within a Supercycle that is used as part of the acceleration cycle for the physics program. During the time the AGS is accelerating the beam for the physics program, the Booster may be idle. However, that time within the Supercycle could be used to study Booster phenomenon. Therefore, another PPM user, say user 2, could be configured accordingly with a group of setpoints and timings for this purpose. This user can be set to occur during the time that the Booster would have been idle, by programming it with the appropriate time in the Supercycle.

System Tools

SuperMan It is a program to set up and make active a Supercycle. It reads and dictates the live magnet function through the controls system and displays the information to the users. The program allows the users to predict what changes need to be made in order to alter the Supercycle. The changes will be verified by the program and proceeded if they are acceptable. The user can then load the changes and give the machine a new Supercycle.

EventLinkDisplay It is a program to display event data for all PPM users in event links. Users can select an event link they want to monitor, and then acquire the event information (including event names, event times, etc.) on that link in real-time for once or continuously. The result is displayed in a list format with those event information. The event information are fetched from the underlying "evMon" ADOs' reports, which makes it convenient to verify the simulation results by showing the simulated timing events' information on the "EventLinkDisplay" through a private CNS.

SIMULATION ARCHITECTURE OVERVIEW

The simulation architecture is shown in Fig. 1. The virtual event link server (the "velServer" block) holds the "relMon" and "evMon" ADOs, interacting with the applications that need event triggers.

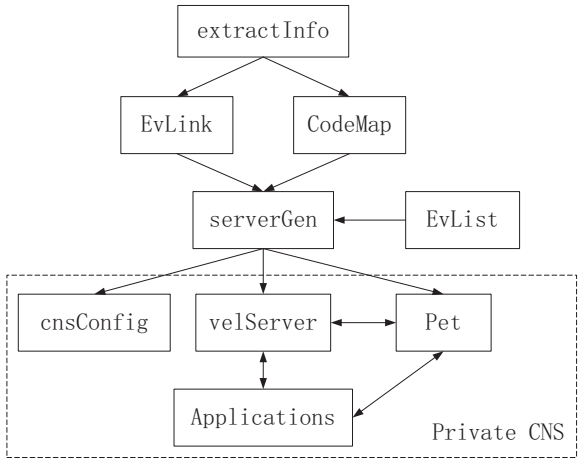

Figure 1: The architecture of the virtual event link server.

The "relMon" and "evMon" ADOs are simulated ADOs. They have the same alias as the real ADOs, so that the existing applications can directly interact with them without changing their codes. On the other hand, the simulated ADOs only reside in the local hosts so as not to interrupt the overall system. The "private" copies of CNS on those local hosts will have their real ADOs' entries replaced with the corresponding simulated ADOs'. Then the applications running on those local hosts will communicate with the simulated ADOs properly. The exact ADOs to simulate are decided based on the users' demands.

To better understand the simulation structure, we first introduce each of its function block in a top-down manner, then describe the simulation procedure.

Build the Configuration Files

Currently, there are four event links that can be simulated, i. e. the AGS event link, the Booster event link, the EBIS event link and the Linac event link. First, users need to run "extractInfo" to extract information about those event links from the database. Those information will be used to run the simulation later. The information are stored in the "EvLink" and "CodeMap" files.

One "EvLink" file is created for each event link, so a total of 4 are created. Each file stores information about all the "relMon" ADOs on that event link. For each "relMon" ADO, the following information are recorded:

- The 3-letter[8] name of the event that the ADO is responsible for (such as *BT*0);
- The corresponding event number (such as 10);
- The ADO's generic name;
- The ADO's alias.

Take the "EvLink" file for Booster for example. One entry it contains is "BT0 10 belMon.simBT0.10 belMon.bt0", where "BT0" is the 3-letter acronym, 10 is the event num-

[7] Here machine refers to the accelerator complex, e. g. AGS, Booster, Linac, etc.

[8] Some event names have more than 3 letters, but most of the names are 3 letters.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA046

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ber, "belMon.simBT0.10" is[9] the ADO's generic name and "belMon.bt0" is the ADO's alias.

Each "EvLink" file also contains event code maps for that event link, which includes the correspondences between event names and event codes. For the "BT0" example, the code map entry is "BT0 10".

The "CodeMap" file stores the code maps for all the Supercycle events. Note that, a Supercycle usually contains events from other event links. Therefore, an event from an event link could have a different event code in the Supercycle. In such cases, the original event codes are applied in the simulation. "BT0" event has the same event code both in the Booster and the Supercycle, thus its entry in "CodeMap" is the same "BT0 10".

Software Generator

Since one server file ("velServer") only supports a specific set of ADOs (which depends on the users' demands), to accommodate various users' demands, a new server file needs to be created for each new user's request. To facilitate the building of such server files, the "serverGen" block acts as a software generator [7] which automatically produces the server files for users.

The event list for the simulation is provided by users in the file "EvList". This file contains what events users want to generate and at what times. The information is organized as pairs. For example, the first line of an "EvList" file could be "SCS 1", which means Supercycle starts at 1 "Jiffy". The second line could be "BT0 4", which means Booster-Time-zero event starts after 3 "Jiffies". The file can contain many other events based on the users' demands.

The "EvList" file can come from an existing SuperMan's history file (for simulating Supercycle events) or can be created based on users' demands. Specifically, users can write it in a plain text file or it can come from the logged data for the selected event link. If no file is specified, a SuperMan's history file[10] is used as the default event list file.

The "serverGen" loads the "EvList" file and uses the information from the "EvLink" and "CodeMap" files to generate the server file, which will be built later and serves as the virtual event link server to interact with applications (by sending event triggers at specific times defined by the users). Note that, multiple "EvLink" files (a total of 4) can be loaded in the same simulation, in which case multiple event links are being simulated.

Control the Events Generation Process

Once the server file is generated, users can build it and start the virtual event link server. The server sends out event triggers to all the asynchronous clients at appropriate times.

The "Pet" is a file that is used to create a customized PET page for the virtual event link server. The PET page lists all the events that are in the server. For each event, the PET page lists its event name, event code, a switch, and times at which this event will be triggered. The switch gives users an easy way to turn on and off a particular event on the simulated event link. For an event that is currently turned on, users can also edit its event times to specify when it appears on the simulated event link. The PET page is expected to act as a major user interface developers could use to conveniently control the events generation process.

Simulation Procedure

First, users run the script "extractInfo" to gather event link information and build the Supercycle code map. Note that this script only needs to be run one time before the simulation starts, since the information can be reused for all the simulations.

Next, users need to specify what events they want to simulate and at what times. They do it by creating a "EvList" file. For simulating Supercycle events, existing SuperMan's history files can be used directly as the event list file. Otherwise, users can either write one in a text file or use the logged data. If no file is specified, a default SuperMan's history file is used. Either way, the file is in a list-of-pairs format. The first element of the pair is the event name or the event code, the second element is the event time at which the event is going to be triggered[11].

After having all the necessary files, the "serverGen" starts to generate the server file. The outputs are "velServer", "cnsConfig", and "Pet".

velServer It is the source code of the virtual event link server, which holds the simulated "relMon" and "evMon" ADOs based on the users' demands. Once the simulation starts, the virtual event link server triggers events at specific times according to the information in the "EvList" file. The events are triggered by calling the ADOs' "EVENT-CODE" [5]. If logged data are used, the server invokes another script "ReadEventLinkLog" to fetch the logged event information[12]. The event list is played in a looped fashion.

cnsConfig The simulation is performed on a private CNS so as not to interrupt the normal operations of the facility. The private CNS directs clients to communicate with the simulated ADOs, which are held by the virtual event link server to generate event triggers. The "cnsConfig" file is used to configure the private CNS so that it gets the necessary information to set up the entries for the simulated ADOs.

Pet As introduced above, the "Pet" file is used to create a PET page, which could serve as the main user interface for developers to interact with the virtual event link server. Through the PET page, users can decide whether or not to

[9] The "simBT0" in the middle of the name implies that this is a simulated ADO, in order to differentiate it from the normal ADO.

[10] A SuperMan's history file is chosen as the default value for it already possesses a proper format that can be directly loaded and used by the simulator.

[11] The unit for the time can be in "Jiffy" or "microsecond", and it represents the time difference that is relative to time 0.

[12] "ReadEventLinkLog" is a C++ application in C-AD to fetch logged event data.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA046

trigger a particular event, and also specify at what times this event can be triggered. Any change made in the current event loop will start taking effect from the next loop.

One thing that is worth mentioning, the PET page renders the simulation structure the flexibility of controlling events on the link without the need of recreating the server's source file. This could be very useful in cases where users just want to modify the existing events that are in the simulation. This also indicates that if all the supported events on a particular event link are listed in the "EvList" file, then users just need to run the "serverGen" once to generate a server file that has the ability to simulate any event on that event link. In this way, as long as doing simulations on that event link, the server file can be reused. Users only need to operate through the PET page to meet the simulation needs.

Simulation results can be verified through viewing the event information in the "EventLinkDisplay" (by selecting the event links that are in the simulation), using the private CNS that is configured with the "cnsConfig".

SUMMARIES

In this work, we propose a new simulation timing system which aims to generate user-specified timing events for software testing in the C-AD control systems. It can be used in various timing-related testing scenarios. For example, to test programs which have PPM features, we can apply the simulator to create timing conditions for different PPM users to verify each one's functionality. It could also be possible to make the virtual event link server to be synchronous with the real Supercycle, so that developers can do experiments using the simulated Supercycle without interrupting the normal operations of the facility.

REFERENCES

[1] Bus (computing), https://en.wikipedia.org/wiki/Bus_(computing)

[2] K. Zeno, "A Rookie's Guide to Booster Operations", BNL, Upton, USA, Rep. BNL-105273-2014-TECH, Sep. 1998.

[3] B. R. Oerter, "Accelerator Timing at the Relativistic Heavy Ion Collider", in *Proc. ICALEPCS'99*, Trieste, Italy, Oct. 1999, paper MC1P06.

[4] D. S. Barton *et al.*, "RHIC control system", *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 499, no. 2–3, pp. 356–371, Mar. 2003.

[5] R. H. Olsen, L. Hoff, and T. Clifford, "Code Generation of RHIC Accelerator Device Objects", in *Proc. ICALEPCS'95*, Chicago, USA, Oct.-Nov. 1995, paper W-PO-58.

[6] Domain Name System, https://en.wikipedia.org/wiki/Domain_Name_System

[7] Automatic programming, https://en.wikipedia.org/wiki/Automatic_programming

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.