Fermilab

Accelerator Division/LHC Accelerator-Fermilab Software (LAFS)

18 September 2006
16:57

SEQUENCED DATA ACQUISITION FOR THE LARGE HADRON COLLIDER

DRAFT

Timofei Bolshakov
Suzanne Gysin
Elliott McCrory
Dennis Nicklaus

Abstract

This document describes Sequenced Data Acquisition (SDA). Based on SDA developed for Fermilab, we propose a possible implementation of SDA for the LHC.

Revision History

Version	Date	Comments
0	18 Sept 2006	First Draft, written at Fermilab.

Table of Contents

Figures

1. INTRODUCTION

Sequenced Data Acquisition (SDA) is a special purpose data logging system that has been developed at Fermilab. This system collects and stores data according to the stages of the accelerator cycles or modes, instead of using wall clock time. It organizes and presents the data so people and programs can read and search it easily. Once the data are organized, presented, and distributed, it becomes a valuable tool to analyze the accelerator's performance or to track down errors and inefficiencies. In short, SDA organizes the accelerator data to facilitate analysis.

More specifically, SDA defines a temporal abstraction for acquiring and storing data about a repeated process, like a collider store. This abstraction encapsulates the progression of time through the process by breaking it down into a specific, repeated series of stages or modes. Data are collected within each of these stages and are stored in a database indexed by the stages. The stages have descriptive names, in addition to a numerical index, facilitating understanding of and communication about the process.

The heart of SDA is the data acquisition and its indexed database. Additionally, SDA provides a suite of application programs, scripts and Application Programming Interfaces (API). These components, which use the collected data and monitor the data flow, are:

- SDA Viewer: a web-based application that allows people to browse SDA data.
- SDA Editor: a web-based application for creating the data acquisition rules
- Store Checker: an application for checking the data within SDA and sending out email notifications if values exceed limits
- Super Table: a spreadsheet that has a row for each instance of the process and contains the most popular data. The table is available in many formats: Excel, HTML, AIDA and ASCII Comma Separated Values formats.
- API: Application Programing Interface to access SDA data programmatically for creating plots, doing special analyses, etc.

SDA is intended to be a general tool, applicable to any repeated process. However, in this document we explore the possibility of applying SDA to the LHC by considering the specific functionality that would be needed to record the progression of the LHC through the modes and cycles that create and maintain colliding beams at the experiments. The ideas presented in this document are intended to serve as a basis for discussion, with a goal of developing a fully functional SDA for the LHC.

This document is divided into four main topics: a description of SDA, an explanation of the advantages of SDA, a description of SDA components, and a discussion of the issues and tasks associated with adapting SDA to the LHC.

2. DESCRIPTION OF SDA

SDA is a special purpose logging system for collecting and storing data for a repeated, multistage process. The data are collected during the process at specific times that are relevant to the temporal evolution of the process, and are stored and indexed by these time markers. The data acquisition is based on customizable and extensible rules. These rules assume that the recorded process has been broken down into a repeated sequence of stages, similar to the states and transitions of a Finite State Machine.

The SDA system consists of several distinct software components that communicate the SDA database and with the user. These components are described in section 4.

The abstractions defined by SDA have been given names at Fermilab. These terms, which are used in this document, are:

- "Shot"—the name of the process for which data are being logged. The shot is designated by a sequence number, which is the index into the data storage for that instance of the process. This term is also called a "store" or a "file."
- "Case"—the stage or mode within the shot. A case has a descriptive name and an index number. A case occurs once and only once within a given shot. The order of the cases is usually repeated in each shot, but this is not necessary. Nor is it necessary for every case to the present in every shot.
- "Set"—a further temporal division within a case; the smallest time division available. A set also is designated by a sequence number. The number of sets can be fixed (as in the number of proton bunches that are injected, one set per injection (if that is relevant)), or variable (for example, a new set may be generated every ten minutes during the physics portion of a store).
- "Event"—the time marker or the condition for data collection. This can be a hardware event generated externally by the accelerator systems or a set of conditions that are met. An event is necessary to trigger the acquisition of a "collection."
- "Collection"—the list of "atoms" that are to be acquired on an event.
- "Atom"—the datum that is to be collected. Different types of atoms can be defined. These include "device" (a scalar reading or an array of readings from a specific accelerator component or sensor), "snapshot" (a fast chunk of data collected at a specific machine time) and "fast-time-plot" (a series of readings from a device over a short time period). Other types of atoms could be, for example, "image", "movie" or "post-mortem-data".

Store, Case, Set and Event are the temporal abstractions within the SDA system, while Collection and Atom are the data abstractions.

To illustrate the SDA abstractions, we present some of the processes that are recorded by SDA at Fermilab and the names assigned to them. Some of the types of shots that are recorded at Fermilab are:

- Collider Shots,
- Accumulator Antiproton Shots To The Tevatron,
- Recycler Antiproton Shots To The Tevatron and
- Accumulator to Recycler Antiproton Shots.

Some of the instances of these shots are linked to each other in SDA. For example, every Collider Shot contains some sort of shot of antiprotons to the Tevatron.

Each of these shots is subdivided into cases that are repeated each time the shot happens. For example, the main cases in a Collider Shot for the Tevatron are:

- Proton Injection porch, 1
- Proton Injection tune up, 2
- Eject Protons, 3
- Inject Protons, 4
- Antiproton Injection porch, 5
- Inject Antiprotons, 6
- Before Ramp, 8
- Acceleration, 9
- Flattop, 10
- Squeeze, 11

- Initiate Collisions, 12
- Remove Halo, 13
- HEP, 14
- Pause HEP, 15

For further information on SDA and how it is implemented at Fermilab, see http://www-bd.fnal.gov/sda and http://lhc001.fnal.gov/SDAII.

3. ADVANTAGES OF SDA

SDA has been an important part of the success of the Tevatron. It allows for the observation of high-level trends in the accelerator complex and for a common medium and vocabulary for discussing low-level details of the process.

Some of the advantages we have gained from SDA at Fermilab are described in this section.

3.1 ANALYSIS OF OVERALL ACCELERATOR PERFORMANCE

SDA enables the development of general, qualitative observations about the accelerator processes. Most results are meaningful only if they can be expressed in some qualitative way, for example: "during fiscal year 2005 we integrated 621.848 inverse picobarns, which is much better than our base plan and almost as good as our most optimistic plan", or "Recycler antiprotons work better than Accumulator antiprotons", or "this shot was almost free from beam-beam effects". SDA is used to help draw conclusions like these. It helps to create reports, to communicate between groups and to locate bottlenecks in the current system.

Since the data in SDA are collected and stored based on specific, named time markers, it facilitates the analysis of the evolution of and communication about the shot. For example, when someone asks the question, "What was the emittance of the 23rd proton bunch at initiate collisions for shot number 4223?" This statement has a very specific meaning because of SDA. Furthermore, since the same cases are repeated within each shot, accurate comparisons can be made easily at a specific stage of the process for several shots.

SDA allows for the quick creation of summary plots, which has been a staple for the evaluation of operations at Fermilab. A sampling of this, generated automatically from SDA data, can be seen at http://mccrory.fnal.gov/tevatronDecayFits.

Details provided by SDA also help to solve difficult technical problems that cross group boundaries, for example, transmission efficiencies. SDA provides common methods for calculating and stating the details of problems and it allows people from different groups to solve complicated problems together.

3.2 ERROR/FAILURE ANALYSIS OF THE ACCELERATOR

Error and failure analysis of shots are enhanced by SDA in several ways:

- The Store Checker issues alerts about failures or problem with SDA data, even when everything seems to be normal. For example, it can mark a negative emittance as an error.
- Analysis of data from the SDA Viewer allows the investigator to understand a failure quickly by providing a detailed view of the evolution of that system.
- Special error analysis shots (that is, a special collection triggered by a specific error) provides information about specific, predefined failures. A particular benefit

of this approach could be to provide meaningful structure to a post-mortem data dump, in parallel to a more general post-mortem system.

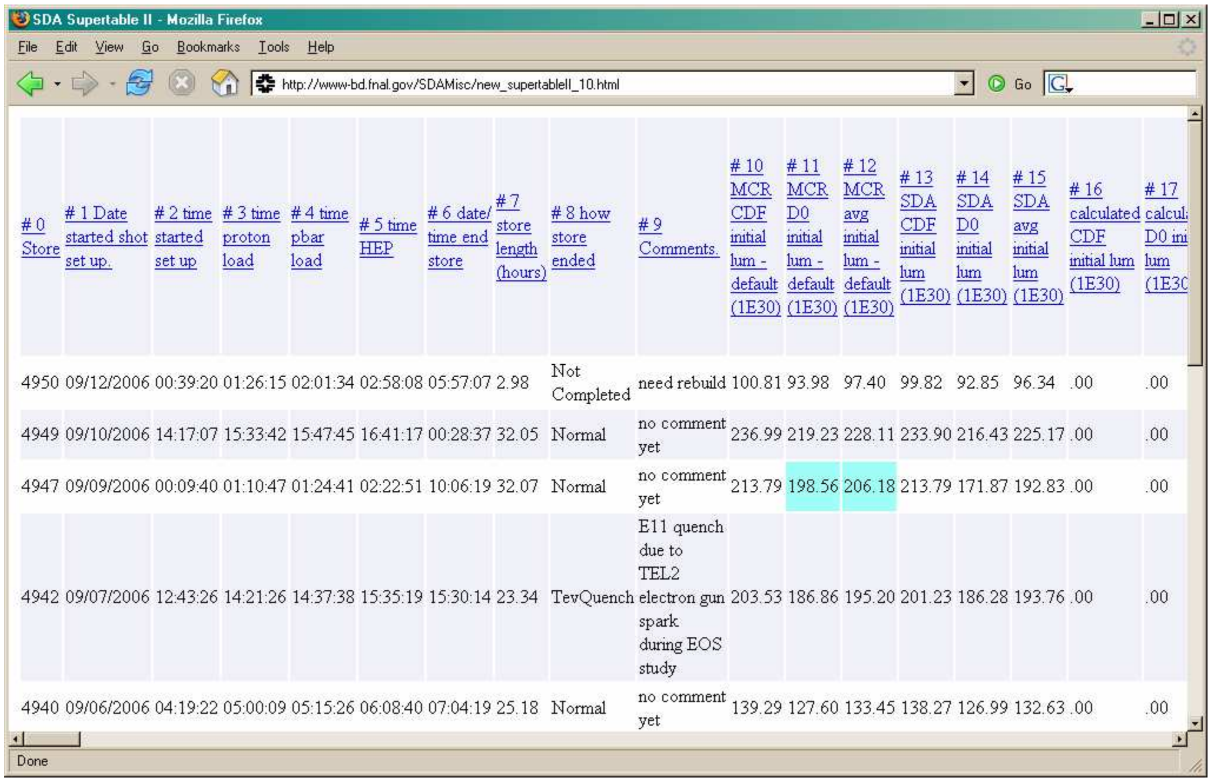
- Computational tables; see below.

3.3 COMPUTATIONAL/SUMMARY TABLES

Computational/summary tables, derived from SDA, have proven to be an effective and popular way to use SDA data. These tables, usually provided as a spreadsheet that can be used with programs like Excel or Java Analysis Studio (JAS), allow anyone to view quickly the status and the details of the process. These tables contain two types of data: summary data taken directly from SDA (e.g., the initial luminosity at an interaction region), and data computed from SDA data (e.g., luminosity lifetime). One prominent use of these tables is to create summary plots that are distributed widely: luminosity plots, integrated luminosity plots, antiproton intensity plots and proton intensity plots.

Organization of SDA data into tables allows a convenient way to access data quickly and meaningfully. When the validity of the tables is accepted, these tables become the basis on which performance is discussed. Since SDA is controlled and maintained by scientists and engineers who are not involved directly with the beam sensors, these people are the liaisons between the people who generate the signals and the people who use them. Since the contents of these tables can be regenerated easily, the specifics of a calculations can change. The most notable of the time-dependent calculations at Fermilab has been the emittances in the Tevatron—it has been and continues to be difficult to agree on the beta functions at the flying wires!

The table that has had the most wide-spread use during the Tevatron Collider Run has been the “Super Table.” This table has one row per collider shot and is available in many formats: Excel, HTML, AIDA and ASCII Comma Separated Values formats. Different columns represent summary information about different properties of accelerator complex. This figure shows a screen shot of the HTML version of the Super Table.


SDA Supertable II - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www-bd.fnal.gov/SDAMisc/new_supertableII_10.html
Go
Done

# 0 Store	# 1 Date started shot set up.	# 2 time started set up	# 3 time proton load	# 4 time pbar load	# 5 time HEP	# 6 date/ time end store	# 7 store length (hours)	# 8 how store ended	# 9 Comments.	# 10 MCR CDF initial lum - default (1E30)	# 11 MCR D0 initial lum - default (1E30)	# 12 MCR avg initial lum - default (1E30)	# 13 SDA CDF initial lum (1E30)	# 14 SDA D0 initial lum (1E30)	# 15 SDA avg initial lum (1E30)	# 16 calculated CDF initial lum (1E30)	# 17 calcul D0 ini lum (1E3
4950	09/12/2006	00:39:20	01:26:15	02:01:34	02:58:08	05:57:07	2.98	Not Completed	need rebuild	100.81	93.98	97.40	99.82	92.85	96.34	.00	.00
4949	09/10/2006	14:17:07	15:33:42	15:47:45	16:41:17	00:28:37	32.05	Normal	no comment yet	236.99	219.23	228.11	233.90	216.43	225.17	.00	.00
4947	09/09/2006	00:09:40	01:10:47	01:24:41	02:22:51	10:06:19	32.07	Normal	no comment yet	213.79	198.56	206.18	213.79	171.87	192.83	.00	.00
4942	09/07/2006	12:43:26	14:21:26	14:37:38	15:35:19	15:30:14	23.34	TevQuench	E11 quench due to TEL2 electron gun spark during EOS study	203.53	186.86	195.20	201.23	186.28	193.76	.00	.00
4940	09/06/2006	04:19:22	05:00:09	05:15:26	06:08:40	07:04:19	25.18	Normal	no comment yet	139.29	127.60	133.45	138.27	126.99	132.63	.00	.00

Figure 1. A Screen Shot of the Super Table

The Super Table currently has 245 columns and thousands of rows (one row per collider store). Most people use the Excel version of the Super Table, but the AIDA version, coupled with JAS, is also very popular.

The Super Table allows for:

- Quick, qualitative understanding of overall performance: which system has problems and where performance can be improved. Having the Super Table available in Excel format has been particularly useful.
- Detailing how the accelerator complex performed during a good shot.
- Creating detailed performance plots like initial luminosity and integrated luminosity versus time, or correlation plots, like the emittance of the protons versus the luminosity lifetime.
- Creating cells that are predictions, based on beam measurements. For example, one interesting cell is the luminosity of the Tevatron as calculated from accelerator measurements; we compare this to the luminosity as measured by the experiments to give us information on the beta functions at the experiments.

The data from the Super Table form the basis for reports Fermilab sends to its funding agency, the Department of Energy. DoE officers have a direct link to the automatically generated performance plots and they check our work daily. This enhancement of communications to our funding agency has been important on a variety of levels, including the establishment of trust to this oversight organization.

More information on the Super Table, including the current version, can be viewed at http://www-bd.fnal.gov/sda/supertable.

There are a number of other computational tables that are used at Fermilab. Several of these tables are automatically included into our "shot scrapbook" – an automatically generated, specialized log book on each shot, created at the request of the operators and written automatically by SDA. Those tables include:

- Recomputed Emittances table,
- Recomputed Intensities table,
- Intensities by transfer 1 (Accumulator view) and Intensities by transfer 2 (Recycler view),
- Emittances by transfer for "Accumulator to Recycler Transfers".

These tables are popular because they represent a common view of several groups to some particular problem or set of problems. (It is interesting to note that the last of these tables, "Accumulator to Recycler Transfers," has been controversial because, at least initially, these two groups disagreed with virtually every number in that table. As time has progressed, they have developed a common language and can now agree on most of these numbers.)

Every table is accompanied by a complete, automatically generated description of the computational algorithm that has been used—one can trace how each table cell has been computed down to the elementary device readings. This increases the credibility of the system and decreases the number of calls to developers with question like "how did you come up with these results?"

It is worth reiterating the sociological role of these tables as they pertain to accelerator performance. These tables provide a common view of the data for different working groups and often serve as a basis for a common language among these groups. In this way SDA has a noticeable influence by improving communication, and thereby contributing to better performance.

4. SDA COMPONENTS

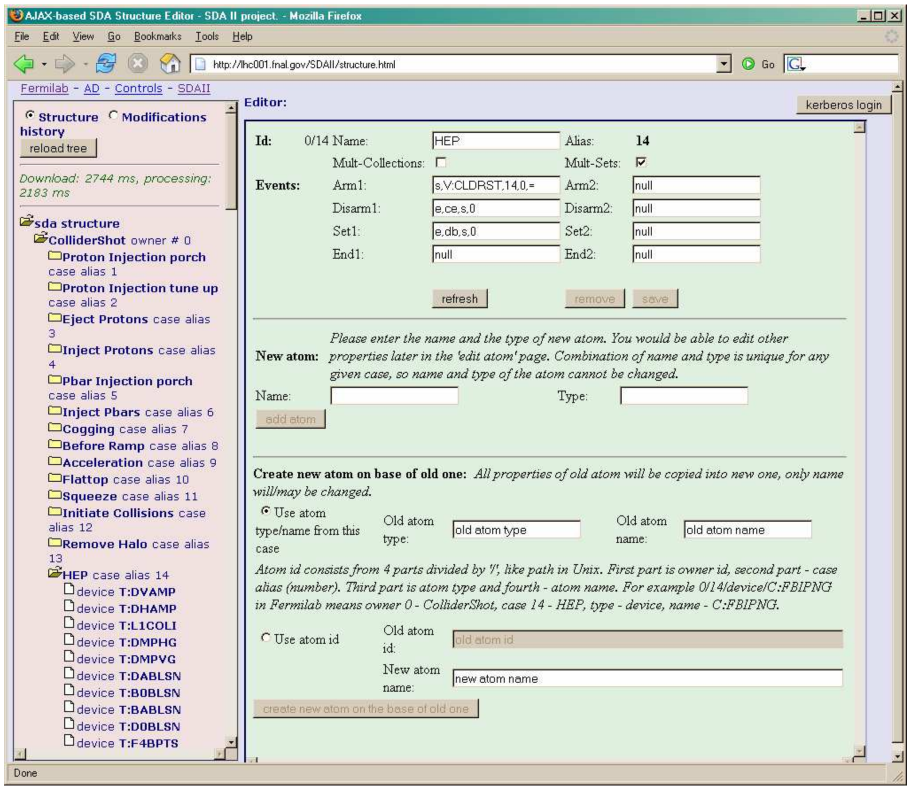
We present here a list of the pieces of the current SDA implementation.

The SDA server and database server run on a PC running Scientific Linux. The DAQ server is implemented in Java 1.5. The name of the class package is gov.fnal.sda. The data acquisition is based on the DAE/SCF technology developed at Fermilab. Data flow within SDA is encapsulated in XML documents. The database chosen for this version is Berkeley DB XML, a native XML database built on top of Berkeley DB. It is imagined that the entire SDA system at LHC will be contained within one PC, provide by Fermilab, on the Technical Network.

The distinct components of SDA are:

- The data acquisition for the local control system.
- The SDA Database, for storing the collected data.
- The SDA Viewer, for browsing the data in the database.
- The SDA Editor, for setting up the data acquisition.
- A Report Editor, for creating automatic summaries of the data.
- The Store Checker, for verifying that the data being collected pass certain simple tests.
- XML Schema for data flow, for the structure of the database and for the returned data.
- An API, for allowing programmatic access to the SDA database. The high-level APIs in JavaDoc format, in the working version currently at Fermilab, can be found at http://lhc001.fnal.gov/SDAII/api/doc.
- A scripting language, built on top of the API.
- Summary and Computational Tables, derived from the SDA data.
- An AIDA/JAS interface to the SDA data.

For example, here is a screen shot of the SDA Structure editor.


AJAX-based SDA Structure Editor - SDA II project. - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://lhc001.fnal.gov/SDAII/structure.html
Go
Fermilab - AD - Controls - SDAII
kerberos login
reload tree
refresh
remove
save
add atom
create new atom on the base of old one
Done

● Structure ○ Modifications history

Download: 2744 ms, processing: 2183 ms

sda structure
ColliderShot owner # 0
- **Proton Injection porch** case alias 1
- **Proton Injection tune up** case alias 2
- **Eject Protons** case alias 3
- **Inject Protons** case alias 4
- **Pbar Injection porch** case alias 5
- **Inject Pbars** case alias 6
- **Cogging** case alias 7
- **Before Ramp** case alias 8
- **Acceleration** case alias 9
- **Flattop** case alias 10
- **Squeeze** case alias 11
- **Initiate Collisions** case alias 12
- **Remove Halo** case alias 13
- **HEP** case alias 14
  - device **T:DVAMP**
  - device **T:DHAMP**
  - device **T:L1COLI**
  - device **T:DMPHG**
  - device **T:DMPVG**
  - device **T:DABLSN**
  - device **T:B0BLSN**
  - device **T:BABLSN**
  - device **T:D0BLSN**
  - device **T:F4BPTS**

Editor:

Id:	0/14 Name:	HEP	Alias:	**14**
	Mult-Collections:	☐	Mult-Sets:	☑
Events:	Arm1:	s,V:CLDRST,14,0,=	Arm2:	null
	Disarm1:	e,ce,s,0	Disarm2:	null
	Set1:	e,db,s,0	Set2:	null
	End1:	null	End2:	null

New atom: *Please enter the name and the type of new atom. You would be able to edit other properties later in the 'edit atom' page. Combination of name and type is unique for any given case, so name and type of the atom cannot be changed.*

Name: Type:

Create new atom on base of old one: *All properties of old atom will be copied into new one, only name will/may be changed.*

● Use atom type/name from this case Old atom type: old atom type Old atom name: old atom name

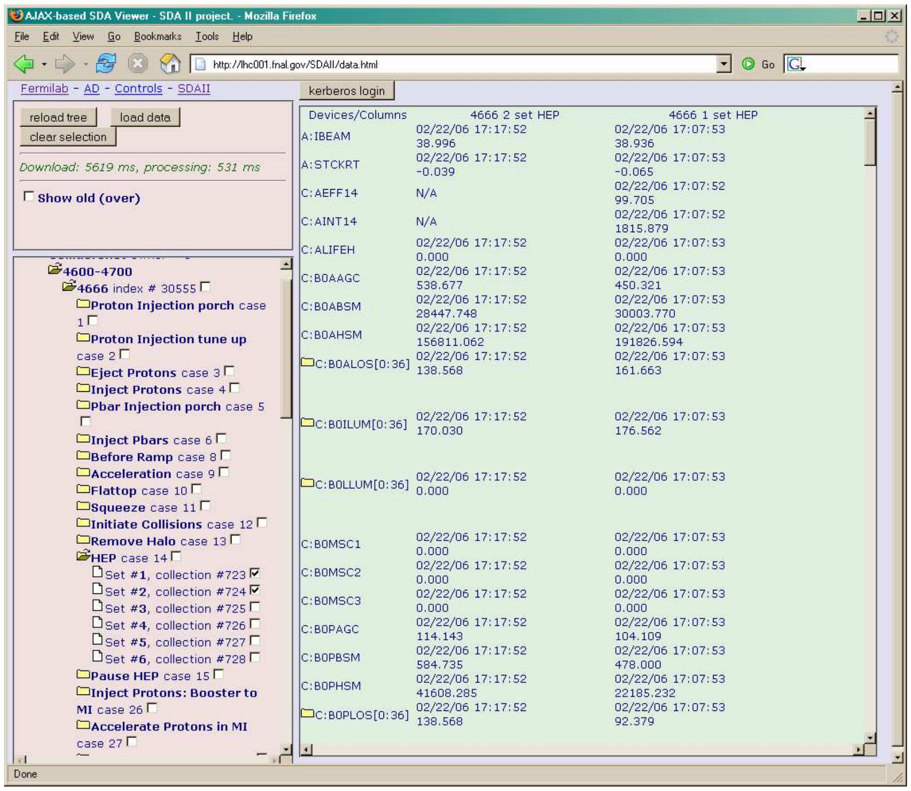
Atom id consists from 4 parts divided by '/', like path in Unix. First part is owner id, second part - case alias (number). Third part is atom type and fourth - atom name. For example 0/14/device/C:FBIPNG in Fermilab means owner 0 - ColliderShot, case 14 - HEP, type - device, name - C:FBIPNG.

○ Use atom id Old atom id: old atom id

New atom name: new atom name

Figure 2. Screen shot of the SDA Structure Editor.

And here is a screen shot of the SDA Viewer, looking at data from the initial part of a recent store.


AJAX-based SDA Viewer - SDA II project. - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://lhc001.fnal.gov/SDAII/data.html
Go
Fermilab - AD - Controls - SDAII
kerberos login
reload tree
load data
clear selection
Done

Download: 5619 ms, processing: 531 ms

☐ **Show old (over)**

4600-4700
4666 index # 30555 ☐
- **Proton Injection porch** case 1 ☐
- **Proton Injection tune up** case 2 ☐
- **Eject Protons** case 3 ☐
- **Inject Protons** case 4 ☐
- **Pbar Injection porch** case 5 ☐
- **Inject Pbars** case 6 ☐
- **Before Ramp** case 8 ☐
- **Acceleration** case 9 ☐
- **Flattop** case 10 ☐
- **Squeeze** case 11 ☐
- **Initiate Collisions** case 12 ☐
- **Remove Halo** case 13 ☐
- **HEP** case 14 ☐
  - Set **#1**, collection #723 ☑
  - Set **#2**, collection #724 ☑
  - Set **#3**, collection #725 ☐
  - Set **#4**, collection #726 ☐
  - Set **#5**, collection #727 ☐
  - Set **#6**, collection #728 ☐
- **Pause HEP** case 15 ☐
- **Inject Protons: Booster to MI** case 26 ☐
- **Accelerate Protons in MI** case 27 ☐

Devices/Columns	4666 2 set HEP	4666 1 set HEP
A:IBEAM	02/22/06 17:17:52 38.996	02/22/06 17:07:53 38.936
A:STCKRT	02/22/06 17:17:52 -0.039	02/22/06 17:07:53 -0.065
C:AEFF14	N/A	02/22/06 17:07:52 99.705
C:AINT14	N/A	02/22/06 17:07:52 1815.879
C:ALIFEH	02/22/06 17:17:52 0.000	02/22/06 17:07:53 0.000
C:B0AAGC	02/22/06 17:17:52 538.677	02/22/06 17:07:53 450.321
C:B0ABSM	02/22/06 17:17:52 28447.748	02/22/06 17:07:53 30003.770
C:B0AHSM	02/22/06 17:17:52 156811.062	02/22/06 17:07:53 191826.594
C:B0ALOS[0:36]	02/22/06 17:17:52 138.568	02/22/06 17:07:53 161.663
C:B0ILUM[0:36]	02/22/06 17:17:52 170.030	02/22/06 17:07:53 176.562
C:B0LLUM[0:36]	02/22/06 17:17:52 0.000	02/22/06 17:07:53 0.000
C:B0MSC1	02/22/06 17:17:52 0.000	02/22/06 17:07:53 0.000
C:B0MSC2	02/22/06 17:17:52 0.000	02/22/06 17:07:53 0.000
C:B0MSC3	02/22/06 17:17:52 0.000	02/22/06 17:07:53 0.000
C:B0PAGC	02/22/06 17:17:52 114.143	02/22/06 17:07:53 104.109
C:B0PBSM	02/22/06 17:17:52 584.735	02/22/06 17:07:53 478.000
C:B0PHSM	02/22/06 17:17:52 41608.285	02/22/06 17:07:53 22185.232
C:B0PLOS[0:36]	02/22/06 17:17:52 138.568	02/22/06 17:07:53 92.379

Figure 3. Screen shot of the SDA Viewer

5. ADAPTING SDA FOR LHC

To adapt SDA for the LHC, several design choices need to be made. Possible preliminary design choices are presented here.

5.1 DESIGN ISSUES AND CHOICES

At this stage of the development of SDA, some design assumptions have been made. This section presents the issues we are facing at this time—some of them we have solutions for, and some we do not.

5.1.1 Suggested Design Choices

All the code will be written using Plugins to maintain the generality of the core parts of SDA. In particular this means that, (1) interfaces that require only simple types will be designed so that the implementing classes do not need to know about the other parts of the system, and (2) there will be an external mechanism to register the plugin.

Data flow: The data sent to the SDA database is encapsulated in an XML document. The data retrieved from the database is returned as an XML document.

We imagine that SDA data acquisition will be done by a stand-alone PC on the Technical Network at CERN. This PC also can be the database server, but this duty may be split out if throughput becomes an issue, or if it is desired to use a relational database instead of the XML database.

The SDA API is implemented in a client/server manner. The server side accesses the database and returns the results through a Java Server Pages (JSP) mechanism. This JSP server complies with the Java Servlet Container specification, currently at 2.5. Nothing on this server is specific to any JSP engine (like Tomcat or Jetty). The clients access the JSP server via XML over HTTP through a mechanism like SOAP, XML-RPC or plain XML.

Sleepy Cat Software's Berkeley DB XML is used as the data storage for the first version of the system. DB XML is being used because one of the more fragile parts of the current version of SDA has been the XML/relational DB interface. The ability to switch to a relational DB is available through plugins.

AJAX has been used for the first version of the support software. The applications may be written in Java at a later date, if this is desired.

Beanshell is the first scripting language for SDA. More scripting languages will be added with the release of Java 6.

5.1.2 Design Considerations

One of the first things that need to be determined is the time structure of the LHC cycles. This structure can change over time (with experience, or as the operation of the accelerator changes). We are interested in helping the LHC with these definitions, since we believe that we have some relevant experience with the Tevatron.

Code Management: The code that represents SDA will be managed in the conventional manner with a tool like CVS. The code that represents SDA should be kept at CERN, but we will need to access it from Fermilab.

What processes will be recorded by SDA? Any repeated process in the entire LHC complex, including the injectors, could be recorded by SDA. These processes need to be identified.

5.2 "TO DO" LIST FOR SDA

Several tasks for implementing SDA have been identified. Here are some of them.

- A requirements document.
- A design document.
- The data acquisition. It seems prudent to have a small team of people from Fermilab and from CERN (maybe 1 FTE a month or so spread over 2 people from each lab) to write the data acquisition plugins. Experience with implementing the Fermilab DAQ plugin will be relevant.
- The SDA Database. While the version we are preparing uses the Berkeley DB XML from Oracle and Sleepy Cat Software, it has been expressed that having a unique, operationally significant database may be inappropriate. Since the data flow is defined by and implemented in XML, it should be straight forward to translate this flow into a relational database.
- Atom definitions and implementations. The data atoms necessary to contain LHC data need to be designed and implemented.

(This list is certainly going to grow!)

6. SUMMARY

SDA collects and organizes shot data around the temporal flow of the shot, and thereby enhances data analysis and trouble shooting. We have identified advantages of SDA, listed the components, and suggested possible tasks and design issues to implement SDA for the LHC. Some of these advantages are:

- SDA is a data acquisition system and a database that collects and stores data on time markers that are relevant to the process being followed.
- It is implemented modularly so that the various pieces of the system do not need to understand very much about the other parts.
- It has several widely useable tools for collecting, examining and analyzing the data and for allowing interested people to craft the system to best suit their needs.
- SDA has been a resounding success at Fermilab and has been a critical aspect of the success of the Tevatron program.

We look forward to discussions about SDA and welcome interaction about and modifications to this proposal. A possible next step would be to develop a requirements document for implementing SDA for the LHC.

REFERENCES

"Portable SDA (Sequenced Data Acquisition) With a Native XML Database", T. Bolshakov, E. McCrory, EPAC '06, http://accelconf.web.cern.ch/AccelConf/e06/Pre-Press/THPCH128.pdf. A presentation of this work can be seen at http://lhc001.fnal.gov/SDAII/CERN_presentation.ppt

Fermilab SDA: http://www-bd.fnal.gov/sda

Fermilab Super Table: http://www-bd.fnal.gov/sda/supertable

SDA-II: http://lhc001.fnal.gov/SDAII; SDA-II API: http://lhc001.fnal.gov/SDAII/api/doc.

AJAX: Asynchronous JavaScript and XML. A programming architecture used extensively by Google for live web pages. See http://en.wikipedia.org/wiki/Ajax_%28programming%29.

AIDA: Abstract Interfaces for Data Analysis, http://aida.freehep.org. From the web site, "The goals of the AIDA project are to define abstract interfaces for common physics analysis objects, such as histograms, ntuples, fitters, IO, etc."

JAS: Java Analysis Studio, http://jas.freehep.org. From the web site, "A general purpose, open-source, data analysis tool."

Beanshell: A scripting language that implements syntax essentially identical to Java and runs within the Java Virtual Machine. See http://en.wikipedia.org/wiki/Beanshell

Sleep Cat Berkeley DB XML: A direct implementation of an XML database. See http://www.sleepycat.com/products/bdbxml.html. Sleepy Cat has been purchased by Oracle.

GLOSSARY

Shot The name for the entire process that is being recorded. This term is also called "store" or "file."

Case The stage within the shot. A case occurs once and only once within any given shot. The order of the cases is usually repeated, but this is not necessary. Nor is it necessary for every case to the present in every shot.

Set A further temporal division within a case; the smallest time division available. The number of sets can be fixed (as in the number of proton bunches that are injected, one set per injection (if that is relevant)), or variable (for example, a new set may be generated every ten minutes during the physics portion of a store).

Collection The list of "atoms" that are to be collected at the outset of a case or a set.

Atom The datum that is to be collected. It is up to the facility to define the types of atoms that are relevant. Plugins are used to control the handling of an atom – its collection, representation etc. Also called a "variable."

Event The time or the condition for data collection.

Plugin A programming style that separates disparate aspects of a complex computer program system. They are widely used in modern web browsers, and we borrow that term from there. See http://en.wikipedia.org/wiki/Plugin.

APPENDIX 1, CONTACT INFORMATION

Timofei Bolshakov, Fermilab Accelerator Division, Accelerator Controls, 630-840-8034, tbolsh@fnal.gov.

Suzanne Gysin, Fermilab Computing Division, Accelerator Modeling R&D, 630-840-8334, gysin@fnal.gov.

Elliott McCrory, Fermilab Accelerator Division Headquarters, *Group Leader,* 630-840-4808, mccrory@fnal.gov

Dennis Nicklaus, Fermilab Accelerator Division, Accelerator Controls, 630-840-6410, nicklaus@fnal.gov.

APPENDIX 2, SDA DATA API

```
public class ApiBase {
    // general utility methods
    static public String resolveOwnerName(int id);
    static public int resolveOwnerId(String name);
    static public String resolveCaseName(int owner, int caseAlias);
    static public int    resolveCaseAlias(int owner, String name);
    // data classes
    public static class Store{
        public int getFileAlias();
        public int getOwnerId();
        public String getOwnerName();
        public Case getCase(int caseAlias);
    }
    public static class Case{
        public int    getCaseAlias();
        public String getCaseName() ;
        public int getOwner();
        public void bind(Store str);
        public Store getStore();
        public void addVariable(Variable var);
    }
    public static class Variable{
       public String getName();
       public Case   getCase();
       public boolean isMultiSet();           // trigger data collection
       public VariableData getData();         // trigger data collection
       public VariableData getData(int set);  // trigger data collection
       public int numSets();
       public int lastSet();
       public int firstSet();
       public boolean isEmpty();
    }
           // data interface
    public static interface VariableData {
    public Object  getObjectData();
    public boolean is(String type);
    public long    getTimestamp();
    public double  op(String whatOp);
    public boolean isError();
    public String  getErrorShortDescription();
    public String  getErrorLongDescription();
}
           // factory methods
           public static Store getStore(String ownerName, int store)l
           public static Store getStore(int owner, int store);
           public static Store getStore(int store);
           public static Store getStore(String store);
           public static Case getCase(String owner, String caseName);
           public static Case getCase(int owner, int store, int caseAlias);
           public static Case getCase(String owner, int store, String caseName);
           public static Variable getVariable(Case _case, String type, String name, String set);
           public static Variable getVariable(Case _case, String name, String set);
           public static Variable getVariable(int owner, int store, int caseAlias, String name,
                        String set);
           public static Variable getVariable(String owner, int store, String caseName, String
                        name, String set);
           public static Variable getAcnetVariable(Case _case, String name);
           public static Variable getAcnetVariable(Case _case, String name, String set);
           public static Variable getAcnetVariable(String owner, int store, String caseName, String
                        name, String set);
           // cache control
           public static void clear(int store);
           public static void clearAll();
           // plugin interface
           public static abstract class AtomConverter {
                  public Variable getVariable(Case _case, String name, String set);
                  public abstract Variable      getVariable(Case _case, String name, String set,
                        Map parameters);
                  public abstract VariableData  produceVariableData(Variable v, String atomContent,
                        long ts, Map <String, String>atomAttributes);
           }
           public static void registerAtomConverter(String type, AtomConverter ac);
           public static AtomConverter getAtomConverter(String type);
}
```

APPENDIX 3, XML SCHEMA

The XML for the data structure is as follows:

```
<shot type='1' alias='4008' index='8672' ...>
...
  <case name='HEP' alias='14' ...>
...
    <set alias='25' collection_index='425' ...>
...
      <atom name='C:B0ILUM' time='...'>
...   data ...
      </atom>
...
    </set>
...
  </case>
</shot>
```

The structure of the SDA data collection XML specification is as follows.

```
<structure>
      <owner  id='{0,1,2,3}' name='{ColliderShot, ...}' test='B'
            current='true'>
          <case alias='N' name='Name' mult-collection='B' mult-sets='B' arm1='E'
                   arm2='E' disarm1='E' disarm2='E' set1='E' set2='E' end1='E'
                   end2='E'>
              <atom name='name1' type='{device, snapshot, fast-time-plot, ...}'
                   desc='description' request='device-str' event='Event'/>
                  ...
          </case>
                  ...
      </owner>
      ...
</structure>
```

Modifications to this structure will be preserved with this sort of XML schema:

```
    <modifications>
      <re-export from='string-source' timestamp='L' by='user'/>
      <inserted timestamp='L' by='user'>
            <path>owner/case/</path>
            <atom>
                  ...
            </atom>
      </inserted>
      <removed timestamp='L' by='user'>
            <path>owner/case/</path>
            <atom>
                  ...
            </atom>
      </removed>
      <changed timestamp='L' by='user'>
            <path>owner/case/...</path>
            <old-value> ... </old-value>
            <new-value> ... </new-value>
      </changed>
    </modifications>
```