History Objects

P. Calafiura and S. Rajagopalan

16 October 2001 - Draft Version 0.1

Abstract

This working document describes our understanding of the requirements for Atlas History objects and provides a conceptual design for them. As part of our Data Model we want to introduce three related History objects: the *JobHistory* to describe when and where a job was run, using which release of the software; the *EventHistory* that will refer to the JobHistory and will also allow to determine which input event data were used, and which conditions and environment databases. Finally, the *DataObjectHistory* will contain specific information on which Algorithms made a data object, and references to its Properties as well as to the EventHistory object of the containing event.

Motivation

The purpose of the History objects is to capture and persistify the conditions under which any given data object was produced, allowing to configure a job to reproduce that data object.

The DataObjectHistory will also allow downstream Algorithms to select among multiple instances of a given data object based on their producer Algorithms (e.g. among a ClusterCollection produced using a cone algorithm and one produced by a nearest neighbour algorithm). At least in principle, the DataObjectHistory also allows to support a "reconstruction on demand" execution model, where the producer Algorithm execution is triggered by StoreGate upon request of a data object produced by that producer Algorithm instance.

JobHistory

We need input from the production bookkeeping and database groups on that

Contents

- platform (hardware and op sys ID)
- production date
- producer (user ID or production ID)
- atlas software release

EventHistory

See also DB architecture document[1]. We need the DB folks to help us understand what the DB information below really is. One possible approach is the CDF one, where each DB table has its own version and a "table of tables", the ValidSet*, indexes a complete, validated set of tables that can be used for production or analysis.*

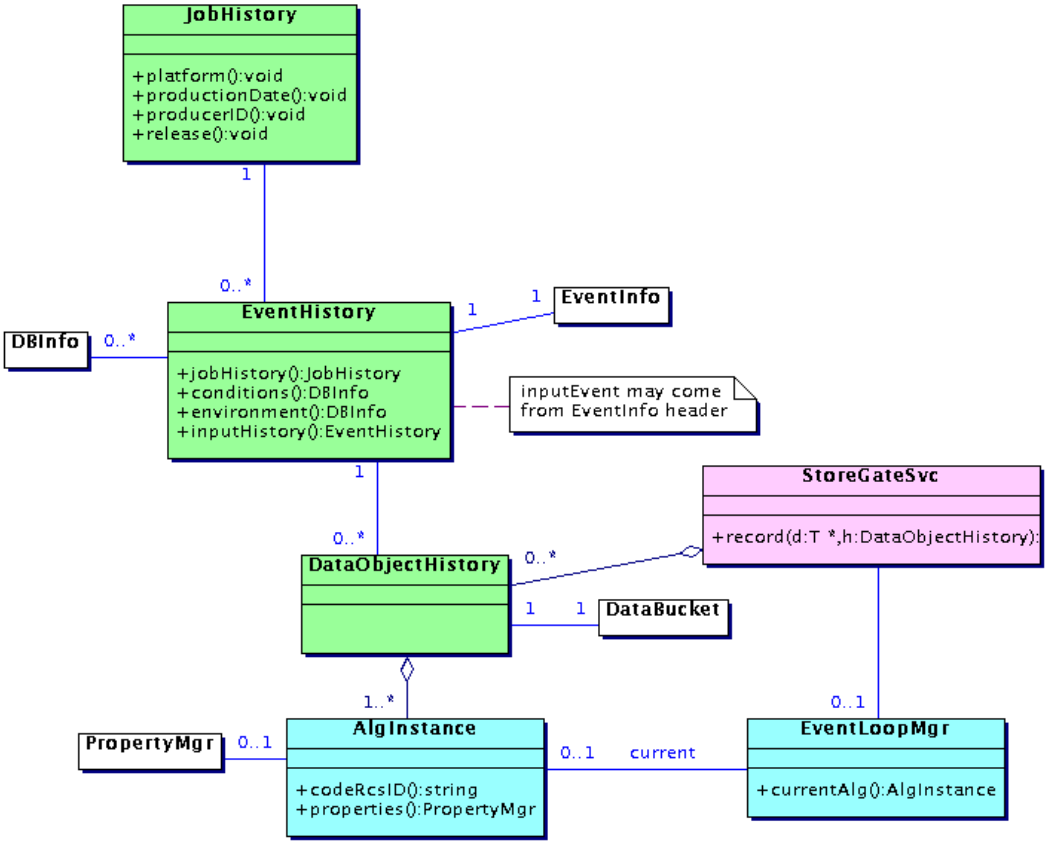

Figure 1: History objects entity diagram

Contents

- Conditions DB information (ValidSet)
- Environment DB information (ValidSet)
- Input Data Sources (may be part of the Event header)
- (a reference to) the JobHistory object of the producer job.

DataObjectHistory

We need input (from the Reco EDM group?) to provide use cases. There are already some at [2, 3]

Each data object in the TDS is associated with a DataObjectHistory object. The DataObjectHistory object can be used as a key to select a data object instance. There must be an efficient way to navigate back from the data object to its history

Contents

The DataObjectHistory contains (references to):

- the producer Algorithm(s) class version (presumably the cvs version)

- the PropertyMgr object that holds the producer Algorithm instance(s) configuration.
- the EventHistory of the containing event.

Creation

A default DataObjectHistory object that captures the configuration of the producer Algorithm is attached by StoreGate to any recorded DataObject. To avoid doing this explicitly in each `StoreGateSvc::record` invocation, we need some kind of "execution context" with a reference to the Algorithm instance that is currently running. The athena `EventLoopMgr'` should provide that.

When a data object is the result of a collaboration among several (sub-)Algorithms, the Property info of each of the producers must be appended to the DataHistoryObjects.

The producer Algorithm can override this default object with another instance (e.g. with a derived one which contains additional informations, such as a different version of the Calorimeter calibration) which would have to be explicitly associated to the data object, presumably when this gets recorded into the store.

Identification and Lifetime Management

Like any other data object, history objects live in the transient store. Their lifetime is strictly dependent on the lifetime of the data objects they are attached to, hence it seems natural to store the history objects alongside their data objects.

A unique identifier for the DataObjectHistory object is necessary. This can also be used as the default key with which we record a data object. This identifier can be read in together with the IOpaqueAddress - the DataObjectHistory object will be read in only on demand, much the same as the data object. A mechanism to generate this unique identifier still needs to be worked out. 32 bits probably is not sufficient.

Using the DataObjectHistoryObject

Since the DataObjectHistory identifies the data object is associated with, it must be possible to use a DataObjectHistory instance as a key in `StoreGateSvc::retrieve`.

`StoreGateSvc::retrieve` may also take as an argument a predicate to select data objects based on the contents of the DataObjectHistory object. A simple example of such a selector would be one that matches the name of the data object producer Algorithm with a StringProperty specified in the job options. Notice that this selector needs to load from persistency only the history objects and not the associated data objects.

Persistence

The DataObjectHistory object can be read on demand with the usual DataProxy/converter mechanism. It is worth noticing that the history object may be persistified even if its associated data object is not.

The SG+DB groups will provide converters for the default DataObjectHistory class.

It remains to be decided who owns what in persistent form. For example should the algorithm properties be persistified with the DataObjectHistory object or should they be stored in a separate "Property" database and the DataObjectHistory object only contains a reference to them.

What can we have for 3.0.0

need feedback from reco group on priorities

We don't expect to have anything more than a crude DataObjectHistory. No persistence support. Most likely the aforementioned mechanism to create a default DataHistoryObject automatically from StoreGate won't be in place. In this case it would be the Algorithm

developers responsability to explicitly instantiate (or retrieve and update) an history object and to associate it to the recorded data object explicitly using the API that later on will be used to override the default creation.

We hope to provide the selection mechanism to get a data object given its producer algorithm.

This limited scope will at least provide a testing ground.

References

[1] `http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/event_store/ev-arch-index.html`

[2] `http://www.usatlas.bnl.gov/~dladams/data_history/`

[3] `http://www.hep.ph.rhbnc.ac.uk/atlas/newsw/and/hlt-design-draft-0.9.ps`