An Object-Oriented Approach to Data Acquisition System Simulation and Modeling

A.W. Booth, M. Botlo, J. Dorenbosch, R. Idate, E.C. Milner, V.S. Kapoor, C.C. Wang & E. Wang
Superconducting Super Collider*
Dallas, TX, 75237

Abstract

This paper describes an object-oriented approach to the simulation and modeling of data acquisition systems for high-energy physics experiments. Using the MODSIM II object-oriented discrete-event simulation language, we have studied the behavior of front-end circuits, data collection circuits, event-building networks and multi-level triggers. Results include a comparison between different "pull" and "push" schemes, in terms of deadtime, throughput, and efficiency as they are influenced by data rates and system architecture.

Introduction

The Superconducting Super Collider Laboratory (SSC) is a new high energy physics laboratory where we expect to be performing experiments by the year 2000. We will search for new physics processes occurring in the collisions of two counter-rotating beams of 20-TeV protons in the 87-km long accelerator ring. The expected data rates, volume, and complexity will be unprecedented in the history of science. In this paper we discuss the design and results of simulation studies of data acquisition (DAQ) systems for collecting these experimental data.

The typical experimental apparatus at the SSC will be a large, complex ensemble of five to ten distinct sub-systems: central tracking (Si micro-strips, straw tubes, scintillating fibers, etc.), calorimetry (electromagnetic and hadronic), muon tracking (wire chambers, timing detectors), triggering (multiple levels), and data acquisition. Sub-system channel counts will range from the small (about 1,000) to the large (about 5,000,000). There will be differences in the data acquired from the sub-systems for each event. For example, it may be necessary to record for each calorimeter channel multiple samples of digitized data, while for Silicon micro-strips each channel gives only one bit of hit information.

The Collider will provide a beam crossing every 16 ns, with a mean of 1.6 collisions per crossing (100 mb cross section) occurring at the design luminosity of 10^{33} cm^{-2} s^{-1}, giving a 100 MHz interaction rate. We can expect data for each event to be about 0.4 MByte, giving a raw data rate of 40 TByte/s. Combined trigger rejection factors of 10^6 to 10^7 would reduce the data recording rate to 4 to 40 MBytes/s. Because of these high rates from large detectors, we anticipate the DAQ systems being more complex than those operating in present-day experiments. The need for simulation and modeling of these complex systems is clear, but the approach and the choice of abstraction level are open to the system designer.

Background

It is well known that object-oriented design is based on "information hiding", and differs from the functional approach to design in that it views a software system as a set of interacting objects, with their own private state, rather than a set of functions. "Information hiding" means that as much information as possible is hidden within the objects themselves, and objects interact with other objects by providing and requesting services.

* Operated by Universities Research Association under contract to the U.S. Department of Energy

It is important to be clear about what an object actually is because it governs one's whole approach to dealing with objects and being disciplined about their use. The best definition we have found is in *Sommerville, 1992 [1]:* -"An object is an entity which has a state (whose representation is hidden) and a defined set of operations which operate on that state. The state is represented as a set of objects attributes. The operations associated with the object provide services to other objects which request these services when some computation is required. In principle, objects communicate by passing messages to each other which initiate operations. A message has two parts: (1) the name of the service requested by the calling object, (2) copies of information from the calling object which are required to execute the required service and, perhaps, the name of a holder for the results of the service execution"

DAQ Objects

In any object-oriented design, the classification of objects is a very important step. Figure 1 shows a partial inheritance tree for some of the objects in our DAQ system model. The objects "NamedObj", "TriggerSignalObj", "ParameterObj" and "ChipObj" are all general purpose objects which define a structure and behavior common to various objects in the DAQ system, and were actually designed with the idea that they would be "inherited" by other objects. The object "DCCObj" inherits all these properties and adds some of its own behavior to them. One very important part of the DCC's behavior is the "ProcessEvent" METHOD, which is also needed by the front-end object ("FEObj") and the Segment Output Buffer Object ("SOBObj"). Three of our objects are now listed in detail with their attributes and messages so that the reader can get a flavor of the design. Comments are enclosed in { }.For a detailed description of these objects and the DCC simulation see *Booth et al. [2].*

```
NamedObj = OBJECT;
        { NamedObj has the following attributes}
                name            : STRING;     {name of this trigger}
                index           : INTEGER;  {index}
                namInd          : STRING;  {name[index]}
                idString          : STRING; {name[index]   :   (fixed length)}
        { NamedObj has the following messages}
                ASK METHOD ObjInit;
                ASK METHOD InitName(IN nam : STRING);  {set  name}
                ASK METHOD AddIndex(IN ind : INTEGER);{name becomes name[index]}
                ASK METHOD InitIndex(IN ind : INTEGER);  {set index}
                ASK METHOD TimNam() : STRING; {returns sim time and IdString}
                ASK METHOD InitDebug(IN lev : INTEGER); {sets debug levels}
                ASK METHOD SetReport(IN lev : INTEGER); {sets report levels}
                ASK METHOD ObjTerminate;
END OBJECT {NamedObj};

ChipObj = OBJECT(TriggerSignalObj);
        { ChipObj has the following attributes}
                myLayer                          : LayerObj; {where parameters for chips reside}
                 inbuf                            : ARRAY INTEGER OF TranspObj;
                opport                           : TranspObj;
                myerror                           : INTEGER;
                notmyerror                        : INTEGER;
                allerrors                          : INTEGER;
                goodEventCount                    : INTEGER;
                badEventCount                     : INTEGER;
                eventsPerSecond                  : REAL;
                instantaneousEventsPerSecond  : REAL;  {cm}
                instantaneousGoodEventCount  : INTEGER;
                chipL2id                         : INTEGER;
```

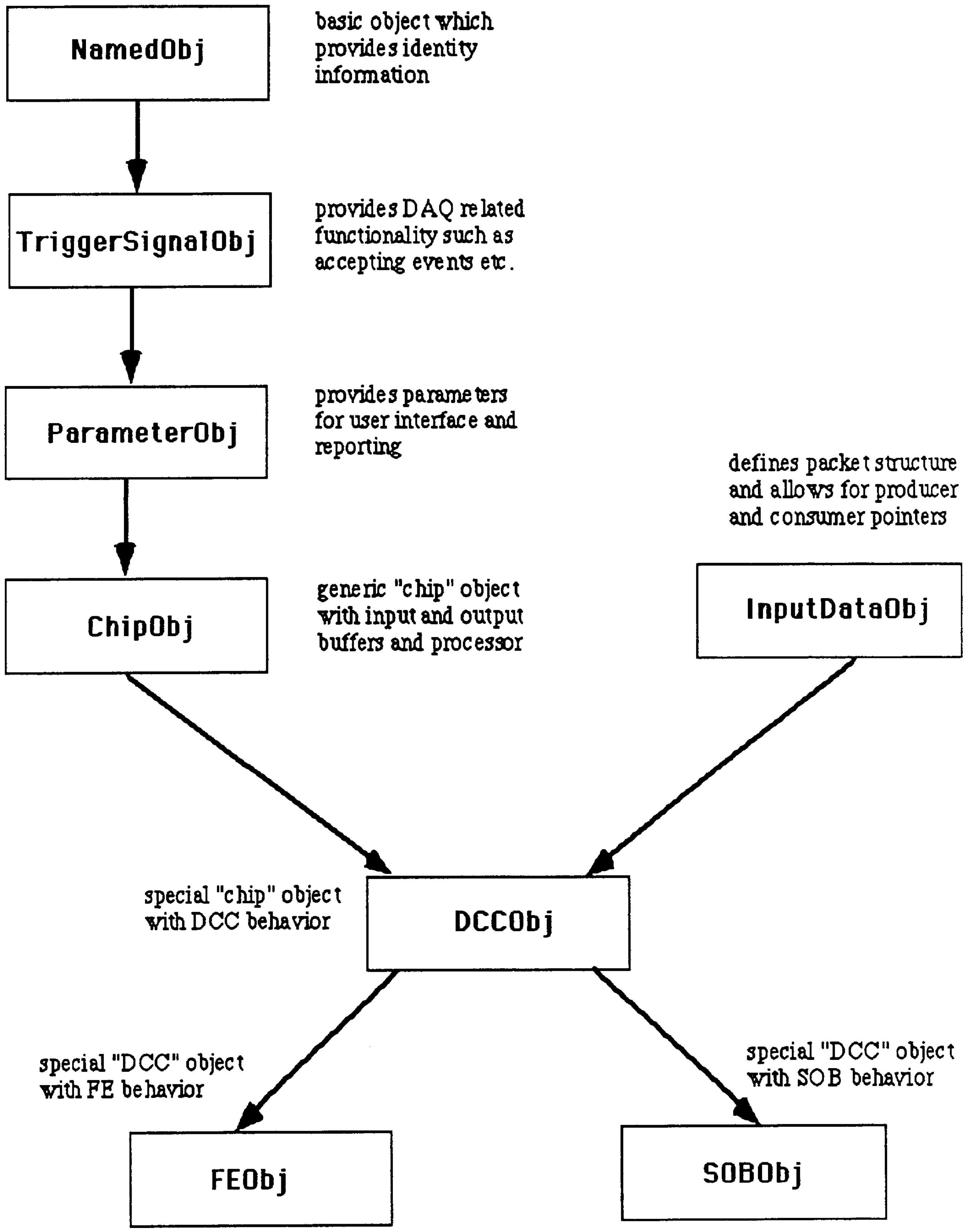
NamedObj
basic object which provides identity information
TriggerSignalObj
provides DAQ related functionality such as accepting events etc.
ParameterObj
provides parameters for user interface and reporting
ChipObj
generic "chip" object with input and output buffers and processor
defines packet structure and allows for producer and consumer pointers
InputDataObj
special "chip" object with DCC behavior
DCCObj
special "DCC" object with FE behavior
special "DCC" object with SOB behavior
FEObj
SOBObj

Figure 1. Partial Inheritance Tree for DAQ Objects

```
{ ChipObj has the following messages}
                ASK METHOD SetMyLayer(IN l : LayerObj);
                ASK METHOD Configure;
                ASK METHOD InitBufferDebug(IN lev : INTEGER);
     { ChipObj also overrides the following messages it inherited from TriggerSignalObj}
     OVERRIDE
        ASK METHOD ObjInit;
        ASK METHOD ObjTerminate;
        ASK METHOD Report;
        ASK METHOD EndWarmUp;
        ASK METHOD Reset;
        ASK METHOD Regulate;
        ASK METHOD CheckToResume;
END OBJECT; {ChipObj}

DCCObj = OBJECT(ChipObj,TranspObj);
     { DCCObj has the following attributes}
                destination         : TranspObj;
                 dccOpbusy          : BOOLEAN;
                 buildbusy          : BOOLEAN;
     { DCCObj has the following messages}
                ASK METHOD SetDestination(IN d:TranspObj);
                ASK  METHOD SetConsumerQMaxBytes(IN bytes : INTEGER);
                TELL METHOD ProcessEventWithImaging(IN in : INTEGER);
END OBJECT; {DCCObj}
```

DCC Simulation

The extensive simulation of the DCC has been reported elsewhere, [2] &[3] for example. Results include plots of trigger rate against throughput for various "pull" and "push" architectures.

References

[1] I. Sommerville, Software Engineering, Addison -Wesley Publishers, 1992.

[2] A. Booth, M. Botlo, J. Dorenbosch, R. Idate, E. Milner, V. Kapoor, E. Wang & C. Wang., “Simulation Studies of Data Acquisition Systems at the Superconducting Super Collider”, SSCL-SR-1148, 1992.

[3] E. Milner, et al, “Data Acquisition Systems at the Superconducting Super Collider”, Proceedings of the 7th Conference on Real-Time Applications in Nuclear, Particle and Plasma Physics, Julich, Germany, 1991.