Overview of Online Software

Murrough Landon – 7 November 2001

http://www.hep.ph.qmw.ac.uk/~landon/talks

Overview

- Introduction
- Core Components
- Other Components
- Installation

Introduction

Scope and History

- Online Software (previously known as Backend Software) handles everything about the online system apart from the high speed transfer of events and event fragments (ie dataflow).
- One of the five main subprojects of ATLAS Trigger/DAQ.
- Work started about four years ago(?) as part of the DAQ/EF prototype "-1" project.
- Uses a fairly detailed software process.
- First public (binary) release approx early 1999.
- Became an open source project some time in 2000.
- Regular releases every few months. Latest release 0.0.15 in October.

General Impression

- A well managed project (whose leader is now leaving!)
- Plenty of (mostly good) documentation on all software components, including APIs and examples.
- Up to date training material available.

Online Components (1)

Organisation

- The Online Software is divided into several components each of which is implemented in one or more CVS packages.
- They identif two main groups of components
- Core components: absolutely required for the system to operate. These were developed first.
- “Integration” components: useful components to build a usable system.

Core Components

- Configuration Databases
- Inter Process Communication (CORBA)
- Message Reporting
- Information Service
- Process Management
- Run Control

Online Components (2)

Integration Components

- Integrated GUI (IGUI)
- Online Bookkeeper
- Resource Manager
- Test Manager
- Diagnostic Verification Service
- Monitoring
- Event Dump
- DAQ DCS Connection
- Integration Tests
- Training Material

Configuration Databases (1)

Introduction

- All aspects of the configuration to be parameterised in databases.
- "Lightweight", in memory database (read the whole database into memory at initialisation)
- Several packages:
  - `database`: contains schema and data files
  - `oks`: basic database library: independent of ATLAS DAQ
  - `confdb`: higher level data access libraries (DALs) providing customised views of ATLAS DAQ system

Configuration Databases (2)

OKS

- OKS is very general, no knowledge of ATLAS.
- The database is implemented using a few generic classes, eg:
  - `Class`: class defining database classes and their attributes (ie the database schema). An instance of a Class object describes one class, eg Module.
  - `Object`: class holding instances of a Class class (ie the database itself).
  - various others for data members, relationships, etc
- Many (federated) schema and data files can be loaded.
- The API provides various queries to locate objects, etc.
- OKS provides two GUI editors (both a bit cumbersome):
  - `oks_schema_editor`: to edit the database schema, basically to create and modify Class objects.
  - `oks_data_editor`: to edit database data files, ie create, modify or delete Object objects.
  - Since both commands need to be given all the relevant schema files, the command lines can be very long. In my script `setup_l1calo.sh` I define some useful aliases: `edschema` and `eddata`.

Configuration Databases (3)

Confdb

- The `confdb` package provides data access libraries (DALs) which load data from OKS into runtime C++ classes.
- So OKS Objects of Class Module are loaded into objects of C++ class ConfdbModule.
- DALs are useful to provide a convenient user view of the data which may be different from the best way to organise the data in the database itself.
- Also, using DALs insulates you from any future changes to the underlying database product (ie OKS at the moment).
- The `confdb` package includes a GUI configuration database editor (`confdb_gui`) which knows about ATLAS DAQ classes and is much more convenient to use than `oks_data_editor`, though still a little awkward.

Inter Process Communication (1)

Introduction

- Online software uses CORBA to communicate between processes running in a distributed system.
- Currently using ILU (free from Xerox) as the CORBA implementation.
- `ipc` package is a thin layer hiding direct use of ILU from the rest of the software, thus allowing ILU to be replaced in future if desired.
- The `ipc` package is also where the basic concepts of partitioning the system are implemented using the CORBA naming service.
- The naming services requires information to be shared via a file (the IPC reference file) whose location is specified by the environment variable IPC_REF_FILE.
- Any process whose objects are accessible via CORBA has to sit in a loop waiting for incoming messages. To do anything else requires another thread (in which you cant run the Online software).

Inter Process Communication (2)

Commands

- The `ipc` package provides some line mode commands to manage partitions (esp if things go wrong).
- `ipc_ls -alF`
  shows top level IPC processes known to the naming system
- `ipc_ls -p partition`
  shows all IPC processes registered in the given partition
- `ipc_rm -i ".*" -n ".*" -v -f`
  kills all IPC processes (on all nodes) – including those belonging to other peoples partitions!
- NB all commands in all Online packages respond to `--help`

Message Reporting Service

Introduction

- Implemented in the `mrs` package.
- Three components: message senders, message server (one), message receivers.
- Senders send messages to the server using a simple stream like interface.
- Message text and expected parameters can be stored in a database.
- Message server forwards messages to connected receivers according to selection criteria.
- Message receiver can subscribe to message server to receive all messages that fit some set of selection criteria.
- In general senders probably wont be receivers and vice versa.

Example

```
#include "mrs/message.h"

IPCPartition partition ("Murrough");
MRSStream mout (partition);

mout << messname1 << MRS_FATAL << ENDM;
mout << messname2 << MRS_PARAM<int>("par",2)
     << MRS_WARNING << ENDM;
mout << messname3 << MRS_TEXT("some arbitrary text")
     << MRS_ERROR << ENDM;
```

Information Service

Introduction

- Implemented in the `is` package.
- Again three aspects: information providers, information servers (many), information subscribers.
- But probably clients will both provide and subscribe to information.
- Multiple servers (identified by name) allow for scalability.
- Basic units of information are ISInfo objects (or subclasses).
- Simple subclasses are provided (ISInfoInt, ISInfoString, etc).
- Users can create subclasses containing complex structures.
- All ISInfo objects are stored and retrieved by name, where the first component is the server name, eg `RunPars.NEvs`, `L1CaloStatus.JEM-0-15`.

Example

```
#include "is/isinfotmpl.h"

IPCPartition partition ("Murrough");
ISInfoDictionary dict (partition);
ISInfoInt isVar;

isVar = 3;
dict.insert("L1CaloPars.simpleValue",isVar);
isVar = 6;
dict.update("L1CaloPars.simpleValue",isVar);
dict.remove("L1CaloPars.simpleValue",isVar);
```

Process Manager

Introduction

- Motivation is to provide an operating system independent layer for starting and stopping processes on distributed machines and for notification of processes crashing, etc.
- A single, partition independent, `pmg_agent` process runs on each machine. At present this is started by rsh/ssh when the DAQ is started, but someday may be started automatically at boot time?
- `pmg` clients request the `pmg_agent` to start processes and can ask for notification of process termination.
- Applications normally defined in the database (but can also be started by providing all information required).
- Agent needs all the required environment variables defined somehow. Maybe via the database or via shell rc script in the account which starts the agent.
- In general applications required by a partition are started by defining them as managed applications in the database.

Run Control

Introduction

- Implemented as a "concurrent hierarchical state machine" (CHSM).
- Tree of run controllers, where each node in the tree can either control a set of child controllers or some bit of the ATLAS readout system (or both?).
- Communication is only between parent and child, **not** between children at the same level.
- Limited synchronisation between different activities can be provided using extra controllers in the tree and altering the order in which the child state transitions are executed.
- Each state transition is associated with a set of actions (one method per transition) returning success or failure.
- Parent executes its actions. If successful its state has changed (but is still locked) and it then asks its children to perform their actions. If all child actions are successful the parent transition has completed successfully. Otherwise the parent sets a (concurrent) error state.
- The main states are `Initial, Loaded, Configured, Running and Paused`. There are also error states.
- The main transitions are sequentially between those states, ie `load, configure, start, pause, resume, stop, unconfigure, unload, terminate`.

Integrated Graphical User Interface

Introduction

- The IGUI, implemented in Java, is intended to be the sole point of access for normal users.
- Ii allows a partition to be booted and shut down, runs started and stopped, etc.
- The IGUI subscribes to and displays all error messages.
- A number of panels set and show IS variables.
- Additional panels can be easily added for subdetector specific information.
- Its a bit slow...

Online Bookkeeper

Introduction

- The OBK records messages, initial values of IS variables and changes to them during a run to an offline database.
- Additional run comments can be added (line mode command).
- The database can be browsed using various tools, including a web interface (requires suitably configured Apache webserver).
- Might be useful to record runs during the slice test?

Resource Manager

Introduction

- Can define resources (which can be anything?) in the database.
- Partitions (ie processes running in a given partition) can lock resources to prevent conflicts if someone tries to start running a different partition which requires the same resources.

Test Manager

Introduction

- Simple, atomic, tests of individual components (or pairs or components) can be defined in a database.
- Tests must return yes/no/undefined results. Plus optional additional messages for humans.
- The idea is that these simple tests can be used for automatic diagnosis of faults by the Diagnostic Service...
- At some point we should consider how to define suitable tests eg for individual modules or crates?

Diagnostic Verification Service

Introduction

- The DVS is an expert system intended to automatically diagnose and if possible correct faults in the DAQ system. Eg processes dying, etc.
- It listens to error messages and watches IS variables.
- It uses the (free) CLIPS package to define sets of rules and actions to be performed when a rule is satisfied.
- The actions may include running tests defined in the Test Manager database to try to pinpoint problems.

Monitoring

Introduction

- The closest the Online software gets to the Dataflow.
- Consists of three components: monitoring provider (factory?), monitoring server (buffer manager), monitoring client.
- The provider is typically part of the dataflow system, eg the ROS or event filter. This can be asked to provide events according to some selection criteria on a statistical basis. All events can be requested, though this will impact on the experiments deadtime so is only intended for test purposes (eg calibration).
- The client is a user monitoring program wanting to analyse events from some part of the system.
- The server acts as a buffer between the user monitoring programs and the performance critical dataflow applications.

Event Dump

Introduction

- The event dump, implemented in Java, is an example of a user monitoring program.
- It decodes and displays the ATLAS event format and a hex dump of the fragment contents.
- Extensibility to allow customised decoding of subdetector event fragments has been requested.

DAQ DCS Connection

Introduction

- DDC handles exchange of commands and information between the DAQ system (Online Software) and the DCS (SCADA).
- Commands to DCS are implemented as MRS messages. DCS data values are mapped to IS variables.
- Mapping of run control partitions and DCS partitions has to be set up by hand I think.

Integration Tests

Introduction

- According to the Online software process, all components including the whole system must be accompanied by tests which check that the software provided functions as specified.
- The integration tests basically put a simple system of a couple of run controllers controlling dummy Readout Subsystems through a number of state transitions.

Installation

Latest Release

- Officially: go to their web page, download several files and perhaps several patches.
  `http://atlas-onlsw.web.cern.ch/Atlas-onlsw/Download`
  (though this is still out of date)
- Or go to our web pages, download one (or two) scripts and run them. See
  `http://www.hep.ph.qmul.ac.uk/l1calo/sweb/software/online.html`
- That web page also has suggestions for what to do next...

Setup (1)

Environment Variables

- The Online software uses a large number of environment variables. Some are required, many are optional.
- TDAQ_INST_PATH: root of local Online software installation.
- IPC_REF_FILE: IPC reference file (for CORBA naming service).
- TDAQ_DB_PATH: top directory for database files.
- TDAQ_DB_SCHEMA: colon separated (PATH style) list of OKS schema files to be loaded.
- TDAQ_DB_PATH: top level partition database file (which may itself load other files).
- TDAQ_PARTITION: name of the default partition to run.
- TDAQ_LIB_PATH: colon separated list of libraries to be added to the default LD_LIBRARY_PATH by remote PMG agents.

Setup (2)

Setup Scripts

- Each release provides setup scripts (`setup.sh` and `setup.csh`) to set all the above environment variables.
- For other than the default `be_test` partition, you will want to customise some of the settings. I do this in a script `l1calo_setup.sh` which calls the default `setup.sh` first.
- If suitable links in $HOME are defined to point to Online software releases and L1Calo packages, it can be used as is. Otherwise customisation at our individual sites is required.

Configuration

Editing Databases

- See demo?

Running It...

`play_daq` Script

- At present the DAQ is started via the `play_daq` script. (A different scheme is envisaged in future).
- Local customisations can be added in a `play_daq_userdefs` file which must be found in the current directory. A template can be found in the same directory as `play_daq` itself. Eg to suppress the OBK for the moment, add `'export OBK="no"'`
- Usage:
  `play_daq L1CaloSlice&`
  where `L1CaloSlice` is the name of the partition (which must also be set in TDAQ_PARTITION).