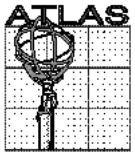


ATLAS
High-Level Triggers, DAQ and DCS

Technical Design Report

Issue:	Draft 1
Revision:	0
Reference:	ATLAS TDR-xx
Created:	12 November 2002
Last modified:	13 February 2003
Prepared By:	ATLAS HLT/DAQ/DCS Group

All trademarks, copyright names and products referred to in this document are acknowledged as such.

ATLAS Collaboration

CERN
European Laboratory for Particle Physics (CERN), Geneva

Acknowledgements

The authors would like to thank Mario Ruggier for preparing the template upon which this document is based and the DocSys group for their help in using it.

Table Of Contents

Part 1

Global View

1 Overview

This chapter is currently being drafted by the management team but as an introduction it relies heavily on knowing the exact content of the other chapters which is only emerging clearly with the first draft. Therfore this chapter will probably only appear after the first draft.

1.1 Main system requirements

1.1.1 From physics

1.1.2 From performance (Read-out, selection)

1.1.3 Functional and operational

1.2 System functions

1.2.1 Detector R/O

1.2.2 Event selection/rate reduction

1.2.3 Movement of data

1.2.4 Storage of data (events, conditions, etc.)

1.2.5 Experiment Operation

1.2.6 Detector controls

The principal task of DCS is to enable the coherent and safe operation of the ATLAS detector. It supervises all hardware of the experimental set-up, not only the different subdetectors of ATLAS, but also the common experimental infrastructure. It also communicates with external systems like the infrastructure services of CERN and most notably with the LHC accelerator.

Safety aspects are treated by DCS only at the least severe level. This concerns mainly questions of sequencing operations or requiring conditions before executing commands. Also tools for interlocks both in hardware and in software are provided by DCS. Not in realm of DCS are situations, which could cause major damage to the detector or even endanger people's lives. The former is the responsibility of a dedicated Detector Safety System (DSS), with which DCS interacts, and the latter is addressed by the CERN-wide safety and alarm system.

It is mandatory that concerning the hardware of the detector all actions initiated by the operator and all errors, warnings and alarms are handled by DCS. It has to provide online status information to the level of detail required for global operation. Also the interaction of equipment experts with their subdetector should normally also go via DCS. DCS has to continuously monitor all operational parameters, signal any abnormal behaviour to the operator and give him guidance. It must also have the capability to automatically take appropriate actions if necessary and to bring the detector in a safe state.

Concerning the operation of the experiment, an intense interaction with the DAQ system is of prime importance. Good quality physics data requires detailed synchronisation between the DAQ system and DCS. Both systems are complementary in as far the DAQ deals with the data describing a physics event and DCS treats all data connected with the hardware of the detector. The former are organised by event number and the latter are normally categorised with a time stamp. The correlation between both is established in offline analysis.

Some parts of the detector will operate continuously because any interruption is costly in time or money or may even be detrimental to the performance of that detector. Hence its supervision by DCS is needed continuously. DAQ in contrast runs only when physics data are taken. Therefore DCS needs complete operational independence. This must however not result in boundaries, which limit functionality or performance. Therefore both share elements of a common software infrastructure. Different modes of operation are foreseen like taking data with colliding beams, detector calibration, and stand-alone operation of a subdetector or even of an individual detector element.

1.3 Types of data TDAQ deals with

1.3.1 Detector control values

1.3.2 Event data

1.3.3 Configuration data

1.3.4 Conditions data

1.3.5 Statistics and monitoring data

1.4 Glossary

1.5 References

1-1

1-2

2 Parameters

This chapter is dedicated to the relevant parameters for the HLT/DAQ/DCS system. These include the detector readout parameters and the trigger selection for the correct dimensioning of the dataflow system and for understanding the data volumes that will need to be stored. These will be the subject of the first three sections.

Other important parameters for the correct definition of the system are the ones coming from the monitoring requirements. These are discussed in the fourth section.

The last section is dedicated to the DCS parameters: the subdivision of the system in detector parts and the amount of configuration data traffic in case of cold configuration and re-configuration of possible faulty elements.

2.1 Detector R/O parameters

This section could be moved to Section 1.2.1, "Detector R/O".

The ATLAS detector organized into three main systems: the Inner Detector, the Calorimetry and the Muon Spectrometer. These systems are then subdivided in sub-detectors.

The Inner Detector is divided in the following sub-detectors: Pixel, SCT and TRT. The Pixel sub-detectors is a detector using the pixel technology with a readout divided in ϕ regions and it is sub-divided in two endcaps, one inner barrel B-layer and 2 outer barrel layers. The SCT sub-detector is a Si microstrip detector subdivided into two endcaps and a barrel part subdivided in two regions for positive and negative η. The TRT sub-detector is a straw tubes tracking detector providing a particle identification based on the transition radiation.

The Calorimetry is a large system made of several sub-detectors based on different technologies. The barrel electromagnetic, the endcap electromagnetic, the endcap hadronic and the forward calorimeters use the LAr as sensible media with different absorbers depending on the particles to be detected. The barrel hadronic calorimeter and two endcaps at larger radii (with respect to the other calorimeters) in the range $|\eta| < 1.7$ is instead based on scintillator-iron technology: the Tilecal calorimeter.

The Muon spectrometer is subdivided in a barrel part where there are precision chambers based on Monitored Drift Tubes (MDTs) and trigger chambers based on Resistive Plate Chambers (RPCs). In the two endcaps up to $|\eta| \leq 2.4$, there are again MDTs as precision chambers and Thin Gap Chambers (TGCs) as trigger chambers. At large pseudo rapidities and close to the interaction point there are Cathode Strip Chambers (CSCs) that are suited to sustain the higher rate and the more severe background contitions.

In terms of readout signals to be transmitted to the Data Acquisition (DAQ) system, the LVL1 Trigger is another source of data and dedicated ReadOut Drivers (RODs) are used.

The organization in terms of readout is in fact slightly different from the pure division of the detector in sub-detectors and it is illustrated in the first sub-section, where a mapping of the ATLAS detector and trigger is specified in terms of data sources (the RODs) for the DAQ system in terms of the partitioning.

The concept of a partition used throughout this chapter coincides with the TTC partition concept introduced by the LVL1 TDR.

2.1.1 RODs per detector per partition

The distribution of the ROD modules and crates per sub-detector and partition generally follows the division of the sub-detectors in parts. This distribution assumes that there is no overlap of hardware among partitions and that each partition can be independently functional.

In Table 2-1 the number of RODs, ROD crates and ROLs are reported per sub-detector per partition.

Table 2-1 The distribution of the RODs per detector per partition.

	Detector	Partition	RODs	ROD crates	partitions	ROLs	Frag size (MB)
Inner Detector	**Pixel**		120	8	3	120	1.3
		B Layer	44	3			
		Disks	12	1			
		Layer 1 + 2	38+26	4			
	SCT		92	12	4	92	1.6
		Left Barrel	22	3			
		Right Barrel	22	3			
		Left Endcap	24	3			
		Right Endcap	24	3			
	TRT		256	22	4	256	1.0
		Barrel A	32	3			
		Barrel C	32	3			
		Endcap A	96	8			
		Endcap C	96	8			

Table 2-1 The distribution of the RODs per detector per partition.

Calorimetry	**Tilecal**		32	4	4	64	1.1
		Barrel A	8	1			
		Barrel C	8	1			
		Ext Barrel A	8	1			
		Ext Barrel C	8	1			
	LAr		192	16	6	768	1.4
		EMB A	56	4			
		EMB C	56	4			
		EMEC A	35	3			
		EMEC C	35	3			
		FCAL	4	1			
		HEC	6	1			
Muon Spectrometer	**MDT**		192	16	4	192	1.0
		Barrel A	48	4			
		Barrel C	48	4			
		Endcap A	48	4			
		Endcap C	48	4			
	CSC		32	2	2	32	0.2
		Endcap A	8+8	1			
		Endcap C	8+8	1			
LVL1 muon	**RPC**		32	16	2	32	1.0
		Half Barrel 1	16				
		Half Barrel 2	16				
	TGC		16	8	2	16	
		Endcap A	8				
		Endcap C	8				
	MIROD		1	1	1	1	0.104
LVL1 calo	**CP/JEP**	RoI		1 or 2		6	0.252
		CP			1	16	1.5
		JEP				16	1.1
		PP		8		16	
	CTP		1	1			0.012—0.038

These are the basic parameters updated during the 3rd ROD Workshop held in Annecy in November 2002. The fragment sizes reported in Table 2-1 have to considered as the maximum expected fragment size during the first phases of the ATLAS data taking and during the calibrations. A more accurate estimation of the fragment sizes is discussed in the next subsection (Section 2.1.2).

Concerning the LAr fragment size a more accurate estimation is on going in the community. It is based on both a compression of the data required and on zero-suppression schemes, but due to their nature it cannot be expressed in a straightforward way in a table like Table 2-1.

2.1.2 Fragment sizes per detector

Includes physics and calibration data.

Should have average values, spread and uncertainties; should be shown against luminosity; and against data compression schemes.

The fragment sizes reported in the previous table are indicative and they have to be seen as the maximum achievable figures.

Investigations are ongoing to obtain more realistic numbers for physics and calibration operations to resolve discrepancies with the values used in the Paper Model. The Detector people have to be contacted and an agreement on the numbers has to be found, based on the latest simulation they have for the sub-detector readout.

Table 2-2 ROBIn raw data fragment sizes in kByte used for the paper model. To each fragment a header with a size of 32 Bytes is added

Subdetector	Low luminosity	Design luminosity
Pixels	0.2	0.50
SCT	0.33	1.20
TRT	0.33	1.20
E.m. calorimeter	0.752	0.752
Hadron calorimeter	0.752	0.752
Muon precision	0.80	0.80
Muon trigger	0.38	0.38

2.2 Trigger parameters

2.2.1 LVL1 rates

For estimating message rates and the volume of data to be transferred to the LVL2 trigger and to the Event Builder, the LVL1 accept rate, the nature and quantity of “regions of interest” (RoIs)

found by the LVL1 trigger, and the accept rate of the LVL2 trigger need to be defined. Two baseline “LVL1 trigger menus” are used in this report:

1. “low luminosity”: for a luminosity of 2.10^{33} $cm^{-2}s^{-1}$ with a LVL1 accept rate of about 20 kHz and a predominantly high transverse momentum trigger. The LVL2 accept rate is about 600 Hz,
2. “design luminosity”: for a luminosity of 10^{34} $cm^{-2}s^{-1}$ with a LVL1 accept rate of about 35 kHz. The LVL2 accept rate is about 1.5 kHz.

Editor: Note that the rates expressed above should be mutiplied by a factor of 2 to 3 uncertainty to get the LVL1 input rates which the TDAQ system should be able to handle, namely 40 kHz at the low luminosity and 75 kHz at design luminosity.

The exclusive rates for the different menu items are specified in Table 2-3. E.m./gamma (EM, the I refers to “isolated”), muon (MU), jet (J) and hadron (TAU) RoIs are distinguished, and labelled with the LVL1 energy or transverse momentum threshold. XE refers to the LVL1 missing energy trigger. For a discussion of these menus see Chapter 4, "Event selection strategy" (NB: 5 kHz of “Other items” are not taken into account).

Table 2-3 Exclusive rates for the LVL1 trigger menu items. For items for which two possibilities are indicated. the latter corresponds to design luminosity

LVL1 Trigger menu item	Low luminosity (kHz)	Design luminosity (kHz)
MU20	0.8	4.0
2 MU6	0.2	1.0
MU10 + EM15I	0.1	0.4
EM25I / EM30I	12.0	22.0
2 EM15I / 2 EM20I	4.0	5.0
J200 / J290	0.2	0.2
3J90 / 3J130	0.2	0.2
4J65 / 4J90	0.2	0.2
J60+XE60 / J100+XE100	0.4	0.5
TAU25+XE30 / TAU60+XE60	2.0	1.0

2.2.2 Parameters relevant for LVL2 processing

The data needed for the LVL2 trigger and the type of processing performed by it depends on the regions of interest supplied by the first level trigger. Each of the four different types of RoIs has its own characteristic type of processing. The processing consists of several steps and after each step a decision is taken on whether data from other subdetectors within the region of interest should be requested for further analysis. In Table 2-4 the subdetectors are indicated from which data are requested in the different processing steps. The associated acceptance factors are also specified in Table 2-4. The data rates can be estimated using these factors and information on the sizes and the locations of the regions of interest, and on the mapping of the detector on the ROBins. The LVL1 trigger defines a finite number of possible RoI locations. A small region in eta-phi space corresponds to each location. A hit in this region satisfying appropriate LVL1 trigger criteria generate a RoI with a location corresponding to the region. The relative RoI rate for each location is assumed to be proportional to the surface of this region, while the sum of the rates for all possible locations should be equal to the LVL1 menu RoI rate. This makes it possible to determine the rate for each possible location. In combination with the RoI sizes (see Table 2-5) and the mapping of the detector on the ROBIns the RoI data request rates for each ROBIn can be calculated. Information on the mapping can be found in ref. .... (backup document on paper modelling).

Table 2-4 Subdetector data requested by different processing steps of the LVL2 trigger for the different types of RoIs and associated acceptance factors. The acceptance factors are relative to the LVL1 RoI rate.

Type of RoI	First step	Acceptance factor	Second step	Acceptance factor	Third step
EM	E.m. calorimeter	0.19 (design lum.: 0.16)	Hadron calorimeter	0.11 (design lum.: 0.16)	TRT /SCT/Pixels
JET	E.m. and hadron calorimeters	1.0			
TAU	E.m. and hadron calorimeters	0.2	TRT /SCT/Pixels		
MUON	Muon precision and trigger detectors	0.39	SCT/Pixels	0.086	E.m. and hadron calorimeters (only for design luminosity)

Table 2-5 LVL2 RoI sizes

Type of RoI	Size in eta	Size in phi
EM	0.2	0.2
JET	0.8	0.8
TAU	0.2	0.2
MUON	~ 0.3 - 0.4 (depends on detector)	~ 0.1 - 0.4 (smallest in muon and in inner detector)

In order to establish the processing resources needed for the LVL2 trigger the algorithm execution times and the overheads for sending requests and receiving data are needed. See Table 2-6 for current estimates, assuming execution on 4 GHz machines. The numbers specified include estimates of the time needed for data preparation. Furthermore for each message sent or received an overhead of 10 microseconds is taken into account, while the processing step resulting in a decision is assumed to take 50 microseconds. Merging of event fragments into a larger fragment suitable for input in the algorithms is assumed to proceed at 160 MByte/s.

Table 2-6 Estimated execution times (in ms) of LVL2 algorithm steps on a 4 GHz processor and for low and design luminosity respectively. The estimated time needed for data preparation has been included in the RoI processing times. The algorithm execution times are the m_95 values (see chapter ...)

Type of RoI or trigger	Muon detectors	Calorimeters	TRT	SCT + Pixels
EM		0.088 / 0.123 (e.m.) 0.023 / 0.032 (hadron)	8.33 / 24.56	1.36 / 3.88
JET		0.68 / 0.68		
TAU		0.044 / 0.061 (e.m.) 0.011 / 0.016 (hadron)	8.33 / 24.56	1.36 / 3.88
MUON	0.5 / 0.5	0.044 / 0.061 (e.m.) 0.011 / 0.016 (hadron)	8.33 / --	1.36 / 3.88

2.2.3 Parameters relevant for Event Builder and Event Filter

Events need to be fully built at a rate equal to the acceptance rate of the LVL2 trigger (0.6 or 1.5 kHz) and then to be analysed by the Event Filter. The Event Filter is expected to reduce the rate by a factor of 10 (see ch. ...) with a typical processing time of 1 second per event, which requires a farm of at least 600 or 1500 processors.

2.3 Data rate summaries

The LVL2 system and the Event Builder both send requests for data to the ROBIns. The rate of the requests from the Event Builder is equal to the event building rate, i.e. 0.6 or 1.5 kHz. The rate of LVL2 requests per ROBIn is presented in Table 2-7. The total data volume output per ROBIn is shown in Table 2-8, the size and properties of the LVL2 farm in Table 2-9 (assuming the use of 4 GHz dual-CPU machines, a utilization of 70% of the available CPU capacity, with the number of machines increased with a safety factor of 20% , and for direct connection of the ROBIns to the network), and the total bandwidth required for data transporting from ROBIns to the LVL2 system and to the Event Builder in Table 2-10.

Table 2-7 LVL2 request rate per ROBIn in kHz, “overall average”: averaged over all ROBIns, “maximum average”: time average for the ROBIn with the highest average number of requests

Luminosity	Muon trigger	Muon precision	E.m. calorimeter	Hadr. calorimeter	TRT	SCT	Pixels
Low (overall average)	0.020	0.044	0.449	0.470	0.034	0.107	0.134
Low (max. average)	0.040	0.061	1.192	0.771	0.044	0.148	0.197
Design (overall average)	0.099	0.215	0.659	0.527	0.012	0.271	0.340
Design (max. average)	0.198	0.298	1.754	0.866	0.016	0.373	0.491

Table 2-8 Output data volume per ROBIn in MByte/s (LVL2 data and data sent to the Event Builder), “overall average”: averaged over all ROBIns, “maximum average”: time average for the ROBIn with the highest average number of requests

Luminosity	Muon trigger	Muon precision	E.m. calorimeter	Hadron calorimeter	TRT	SCT	Pixels
Low (overall average)	0.516	0.265	0.822	0.839	0.229	0.256	0.170
Low (max. average)	0.533	0.272	1.405	1.075	0.233	0.271	0.185
Design (overall average)	1.331	0.707	1.692	1.589	1.863	2.182	1.245
Design (max. average)	1.413	0.741	2.551	1.855	1.868	2.307	1.325

Table 2-9 Total bandwidth required for transport of event fragments in MByte/s

	Low luminosity	Design luminosity
Total bandwidth LVL2 traffic (MByte / s)	318	510
Event Building rate (kHz)	0.6	1.5
Total bandwidth traffic to Event Builder (MByte / s)	604	2017

Table 2-10 LVL2 farm size, message rates and data volumes per L2PU

	Low luminosity	Design luminosity
LVL2 farm size	37	72
Fragment rate in = request rate out per L2PU (kHz)	11.6	9.0
Fragment volume in per LVL2 processor (MByte / s)	8.6	7.1
Decision rate per L2PU (kHz)	0.54	0.48

2.4 Monitoring requirements

2.5 DCS parameters

2.5.1 Data Volumes and rates

The DCS systems will be configured using ConfDB. PVSS (already mentioned and ‘defined’?) systems (DPTs, DPs, managers/drivers used etc.) will be configured and all associated software, such as hardware drivers(?), OPC servers, configuration files for ‘external’ applications (e.g. DDC) set up using information from ConfDB.

This ‘system’ level configuration will not be done very often, i.e. only at times of hardware modification (new equipment added, possibly equipment changed). Data volume large due to high number of separate systems (100 PCs?) though fast data rate not required as this is not a real time operation and should only be performed during shut-down periods.

During operation of the LHC, various operating modes are required for each sub-system. These modes require hardware to have different settings. A recipe is defined as a collection of settings used for a given operating mode. For a given run, more than one recipe may be required at different stages of the run (e.g. beam starting, beam on, beam stopping).

This ‘operating mode’ configuration is completed more often than the system level configuration. Before a run starts, any recipes used will be downloaded from the ConfDB and stored locally with each system (data consistency problem - may need to lock ConfDB at a given time before download). The data is then loaded into the running system and applied at the time it is required. The data volume is lower than that for system configuration, though still quite large as there are many systems, each requiring a number of ‘stages’ containing all values to be set. Data rate required not high as the downloading is completed before a run starts (how long before?) and therefore, real-time download is not required.

Information held in ConfDB. System set up (system names in which PCs, manager lists, external application lists, configuration file(?) for external applications or at least information to allow these files to be produced, DDC, DPTs, DPs, addressing, dp functions, command transfer, message transfer). Recipe set up (original values, archiving, CondDB output configuration, alert limits, action scripts).

The CondDB will be used to store data read from DCS system. Bi-directional link desirable to allow data from both DCS and TDAQ to be correlated and displayed in PVSS. However, main direction is from DCS to CondDB. Desirable to put all values into CondDB (i.e. not only that ‘data required for off-line’). Data volume will be high (~1,000,000 channels plus others) though data rate could be low if data not sent in real time (possibly downloaded every 15mins/1 h/12 h/ etc.).

2.6 References

2-1 3rd ROD workshop

2-2 Inner Detector TDR

2-3 LAr Calorimetry TDR

2-4 Tilecal TDR

2-5 Muon Spectrometer TDR

3 Operationial requirements for the TDAQ system

Why this chapter? While Chapter 2, "Parameters", somehow says what is required from TDAQ in terms of performance, this chapter should highlight what is expected by TDAQ when it is "used".

The chapter contains the description of:

a. how an event is identified, at different levels in TDAQ.

b. What are the global TDAQ states. What are the detector and machine states. Including how TDAQ states relate to the detector and machine states. The global state transition

c. The defition of a run. How runs are identified. How an event is uniquely identified throughout the life of ATLAS. Types of runs. The question of the transition between runs. What is allowed during a run, what is done outside the run.

d. Partitioning: definition, operations

e. The general strategy to react to faults and errors (in TDAQ but also, and mainly, caused by external systems, such as the detector).

f. The role of data bases, what kind of data is permanently stored for what purpose (and where?).

3.1 Event identification

Up to acceptance by level-2 (or event building in the case of a partition without level-2) an event is identified by an extended (32-bit) level-1 ID (generated by level-1 as 24-bit number and extended to 32).

A Global Event Number (GID) uniquely identifies an event, accepted by the level-2 trigger, within a run. It is generated by a central element (today it could be the DFM) after the LVL2 decision and it is made available, to be tagged into the event, to the element responsible for building the full event (today this would the SFI).

3.2 TDAQ states

We define 1) the DAQ states, 2) the detector (i.e DCS) states (relevant to TDAQ) and the machine states relevant to TDAQ. Followed by the global TDAQ state machine.

The DAQ states as currently defined by the TDAQ Global Issues Working group in the the document "Run and States" are: Idle, Initial, Loaded, Configured, Running, Paused. There is also a transition which is sometimes refered to as a state and it is called Checkpoint. These are illustrated in the state transition diagram shown in Figure 3-1

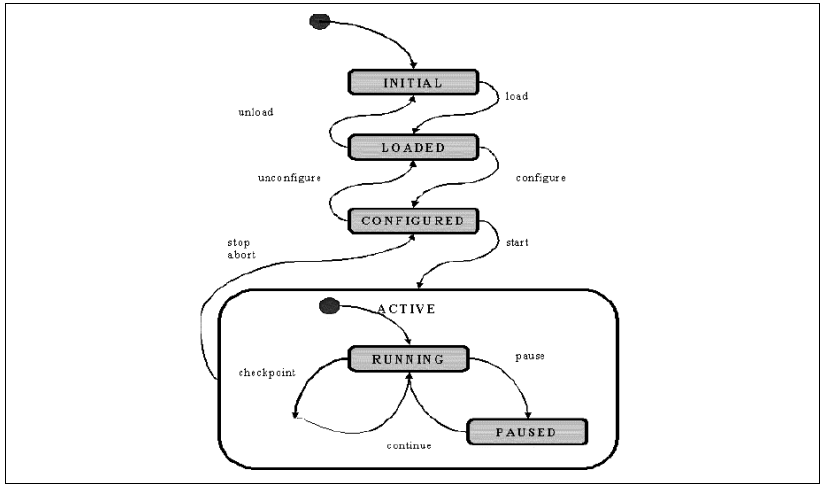


Figure 3-1 DAQ states and transitions diagram

3.3 The run

3.3.1 Run and Run Number

Definitin, purpose, where it is set into the event, who does it, how it is done.

A run is a period of data taking in a TDAQ partition with a defined set of stable conditions related to quality of physics. *Note: define what are conditions*

The run number uniquely identifies a run (today it is a 32 bit number), and associates an event to a set of stable conditions.

A run number is unique throughout the lifetime of the experiment. A run number is generated by a central service, upon request by the run control application.

The proposed mechanism to include the run number into the event is based on the TDAQ run control system (or some other online application program) to distribute the run number (at the beginning of a run) to the ROD crate controllers. The RODs then insert the run number into the fragment header for each event.

In this way, any event fragment is identified anywhere in the system by its associated run number.

3.3.2 Requirements

The ATLAS TDAQ system is required to minimise its contribution to the experiment down time; although a quantitative definition of this requirement is not yet available, one can anticipate that the experiment down time due to TDAQ must be well below 1%.

There are two contributions to system down-time which are relevant to the subject of this document:

- the time spent by the system to initiate (start) or terminate (stop) a run, and
- the down time when coping with malfunctioning TDAQ components.

The two contributions above have an important impact on:

- how a run is defined, that is what configuration and parameter changes force a new run and what changes do not force a new run,
- how the transition between runs should be implemented, and
- how faults should be handled during a run.

It is difficult to identify precisely the conditions which characterise the quality of physics, hence a run. Nevertheless we could identify three classes of such conditions: the parameters defining or affecting the selectivity of the triggers (LVL1, LVL2 and EF), the set of sub-detectors participating to the TDAQ partition, the operational parameters of sub-detectors. A modification of any of the above conditions forces a new run, that is the events following the change of the conditions are tagged with a new run number.

Conditions whose change forces a new run are stored in a conditions data base, whose contents are saved to permanent storage prior to the start of a new run.

Changes which do not force a new run include for example the removal or the insertion of processors or the disabling of FE channels (insofar as the physics is not affected)[1]. Those changes which do not force a new run are not stored in the conditions data base. The change may be entered for example in an electronic experiment logbook if it affects the performance of the TDAQ system (e.g. removal of an EF processor) or tagged into the event if it affects the data from the detector. As an example of this latter: the removal of a ROD/ROB may be flagged by the corresponding ROS by appropriately setting a “quality flag” in the fragment header.

A run, when conditions do not change, may extend throughout an entire machine fill.

1. The number of e.g. FE channels (or ROBs) which can be removed from the read-out without affecting the physics is bounded by some threshold which is sub-detector dependent. When the amount of unavailable read-out exceeds the threshold, the physics is affected and the run should be stopped.

3.3.3 Physics and calibration runs

Define what they are: what is their purpose, the actors (i.e. what a run type needs), where output goes.

Backup document on use cases?

3.3.4 Operations during a Run

There is the need for an active[1] run, i.e. following the successful execution of the run start command, to be interrupted temporarily: level-1 triggers are not generated so that some change or intervention on the detector can be done. Changes and interventions are such that they do not affect the physics, that is they do not force a new run. The global system state associated to the temporary interruption of a run is called Paused.

Two commands are available to respectively enter and exit the Paused state: pause and continue. Pause and continue commands may be issued: by an operator or by software (viz. an expert system).

When the pause command is issued:

- The level-1 triggers are blocked, by raising the global busy signal.
- All TDAQ elements are issued with the Pause command. Each element will execute it locally as soon as the handling of the current event is terminated (i.e. TDAQ elements will not empty their buffers before entering the paused state).
- TDAQ completes the transition to Paused as soon as all the TDAQ elements have entered the Paused state.

When the continue command is issued:

- All TDAQ elements are issued with the continue command, each element returns to the running state.
- TDAQ completes the transition to the running state as soon as all the TDAQ elements have returned to the running state.
- At this point level-1 triggers are unblocked.

There is another special command which may be issued to a running TDAQ system, the Abort command. This command is reserved for very special cases and it entails a fast termination of the run; for example TDAQ elements will not complete the processing of events in their buffers.

3.3.5 Transition between Runs

From the operational point of view, a run is bracketed by a (run) start and a (run) stop command. These commands have to be sent to all[1] the TDAQ elements (viz. processors) for synchronous local execution prior to the transition to the running state or to the stopped state.

1. TDAQ is said to be in the running state.

Prior to the start of a run, or as the immediate consequence of a (run) stop command, the LVL1 Busy is asserted. The LVL1 Busy is removed upon the transition to the running state, this latter implies that all[1] TDAQ elements have completed their local execution of the (run) start command.

The completion of a run is also a process which needs synchronous local processing of all[1] the TDAQ elements: they receive the stop command, complete the processing of the contents of their buffers, produce end of run statistics etc. and leave the running state.

In addition to the run control command which signals a TDAQ element when a run is requested to complete, a mechanism is necessary to determine when the last fragment or event of the terminating run has been processed. This mechanism cannot be part of the run control, it is tightly related to the flow of the event data in the TDAQ system. This is done by means of a time-out: a TDAQ element will consider that the last event has been processed when 1) it has received the "stop run" command and 2) it has not received events for a certain time (for example a time out of some 10s of seconds).

The transition between two runs (i.e. stopping the previous and starting the next) includes two potentially time consuming processes:

- the completion of the processing of the contents of all the fragment / event buffers in the system: front-end buffers, RODs, ROBs, LVL2 and EF nodes;
- the synchronisation of all[1] the TDAQ elements to complete the transition stopped / running or running / stopped. That is, before the TDAQ partition may complete a state transition, all[1] the TDAQ elements have to have completed the transition locally.

There are conditions, for example the LVL1 trigger masks, thresholds and pre-scaling factors, or sub-detector calibration operating parameters, such that:

3. the modification of their values forces the change to a new run and
4. their value may be required to change relatively often (may be several times per machine fill).

The same considerations may also be applied to calibration runs, when some detector operating parameter may be required to change frequently.

In these cases the transition between runs is not adequate in terms of the potentially long TDAQ system down time. A more efficient transition between runs is required and we define:

Checkpoint

> a transition in a running TDAQ system, triggered by a change in conditions or by an operator, which 1) results in the following events to be tagged with a new run number and 2) does not need the synchronisation, via run control start / stop commands, of all TDAQ elements.

The checkpoint transition is intended for those changes in conditions which require that events be correlated to the new conditions via a new run number but the change has a light implication

1. It is envisageable that, in the case of the LVL2 and the EF, only a (to be defined) percentage of the farm needs to successfully perform the transition. The rest may do it "in the background" and join the new run afterwords.

on most of TDAQ. It is a mechanism to associate a new run number to events characterised by new conditions with minimal synchronisation within TDAQ.

A checkpoint transition is started automatically by the TDAQ control system when certain conditions are modified, it may also be initiated manually by an operator or automatically by some other software component (viz. an expert system). It should be noted that, for a transient time, events belonging to more than one run could be simultaneously present in the system. In particular given that LVL2 accepts are not time ordered, a EF node might have to process events belonging to two (or in principle even more) different runs.

The main feature of the checkpoint transition is the fact that events keep flowing in the system continuously: a mechanism is needed for a TDAQ element to detect when the new run begins. That is to say when the new run number becomes applicable, when the Global Event ID should be reset to 0 and when a TDAQ element should perform run completion processing and the initialisation necessary for a new run (for example a LVL2 processor may require to read the new conditions).

The run number may be used for this purpose, i.e. a TDAQ element recognises a new run whenever a piece of data (fragment or full event) is tagged with a new run number. TDAQ elements may therefore perform the "transition" from the old to the new run at their own pace and time. Note that the same mechanism is also applicable to analysis and monitoring software dealing with a statistical sample of the event data: an e.g. monitoring program recognises a new run whenever it samples an event with a new run number (with the caveat that, as for EF processing units, programs sampling events after Level-2 might have to handle events belonging to more than one run). A condition (belonging to a well defined sub-set of the possible run conditions) is changed or an operator asks for the execution of a checkpoint command.

3.4 Partitions and related operations

Defintion of what a partition is: what for, who are the actors participating to the partition.

Allowed partitons (here we should make reference to the constraints imposed by the TTC system and include the table of the detector partitions).

What can be done with partitions: join and split.

Material from [3-2].

How partitioning is realised on the system is reserved to Chapter 5, "Architecture" (and possibly Part 2 System Components).

3.5 Operations outside a run

Define what are the operations allowed when a run is stopped or when LHC is off. "Define" should include: the purpose of the operation, the actors, the expected result, the effect on TDAQ.

Initialisation, configuration.

Operations on partitions: split / join

A backup document with use cases would be useful (if can be done).

This section was moved wrt the original layout so as to come AFTER the section on partitioning (since some of the operations might be done on partitions). It was also promoted one level up in the section hierarchy

3.6 Error/Fault reporting/handling strategy

Brief description of the global strategy here as the details are in Chapter 6. Emphasis should be given to 1) what TDAQ does when an internal error happens and 2) what TDAQ does when a fault happens outside TDAQ (but the fault affects the operation of the system).

3.7 Data Bases

What has to be stored permanently? at least give some broad categories and the source of the data. Then list what is the required functionality of the data base system(s). For example data related to configuration, conditions, monitoring, etc.

Material from this section should come from the efforts going on to collect requirements on data bases.

3.8 References

3-1 GIWG. Run and States

3-2 GIWG. Partitioning

4 Event selection strategy

4.1 The approach

HLT as a coherent entity, use of complementary features of LVL2 and EF (if not discussed before)

emphasis on inclusive signatures, have more refined tools (other selection algorithms, more exclusive / topological criteria etc) at hand

4.2 Selection objects

define (physics oriented) objects (e.g. e, mu, ...) to be used for the selection, describe in the following sub-sections the high-level (algorithm) steps to define candidate objects

4.2.1 Electron/photon

4.2.2 Muon

4.2.3 Tau/jets/$E_T$miss

4.2.4 b-tagged jets

4.2.5 B-Physics

4.3 Trigger menus

define the basic trigger menu(s), covering the major part of the physics program

need to address various scenarios

4.3.1 Physics triggers

unprescaled signatures go here

4.3.2 Pre-scaled physics triggers

4.3.3 Monitor and calibration triggers

4.4 Physics coverage

describe the coverage (essentially impact of thresholds) on various physics processes of interest

4.5 Determination of trigger efficiencies etc.

4.6 References

4-1

4-2

5 Architecture

The purpose of the chapter is to describe the top level architecture of TDAQ, in terms of:

its place with respect to the other parts of ATLAS, as well as systems and services external to ATLAS,

how the system is organised: functionally, in terms of sub-systems and in terms of more abstract elements,

a generic architecture with a a definition of the abstract components that are visible at the architectural level

how sub-systems map onto the generic architecture ("views").

how the scalability and partitioning can be performed and

finally it proposes a baseline architecture expressed by the realisation of the abstratc components.

DCS is considered, as regards this chapter, as a black box with interfaces to TDAQ and external systems. The internals of DCS do not belong to this chapter.

5.1 TDAQ context

5.2 Context Diagram

The ATLAS TDAQ context diagram is shown in Figures 5-1. The LVL1 trigger provides LVL2 with region-of-interest(RoI) and other data needed to guide the LVL2-trigger data selection and processing; this interface is discussed in detail in part 2. The Timing, Trigger and Control (TTC) system provides signals associated with events that are selected by the LVL1 trigger. ReadOutDrivers(RODs), associated with the detectors, provide event fragments for all events that are selected by the LVL1 trigger. In addition, the LVL1 system contains RODs which provide data to be read out for the selected bunch crossings. The LVL1 trigger system, the TTC system and the ROD systems of the detectors all need to be configured by the DAQ system, for example at the start of each run. These components are shown in the top part of the diagram.

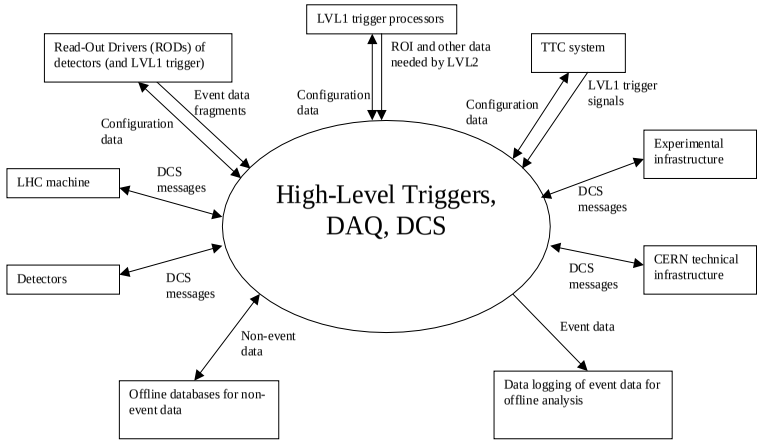


Figure 5-1

Interfaces to other external systems are also illustrated in Figure 5-1. These connect to the LHC machine (e.g. to exchange information on beam parameters), to the detectors (e.g. to control voltages), to the experimental infrastructure (e.g. to monitor temperatures of racks), and to the CERN technical infrastructure.

The remaining interfaces relate to long-term storage of data that must also be accessed for offline analysis of the event data. For events that are retained by the high level triggers, the event data have to be stored for offline analysis. In addition, a large amount of non-event data has to be stored: alignment and calibration constants, configuration parameters, etc. Not shown in the figure is the importation of programs from the offline software for use by the high level triggers.

5.2.1 TDAQ Interfaces

More in detail how the context elements and TDAQ interface; we also define what data is exchanged (who is generating it, who is using it).

An interface is defined in terms of the partners (in TDAQ and outside TDAQ), who is responsible for the interface (within TDAQ and outside TDAQ), what data flows in/out of the intreface and where the interface is documented. It is proposed to split things in 2 parts: I/F internal to ATLAS, I/F with external (wrt ATLAS) services and sub-systems.

Strong link of this section to Chapter 2, "Parameters".

5.2.1.1 TDAQ interfaces to ATLAS

Indicate what are the interfaces between TDAQ and ATLAS and reference where they are documented.

5.2.1.2 External interfaces

Given the external (i.e. non ATLAS) system and services as highlighted previously, define and reference the interfaces (and indicate the responsibilities).

Finally a summary table, which for any given interface, as defined in the previous section, say what type of data (viz. raw data) is exchanged. Possibly make references to data volumes/rates (where applicable) as documented in Chapter 2, "Parameters".

5.3 TDAQ Organisation

The purpose of the section is to show how TDAQ is organised (the system as such, not necessarily managerially) internally. The internal organisation is looked at from three perspectives: what function are performed by TDAQ, how functions are associated to TDAQ blocks, and a very abstract categorisation of internal elements. Generality (as opposed to implementation) and complementarity of views is stressed.

5.3.1 Functional decomposition

The TDAQ system provides the ATLAS experiment with the capability of: moving the detector data (physics events) from the detector to mass storage, selecting, between detector and mass storage, those events which are considered of physical interest, controlling and monitoring the whole experiment.

The following functions are identified:

- Detector read-out: the data produced by one bunch crossing are stored in detector memories (RODs), an event is therefore split in a number of fragments: there are ~ 1600 of such memories which have to be read-out at a rate of 75KHz into a set of TDAQ buffers (the event memory for TDAQ).

- Movement of event data: once buffered event fragments have to be moved to the high level triggers and, for selected events, to mass storage. This is a complex process which involves both moving small amounts of data at the level-1 trigger rate (the region of interest data for the level-2 trigger at 75 KHz) and the full event (i.e. ~ 1MB) at the level-2 trigger accept rate (few KHz).
- Event selection: TDAQ is repsonsible to reduce the rate and the data volume to the manageable amount of ~ 100MB/sec; this is achieved by a sophisticated, 2-level, trigger system.
- Event storage: events selected by the high level trigger system are written onto permanent storage for further offline analysis.
- Controls and monitoring: this refers to the capability of i) operating and controlling the experiment (detector, infrastructure, TDAQ) and ii) monitoring the state and behaviour of the whole of ATLAS.

5.3.2 TDAQ building blocks and sub-systems

The ATLAS TDAQ system is designed to provide the above functions in terms of the following building blocks:

- Read-Out System (ROS): event data is buffered, by the ATLAS detectors, in the RODs; each ROD holding a fragment of the whole ATLAS event. The ROD fragments are read by TDAQ into its own buffers, the “Read-Out Buffers” (ROBs). Logically, but not necessarily-implementation-wise, there is an equal number of ROB buffers as there are ROD fragments (indeed, see below, the level-2 trigger needs to access data at the level of the individual ROD fragments). Event fragments are kept in the ROB buffers until they are either moved downstream (accepted by the level-2 trigger) or they are removed from the system (rejected by level-2). The depth of the ROB buffers is determined by the time needed by level-2 to select events. The ROS provides individual event fragments, out of the ROBs, to the level-2 trigger and to the event builder: in this latter case a further level of buffering, multiplexing several individual ROBs into a single event builder input, may be provided by the ROS.
- Level-2 trigger: the level-2 trigger, as detailed in XXXXX, uses a mechanism to selectively read-out an event; that is, the level-2 trigger requests, as directed by the findings of the level-1 trigger, a small fraction of the event fragments in order to take a decision on the acceptance/rejection of the event. The ROI mechanism, using input from level-1, defines what fragments the level-2 trigger will need for a particular event. Appropriate fragments are requested from the ROBs and used to decide on the acceptance or rejection of that event. It is remarked that the level-2 trigger requests fragments on the basis of i) the level-1 identifier and ii) the ROL number (as opposed to a ROB number).
- Event Builder: the event is kept in the form of many (~1600) parallel streams up to the decision by the level-2 trigger. Any further reduction in the event rate needs working on the complete event, hence the requirement for a component which merges all the fragments of an event into a single place: the event builder.
- Event Filter (EF): another level of event rate reduction is provided by the event filter which requests complete events from the SFI buffers and performs on them complex selection algorithms.

- TDAQ controls: the function in charge of the control and supervision of the whole TDAQ system; this includes the initialisation and configuration of the TDAQ components. It also includes those ancillary functions, such as sharing (non event data) information between components.
- Detector controls: it represents the function in charge of controlling and monitoring all aspects of the ATLAS detector; it also includes the function of initialisation and configuration of the ATLAS detector.
- Monitoring: it is the part of TDAQ in charge of i) the (event data based) monitoring of the experiment and the operational monitoring of TDAQ.

Now explain that the work has been organised in terms of a dataflow, dcs, hlt, pesa, online sub-systems

5.3.3 Component categories

Here we characterise the components of TDAQ in terms of broad categories of elements: buffers, processors, supervisors and networks. It will be indicated what buffers (decouple parts of TDAQ, smooth differences in performances between parts of TDAQ), processors (selection at HLT level, monitoring, control), supervisors (L2SV, DFM to control the flow of the data) and networks (transport the data) do in the system.

In very broad terms the ATLAS TDAQ system is composed of :

- Buffers: they are used to decouple the different parts of the system: detector R/O, level-2, event builder and event filter. Because of the parallelism designed into the system, buffers belonging to the same function (e.g. ROBs) are independent.
- Processors: to run event selection algorithms, to monitor and control the system. They are organised in farms, groups of processors performing the same function.
- Supervisors: these elements coordinate the parallelism, in terms of assigning events to processors and buffers, at the different levels: the level-2 trigger (RoIB and L2SV), the event builder (DFM) and event filter.
- Communication systems: they connect buffers and processors to provide a path for tarnsporting event data or a path to control and operate the overall system. Communication systems are present at different locations in the system, some of them are switching networks, others may be point to point links.

5.4 TDAQ generic architecture

Now we put together and refine what we have said in Section 5.3, "TDAQ Organisation". The generic architecture is built upon and justified on the basis of that section. A backup document may be needed if more detail is required.

5.4.1 Architectural components

This is the list of the components which are visible at the level of the architecture, the list should include what is relevant in terms of functions, building blocks and abstract elements. Should include the functions and components identified in Section 5.3, "TDAQ Organisation": ROD, ..., RoIB, L2SV, online software major components such as control, DB, etc. For each component the following information should be provided: a definition or purpose (i.e. its function), and required performance. This section is neutral with respect to possible implementations. Why generic (and "unorthodox") names such as RRC and RRM, ROB instead of ROBin? The intent is to indicate that at that point in the system something is needed with a certain functionality (to connect and possibly mpx ROLs to ROBs, etc.).

The general, implementation independent, ATLAS TDAQ architecture is presented. *Note that we have carried forward most of the design originally presented in the ATLAS TP (1994), DAQ/DCS/HLT TP (2000); note were choices have been made (e.g. level-2 requesting data).*

The architecture is presented in terms of the functional breakdown of the previous sections. Reference to Chapter 2 is done to use / derive requried performance figures (based on the design L1 rate of 75KHz, some reference to the expected behaviour at 100KHz as well?).

5.4.1.1 Detector read-out

ROL (Read-Out Link): the communication link out of the detector buffers (RODs). Each ROD may have one or more ROLs; each ROL corresponds to one event fragment. The ROL is expected to transport data at a rate equal to the maximum event fragment size times the maximum level-1 rate (i.e. XXX MB/sec).

RRC (ROD to ROB connection): the connection between the ROL and the ROB may be multiplexed, that is to say one or more ROLs may be connected to a single ROB. Hence a functional element in the system which represents how ROLs are multiplexed into ROBs. Figures on required bandwidhts

ROB: the detector fragments are read out of the RODs and stored into TDAQ buffers; depending on the level of multiplexing provided by the RRC component, one or more fragments may be stored into a single ROB for the same event. *Figures on buffer depth; input and output bandwidth.*

RRM (ROB to ROS Multiplexor): in order to reduce the number of connections into the level-2 and event builder networks it is possible to funnel a number of ROBs into a single component. The RRM represents this ROB multiplexing capability. *Give figures on multiplexing capability (based on ROB output bandwidth).*

ROS (Read Out System): a component for serving data to the level-2 and event builder. It may also be used to introduce a further level of buffering before the event builder.

5.4.1.2 Level-2:

RoIB: the component which determines which fragments ought to be analyised by level-2 for a particular event, based on information received from the level-1 trigger. *(Note it runs at the L1 rate)*

L2SV: The level-2 trigger supervisor (L2SV): the component which, for a given event accepted by level-1, receives the information produced by the RoIB, assigns a L2PU to process the event and inputs the L2PU with the information provided by the RoIB.

L2PU: the component which, using the information produced by the RoIB, requests event fragments from the ROS, process them and produces a decision (accept/reject) for an event. The decision is passed to the ROS in order for this latter to remove (from the ROS buffers) or forward (to the final part of TDAQ, the event filter) the event.

L2N (level-2 netywork): the switching network used to connect all the ROSes, level-2 processors and supervisors for the purpose of moving ROI data and level-2 decisions between the TDAQ buffers, level-2 processors and supervisory components.Note that data and control share the same network. *Figures on expected bandwidhs and rates*.

5.4.1.3 Event Builder

DFM: the supervisory element which assigns an event, accepted by level-2, to an SFI.

EBN (Event Builder Network): the event builder will handle events at a rate of a few KHz; to achieve this performance several events are built concurrently into many SFI's by means of a switching network which connects ROSes, SFIs and DFM. Note that data and (event builder) control share the same network. *Figures on expected bandwidhs and rates*

SFI: the buffer where a full event is built prior to being moved to the event filter for further selection. *Target performance ~ 70MB/sec.*

5.4.1.4 Event Filter:

EFP (Event Filter Processors): A farm of processors, to run the algorithms; including possibly a supervisory component to assign events, available in the SFIs, to event filter processors. *Figure on expected time/event.*

EFN: A communication system connecting SFIs, event filter processing unit and SFOs. *Note that the issue here will be one of connecting a lot of processing units more than actual volumes of data (ratio processing to communication).*

SFO: A set of memories, sub-farm output (SFO), to buffer the events accepted by the event filter prior to writing the events to permanent mass storage. *Again expect performance ~ 70 MB/sec*

DCS (Detector Control System): *at this level of architectural detail the detector control system is seen as an unstructured entity which interfaces with the rest of TDAQ via the online network.*

5.4.1.5 Online:

OSF (Online Software Farm): the farm of processors on which the TDAQ software services, such as the run control and the monitoring facilities. A single partitions and the whole experiment are operated out of this farm.

DBS (Data Base Servers): the set of servers used to hold the data bases.

OSN (Online Software Network): a network connecting the Online software farm, the detector control system as well as the controller and supervisors local to the TDAQ components. A more detailed organisation of this network, showing which TDAQ elements have a controlle etc., is provided below in the detailed component views. *Some figures on expected performance and size of the network.*

The generic architecture is shown pictorially in Figures 5-2.

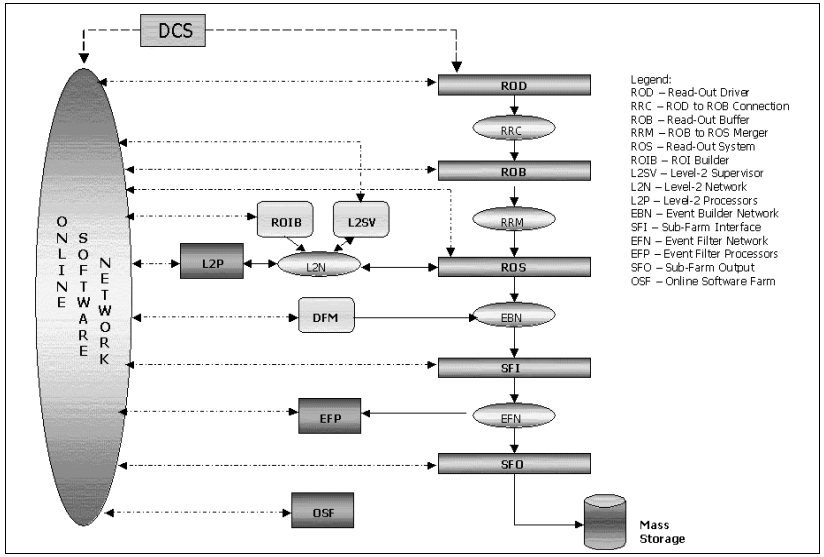


Figure 5-2 Generic diagram.

For missing details, one should refer to the "views" below to e.g. go into more details as regards their (the view's) parts. For example a ROB has a data flow view as buffer, but also a control view and a data base view.

The drawing has to be corrected to show LVL1 and its connection to the ROIB, the relationship L2SV/DFM.

5.5 TDAQ data flow architectural view

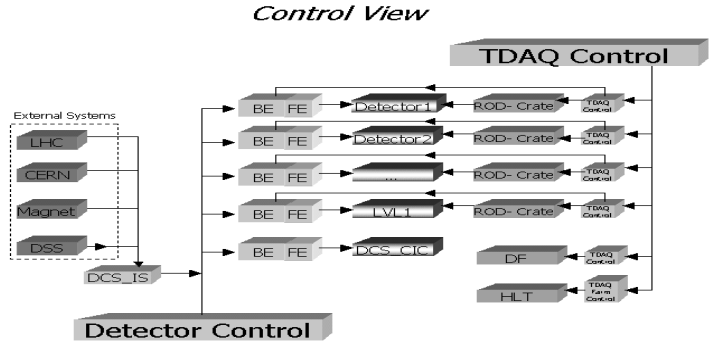
Specialise generic architecture for the purpose of Data flow.

Shall contain: functional decomposition into DF packages and sub-packages; interfaces and boundaries between DF packages and sub-packages; main use-cases realisation; "Event control and event flow" view which will include the rates and data volumes between DF packages and sub-packages (including type of communication).

5.6 TDAQ controls and supervision view

Specialised generic architecture for the purpose of control and supervision (eg show local controllers).

Includes both DCS and Online controls.



Remarks to this view:

This view has been chosen to illustrate the relation between the TDAQ Control and the DCS Control. Explanations and some reasons for choosing this view are listed below.

- *Arrows represent the direction of command flow.*
- *Lines without arrows represent infomation exchange which is vital for correct decision making by the master control unit (see later for which control is the master depending on the operational status). Component boxes represent logial entities.*
- *Systems expernal to Atlas are shown where they have a vital importane to the experiment control. The master control must have knowledge of the machine status in order to take correct decisions.*
- *LHC: LHC machine status; CERN: Cern infrastructure; Magnet: Magnet status; DSS: Detector Safety System*
- *During data taking periods when TDAQ Control is active it has master control over the TDAQ system and the Detector control system.*
- *Outside data taking periods, when TDAQ Control is not active, the Detector Control system stays fully operational and controls all its connected units.*
- *Each detector can be controlled independently both from the TDAQ Control including the Detector Control during data taking periods, for example during installation and test phases, or outside data taking periods via Detector Control.*
- *The Command flow from TDAQ Control to Detector Control is performed from the TDAQ control at the level of the different detectors.*
- *The presented components will be expanded and explained in more detail in the chapter on Experiment Control, when necessary details have been explained in the chapters on components and interfaces.*

5.7 Information sharing services view

Specialise the generic architecture for the purpose of information sharing services provided by the online software.

There are several arias where the information sharing is used in the TDAQ system: synchronisation between processes, error reporting, operational monitoring, physics event monitoring, etc. There are different types of information which TDAQ applications may share in different cases. The Online Software provides a number of services to support all the possible types of information exchange between TDAQ software applications.

As it is shown on Figure 5-3, each of those services acts as a common communication bus for all the TDAQ systems and detectors. Information can be shared between applications belonging to the same TDAQ system, among several TDAQ systems, and to each of the TDAQ systems and detectors.

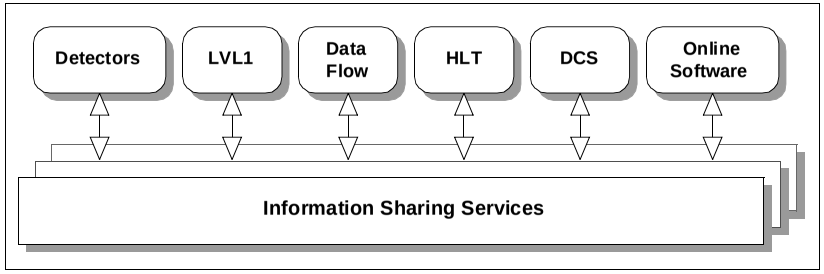


Figure 5-3 Information Sharing context diagram

All the Information Sharing services are partitionable in a sense that different instances of the same service are able to work in different TDAQ Partitions concurrently and fully independently.

5.8 TDAQ data base view

Data base architecture: including where access to (in and out of) databases is done.

Remark: This is a very basic view of the databases in TDAQ. It is expected that more details can be presented when a common understanding is reached on the sharing of non-event data accross DCS, DAQ, HLT and offline systems. This will be the topic of discussion at the next ATLAS week and the next Atlas Software workshop.

TDAQ and detectors are using configuration databases to describe their system topology and the parameters which are used for data-taking. A variety of configurations can describe and combine different combinations of existing partitions which are prepared for different types of runs (physics, calibration, debug, shutdown, etc.).

The TDAQ and detectors are using offline conditions databases to read and to store conditions under which the event data were taken. Such databases are used by the offline group for analysis and reconstruction of physics data and by the TDAQ experts to analyse logs of the operational monitoring information stored during data taking by the online bookkeeper.

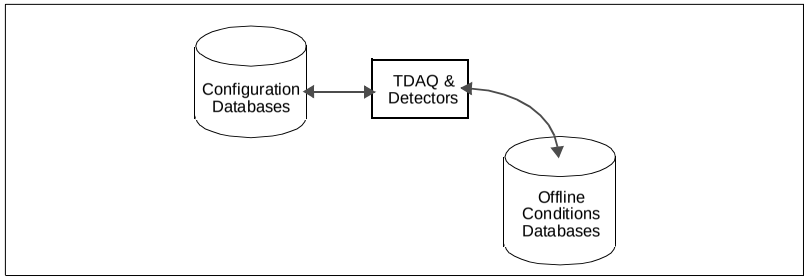


Figure 5-4

5.9 HLT view

The HLT issues relative to the generic architecture. It should probably include the organisation of the EF and LVL2 blobs, how HLT gets at the data.

5.10 Partitioning

The definition of partitions and the allowed operations are defined in Chapter 3, "Operationial requirements for the TDAQ system". We remind that partitioning refers to the capability of providing the functionality of the complete TDAQ to a subset of the ATLAS detector.

The definition of the detector subset defines, because of the connectivity between RODs and ROBs, which ROBs belong to the partition. Downstream of the ROBs a partition is realised by assigning part of TDAQ resource (EBN, SFI, EF, online farm and network) to the partition: it is a resource management issue. In particular a subset of the ROBs, as mentioned above, and a subset of the SFIs *Note that this assumes that assigning SFIs implies associating a sub-set of the EF farm; if this is not the case then routing of events has to be done by the SFIs (see connection to TTC below).*

As regards the transport of the data across the allocated resources, the DFM plays the key role of routing subsets of ROBs to the associated subsets of the SFIs. In order for this to happen, in the case of partitions associated to non physics runs (i.e. when there is no level-2), the DFM must receive, via the TTC, the triggering information for the active partitions. *Need for connections of up to 35 TTC systems to DFM*

5.11 Scalability of the system

How the generic arch. can scale in performance (probably with respect to LVL1 rate). Viz. what has to be expanded, and how. A strategy which show that the architecture can scale.

5.12 Baseline architecture implementation

The baseline would be defined in terms of a concrete implementation of the generic components defined above (e.g. what is used to connect ROD to ROBs, the rob is a ROBIN, etc). It certainly should spell out what the generic components are (for example a bus based ROS, a point to point link for the ROL) and probably suggest a physical implementation (e.g. this switch is gigabit ethernet with this size). We could also have a sub-section which indicates options (a small number) which one might wish to consider later on.

The justification of the validity of the overall architecture is to be spelled out in Part 3 System Performance.

5.13 References

5-1 Document from Architecture working group on global architecture.

5-2 DataFlow Architecture document.

5-3 ROS Architecture document.

5-4 Data Collection Architecture document.

6 Fault Tolerance and Error Handling

6.1 Fault Tolerance and Error Handling Strategy

Error handling and fault tolerance are concerned with the behaviour of the TDAQ system in the case of failures of its components. By failure we mean the inability of a component to perform its intended function. This includes both hardware and software caused problems.

The overall goal is to *maximize system up-time, data taking efficiency and data quality* for the ATLAS detector. This is achieved by designing a *robust* system that will keep functioning even when various parts of it are not working properly.

Complete fault tolerance is a desired system property which does not imply that each component must be able to tolerate every conceivable kind of error. The best way for the system to achieve its overall goal may well be to simply reset or reboot a component which is in an error state. The optimal strategy depends on the impact the faulty component has on data taking, the frequency of the error and the amount of effort necessary to make the component more fault tolerant.

The fault tolerance and error handling strategy is based on a number of basic principles:

- Minimize the number of single points of failure in the design itself. Where unavoidable, provide redundancy to quickly replace failing components. This might consist of spare parts of custom hardware or simply making sure that critical software processes can run on off-the-shelf hardware which can be easily replaced.
- Failing components must affect as little as possible the functioning of other components.
- Failures should be handled in a hierarchical way where first local measures are taken to correct it. Local recovery mechanisms will not make important decisions, e.g. to stop the run, but pass the information on to higher levels.
- All errors are reported in a standardized way to make it easy to automate detection and handling of well-defined error situations (e.g. with an expert system).
- All errors will be automatically logged and be available for post-mortem analysis if necessary. Where the error affects data quality the necessary information will be stored in the condition database.

We distinguish the following cases:

Error detection describes how a component finds out about failures either in itself or neighbouring components. Errors are classified in a standardized way and may be transient or permanent. A component should be able to recover from transient errors by itself once the cause for the error disappears.

Error response describes the immediate action taken by the component once it detects an error. This action will typically allow the component to keep working but maybe with reduced functionality. Applications which can sensibly correct errors that are generated internally or occur in hardware or software components they are responsible for should corrected them directly.

In many cases the component itself will not be able to take the necessary action about failures in a neighbouring component. Even if the component is unable to continue working, this should not be a fatal error for the TDAQ system if it is not a single point of failure.

Error reporting describes how the failure condition is reported to a higher level which might be able to fix the error condition. The mechanism will be a standardized service which all components use. The receiver of the error message might be persons (like a shifter or an expert) or an automated expert system.

Error recovery describes the process of bringing the faulty component back into a functioning state. This might involve manual intervention by the shifter or an expert or an automated response initiated by the expert system. The time-scale of this phase will typically be longer than the previous ones and can range from seconds to days (e.g. in the case of replacing a piece of hardware which requires access to controlled areas).

Error prevention describes the measures to be taken which prevent the errors to be introduced to hardware or software. Good software engineering, the use of standards, training, testing and the availability and use of diagnostic tools help in making the TDAQ system fault tolerant.

6.2 Error Definition and Identification

In order to respond to error conditions it is important to have a clearly defined TDAQ wide classification scheme that allows proper identification. It is assumed that error conditions are detected by data flow applications, controllers, event selection software and monitoring tasks. These conditions may be caused by failures of hardware they control, of components that they communicate with or these may occur internally.

The sources have a dual responsibility: correct anomalous conditions immediately or issue an error message, suitably classified and containing all necessary information for subsequent action by human or expert system.

Error messages are classified according to severity. The classification is necessarily based on local judgement; it is left to human/artificial intelligence to take further action, guided by the classification and additional information provided by the applications that detect the errors:

Additional information consists of a unique TDAQ wide identifier (note that status and return codes, if used, are internal to the applications), determination of the source and additional information needed to repair the problem. All messages are directed to an Error Reporting Service, never directly to the application that may be at the origin of the fault.

For successful fault tolerance, it is essential that correct issuing of error messages is enforced in all TDAQ applications.

6.3 Error Reporting Mechanism

Applications encountering a fault make use of an error reporting facility to inject an appropriate message to the TDAQ system. The facility is responsible for the message transport and message distribution. Optional and mandatory attributes can be passed with the message. The facility allows receiving applications to subscribe to a message according to the severity or other qualifi-

ers independent of its origin. A set of commonly used qualifiers will be recommended. These can for example include the detector name, the failure type like hardware, network, software failures, or finer granularity indicators like ‘Gas’, ‘HV’ etc. They provide together with mandatory qualifiers like process name and id, injection date and time, and processor identification provide a powerful and flexible system logic for the filtering and distribution of messages.

6.4 Error Recovery Mechanisms

Error recovery mechanisms describe the actions which are undertaken to correct any important errors that a component has encountered and can not handle on its own. The main goal is to keep the system in a running state and minimize the consequences for data taking.

There will be a wide range of error recovery mechanisms, depending on the subsystem and the exact nature of the failure. The overall principle is that the recovery for a failure should be handled as close as possible to the actual component where it occurred. This allows both to isolate failures to subsystems without necessarily involving any action from other systems, to decentralize the knowledge required about properly reacting to a failure and to allow experts to modify the error handling in their specific subsystem without having to worry about the consequences for the full system.

If a failure cannot be handled by a subsystem at a given level, it will be passed on to a higher level in a standardized way. While the higher level will not have the detailed knowledge to correct the error, it will be able to take a different kind of action which is not appropriate at a lower level (e.g. it might be able to pause the run and draw the attention of the shifter to the problem, or to take a subfarm out of the running system and proceed without it etc.)

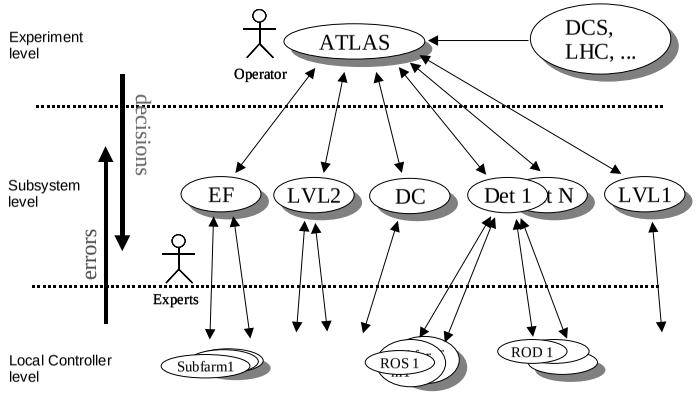
The actual reaction to the failure will strongly depend on the type of error. The same error condition (e.g. timeouts for requests) may lead to quite different actions depending on the type of component. A list of possible reaction is given in Chapter 6.5, "Typical Use Cases".

Each level in the hierarchy will have different means to correct failures. Only the highest levels will be able to pause data taking or decide when to stop a run.

6.4.1 Verification, Diagnostic and Automatic Recovery

Regular verification of the system status and its correct functioning will be a vital operation to help avoiding failures to occur. A customizesable diagnostic and verification framework will allow to verify the correct status of the TDAQ system before start-up or between runs, automatically or on request. It will make use of a suite of custom test programs which are specific for each component type in order to diagnose eventual faults. It will be integrated into the hierarchical structure of the TDAQ supervision framework and can (optionally) take automatic action for the recovery of a fault. It provides multi-level decentralized error handling and allows actions on failures on a low level. A knowledge base containing the appropriate actions to be taken will be established at system installation time. Experience from integration tests and in test beam operation will initially provide the basis. Each supervision node may contain a rules customized to its specific role in the system. Table 6-1 gives an example on the hierarchical distribution of supervision levels.

Table 6-1 Error handling levels



Remark: the diagram will have to be adjusted in style etc. to map the definitions made in earlier chapters.

6.5 Typical Use Cases

This paragraph will describe how a component would react to some typical faults both in a global approach and discussing any system specifics. The current colection is incomplete. Contributions should be provided by the systems. It is the intension to include details which would go bejond the scope (or space) in the TDR in a supporting document.

ROL (flow control, missing ROD fragments, failure); DF applications (failure of one or more); control and/or event data messages (packet loss, flow control, QOS (peer to peer or switches). Results from modelling may be used to justify.

A short list of possible reactions on different levels (from inside an application to system wide) and their impact on data taking follows:

- Symptom: Read-out link not working properly.
  - Action: Reset of local hardware.
  - Impact: Some parts of the event might be missing. If successful only an informational message would be send to the higher level. If not successful a error message would be issued.
- Symptom: Timeout for requests to a ROS inside a LVL2 node.
  - Action: Retry a configurable number of times.

- Impact: Parts of an event might be missing. If not successful, the LVL2 trigger might not be able to run its intended algorithms and the event has to be force-accepted. If the error persists, the data taking efficiency might drop because the event building will be mostly busy with forced-accept events.

- Symptom: Dataflow component reports that ROS times out repeatedly.
  - Action: Pause the run, remotely reset the ROS component and if successful resume the run. If not successful, inform all concerned components that this ROS is no longer available and inform higher level (who might decide to stop the run and take other measures like calling an expert).
  - Impact: Data missing in every event.
- Symptom: LVL2 supervisor event request to LVL2 node times out.
  - Action: retry a configuration number of times. Then take node out of scheduler and report to higher level.
  - Impact: Available LVL2 rate is reduced
- Symptom: LVL2 Supervisor reports that LVL2 node repeatedly timed out.
  - Action: Remotely reset the offending node. If successful, the node should come back into the run. If not successful the
  - Impact: LVL2 rate is reduced while node is reset.
- Symptom: None of the nodes in a Eventfilter subfarm can be reached via the network (e.g. in case of a switch failure).
  - Action: Take all affected nodes out of any scheduling decisions (e.g. in the DFM) to prevent further timeouts. Inform higher level about the situation.
  - Impact: Data taking rate is reduced.

As can be seen, the same error condition (e.g. timeouts for requests) leads to quite different actions depending on the type of component. Each ROS is unique in that its failure leads to some non-recoverable data loss. A LVL2 node on the other side can be easily replaced with a different node of the same kind.

6.5.1 Reliability and fault tolerance in the Data Flow

This section was moved from Chapter 8, "Data-flow" and inserted as is here. It is most likely that this will be reworked into Section 6.5, "Typical Use Cases".

This section presents the major error use cases of the DataFlow. As a guideline to identifying the error use cases, major should be interpreted as referring to those that have directly influenced the design of the DataFlow. Each use-case is described as having transient, accumulative or persistent effect on the behaviour of the DataFlow. The handling of each use-case is presented based on results of real life tests. The exact layout of this chapter is subject to the identification of the major error use cases.

Each subsection groups related error use cases.

6.5.1.1 Detector read-out

Possible error use cases here are: ROL failure; assertion of one or more of the error bits in the S_LINK end of frame control word, i.e. ROD fragment corruption; assertion of S-LINK LDOWN; missing or out of sequence ROD fragments.

6.5.1.2 Level 1 to RoI builder

This sub section should be a summary of what is detailed in [8-27].

6.5.1.3 Control and event data messages

How the system handles the loss of each type of control message and event fragments separately.

6.5.1.4 Applications

How the system handles the failure of one or more of the applications.

6.5.2 Reliability and fault tolerance in the XXX system

describe important reliability and fault tolerant aspects in the respective system

6.6 References

Working Group Report of the Atlas TDAQ Error Handling and Fault Tolerance Working Group

7 Monitoring

7.1 Overview

A fast and efficient monitoring is essential during the data taking periods. Any malfunctioning part of the experiment must be identified and signalled as soon as possible so that it can be cured. The sources of monitoring information may be events, fragments of events or any other kind of information (histograms, counters, status flags, etc...). They may come from the hardware, processors or network elements, either directly or via the DCS. Some malfunctions can be detected by the sole observation of a single piece of information and could be performed at the level of the source of the information. An infrastructure has to be provided to process the monitoring information and bring the result to the end user (normally the shift crew).

The monitoring can be done at different places of the Data Flow in the DAQ system: ROD Crate, ROS, and SFI. Moreover, additional monitoring can be provided by the LVL2 trigger and by the Event Filter due to the fact that these programs will decode data, compute tracks and clusters, count relevant quantities for simple event statistics or to monitor the functioning of the various trigger levels and their selection power. It is possible to foresee that a part of the Event Filter is dedicated to monitoring activities where events tagged at read-out level or at previous stages of the selection are routed by the Data Collection (Event Builder) for special treatment.

The ideas which are expressed in the present text reflect the very preliminary stage of the work of the ad hoc working group on Monitoring. Many of them are likely to evolve until the final edition of the TDR, when discussions with all concerned communities (TDAQ, detectors) bring new input.

7.2 Monitoring sources

which ones of the ATLAS systems and sub-systems need to be monitored during data taking

7.2.1 DAQ monitoring

7.2.1.1 Front-end and ROD monitoring

sub-detector front end electronics specific monitoring

- data integrity monitoring
- operational monitoring (throughput and similar, scalers histograms)
- hardware

7.2.1.2 Data Collection monitoring

DAQ specific monitoring

- data integrity monitoring
- operational monitoring (throughput and similar, scalers histograms)
- hardware

7.2.2 Trigger monitoring

7.2.2.1 Trigger decision

simulate the decision of the trigger stages to confirm the quality of the decision

7.2.2.1.1 LVL1 decision

sample of rejected events passed on a pre-scaled basis to dedicated processing tasks (possibly in a dedicated partition of the Event Filter)

7.2.2.1.2 LVL2 decision

same as LVL1

7.2.2.1.3 EF decision

detailed information appended to the event, for a sub-set of accepted and rejected events for offline analysis.

7.2.2.1.4 Classification monitoring

In terms of monitoring, classification is a very important output of both LVL2 and EF processing. It consists of a 128-bit bitmap which records which signatures in the trigger menu were passed. Histograms can be filled locally on the processors where the selection is performed. With an accept rate of 1 kHz for LVL2 and 200 Hz for EF, and assuming a sampling rate of 0.1 Hz, a 1 byte depth is sufficient for the histograms. For both LVL2 and EF farms, the rate for the transfer of the histograms is therefore 1.2 kbyte/s.

7.2.2.2 Physics monitoring

The most simple approach to monitor the quality of the physics which is sent to permanent storage will consist in measuring the rates for some physics channel. It may be performed easily in the EF. A part of the results of these monitoring operations could be appended to the event bytestream for offline analysis. Others could be sent to the operator via the standard Online Software media for an online analysis.

Histograms of the rates for every item of the trigger menu as a function of time should be recorded, with the relevant variables with which they must be correlated (e.g. the instantaneous luminosity). Such histograms can give very quickly an evidence for malfunctioning, although their interpretation may be quite tricky.

Well-known physics channels could be monitored so that one could permanently compare the observed rates with the expected ones. The list of such channels should be established in collaboration with the physics groups.

Information coming from the execution of reconstruction algorithms may be of interest. One could monitor e.g. the number of tracks found in a given detector on a per event basis. There again, a comparison with reference histograms may be of great help to detect malfunctioning. Input is required from offline reconstruction groups.

7.2.2.3 Operational monitoring

Everything related to the “system” aspects, e.g. transportation of the events or event fragments, usage of computing resources, etc...

7.2.2.3.1 LVL1 operational monitoring

The integrity and correct operation of the LVL1 trigger will be monitored at both the hardware level by processes running in trigger crate CPUs and also by monitoring tasks in the Event Filter.

The LVL1 trigger is the only place where every bunch crossing is processed and where a crude picture of the real beam conditions can be found. For example, the calorimeter trigger fills histograms, in hardware, of the "level 0" rates and spectra of every trigger tower and can quickly identify, and if necessary suppress, hot channels. Hardware monitoring is also used to check the integrity of links between the successive steps in the trigger processor pipeline.

At the Event Filter, monitoring tasks will check for errors in the trigger processors at a lower rate than hardware monitoring, but with greater diagnostic power. Event Filter tasks will also produce various histograms of trigger rates, their correlation and history.

7.2.2.3.2 LVL2 operational monitoring

The LVL2 selection software runs as part of the Data Collection (DC) in the L2PU [L2MonitMRS]. It will therefore use the DC infrastructure and hence the monitoring tools foreseen for this system. The following aspects, relevant of DC, will be monitored :

- trigger, data and error rates
- CPU activity
- queue occupancies (load balancing)

Monitoring of the quality of the data by LVL2 processors is not envisaged. Indeed, the available time budget is limited because of the necessity to release data from the ROB. Monitoring a fraction of the events in the L2PU is not desirable since this would introduce large variations in LVL2 latencies as well as possible points of weakness in the LVL2 system. The necessary monitoring of the LVL2 quality shall therefore be delegated to the downstream monitoring facilities, i.e. the EF (or online monitoring farm) and the offline analysis. One should however discuss very carefully the opportunity to fill in L2PU some histograms, possibly read at the end of the run, so that a high statistics information is given, which could not be reasonably be obtained by

using forced accepted events on a pre-sampled basis. The evaluation of the extra CPU load for such operations should be made.

7.2.2.3.3 EF operational monitoring

The monitoring of the data flow in the Event Filter will be primarily done directly at the level of the EFD process. Specific EFD tasks, part of the main data flow, will be in charge of producing relevant statistics in terms of throughput at the different levels of the data flow. They have no connection with other processes external to EFD.. The detailed list of information of interest for the end user has not yet be finalised and will continue to evolve all along the lifetime of the experiment.

Among the most obvious parameters which are going to be monitored, one might quote :

- the number of events entering the Farm
- the number of events entering each sub-farm
- the number of events entering each processing host
- the number of events entering each processing task
- the number of events selected by each processing task, as a function of the physics channels present in the trigger menu
- the same statistics as above at the level of the processing host, the sub-farm and the Farm

Other statistics may be of interest such as the size of the events, as a function of different parameters (the time, the luminosity of the beam, the physics channel). As stated above, the identification of these statistics will be formulated more precisely at a later stage of the development of the experiment.

The results of the data flow monitoring will be sent to the operator via standard Online SW media (e.g. IS or Histogram Service in the present implementation).

7.2.2.3.4 PESA SW operational monitoring

Very preliminary ideas on monitoring in PESA SW have been presented during the HLT integration Workshop. A first list of parameters which could be monitored for debugging purpose and comparison with modelling results has been given:

- length of time spent in each algorithm
- frequency at which each algorithm is called
- number of steps in the step sequencer before rejection
- info and debug messages issued by the PESA SW

Some of these points could be also monitored all along the duration of normal data taking.

Hooks necessary for these profiling measurements should be implemented directly at the level of the base class of the GAUDI algorithm. Profiling tools such as NetLogger for coarse meas-

urements and TAU have already been studied in the context of LVL2 and their use at a larger scale will be considered in PESA SW.

An Online Event Display, with the whole functionality of the offline version but working with events sampled after the Event Builder, including accepted as well as rejected events, is considered as a very powerful monitoring tool.

7.2.3 Detector monitoring

The detector monitoring can be done at different places of the Data Flow in the DAQ system: ROD Crate, ROS, and SFI. Moreover, additional monitoring can be provided by the LVL2 trigger and by the Event Filter due to the fact that these programs will decode data, compute tracks and clusters, count relevant quantities for simple event statistics or to monitor the functioning of the various trigger levels and their selection power.

For small test systems a monitoring at the ROD Crate level would be used. This has been achieved already during the recent test beams using the current version of the DAQ-1 software, writing an event sampler at the ROD crate. The Online Software Sampler is then used to make these monitored events available to Monitoring tasks implemented by the users.

Monitoring at the ROD crate would help in the experiment to check the functioning of the readout chain or to debug specific problems at this stage of the data flow. Some sub-detectors will make use of monitoring at this stage to monitor the possible additional tasks performed at this level as the fitting procedure to find the weights to be applied to the ADC samples for the optimal filtering procedure (e.g. LAr and Tilecal).

The monitoring at the ROS level will be also required in experiment for several uses. It will be required to monitor at the ROS level the events that are rejected by the LVL2 to monitor the correct functioning of the selection system. Moreover the monitoring at the ROS level will be very useful to get useful data of a given sub-detector. In effect, according to the present detector readout organisations, a ROS may represent a significant part of a sub-detector and therefore monitoring at the ROS level may give very useful feedback on a high statistical basis.

It is worthwhile to stress the fact that monitoring at the level of SFI is different from that at the level of EF, the first one being much more general and therefore very useful, the other one being CPU consuming and obliging all detectors to have an EF system which is not presently the case. To build meaningful Processing Tasks at the EF level is not a triviality and in these CPU intensive calculations one cannot include monitoring at zero cost.

The monitoring at the level of the SFI will be needed in experiment. This is the first place where the complete assembled event is in the dataflow chain. An high statistic monitoring at this stage can help in monitoring the functioning of the detector and to establish the first correlation between sub-detectors looking at simple quantities and without expensive calculations that can be instead performed at the EF level. At the SFI level one could also think of implementing an Event Display that is an heavy task for the EF tasks already busy for the last level decision. Moreover monitoring events at the SFI level will help in understanding also the rejection of the Event Filter.

It is worthwhile to stress the fact that monitoring at the level of SFI is different from that at the level of EF, the first one being much more general and therefore very useful, the other one being CPU consuming.

The monitoring at the level of the EF is very promising, but it requires a lot of effort on the software side for the detectors experts for developing detector calibration and reconstruction tasks that have to be fast enough to cope with the EF latency. The same programs used for offline tasks may turn out to slow down the system put a back pressure on the Data Flow. This aspect has to be further developed with the latest measurements.

More input expected from Monitoring Workshop

7.3 Monitoring destinations and means

Where and how (which tools) to perform monitoring operations

7.3.1 Online Software services

The Online Software provides a number of services which can be used as monitoring mechanism which is independent of the main data flow stream. The main responsibility of these services is to transport the monitoring data requests from the monitoring destinations to the monitoring sources and to transport the monitoring data back from the sources to the destinations.

There are four services provided for a different types of the monitoring information:

- **Event Monitoring Service** - is responsible for transportation of physical events or event fragments sampled from well-defined points in the data flow chain to the software applications which can analyse them in order to monitor the state of the data acquisition and the quality of physics data of the experiment.

- **Information Service** - is responsible for exchange of user-defined information between TDAQ applications and aimed to be used for the operational monitoring. It can be used to monitor the status and various statistics data of the TDAQ sub-systems and their hardware software elements;

- **Histogramming Service** - is a specialisation of the Information Service with the aim of transporting histograms. It accept several commonly used histogram formats (lik3 ROOT histograms for example) as the type of information which can be send from the monitoring sources to the destinations;

- **Error Reporting Service** - provides transportation of the error messages from the software applications which detect these errors to the applications which are responsible for their monitoring and handling.

Each service offers the most appropriate and efficient functionality for a given monitoring data type and provides specific interfaces for both monitoring sources and destinations.

7.3.2 Monitoring in the Event Filter

From the beginning of the design of the EF, it has been foreseen to perform some monitoring activities in it, in addition to the ones related directly to the operation. EF is indeed the first place in the data taking chain where the full information about the events is available. Decisions from the previous levels of the trigger system can be checked from both accepted and rejected (on a pre-scaled basis) events. Information coming from the reconstruction phase, which generally requires a large amount of CPU power, can be rather easily re-used, leading to large savings in

terms of computing resources. Finally, the fact that EF is part of the data taking chain ensures that pertinent information will be made available to the shift crew in the best time delays.

Monitoring in the Event Filter, or more generally monitoring after the Event Builder, can be performed in different places :

- directly in the filtering tasks (which raises the problem of the robustness of the monitoring code),
- in dedicated monitoring tasks running in the context of the Event Filter (then, one should think of passing the information gathered in the filtering task to take profit of the already used CPU)
- or in a dedicated, independent sub-farm. In that case, that is the Data Collection DFM which takes care of directing potentially interesting events (tagged at read-out, LVL1 or LVL2 levels).

Those different possibilities are not mutually exclusive.

7.4 Archiving monitoring data

Data which is produced by monitoring activities should be archived by some bookkeeping service so that it can be cross-checked offline with more detailed analysis. One should also store (in a dedicated channel ?) events whose acceptation has been forced at any level of the selection chain. These events are necessary to evaluate precisely the acceptance of the trigger.

7.5 Monitoring requirements on networks

Monitoring matrix

Part 2

System Components

8 Data-flow

8.1 (Possible introduction)

8.2 Detector read-out and event fragment buffering

8.2.1 Read-out link

Each sub-detector reads data from the detector over Front-End links and uses a ROD to multiplex the data. Each of the sub-detectors has different requirements (and different designers;-) consequently the implementation of the ROD varies between sub-detectors. The guidelines for designing the ROD are set out in the Trigger & DAQ Interfaces with Front-End Systems: Requirement Document [8-1]. The purpose of the ROL is to connect the sub-detectors to the TDAQ system and it is responsible for transmitting error-free data from the output of the ROD to the input of the ROS, the first element in the TDAQ chain.

The ROL requirements have been stable since the High-level Triggers, DAQ and DCS Technical Proposal TP [8-2]:

- 32 bit data at 40.08Mhz, (~160 MByte/s)
- A control bit to identify the start and end of an event
- Xon/Xoff flow control
- Error detection, error rate < 10^{-12}
- A maximum length of 300m for the fibre version, 25m for the electrical version.

To ensure homogeny, the output of the ROD is defined by the S-LINK specification [8-3]. In addition, The raw event format [8-4] defines the order and content of the information transmitted from the ROD. At the other end of the ROL, the ROS inputs are identical for all sub-detectors and also conform to the S-LINK standard.

The S-LINK specification has been stable since 1996. It is used in COMPASS and in other LHC experiments, e.g. CMS. S-LINK is an interface definition; it only defines protocols and recommends connector pin-out. As shown in Figure 8-1, the ROD end of the ROL is called the Link Source Card (LSC) and the ROS end the Link Destination Card (LDC). They are connected by optical fibres or copper cables. Event data flows from the LSC to the LDC on the forward channel. Flow control information, i.e. the ROS can stop the ROD sending data if it's input buffers are almost full, flows from the LDC to the LSC on the return channel.

The DIG - ROD Working Group have also recommended that the LSC be placed on a mezzanine card to facilitate support and upgradeability [8-5]. The form factor of these mezzanine cards is based on the CMC [8-6] standard.

The LSC plugs onto the S-LINK connector on the ROD (or its associated rear transition card). For the forward channel, a Field-programmable gate array (FPGA) handles the protocol and delivers words to a serial/deserialiser (SERDES) chip which performs parallel-to-serial data con-

Figure 8-1 The relationship between the S-LINK and the ROL.

version and encoding. The output of the SERDES drives an optical transceiver that in turn feeds the optical fibre. The operation of the receiving card, the LDC, is a mirror image of the LSC. In fact the current LSC and LDC are physically the same card with different programs in the FPGA.

Various prototype implementations of the ROL have been built to prove the concept and measure the performance. The previous version of the ROL, the ODIN, used a physical layer that was based on the Hewlett Packard G-LINKs (HDMP-1032/1034). They have also been used successfully in ATLAS test-beams and eighty of these ROLs are being used in the COMPASS experiment. However, the maximum bandwidth is limited by the G-LINK at 128 MByte/s. Following the second ROD workshop, the requirements of the ROL were increased to 160MByte/s and a second version of this link was designed which used two G-LINKs chips per channel. Unfortunately, this raised the cost as two pairs of fibres and associated connectors were required.

Another recommendation of the ROD Working Group was to build a ROL that would use only one pair of fibres. This has been achieved by using 2.5 Gbit/s components in the current design, the High-speed Optical Link for ATLAS (HOLA) [8-7]. In this implementation a small FPGA, the EP20K30E APEX 20K, handles the S-LINK protocol. The SERDES chip is a Texas Instruments TLK2501 running at 2.5 Gbit/s for both for the forward and for the return channel (one per card). For the optical transceiver, the Small Form Factor Pluggable (SFP) Multimode 850 nm 2.5 Gbit/s with LC Connectors is recommended, e.g. the Infineon V23818-N305-B57. The use of pluggable components allows the optical components to be changed in case of failure.

Test equipment has been developed for the ROD/ROL/ROS. This includes an emulator that can be placed on the ROD to check that the ROD conforms to the S-LINK specification. Similarly, an emulator exists that can be placed on a ROS to emulate a ROL connection. The emulators allow ROD, ROL and ROS designs to be tested at full bandwidth and errors to be introduced in a controlled manner. The HOLA was produced and tested in 2002 and satisfies all requirements of the ROL.

In addition, for the purposes of exploitation in laboratory test set-ups and in test-beams, i.e. further testing, cards exist which allow the ROL to be interfaced to the PCI Bus in a PC. Performance measurements of this interface [8-8] have shown that data can be transferred into a PC at

160 MByte/s using a single ROL input. Modifications to the firmware have allowed the emulation of an interface with four ROL inputs. Measurements using this emulator have demonstrated a bandwidth of 450 MByte/s into a PC. The next version of the interface, the FILAR, will have four ROLs on-board and should be ready for the April 2003 test-beam.

The purchase of the cards, in small quantities, is handled by the CERN stores. For quantities required for ATLAS a tendering process will be initiated in 2003 thus ensuring the availability of larger quantities during 2004. The production schedule will be adapted to the requirements of the sub-detectors who have been asked by the DIG to provide estimates of quantities for the years up to the start of the LHC. Maintenance and short-term loans of equipment will probably be handled by EP/ESS.

8.2.2 Read-out subsystem

8.2.2.1 High Level Design

The ROS has three major components: the RobIn, the IOManager and the LocalController. Figure 8-2 shows the relationship between the three ROS components and any other relevant TDAQ component. A complete high level design of the ROS can be found in [8-10], only a summary is presented here.

Figure 8-2 Main ROS components and their relationship with the other TDAQ components.

The RobIn component provides the temporary buffering of the individual data fragments produced by the RODs. All incoming ROD event fragments are buffered for the duration of the LVL2 trigger decision time. It thus needs to receive and buffer incoming ROD event fragments at the full LVL1 trigger rate. The ROD event fragments are buffered The date In addition, also sends on request the accepted ROD event fragments.

Due to these very demanding requirements the baseline RobIn is designed and implemented as a custom hardware module. Section 8.2.2.2 describes the high level design of the RobIn component and the results of measurements obtained with a prototype RobIn.

The main function of the IOManager is the servicing of requests for data by the High Level Triggers. According to the criteria specified in the data request, the IOManager collects ROB fragments from one or more RobIns and builds a ROS Fragment. This fragment is then sent to a destination specified in the original request. It also receives from the High Level Triggers the request to release buffer space occupied by ROD event fragments. These event fragments have either been rejected by the LVL2 trigger or successfully built by the Event Building. So as to maximise the overall performance of the ROS, the design of the IOManager allows a number of data requests and releases to be handled concurrently that is to say, it overlaps I/O operations with the processing of requests.

In the baseline TDAQ implementation the IOManager units are implemented as multithreaded software processes.

The LocalController also provides a single interface point between the ROS and the Online software (configuration database, run control, message reporting, monitoring and process control). In particular the LocalController is responsible for retrieving the relevant information from the configuration database and sending it to the IOManager component. The IOManager also provides for the configuration, control and operational monitoring of its associated RobIns.

As the IOManager and the RobIn components, the LocalController are organized in specific units, each of which is connected to one or more IOManager units. In the TDAQ baseline implementation the LocalController is implemented as a multithreaded software processes.

In one of the possible future ROS deployment scenarios there would not be any IOManager component and all the RobIn units would be directly visible outside of the ROS system. In such a scenario all the control, monitoring, error handling functionalities provided by the IOManager will have to be provided by the other components or sub-systems, i.e. LocalController and DataCollection. No data multiplexing will be provided at the level of the ROS, and the different systems will always have to send data requests to the individual RobIn units and handle the individual ROB Fragments coming back from all of them.

8.2.2.2 Design of the ROBIN

The RobIn is located at the boundary between the detectors and the ROS. It receives ROD event fragments from a number of ROLs. Its basic functionality can be described by the following four tasks:

- Receive ROD event fragments from the ROL
- Buffer ROD event fragments
- Send ROD event fragments, on request, to the High Level Triggers
- Release ROD event fragments, on request, from the buffer.

Figure 8-3 shows the context of the RobIn.

The baseline RobIn takes into account the experience and results of studies from previous prototyping studies [8-12], [8-13] and the requirements on it are documented in the ROS-URD [8-9]. The final design of the RobIn will based on the final prototype whose complete design is docu-

Figure 8-3 Context of the RobIn.

mented in a set of documents [8-14], [8-15] and [8-16]. Here only a summary description is given.

Andreas: I intend to omit all the details of the function description which can be taken from HLDD and DLDD.

Editor: A reduced description needs to be given here.

Andreas: Should we elaborate on the issues of changing the ROL to GE, multiplexing etc?

Editor: What we finally put here should reflect the baseline choice, not an either or.

Referring to Figure 8-4, the primary functions of the RobIn (receive, buffer, send and release) are mapped onto a small number of specialised building blocks: ROL-IF - CORE - MEM - TDAQ-IF. It supports two ROLs, the data from which are buffered in separate buffers. All functionality related to the receiving of ROD fragments from the ROLs are realised in an FPGA, the CPU handles the issues related to management and monitoring.

The TDAQ-interface block implements two interfaces, a PCI bus and a network interface, allowing various DataFlow architecture options to be studied. The design of the final RobIn can be realised by removing (and not by adding) functionality, i.e. the PCI bus or network interface.

The Software Interface [8-16] to the prototype RobIn is generic enough to allow the RobIn to be studied in all DataFlow architectural options. The same basic services and messages are used in all cases, and functionality particularly required for e.g. the network is encapsulated in appropriate modules. The software interface comprises mainly the definition of a set of services provided by the RobIn and the messages to request these services and to transfer the responses.

Figure 8-4 Basic Functional Diagram.

The ROS LocalController, via the IOManager, performs the configuration and control of the RobIn via the TDAQ-interface block.

Andreas: Should we add more information about the production status?

Editor: It will be out of date at the time of print.

8.2.2.3 Implementation and performance

The baseline deployment of the ROS is shown in Figure 8-5.

Figure 8-5 Baseline deployment of the ROS.

Benedetto: for now I put the bus based ROS but this may change!!!!

It is deployed on two nodes: a ROS PC and a RobIn module. The former is a desktop PC running the Linux operating system and has at least one Ethernet connection for the purpose of communication with the Online system. In addition, it has at least four 64 bit / 33 MHz and 3.3 V PCI slots. These slots are used to host the RobIn modules. Each RobIn node has two SLINK input connections. The IOManager via the DC Message Passing interface receives data requests and release messages, and returns ROS event fragments to the High Level Trigger components. These communications occur via Ethernet.

Figure 8-6 shows an alternative way of deploying the same ROS components. In this case the RobIn devices are now implemented on dedicated boards, e.g. VMEbus 9U, and the connection with the ROS PCs is made through an Ethernet switch. The figure also shows that in this scenario the LVL2 trigger requests data fragments directly from the RobIn modules, without passing through the IOManager process.

Extensive measurement have been made on the performances of the ROS for the deployment scenarios previously described, bus-based and switch-based, here only the main results are presented, the complete set of results can be found in [8-20].

Figure 8-6 Alternative ROS deployment scenario.

Figure 8-7 shows the setup for the bus-based testbed. In this testbed an IOManager and a LocalController process were deployed on a standard 2 GHz Xeon PC with a single processor and a 66 MHz / 64Bytes PCI bus, running RedHat Linux 7.3.

As the final prototype RobIns were unavailable at the time of these measurements, I/O with a RobIn was emulated using a number of RACE boards [8-19] which have the same physical PCI bus interface as the final prototype RobIn, and thus provide a very accurate emulation of the final devices. The RACE boards were not connected to any external data source and were programmed to generate ROB Fragments on demand.

Figure 8-7 Setup of the testbed for studying the bus-based ROS system.

The testbed has been run both in a standalone configuration, where the IOManager was generating triggers internally and the produced ROS Fragments where sent nowhere, and in a config-

uration where the IOManager was receiving real data request messages from the network and sending back the ROS Fragments to the requester process.

Figures 8-8 and Figures 8-9 show the maximum LVL1 rate that an IOManager was able to sustain for different fractions of LVL2 and Event Building requests and for different number of RACE boards connected to it. As we had only 6 RACE boards available, we developed a software simulation of the RobIn to allow the test of the IOManager performances with larger numbers of connected RobIn modules

The "software emulation" probably needs to be elaborated.

..

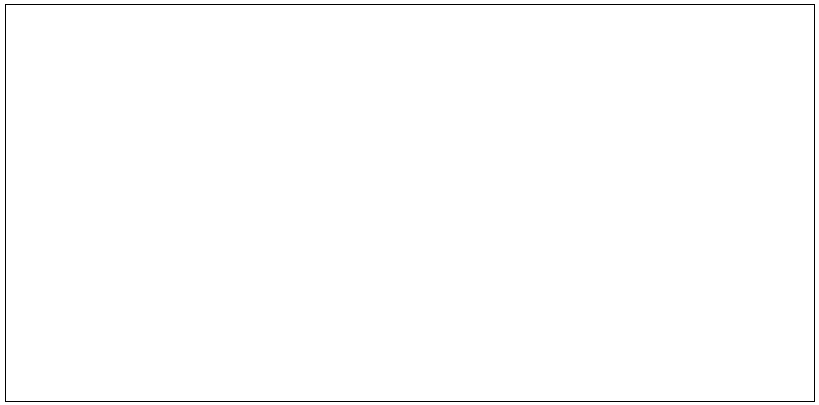


Figure 8-8 Maximum sustainable LVL1 rate for different fractions of LVL2 and Event Building requests, for a standalone bus-based ROS connected to a different number of RobIn modules.

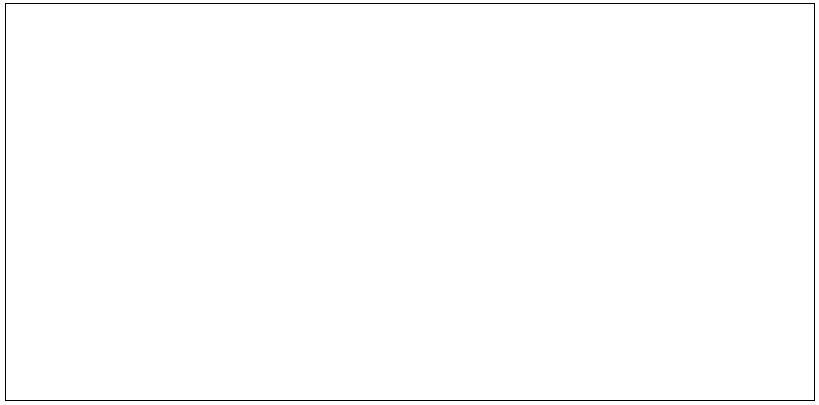


Figure 8-9 Maximum sustainable LVL1 rate for different fractions of LVL2 and Event Building requests, for a standalone bus-based ROS with real I/O to other DataFlow components.

Figures 8-10 shows how the simulation reproduces the measured results for up to 6 RobIn modules and the results that one obtains for a larger number of RobIns.

Figure 8-10 Maximum sustainable LVL1 rate for different fractions of LVL2 and Event Building requests. for a standalone bus-based ROS with simulated PCI RobIn input.

The system is thus shown to fulfil the requirements for the final ATLAS for up to XXXX PCI RobIn modules connected to a same IOManager.

Figures 8-11 shows the setup for the switched-based testbed. The IOManager and a LocalController process were deployed on a standard PC and the RobIns were emulated with a number of FPGA emulators [8-21] that were programmed to receive data requests over the network with the same message passing interface as the final RobIn prototype and to generate ROB Fragments on demand. Similarly to the tests performed on the bus-based ROS, this testbed has been operated both in a standalone configuration, where the IOManager was generating triggers internally and the produced ROS Fragments where sent nowhere, and in a configuration where the IOManager was receiving real data request messages from the network and sending back the ROS Fragments to the requester process.

Figure 8-12 and Figure 8-13 show the maximum LVL1 rate that the system was able to sustain for different fractions of LVL2 and Event Building requests and different number of connected RobIn sources. Also in this configuration the system is shown to fulfil the requirements for the final ATLAS, see SOME SECTION IN PART 1..

8.2.2.4 pROS

Main description here. Short addition in Chapter 9, "High-level trigger".

8.2.3 ROD crate data acquisition

The ROD is a sub-detector specific front-end element. It is located, in the event data flow, after the first level of on-line event selection, between the Front-end Electronics (FE) and the ROS. The ROD receives data from one or more Front-end Links (FELs) and sends data over the ROL

Figure 8-11 Setup of the testbed for studying the switched-based ROS system.

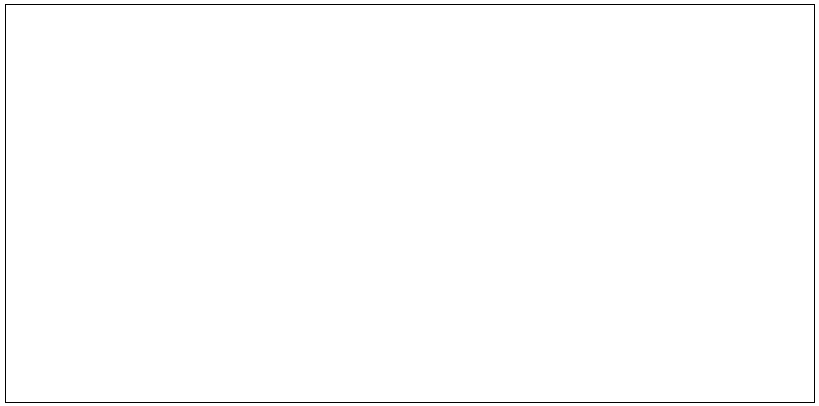


Figure 8-12 Maximum sustainable L1 rate for different fractions of L2 and EB requests, for a standalone network-based ROS connected to a different number of RobIn modules.

to the ROB. The ROD System covers all RODs and other functional elements at the same hierarchical level in the event data flow between the FE and the ROS. Those elements are grouped in crates. The crates contain ROD Crate Modules (RCMs) which can be: RODs, modules other than RODs, e.g. for control of the FE, for processing event data upstream of the RODs or for driving a TTC partition, as well as not fully functional ROD prototypes in laboratory setups or at test beam, and one or more ROD Crate Processors (RCPs). Each ROD Crate is connected to one or more ROD Crate Workstations (RCWs).

The sub-detectors need common DAQ functionality at the level of the ROD Crate for single or multiple ROD Crates in laboratory setups, at assembly of detectors, at test beam, and at the experiment during commissioning and production. ROD Crate DAQ [8-17] is part of the TDAQ system. It comprises all software to operate one or more ROD Crates and runs inside the ROD

Figure 8-13 Maximum sustainable L1 rate for different fractions of L2 and EB requests, for a network-based ROS with real DC I/O.

Crate as well as on the RCWs. It provides the functionality for configuration and control, ROD emulation, monitoring, calibration at the level of the ROD Crate, and event building across multiple ROD Crates.

8.2.3.1 High Level design

The ROD Crate configuration describes all necessary data required to fully configure all modules of the ROD Crate and the RCW(s). All ROD Crate configuration data are stored in one or more databases with the configuration database of the Online Software being the driving one. Several different ROD Crate configurations are stored in the database(s) concurrently. At initialisation of a run, one configuration is selected and loaded.

The ROD Crate control takes all necessary actions required to fully control all modules of the ROD Crate and the RCW(s). It is based on the run control of the Online system and implemented as a tree of run controllers, one per ROD Crate and others on the RCW(s) as necessary. The ROD Crate controller (RCC) drives all RCMs of the ROD Crate into well-known states and investigates their status. It may require interaction with the TTC system and/or DCS.

The primary event data flow of the ROD Crate transports event data from the FE over the FEL, the ROD and the ROL to the ROS. The secondary event data flow of the ROD Crate transports sampled event data from the RODs over VMEbus, optional ROD Crate data collection, optional ROD Crate event building, and ROL to the ROS, or optionally to data storage for recording. ROD Crate DAQ provides collection of sampled event data from multiple RODs in the same ROD Crate and building of sampled event data from multiple ROD Crates.

Some ROD prototypes are not fully functional RODs and some are non-ROD modules, in particular at test beam, have to be read out at the same hierarchical level in the event data flow as a ROD. ROD emulation provides the missing functionality. ROD emulation may be based on the primary or on the secondary event data flow. In both cases, an RCP is required.

Monitoring is another basic function of the ROD Crate DAQ. Different types of monitoring have to be distinguished depending on the different types of monitoring data they are collecting. Event data monitoring provides event data coming from the secondary data flow. Scaler and histogram monitoring provides scaler and histograms derived from event data. Operational monitoring reads operational values not directly derived from event data.

ROD Crate calibration provides sub-detector calibration at the level of the ROD Crate. It reads all event data from the secondary event data flow, processes them and calculates calibration data. The calibration data are written to data storage for recording or to the calibration database.

ROD Crate event building is achieved by event building sources, one for each ROD Crate which participates in the event building, and one event building destination. An event building source is an output of a ROD emulation, monitoring or calibration activity. It sends all event data of the secondary event data flow over Local Area Network (LAN) to the event building destination. The event building destination is the input of a dedicated ROD emulation, monitoring or calibration activity and usually runs on the RCW.

The basic functions of ROD Crate DAQ can be combined to provide high-level functionality for physics and calibration runs. They can also be used in different setups for physics data taking, ROD emulation, event building from multiple ROD crates, and small laboratory setups.

The framework of ROD Crate DAQ is organized into four layers: hardware, operating system, low-level services, and high-level tasks. A call for tender for the hardware of the RCP is under way. It is assumed that PCs will be used for the hardware of the RCWs. Linux is the first choice of operating system. LynxOS will be used for the RCPs in case real-time performance is required. The low-level services, like libraries and drivers, are organized into three different layers for hardware access, high-level task support and support for the Online Software.

A ROD Crate DAQ task is a high-level task for ROD Crate controller, ROD emulation, monitoring, calibration or event building. ROD Crate DAQ tasks are provided as skeletons made of generic functions which may be extended by the sub-detector groups. Some standard functions are provided, e.g. for event building, which probably do not require extension.

The generic functions of the ROD Crate dataflow task are: the “input function” which reads data from FEL, RCM or LAN, the “processing function” which processes and selects data, the “output function” which sends data over the ROL or LAN, or to data storage, the “control function” which communicates with the ROD Crate controller, and the “monitoring/histogramming function” which communicates with the monitoring/histogramming of the Online Software. The specific tasks for ROD emulation, monitoring, calibration or event building are distinguished by the implementation of their individual functions

8.2.3.2 Implementation

ROD Crate DAQ re-uses existing software where possible. The Online Software is used as is, with some adaptation to sub-detector specific needs, in particular, for configuration. The ROD Crate controller is adapted from the controller developed for the ROS. The ROS software is also used to provide skeletons for the different tasks of ROD emulation, monitoring, calibration and event building.

The initial software development involves several real users and ROD Crates containing ROD prototypes as well as fully functional RODs. The workplan [8-18] allows for a first distribution of ROD Crate DAQ to be available by June 2003.

8.3 Boundary and interface to the level 1 trigger

Because ATLAS depends on data collection guided by RoIs the level 2 system needs information from the level 1 trigger decision. This information includes both the triggers which passed and the details of where, in eta and phi, the trigger primitives that caused the accept came from. This requires information internal to the level 1 system to be passed on to the HLT. Collecting this information and passing it on to the HLT is the responsibility of the RoIB.

Figure 8-14 shows the RoIB and its connections to the level 1 system. Since the level 1 accept rate is fairly high as an input transaction rate for a single processor, the RoIB is designed to spread the level 1 events over a small farm of processors which are referred to as supervisor processors. The supervisor processors receive a single S-link record containing the summary information for each event from the RoIB. Since the RoIB sends complete records to several supervisor processors no single processor has to deal with a full level 1 rate. The supervisors pass the level 1 data on to the level 2 processor that will make a decision on the event. The supervisors are responsible for a rudimentary form of load levelling. They are aware of the disposition of events that they send to level 2 processors and need to make sure that events are dealt with in a timely way. They also need to be assured that no single processor is overloaded with pending events. The operation of the supervisors is described in the Section 9.2.3.

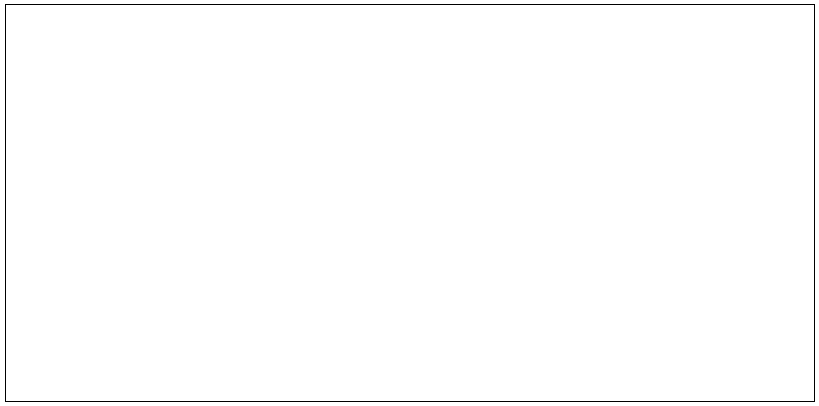


Figure 8-14 [reb001v000_lvl1lvl2.eps]

8.3.1 Description

This sub section should be a summary of what is detailed in [8-27].

A block diagram showing the level 1 system and its interconnection with the RoIB is shown in Figure 8-15. Each link from the level 1 system is an independent S-link input that sends a compact description of the event for that component. Each link is limited to sixty three 32 bit words or less per event. The various pieces of an event (referred to as fragments) will all arrive at the RoIB within one millisecond of each other. The RoIB will assemble the event data and send it to a supervisor processor which will then initiate the level 2 processing by passing a record to a level 2 processor which includes the pertinent level 1 RoI data.

Figure 8-15 [reb001v000_lvl1block.eps]

8.3.2 Region of interest builder

There is overlap with Chapter 9, "High-level trigger" on this component and only one chapter should describe it in detail with the other just mentioning the specifics for that chapter.

The RoIB is a VME based system which uses FPGAs to combine the level 1 fragments into a single record. It is composed of two parts. A pair of cards buffer the level 1 input and direct fragments to cards which assemble individual events. Twelve inputs are considered adequate. This will include both the level 1 fragments and an independent TTC input to assure consistency between level 2 and the read-out system. The input cards will communicate over a dedicated backplane connection to one or more `builder' cards that provide four outputs for assembled events. Figure 8-16 shows the system organization. The system can service four supervisors with a single `builder' card and can be expanded in units of four by adding `builder' cards.

8.3.2.1 Detailed design

This sub section should expand on the High level design described in Section 5.5. It should be a summary of what is detailed in [8-27].

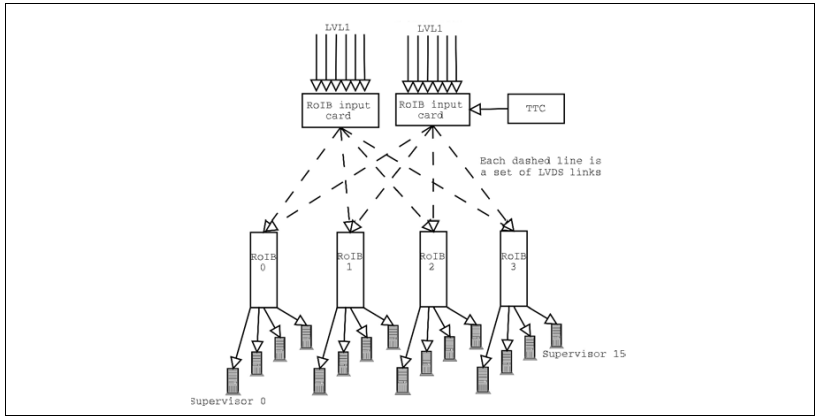


Figure 8-16 RoIB System organisation.

8.3.2.2 Performance

Based on results with (12 U) prototype, including results of integration studies with Level 1.

A prototype of the RoIB was fabricated and tested in 1999. This version was built using a pair of 12U VME cards, an input card capable of handling six S-link inputs and a pair of builder cards able to output to a pair of processors. This system utilized 76 Altera 10K40 FPGA's and 8 10K50's. Figure 8-17 shows the pair of boards that were built. The system and early performance measurements are documented in [8-22].

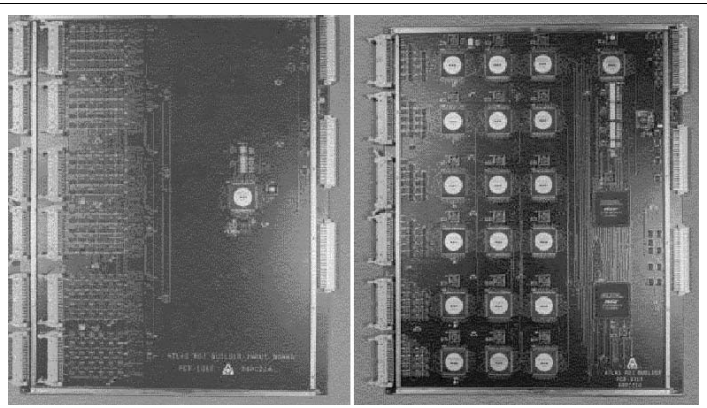


Figure 8-17 12U VME card prototype of the RoIB.

This system was adequate to demonstrate a number of critical points. It showed that the idea of combining records from several sources using an FPGA based device is feasible. It showed that the communications overhead for processors would not result in unmanageable numbers of processors just to handle the 100kHz event rate; four 300 MHz pentium two machines were adequate to handle the 100kHz rate. Subsequent tests with several prototype pieces of the level 1 system (the muon-CTP interface and the calorimeter CPROD) made a start on debugging the component interfaces and further demonstrated that external inputs could be handled at the expected rates [8-23].

The scale of a full system will need to be set in the future, but current indications from the early prototype make it clear that the full system will involve the number of processors needed to satisfy the HLT functions of the supervisors documented in the HLT description and testing section plus a few processors (less than four) to cover the additional communications from the RoIB.

8.4 Control and flow of event data to high level triggers

8.4.1 Message passing

8.4.1.1 Control and event data messages

Introduce the types of messages, the flow of messages, message rates and the bandwidths required. Concluding with the choice of link technology.

The flow of event data from the ROS, where data are buffered during the LVL2 and event building latencies, to the HLT is achieved by the exchange of control messages and subsequent event data messages between components of the DataFlow system. This is described in detail in [8-24] and here only its major features are summarized. Figure 8-18 shows a sequence diagram detailing the base interactions between DataFlow components.

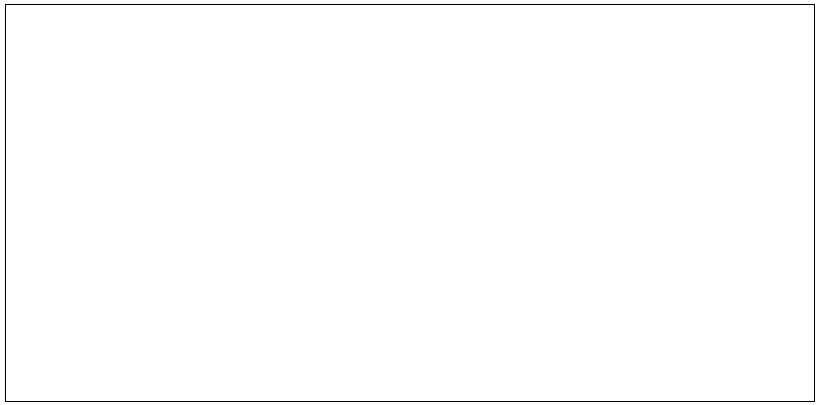


Figure 8-18 Sequence diagram showing the interactions between DataFlow components

8.4.1.1.1 L2SV

The LVL2 Supervisor receives LVL1 Results containing the RoI information from the RoI Builder. It assigns according to a load balancing algorithm a L2PU to analyse the event. It will then receive the LVL2 decision from the L2PU which it forwards to the DFM. In case no LVL2 decision will be received within a pre-defined timeout (e.g. the L2PU crashed, or a message was lost), the L2SV will treat the event as if accepted by the L2PU. There will be several L2SV deployed in the final system.

A LVL1 Result message does not exceed 512 bytes [8-27], leading to a maximum bandwidth requirement of O(50 MB/s) at 100 kHz LVL1 rate. This bandwidth can be handled easily with any gigabit capable link technology, furthermore, it has to be divided by the number of L2SVs deployed.

8.4.1.1.2 2.2 L2PU

A LVL2 Processing Units receives a LVL1 result containing RoI Information from the L2SV, which it uses to seed its processing algorithms. It requests specific RoI data from selected ROSs for analysis, more RoI data is requested repeatedly until the sequential processing results in its final decision to either reject or accept the event.

This includes also the decision whether an accepted event should be prescaled, or a rejected event should become forced-accept. The L2PU then sends the LVL2 decision (accept, reject, prescaled, forced-accept) back to the L2SV and in case of an (forced-) accepted event it also sends a detailed record to the pseudo-ROS. There will be many hundreds of L2PUs deployed in the final system.

As any individual L2PU processes events at a rate not higher than O(100 Hz), the bandwidth requirement for receiving the LVL1 results are O(1 kB/s). However, the data rate of the received RoI data can be substantial and is estimated to reach maximum values as high as O(10-20 MB/s).

8.4.1.1.3 ROS

The Read-out System holds event data fragments inside Read-out Buffers (ROBs) and makes them available to HLT, following data requests. The ROBs are cleared on receipt of clear messages.

The maximum bandwidth out of an individual ROB is estimated to O(10 MB/s) and data fragments are requested at up to xxx kHz [8-25]. Depending on how many ROBs are hold in a ROS and of the access mode to the ROBs, the ROS needs to provide O(Y MB/s) at a rate of yyy kHz.

8.4.1.1.4 2.5 pROS

The Pseudo-ROS receives the detailed result records of the L2PUs for accepted events and participates to the event building process, such that the LVL2 detailed result appears within the full event record. From the point of view of the SFI there is no difference between the pROS and a normal ROS component. However, the pROS has no detector specific input at all and requires no special hardware like e.g. a ROBIn.

Given a LVL2 accept rate of O(2 kHz) and an estimated size of the LVL2 detailed result record of O(1 kByte), the bandwidth in and out of the pROS will be a O(2 MB/s). One pseudo-ROS will therefore be sufficient for the final system.

8.4.1.1.5 2.6 DFM

The Dataflow manager receives (grouped) LVL2 decisions from the L2SVs and assigns an SFI, following a load balancing algorithm, for the event building of every accepted event. It multicasts the (grouped) clear messages to all ROSs (incl. pROS).

The bandwidth requirements for the exchange of control messages with the DFM are small, given the grouping factor of the LVL2 decisions is O(100) results at a LVL1 rate of 100 kHz to O(1 kHz) for the reception of the LVL2 decision messages. The communication with the SFI adds an additional two times O(2 kHz) message rate. The distribution of the clear messages

from the DFM to the ROSs will also be grouped. with an assumed grouping factor of O(300), only O(300 Hz) of message rate will need to be added - given a multicast mechanism for the distribution of clear messages. The total message rate to be handled by the DFM is therefore O(6 kHz); only small messages O(few 100 Bytes) need to be exchanged, leading to an aggregated bandwidth requirement of O(3 MB/s) to be handled by the DFM.

8.4.1.1.6 SFI

The SubFarm Input assembles event fragments from the ROSs (incl. pROS) and serves these to EventFilter SubFarms. It is the event building component in the DataFlow system and has to stand relatively high data and control rates, and conversely the bandwidth (for the data). The event size for complete events is O(2 MB) and the event building rate O(2 kHz), resulting into an aggregated bandwidth requirement of O(4 GB/s). This load needs to be distributed over many SFIs. Assuming an event building rate of O(50 MB/s) handled by one SFI, O(80) SFIs, each building events at O(40 Hz), will need to be deployed in the final system. The message rate to be handled by an SFI depends on the number of ROSs deployed, and is O(40 Hz) times two times the number of ROSs. It will not be above O(64 kHz) in case of 1600 ROS deployed.

For the communication with the EventFilter SubFarms, only a few messages need to be exchanged per event. However, a bulk transfer of the full event record is needed.

None of the afore mentioned messages require rates and bandwidth which cannot be handled by a wide range of link technologies. A commodity solution of widely available products on the world market can be deployed. Here the choice is dictated by price, long term availability, support, inter-operability and suitability for ATLAS DataFlow. Ethernet in its varieties of 100 Mb/s and 1000 Mb/s is the prime candidate and has been evaluated to proove its suitability for exchange of control and event data messages in the Atlas DataFlow.

8.4.1.2 Ethernet

This section should introduce the key features (i.e. VLANS, QoS, switches, flow control) supporting its selection and how they will be used. Should also summarise, based on [8-29], the basic message passing capabilities in terms of achieved rates, overheads and CPU loads.

8.4.1.3 Design of the message passing component

Presents the main features of the design (high Level enough?) based on [8-30].

8.4.1.4 Performance of the message passing

Presents, based on [8-29], the performance of the message passing component in terms of achieved rates, overheads and CPU loads.

The message passing layer of the data flow software is responsible for the transfer of all control and event data between different components. It provides a common technology-independent API across all applications.

The message passing layer itself imposes no structure on the data which is exchanged. Rather, this structure is defined by the message types in [see Section 8.4.1.1] which can be changed without affecting the message passing per se.

The service it provides is the transfer of up to 64 kByte of data with only a best-effort guarantee. No re-transmission or acknowledgement of data is done by this layer. This allows to implement the API over a wide range of technologies without imposing un-necessary overhead where not needed or duplicating existing functionality. The API supports the sending of both unicast and multicast messages. The latter has to be emulated by the implementation if it is not available (e.g. for TCP).

The message passing layer interface has been implemented over raw ethernet frames, UDP and TCP. The latter two implementations are technology-independent per se, although systematic measurements were only done for switched ethernet configurations (i.e. the routing aspects of IP are not necessary for the ATLAS architecture). TCP provides additional reliability compared to UDP and raw ethernet. However, applications and message flow have been designed in such a way that the system will still work when running over an unreliable technology. The raw ethernet implementation adds message re-assembly on the receiver side, similar to what IP provides. Otherwise the maximum message size would be restricted to a single ethernet frame which was seen as too restrictive for the range of data sizes intended.

Internally all implementations support scatter/gather transmission and reception of data. This allows to build a logical message out of a message header and additional user data that doesn't need to be copied inside the application.

The basic operations of allocating and de-allocating a buffer to send or receive are dominated by the need to make the interface thread-safe. On a 1 GHz dual processor SMP machine they take in the order of 0.6 µs each. Since they require main memory access on a real multiprocessor system this does not scale with the CPU frequency but with the memory speed. On a 2.2 GHz machine the numbers are only slightly smaller.

The basic measurements can be compared to the direct socket measurements of [see Section 8.4.1.2]. Note that the latter are done in a single-threaded environment without any dynamic memory allocation and always send and receive data from a fixed location and with a fixed size that is known in advance. Raw ethernet measurements are only done with a maximum of 1460 bytes, since no re-assembly of larger packets has been implemented. They therefore provide an upper bound for the possible performance which we don't expect to reach with the additional functionality in the message passing layer.

Among the reasons for lower performance are:

- Support for scatter/gather operations requires the kernel to copy an additional user data structure across kernel boundaries.
- The varying message length supported by the message passing API requires at least two read system calls for TCP, and two read system calls for every raw ethernet packet [check with David Botterrill, maybe just for first packet].
- For raw ethernet the re-assembly of frames into large messages.
- Thread-safety for the sender side: multiple threads can send at the same time as long as their destinations differ. They are serialized when they both send to the same destination.

Problems with scalability are only expected for the TCP implementation where a potentially large number of open sockets has be handled. The default TCP code uses the select() system call which is known not to scale. However, alternatives are available either in the form of POSIX conforming real-time signal in combination with non-blocking sockets or in a non-standard form by ongoing developments in the latest Linux kernel (epoll() interface[ref]). The UDP and raw ethernet implementations use only a single socket for receiving data.

Repeating the measurements for request/response and streaming shows an overall overhead of about 10 µs compared to the low-level tests. This translates into a time of about 22 µs to serve an incoming message or a rate of 45.5 kHz for receiving. These numbers are for a dual 2.2 GHz machine and require proper settings of socket buffer space and interrupt coalescence for the driver. E.g. with default settings for the interrupt coalescence this rate drops to 7 kHz. The difference between the various implementations are negligible: they are about 22.9 µs for UDP and 21.09 µs for raw ethernet.

Finally we can compare the numbers measured above with the observed rates of one of the applications in the data flow. The SFI application can do event building with raw ethernet frames at an overall rate that corresponds to a 40 kHz of data packages (ca. 1400 bytes) input and 40 kHz of requests (<64 bytes) output rate [8-32] which is in overall alignment with the low-level measurements.

8.4.2 Data collection

8.4.2.1 General overview

This section describes the common model to collecting data for level 2 processing and event building.

DataCollection is a subsystem of the Atlas TDAQ DataFlow system responsible for the movement of event data from the ROS to the LVL2 Processing and to the EventFilter and also to MassStorage. See.

It includes the movement of the LVL1 RoIs to the LVL2 PU (via the LVL2 SuperVisor) and the LVL2 result (decision and detailed result) to the EventFilter as well as the EventBuilding and feeding the complete events to the EventFilter.

However, DataCollection is not responsible for initializing and formatting (or preprocessing) of event fragments inside the ROS, neither is it responsible to do preprocessing nor to perform trigger decisions in the LVL2 Processing Unit or in the EF SubFarm.

Figure 8-19 shows a context diagram of the two main components of DataCollection (LVL2 DataCollection and EventBuilding) and its interfaces to other systems and subsystems of Atlas TDAQ.

The following lists the applications to be provided by DataCollection:

L2SV	LVL2 SuperVisor
L2PUA	LVL2 Processing Unit Application (i.e. L2PU low layer functionality)
DFM	DataFlow Manager
pROS	Pseudo ROS

Figure 8-19 DataCollection context diagram.

SFI SubFarm Input

SFO SubFarm Output

In order to deploy this variety of components, a common approach in design and implementation is envisaged. This approach lead to the definition of the common DataCollection framework, implementing a suite of common services. These were found to be:

- OS Abstraction Layer
- Configuration Database
- Error Reporting
- System Monitoring
- Run Control
- Message Passing

All applications in the DataCollection software share the need for a common set of typical operations. This includes error logging, configuration, system monitoring, run control and message passing. All these capabilities are provided by a set of packages which is usually referred to as the DataCollection Application Framework. This design leads to a large code reuse in practice. A typical application is built on top of a skeleton application and only has to provide the actual additional functionality.

Services are built from packages following a modular approach. Many of these packages consist of interfaces only, whose implementation is provided by other packages which can be changed at configuration or run-time. Examples are the error reporting (switching between simple stdout/stderr and MRS), the configuration database (switching between OKS files and remote database server), the system monitoring (providing an interface to the Information Service of the Online Software and a local independent version). The message passing interface allows the concurrent existence of multiple implementations at the same time. E.g. all of UDP, TCP and raw ethernet sockets can be used by a single application in a given setup.

This clear separation between interfaces and implementations exists down to the lowest levels like the thread interface and access to clocks and timers.

8.4.2.1.1 OS Abstraction Layer

The OS abstraction layer consist of packages hiding all OS specific interfaces. E.g. the *threads* package hides the details of the underlying POSIX thread interface.

8.4.2.1.2 Error Reporting

The ErrorReporting package allows to log error messages either to stdout/stderr or to MRS. Each package can define its own set of error messages and error codes. Error logging can be enabled/disabled on a package by package basis, with a separate debug and error level for each package. Furthermore debug logs and normal error logs are treated logically differently, so the debug message could go to stderr while all normal application logs go to MRS. The user only interfaces via a set of macros to the ErrorReporting system. This allows to compile out the debug macros for optimized builds.

8.4.2.1.3 Configuration Database

All applications make use of the Online Software configuration database through the API provided by these packages. The design uses the Bridge pattern described in Gamma et al. This allows to change the underlying implementation without the client code noticing it.

The user's view of the database is hidden by configuration objects, which read the database and provide a more convenient way to access the information. Database schema file evolutions are coped with an automated re-creation of the C++ code for these configuration objects out of the schema file only

8.4.2.1.4 System Monitoring

This package allows every component to make arbitrary information available to some outside client. In practice this is used to publish statistics like counters and histograms. Users inherit from the *Resource* class and implement a virtual function. The packages makes this information available in various different ways, including the Information Service of the Online Software.

Again the interface is strictly separated from the different implementations, so users are unaware of it and the implementation can change without them noticing it.

8.4.2.1.5 Run Control

The run control interface is responsible for translating the requests from the Online Software about state changes into commands for the application. It also provides a skeleton around which one can build an application.

These classes realize most of the use cases for run control. They talk to a special DataCollection Run Controller on the one side and to user code on the other side.

8.4.2.1.6 Message Passing

The Message Passing Layer defines a couple of classes to allow the sending and receiving of messages. The *Node*, *Group* and *Address* classes are used at configuration time to setup all the necessary internal connections.

The *Port* class is the central interface for sending data. All user data has be in part of a Buffer object to send or receive it. The *Buffer* interface allows to add user defined areas which are not under the control of the Message Passing layer to avoid copying.

The *Provider* class is an internal interface from which different implementations have to inherit. Multiple *Provider* objects can be active at any given time. A *Provider* is basically the code to send and receive data over a given protocol / technology, e.g. TCP, UDP or raw ethernet.

Using the DataCollection Application Framework, the DataCollection components were implemented efficiently with maximum reuse of code and coherency in system aspects.

The interaction of the DataCollection components is detailed in the following to sections for LVL2 DataCollection and event building.

8.4.2.2 RoI data collection

8.4.2.2.1 Design

This section should describe the interaction between applications which results in the collection of data at the level 2 processing unit.

8.4.2.2.2 Performance

8.4.2.3 Event Building

8.4.2.3.1 Design

This section should describe the interaction between applications which results in the collection of event fragments to form a complete event at the SFI. Should also include the aspects related to traffic shaping.

8.4.2.3.2 Performance

8.5 Reliability and fault tolerance: this section has been moved to Chapter 6, "Fault Tolerance and Error Handling"

8.5 Configuration, control and operational monitoring

The sub-sections in this section have been moved to other chapters.

Local Controller: moved to Chapter 13, "Experiment control".

Configuration data: moved to Section 10.2, "Databases".

Operational monitoring: moved to Chapter 7, "Monitoring"

8.5 Scalability

8.5.1 Detector read-out channels

This section describes quantitatively how the physical size, performance and control and configuration of the system scales with the “amount” of detector to be read-out.

8.5.1.1 Control and flow of event data

How the number of applications, messages and data volume changes.

8.5.1.2 Configuration and control

Amount of configuration data a function of the amount of detector.

8.5.2 Level 1 rate

How the system performance and physical size scales with respect to the level 1 rate.

8.6 References

8-1 Trigger & DAQ Interfaces with Front-End Systems: Requirement Document http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/DIG/archive/document/FEdoc_2.5.pdf

8-2 ATLAS High-level Triggers, DAQ and DCS Technical Proposal, CERN/LHCC/2000-17, 31 March 2000.http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/SG/TP/tp_doc.html

8-3 The S-LINK Interface Specification. http://edmsoraweb.cern.ch:8001/cedar/doc.info?document_id=110828

8-4 The raw event format, http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/ROS/DAQ-1_Note_050_update_1a.pdf

8-5 Recommendations of the Detector Interface Group - ROD Working Group https://edms.cern.ch/document/332389/1

8-6 The CMC standard. Common Mezzanine Cards as defined in IEEE P1386/Draft 2.0 04-APR-1995, Standard for a Common Mezzanine Card Family: CMC (the CMC Standard).

8-7 Design specification for HOLA, https://edms.cern.ch/document/330901/1

8-8 Procedures for Standalone ROD-ROL Testing, G. Lehmann *et. al.*, 27 July 2001, ATC-TD-TP-0001 http://edmsoraweb.cern.ch:8001/cedardoc.info?document_id=320873&version=1

8-9 ROS URD

8-10 Read out system high level design

8-11 ROS Local Controller

8-12 ROBIN Summary document

8-13 Readout sub-system test report (using DAQ -1.

8-14 ROBIN HLDD

8-15 ROBIN DLDD

8-16 ROBIN SWID

8-17 ROD crate DAQ design

8-18 ROD crate DAQ woekplan

8-19 Reference to the RACE board

8-20 ROS Test Report

8-21 Reference to FPGA emulators

8-22 ATL-DAQ-99-016

8-23 cite ATL-DA-ER-0016 & the corresponding muon-CTP i/f document

8-24 Data collection note # 012 ... to be moved into EDMS

8-25 Paper model results

8-26 Data Collection URD

8-27 Level 1 - Level 2 interface document

8-28 RoI Builder URD

8-29 Results of basic comms tests

8-30 Design of the message passing component

8-31 Documents supporting technology choices

8-32 DataCollection test report

8-33 DataCollection Local Controller

9 High-level trigger

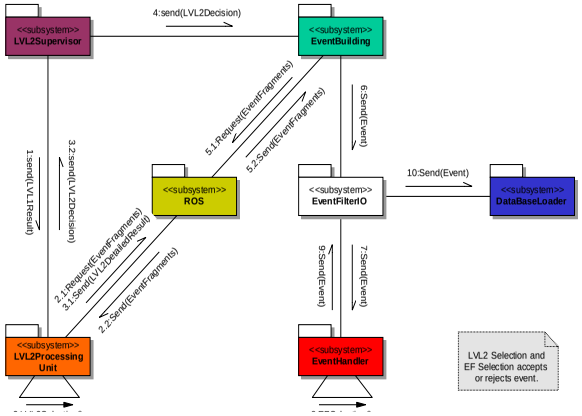
The chapters of Part 2 should contain the major components as identified by the architecture.

Details should be provided on design, implementation and supporting measurements. For each component describe: the purpose/function/scope of the component, the performance requirements of the component, the architecture of the component, a proposed implementation, and performance and validation measurements.

The commonalities and differences between LVL2 and EF should clearly be shown.

Detailed design, and performance (where appropriate - performance of some components only relevant/directly measurable as a set-of components) of each component should be described in the sections below.

9.1 HLT Overview



9.2 Level 2

9.2.1 Overview

Includes use RoI mechanism (*i.e. selective Read-out*), requirements and interplay between components.

9.2.2 RoI Builder

There is overlap with Chapter 8, "Data-flow" on this component and only one chapter should describe it in detail with the other just mentioning the specifics for that chapter.

Main description of design, implementation, interfaces etc should be in Chapter 8 (DataFlow). Here limited to a brief description of the functions (i.e. gathering together of the LVL1 RoI information and then routing to a Supervisor processor)

9.2.3 LVL2 Supervisor

Main description of functions, design and implementation here in this chapter, should refer back to the DataCollection Framework described in Chapter 8. The description here to include the load balancing aspects. (The DataFlow chapter should be limited to a description of function of the Supervisor in DataFlow and the rate at which it handles messages - i.e. the performance measurements (30 kHz rate).)

9.2.4 LVL2 Processors

Need a description of how the software inside a LVL2 processor is structured. i.e. L2PU hosting the PSC, which hosts the event selection code. Also how the algorithms access data data from the ROB's with the interface layers provided between the algorithm and the L2PU. Diagrams to be included here are ones showing the multi-threaded nature of the Worker threads and the sequence diagram of what happens during configuration and in the event loop.

Should include a statement about the possible use of FPGA's here with a reference to the FPGA implementation back-up document, but note that this is not included in the baseline option.

9.2.4.1 L2PU

Main description of functions, design and implementation of L2PU given here in HLT, should refer back to the DataCollection Framework described in Chapter 8. (The DataFlow chapter should cover the (messaging) performance of the L2PU - i.e. the measurements of bandwidth and message rate in etc., including RoI data gathering capability and scaling as extra processors are added.) Only performance numbers to be recapped here are the rate at which a single processor can handle events.

9.2.4.2 PSC (PESA Steering Controller)

Decscription of the design and implementation of the PSC, how it sits inside the L2PU (including receipt of LVL1_Result and return of LVL2_Result) how it provides various an ATHENA like environment for the Event Selection code and the mechanisms for the algorithms to be configured. Should include some performance results when sitting inside L2PU without running algorithms

9.2.4.3 Data access i/f's

Description of how the PESA DataManager can access RoI data from the L2PU. Not clear that there are meaningful separate performance measurements for this.

Need a diagram here to showing how the London scheme is implemented in LVL2.

9.2.5 pROS

Main description is in Chapter 8, "Data-flow". Here just a brief note to describe the function from an HLT perspective. i.e. The mechanism to receive the LVL2 result for inclusion in the built event - thus passing the result from LVL2 to the EF.

9.2.6 LVL2 Operation

Here a brief description of how the LVL2 processors are configured, controlled and monitored, how they are organised into sub-farms and how the farm-fabric is managed.

9.3 Event Filter

Description of EF DataFlow - event distribution, use of PT's (providing ATHENA environment), use for Event Selection, for Calibration and data monitoring, generation of EF_Result, appending EF_Result to the built event, passing accepted events back to main DataFlow.

3rd and last step of the event selection procedure

Works on the full available information (fully built event, downstream the Event Builder)

Make the largest possible use of offline environment and algorithms, possibly including the offline implementations

Implementation very likely to happen on farms of standard computers (e.g. PC based)

9.3.1 High Level design

9.3.1.1 Functionality

logically decomposed in two parts :

- Event Handler
  - Data flow
  - processing tasks
- Supervision
  - control
  - monitoring

additional functionality may be envisaged in the fields of

- detector (including trigger) monitoring
- online calibration and alignments

9.3.1.2 Operational Analysis

Details in HLT operational analysis (EDMS note)

Use cases and requirements to other sub-systems

9.3.2 Event Handler

9.3.2.1 Requirements

details in ATLAS-DAQ-2002-003

Functionality

- provide the functionality to the DAQ Data Collection to inject and receive events
- distribute events to specific processing tasks according to an event type contained in the event header
- provide the software infrastructure to perform the required treatment : filtering, monitoring, calibration check, etc... inside the Processing Tasks.
- collect selected events for offline production according to the streams determined by the classification.

Robustness

Reliability

9.3.2.2 Detailed design

EFD design

details in EF Dataflow design (EDMS note)

- 1 EFD process per processing node
- one or several external processing tasks running on the processing node as independent processes (robustness)
- event passing via shared memory on memory-mapped files (efficiency)
- synchronisation ensured by EFD worker thread (next task)

PT design

- offline framework
- bytestream
- EF result

9.3.3 EF Supervision

9.3.3.1 Requirements

Details in ATLAS-DAQ-2002-004

Functionality

- process control
- process monitoring
- possibly hardware control and monitoring

Constraints

- connection with ATLAS general control
    - finite state machine mapping
    - partitioning
- interaction with end user
    - via Online Software services

Robustness

Reliability

9.3.3.2 Detailed design

- Tree structure
    - sub-farms organised around the data flow architecture (SFI)
    - control at the sub-farm level
- use Online Software tools as much as possible

9.3.4 Extra functionality possibly provided by the Event Filter

- Directly in EFD context (by-products of calculations performed for selection, in the filtering tasks or in independent monitoring tasks)
- or in dedicated parts of the Farm, specially fed by Data Collection, and working under the control of the EF supervision
- Functionality in the fields of
    - trigger monitoring
    - detector monitoring
    - online calibration and alignment (checks and / or computation)

9.4 Event selection software

ESS Architecture - requirements, design and implementation - including the main internal components

This is the only place in the TDR where the overall design and implementation of the ESS is described. This section should be a shortened form of what is in the PESA Framework requirements and design documents. However, given the complexity and importance of the material it will be substantial in length.

Configuration, control, supervision and operational monitoring: moved to several other chapters.

Describe here the HLT specific items and issues.

Thus:

1) LVL2 use same LocalController as DataCollection. Need to add something about the operational supervision of the LVL2 processors.

2) EF has the EF Supervision sub-system

3) Concept of sub-farms - how they are defined, and application configured

As far as possible common issues should be described in the OnLineSW chapter.

Issue - Should we include management of the sub-farm fabrics here? (Presumably yes)

Issue - Where do we cover algorithm configuration

(I assume that data and algorithm related monitoring is included in the monitoring chapter).

9.5 References

9-1 LVL2 URD

9-2 EF DataFlow URD

9-3 EF Supervision URD

9-4 ESS Requirements Doc

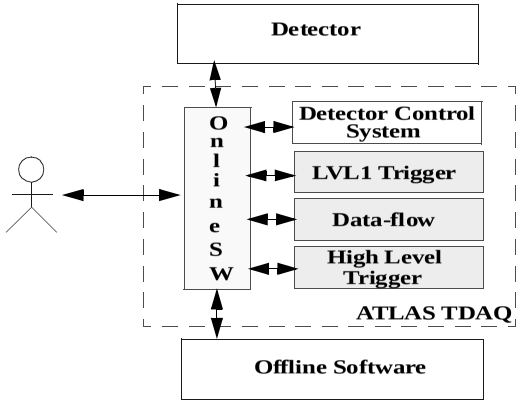
9-5 ESS Design Doc

10 Online Software

10.0.1 Introduction

The Online Software encompasses the software to configure, control and monitor the TDAQ but excludes the management, processing and transportation of physics data. It is a customizable framework which provides essentially the "glue" that holds the various sub-systems together. It does not contain any elements that are detector specific as it is used by all the various configurations of the TDAQ and detector instrumentation. It co-operates with the other sub-systems and interfaces to the Detector Readout Crates, the Detector Control System (DCS), the Level 1 Trigger, the Data-flow, the High Level Trigger processor farms, the Offline Software and to the human user as illustrated in Table 10-1.

Table 10-1 The Online Software in relation to the TDAQ system



An important task of the Online Software is to provide services to marshal the TDAQ through its start-up and shutdown procedures so that they are performed in an orderly manner. It is responsible for the synchronisation of the states of a run in the entire TDAQ system and for process supervision. These procedures are aimed to take a minimum amount of time to execute to reduce the overhead since this affects the total amount of data that can be taken during a data-taking period. Verification and diagnostic facilities help for early and efficient problem finding. Configuration database services are provided for holding the large number of parameters which describe the system topology, hardware and software components and running modes, based on the partitioning model. During data taking, access to monitoring information like statistics data, sampled data fragments to be analysed by monitoring tasks, histograms produced in the TDAQ system, and also to the errors and diagnostic messages sent by different applications is provided. User interfaces display the status and performance of the TDAQ system and allow the user to configure and control his operation. These interfaces provide comprehensive views of the various sub-systems at different levels of abstraction.

The Online software has various types of users. The *TDAQ Operator* runs the *TDAQ system* in the control room during a data-taking period, the *TDAQ Expert* has system-internal knowledge and can perform major changes to the configuration, the *Sub-system or Detector Expert* is responsible for the operation of a partcular sub-system or detector and *TDAQ and detector software applications* use services provided by the Online software via application interfaces.

remark: the following definitions will be available from the TDAQ glossary to which only a reference will be made.

The following types of Online Software users have been identified:

1. *TDAQ Operator - this user is responsible for using the TDAQ system to take data during a particular data taking session, for example during a shift. He has to be able to start, monitor and stop data taking. He is not expected to perform any programming tasks related to the Online Software.*
2. *TDAQ Expert - this user is responsible for running and maintaining the TDAQ system itself. He is responsible for the initialisation and shutdown of the TDAQ system or parts of it. He shall have a knowledge of the TDAQ structure and its components.*
3. *TDAQ Sub-System or Detector Expert - this user is responsible for the operation of a particular sub-system of the TDAQ or particular sub-detector of the ATLAS detector. He should be capable of describing the specific TDAQ sub-system or detector configuration and diagnosing the specific sub-system or detector problems which may appear during operation.*
4. *TDAQ Sub-System or Detector - this user represents a software produced by the TDAQ Sub-System or Detector developers. This software will use the services provided by the Online Software for the sub-systems and detectors configuration and control.*

The user requirements to the Online Software are collected and described in the corresponding document [10-1].

10.0.2 The Architectural Model

The Online Software architecture is based on a component model and consists of three high-level components, called packages. Details on the architecture and a comprehensive set of use cases are described in [10-2]. Each of the packages is associated with a functionality group of the Online software. For each package a set of services which it has to provide is defined. The services are clearly separated one from another and have well defined boundaries. For each service a low-level component, called sub-package, is identified. Each sub-package contains as many classes implementing a specific functionality as boundary classes are necessary providing interfaces for a variety of the Online Software users.

Each package is responsible for a clearly defined functional aspect of the whole system.

1. Control and Supervision - contains sub-packages for the control of the TDAQ system and detectors. Control sub-packages exist to support TDAQ system initialisation and shutdown, to provide control command distribution, to start and stop processes within the TDAQ system and to execute tests requested by a user.
2. Databases - contains sub-packages for configuration of the TDAQ system and detectors. Configuration sub-packages exist to support system configuration description and access to it, record operational information during a run and access to this information. There are also boundary classes to provide read / write access to the conditions storage.

3. Information Sharing - contains classes to support information sharing of the TDAQ system and detectors. Information Sharing classes exist to report error messages, to publish states and statistics, to distribute histograms built by the sub-systems of the TDAQ system and detectors and to distribute events sampled from different parts of the experiment's data flow chain.

The following sections describe these packages in more details.

10.1 Control and Supervision

The main task of the control and supervision package is to provide the necessary tools to perform the system operation as they are described in Chapter 3. It provides the functionality of the TDAQ Control as shown in the controls view of the Chapter 5.

In addition the package has the responsibility for functionalities necessary in the computing environment for user interaction, process management and access control.

10.1.1 Functionality of the Control and Supervision

Control and Supervision encompasses software components responsible for the control and supervision of other TDAQ systems and the detectors. The functionalities have been derived from the user requirements and are described in turn.

- **User Interaction:** Interaction with the human user like the operator or system expert of the TDAQ system
- **Initialization and shutdown:** Initialisation of TDAQ hardware and software components is foreseen. The operational status of system components must be verified and the initialisation of these components in the required sequence is ensured. Similar considerations are required for the shutdown of the system.
- **Run control:** System commands have to be distributed to a range of several hundred to thousand of clients programs. The control sub-package is responsible for the command distribution and the required synchronisation between the TDAQ sub-systems and detectors.
- **Error handling:** Malfunctions can interrupt system operations or deteriorate the quality of physics data. It is the task of the control sub-package to identify such malfunctions. If required the system will then perform autonomously recover operations and assist the operator with diagnostic information.
- **Verification of System status:** The control and supervision package is responsible to verify the functionality of TDAQ configuration or any subset of it.
- **Process Management:** Process management functionality in a distributed environment is provided.
- **Resource Management:** Management of shared hardware and software resources in the system is provided.
- **Access Management:** The control and supervision package provides a general Online software safety service, responsible for TDAQ user authentication and the implementation of an access policy for preventing non-authorised users to corrupt TDAQ functionality.

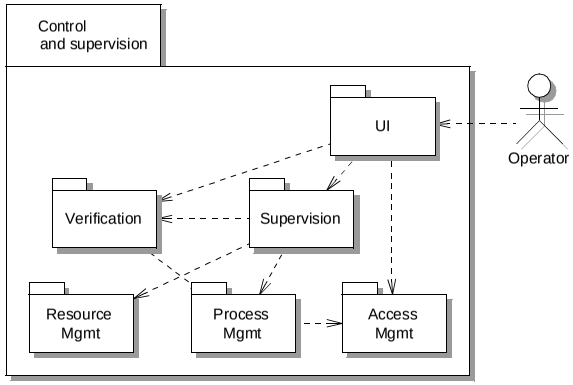
10.1.2 Performance and Scalability Requirements on the control and supervision

The control of the TDAQ system will be performed in a hierarchically distributed manner in so-called local controllers. Their number is estimated to be in the range of 500 to 2000. The responsibilities of these controllers depends on its task in the TDAQ system and in the detectors. Some of these local controllers will control the operation of hardware devices, others interact with large processing farms. A detailed explanation can be found in the chapter on Experiment Control.

10.1.3 Architecture of Control and Supervision

The Control and supervision package is divided into a number of sub-packages as shown in Figure 10-1.

Figure 10-1 The Organization of the Control and Supervision package



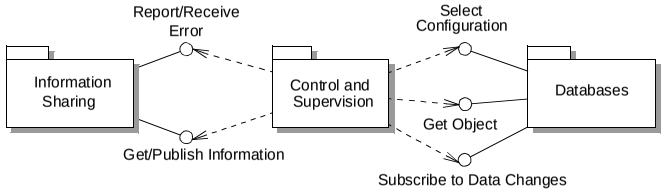
The functionality described in Section 10.1.1, "Functionality of the Control and Supervision" has been distributed to several distinct sub-packages:

- The User Interface (UI) for interaction with the operator
- The Supervision for the control of the data-taking session including initialization and shutdown, and for error handling
- The Verification framework for analysis of the system status
- The Process Management for the handling of processes in the distributed computing environment
- The Resource Management for coordination of the access to shared resources
- The Access Management for providing authentication and authorisation when necessary

10.1.3.1 Interaction of the Control and Supervision System with other Online SW packages

The Control and Supervision package interacts with the other Online Software packages as shown in Figure 10-2. The Databases package is used to describe the system to be controlled. This includes in particular the configuration of TDAQ Partitions, TDAQ Segments and TDAQ Resources. The Information Sharing package provides the infrastructure to obtain and publish information on the status of the controlled system, to report and receive error messages and to publish results for interaction with the operator.

Figure 10-2 Interaction of the Control package with other Online Software packages



10.1.3.2 User Interface

The **User Interface** (UI) provides an integrated view of the TDAQ system to the operator and should be the main interaction point. It is foreseen to provide a flexible and extensible UI that can accommodate panels implemented by the detectors or TDAQ systems.

Web based technologies will be used in particular for the access to the system for off site users and text based utilities will be provided in addition. Other technologies are under consideration.

10.1.3.3 Supervision and Verification

The **Supervision** sub-package realizes the essential functionality of the Control package. The sub-package itself has a coordinating role and interfaces via a Local Controller to other TDAQ system or detector specific controllers. Via the User Interface sub-package it provides to the Operator all TDAQ control facilities. These are expressed as interfaces in Figure 10-3 and discussed below in more details.

Several main functionality domains have been identified: the *Setup* is concerned with the initialization and the verification of the system, the *Run Supervisor* is responsible for the data-taking activity and the *Error Handling* with the error response and the recovery from system failures.

The *Setup* is responsible for

- initialization of TDAQ hardware and software components, bringing TDAQ partition to the state when it can accept Run commands.
- re initialization of a part of the TDAQ partition
- shutting the TDAQ partition down gracefully

The *Run Supervisor* is responsible for

- controlling the Run by accepting commands from the user and sending commands to TDAQ sub-systems
- analysing the status of controlled sub-systems and presenting the status of the whole TDAQ to the Operator

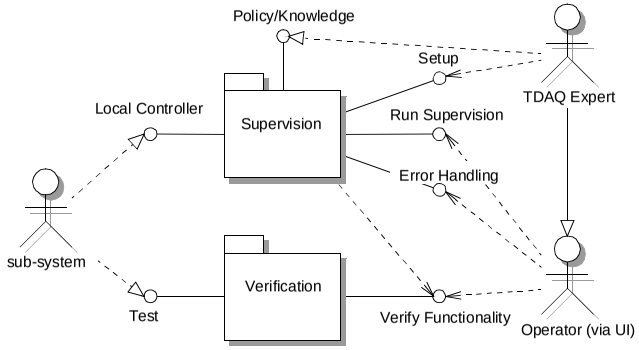
The *Error Handling* is concerned with

- analysing run-time error messages coming from TDAQ sub-systems
- diagnosing problems, proposing recovery actions to the operator or performing automatic recovery if requested

Most of the above defined functionality can reside on the local controller and be extended by defining specific policies which the TDAQ sub-systems and detector expert developers implement. Currently a rule based Expert System is the preferred design choice.

The **Verification** sub-package is responsible for the verification of the functionality of the TDAQ system or any subset of it. It uses developer's knowledge to organize tests in sequences, analyse test results, diagnose problems and provide a conclusion about the functional state of TDAQ components. An expert system is a likely design choice for knowledge representation and reasoning.

Figure 10-3 Interfaces of the Supervision and Verification sub-packages



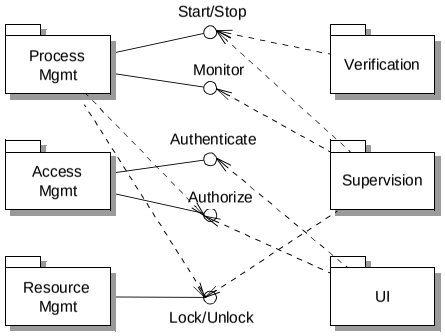
A TDAQ sub-system developer implements and describes tests which are used to verify any SW or HW component in a configuration. This includes also complex test scenario, where the component functionality is verified by the simultaneous execution of processes on several hosts. The sub-system uses the Process Management sub-package for the execution of tests.

The Verification sub-package is used by the Supervision to verify the state of the TDAQ components during initialization or recovery operations. It can also be used directly by the Operator via UI, as it is shown on Figure 10-3.

10.1.3.4 Process, Access and Resource Management systems

The Verification and Supervision sub-packages connect via interfaces to other Control sub-packages, as shown in Figure 10-4.

Figure 10-4 Interfaces of the Process management, resource Management and the Access Management



The **Process Management** provides basic process management functionality in a distributed environment. This functionality includes starting, stopping and monitoring processes on different TDAQ hosts.

The **Resource Management** is concerned with the allocation of software and hardware resources between running partitions. It prevents the operator from performing operations on resources which are allocated to other users.

The **Access Management** is a general Online “software safety“ service, responsible for TDAQ user authentication and implementation of an access policy, in order to disable non-authorised persons to corrupt eventually TDAQ functionality.

10.1.4 Prototype Evaluation

Main parts of the required functionality for the final system have been evaluated in the prototype implementation of the Online System [10-4].

- A Run Control implementation is based on a State Machine model and uses the State machine compiler CHSM as underlying technology.
- A Supervisor is mainly concerned with process management. It has been build using Open Source expert system CLIPS.
- A verification system, DVS, performs tests and provides diagnosis. It is again CLIPS based.
- A Java based graphical User Interface, IGUI
- Process Management and Resource Management are implemented based on other components provided by the other current implementation of the Online Software packages.

Large Scale scalaility tests have been performed [10-3] to verify the functionality of the prototype [10-4]. It could be shown that a test configuration in the range of the foreseen system can successfully be controlled.

Some plot form the scalability test

More description of the scalability test

For the final system a new evaluation of expert system technologies has been started. CLIPS has been evaluated in the prototype implementation and related products like Jess, a similar implementation based on Java and the commercial alternative Eclise by Haley. Inc. are studied.

In the evaluation we have also investigated other alternatives, like scripting based solutions. We have studied the use of SMI++, a scripting language for the description of interaction state machines, which is used by several HEP experiments. Furtheron we have considered a possible implementation based on other scripting languages like Python. While each of these approaches has its particular merits, our evaluation showed that the CLIPS based solution is better suited for our environment and is the favoured implementation choice.

10.2 Databases

The TDAQ systems and detectors require several persistent services to access description of the configuration used for the data taking as well as to store the conditions under which the data were taken. The online software provides common solutions for such services taking into account requirements coming from the TDAQ systems and detectors.

10.2.1 Functionality of the Databases

There are three main persistent services proposed by the online software:

- the **configuration databases** to provide the description of the system configurations,
- the **online bookkeeper** to log operational information and annotation,
- the **conditions databases interface** to store and read conditions under which data were taken.

10.2.1.1 Configuration Databases

The configuration databases are used to provide overall description of the TDAQ systems and detectors hardware and software. It includes description of partitions defined for the system and parameterized for different types of runs describing the hardware and software used by a given partition and their parameters.

The configuration databases are organized in accordance with the actual hierarchy of the TDAQ system and detectors. The configuration databases give a possibility for each TDAQ system and detector to define their own format of the data (i.e. the database schema), to define the data themselves and to share the schema and the data with others. The configuration databases provide graphical user interfaces to browse the data by any human user and to modify the data by authorized human experts. The configuration databases give possibility to generate data access

libraries which hide the technology used for the databases implementation and can be used by any TDAQ or detector application to read the configuration description or to be notified in case of it's changes. An application started by the authorized expert can use the data access libraries to generate or to modify the data.

The configuration databases provide efficient mechanism for fast access to the data for huge number of clients during data taking. They do not store the history of the data changes but provide archiving options. Configuration data which are important for offline analysis must be stored in the conditions databases.

10.2.1.2 Online Bookkeeper

The online bookkeeper is the system responsible for the online storage of relevant operational information and configuration description provided by the TDAQ systems and detectors. The OBK organizes the stored data on a per-run basis and provides querying facilities.

The online bookkeeper provides graphical user interfaces to allow human users to browse contents of the recorded information or append such information. The append access is limited for authorized users only. Similarly, the online bookkeeper provides application programming interfaces for user applications to browse the information or to append annotations.

10.2.1.3 Conditions Databases Interfaces

The TDAQ systems and detectors use the conditions databases to store conditions which are important for the offline reconstruction and analysis. The conditions data are varying with time and physical parameters such as temperature, pressure and voltage. These conditions are stored with a validity range (typically time or run number) and retrieved using time or run number as a key.

The conditions databases are implemented and supported by the offline software group. The online software group provides application programming interfaces for all TDAQ systems and detector applications and mechanisms to insure required performance during data taking runs.

remark: details of the functionality of the Conditions Database interface are still under discussion and the therefore the content of the explication given above may change.

10.2.2 Performance and Scalability Requirements on the Databases

remark: The performance and scalability requirements to the databases provided by the online software are not finalized by the users at the moment of the document writing.

10.2.2.1 Configuration Databases

The performance and scalability requirements to the configuration databases are strongly dependent on the strategies chosen by the TDAQ systems and detectors to get the configuration to their applications. For detectors, ROS, LVL1 and DC the number of applications reading the configuration databases is defined by the number of run controllers and supervisors, and it is less than 700 in total. For LVL2 it depends on the size of the system and varies from 100 to 1000.

For EF the situation is not defined yet and the number of database clients in the worse scenario can $O(10^4)$, if each processing task will read configuration description itself.

remark: What should we say about DCS?

The configuration information can be split on several non overlapping domains. The complete configuration description is required only to few applications in the system, while most others require to read only a small fraction of it. Each TDAQ system and detector in a worse scenario requires not more than $O(10^2)$ MBytes of configuration data and maximum number of clients per system and detector is limited by $O(10^2)$ except EF.

The configuration data are read once during starting of the data taking and an acceptable time to get full configuration for the whole system is around one minute. During the data taking of physics data the configuration may slightly be changed and an acceptable time to get the configuration changes by registered applications is several seconds. The configuration can be changed considerably during special calibration runs. The maximum rate required in this case is 10 Mbytes produced in one hour.

10.2.2.2 Online Bookkeeper

remark: no final requirements yet

10.2.2.3 Conditions Databases

remark: requirements to be written

10.2.3 Architecture of Databases

remark: the content of this paragraph may be moved to Section 10.2.1, "Functionality of the Databases".

10.2.3.1 Configuration databases

The ConfDB (Configuration Databases) provide user and application programming interfaces.

Via a user interface the software developer defines the data object model (i.e. the database schema) describing required configurations. The expert uses the interface to manage databases, to put the system description and to define configurations, which can be browsed by a user.

A TDAQ or detector application accesses databases via data access libraries (DAL). A DAL is generated by the software developer for a part of the defined object model relevant to his subsystem. The DAL is used by any process required to get the configuration description or to receive a notification in case of it's changes. Also, the DAL is used by an expert process needed to produce the configuration data.

The ConfDB system contains the following classes:

- **ConfDB Repository** - defines low-level interfaces to manipulate the configuration databases including databases management, schema and data definitions and notification subscription on data changes

- **ConfDB UI** (User Interface) - defines user interface for object model definition, configurations definition, database browsing and population by human actors
- **ConfDB DAL Generator** - defines interface to produce DAL
- **ConfDB DAL** - defines interfaces for configurations selection, reading and writing configuration objects and subscription for notifications on data changes by the user and expert processes (to receive notification a process shall realize **ConfDB Data Monitor** interface)

The **ConfDB Repository** class defines interfaces above a DBMS used for implementation and hides any specific details, so any other ConfDB classes shall never use DBMS-specific methods directly.

The Figure 10-1 shows interfaces provided by the configuration databases and their users.

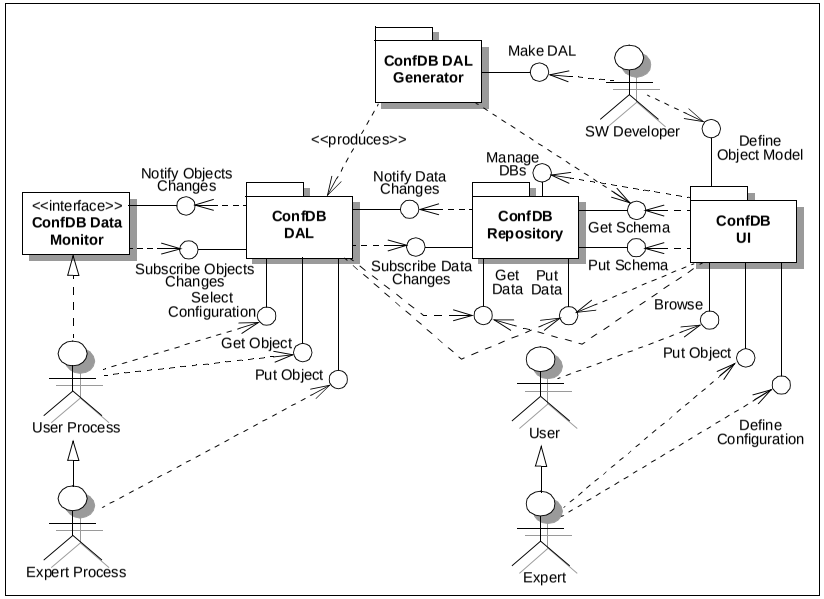


Figure 10-5 Configuration databases users and interfaces

10.2.3.2 Online bookkeeper

The OBK provides several interfaces to its services, being that some of them are human oriented, while others are Application Programming Interfaces (APIs) to allow interfacing with client applications. The access to these interfaces depends on the user's (human or system actor) privileges.

The OBK uses as persistency backbone the Conditions and Time-Varying offline databases services. In this sense, it counts on those services to provide the necessary means to store and retrieve coherently data that changes in time and of which there may exist several versions (e.g. configurations). In Figure 10-2 it is possible to observe the logical subdivision of the OBK system into abstract classes. Of these, the main ones are:

- **OBK Repository** - defines the basic methods to allow the storing, modifying and reading of online log data, as well as the methods to set the OBK acquisition mode and also to request log data from the several TDAQ processes providing them. It allows a human or system actor to start or stop the acquisition of log data. In order to become a log data provider a TDAQ application will have to realize the **OBK Log Information Provider** interface. This interface will allow a TDAQ application to accept subscriptions for log information from the OBK, as well as for the OBK to access log information in a TDAQ application;
- **OBK Browser** - this is the class responsible for providing the necessary querying functionality for the OBK database. Since the data browsing and data annotation functions are tightly coupled, the class also includes functionality to add annotations to the database;
- **OBK Administrator** - the OBK Administrator class provides to the users which are granted enough privileges the functionality to alter (delete, move, rename) parts or all of the OBK database. These users also given the possibility of changing the OBK acquisition mode (e.g. data sources, filters for the data sources).

Apart from the main classes depicted in Figure 10-2, OBK's architecture also includes four other classes (not shown in the diagram for reasons of clarity): **OBK UI Browser** and **OBK Browser API** both inherit from the OBK Browser class and define the human client oriented and the application client oriented versions of that class; the same thing happens for the **OBK UI Admin-**

istrator and the **OBK Administrator API** classes which defines the human client and application client oriented versions on the OBK Administrator class.

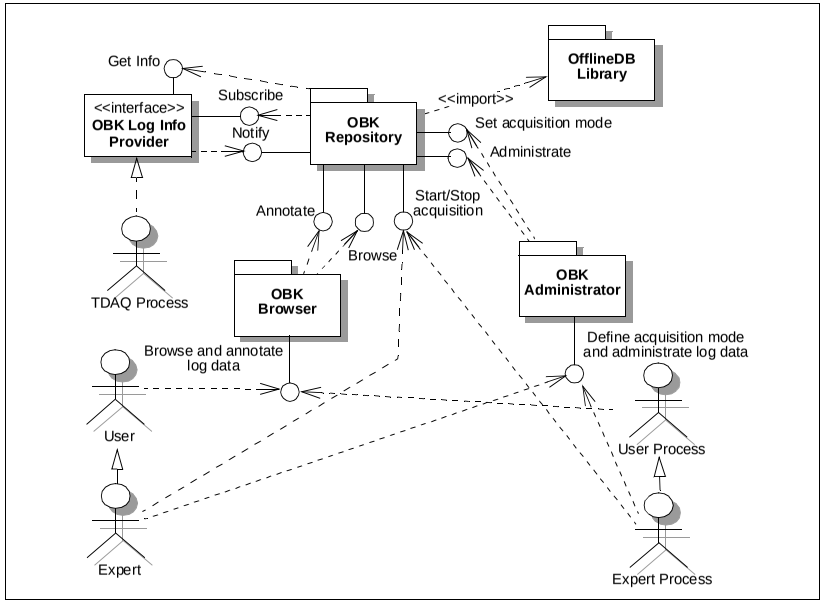


Figure 10-6 OBK users and interfaces

10.2.3.3 Conditions Database Interface

to be clarified after Databases workshop and the following workshop on 'non-event-databases'

10.2.4 Application of Databases by the TDAQ Sub-systems

Usage of the databases by the other TDAQ systems, concentrating on differences with general use.

Should be provided by TDAQ systems

10.2.5 Prototype Evaluation

10.2.5.1 Configuration Databases

The prototype [10-4] of the configuration databases is implemented on top of the OKS persistent in-memory object manager. It allows to store the database schema and data in multiple XML

files of which subset can be combined to get a complete description. The access to the configuration description can be made via file system (C++ interface) and via dedicated remote database server built on top of ILU (freeware CORBA implementation) and tested for C++ and Java interfaces.

Below there are results obtained during online software performance and scalability tests [10-3]:

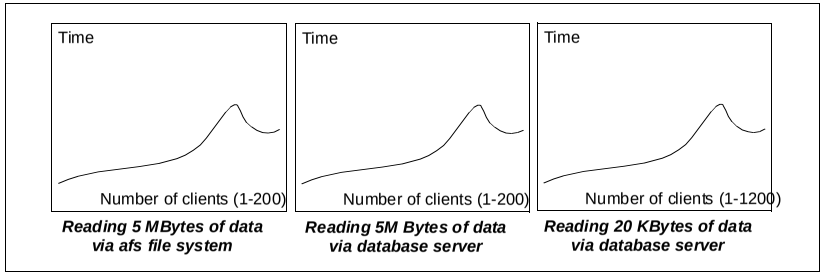


Figure 10-7 The results of the configuration databases performance and scalability tests

The tests with direct access of the configuration via common afs file system had shown good scalability and performance and such approach can be used if a common file system will be available. The graphs of the tests with access via the remote database server indicate the number of the servers which need to be started depending on number of clients, amount of data they read and time requirements.

The proposed architecture of the configuration databases allows to switch between implementation technologies without affecting user code. Other database technologies are studied as possible alternative of the one used in the prototype including relational databases like MySQL and ORACLE with possible object extensions and the POOL project supported by CERN IT division.

10.2.5.2 Online Bookkeeper

The prototype of the online bookkeeper was implemented on OKS persistent in-memory object manager and MySQL freeware implementation of relational database management system. Results obtained with the current MySQL implementation during performance and scalability tests [10-3] have shown, that it allows to reach a rate of 20 KBytes per second when storing monitoring data (100 bytes per data item) produced by up to 100 providers.

remark: no technology evaluation, we will use offline solution for Conditions & Time-Varying databases

10.2.5.3 Conditions Databases Interfaces

remark: we have no prototype of the Interface, there is prototype of the Conditions Databases only. Should we mention it?

10.3 Information Sharing

The choice of name for this section is not final. A possible alternative could be "Monitoring services". This would then be applied to the whole of the document where one talks about these services.

There are several areas where information sharing is necessary in the TDAQ system: synchronisation between processes, error reporting, operational monitoring, physics event monitoring, etc. The Online Software provides a number of services which can be used to share information between TDAQ software applications. This chapter will describe the architecture and prototype implementation of these services.

10.3.1 Functionality of the Information Sharing Services

Any TDAQ software application may produce information which is of interest for the other TDAQ applications. The first application will be called in this chapter Information Provider, the later ones will be called Information Consumers, which indicates that they are users of the information. Any TDAQ software application may be Information Provider and Information Consumer at the same time. The main responsibility of the Information Sharing services is:

- transportation of the information from the Information Providers to the Information Consumers
- delivery of information requests from the Information Consumers to the Information Providers.

Figure 10-8 shows main interactions which providers and consumers may have with the Information Sharing services.

Figure 10-8 Information Sharing in the TDAQ system

10.3.2 Performance and scalability requirements on Information Sharing

It is expected that the TDAQ system will contain about $O(10^3)$ processes. Each of those processes can produce information of different types. Therefore each Information Sharing service shall be able to serve $O(10^3)$ Information Producers simultaneously.

The number of Information Consumers for any single information item is expected to be about O(1) processes. Therefore each Information Sharing service shall be able to serve O(1) Information Consumers of each information item simultaneously.

The time to transport a single information object from the Information Provider to all the interested Information Receivers shall be about O(1) milliseconds.

10.3.3 Architecture of Information Sharing Services

The Online Software provides four services to handle different types of shared information. Each service offers the most appropriate and efficient functionality for a given information type and provides specific interfaces for both Information Providers and Consumers. Figure 10-9 shows the existing services.

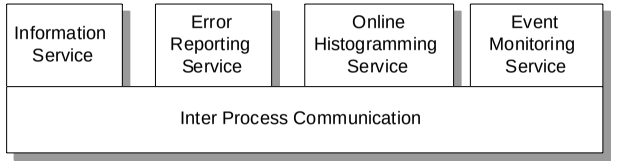


Figure 10-9 Information Sharing services

The Inter Process Communication (IPC) is a basic communication service which is common for all the other Online Software services. It defines high-level API for the distributed object implementation and for remote object location. Any distributed object in the Online Software services has common basic methods which are implemented in the IPC. In addition the IPC implements partitioning, allowing to run several instances of the Online Software services in different TDAQ Partitions concurrently and fully independently.

10.3.3.1 Information Service

The Information Service (IS) allows software applications to exchange user-defined information. Figure 10-10 shows interfaces provided by the IS.

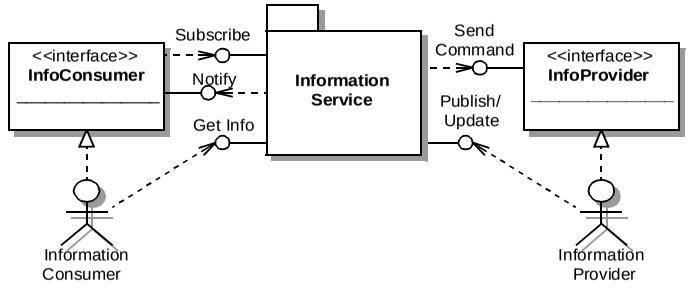


Figure 10-10 Information Service interfaces

Any Information Provider can make his own information publicly available via the Publish interface. Then there are two possibilities. The Information Provider, which does not implement the InfoProvider interface, has to inform the IS about all the changes of the information via the Update interface. The Information Provider, which implements the InfoProvider interface, up-

dates the information only when it is explicitly requested by the IS via the Send Command interface.

There are also two types of Information Consumers. One can access the information by request via the Get Info interface. This one does not need to implement the InfoConsumer interface. The Information Consumer, which implements the InfoConsumer interface, is informed about changes of the information, for which it subscribed via the Subscribe interface.

10.3.3.2 Error Reporting Service

The Error Reporting Service (ERS) provides transportation of the error messages from the software applications which detect these errors to the applications which are responsible for their handling. Figure 10-11 shows interfaces provided by the Error Reporting Service.

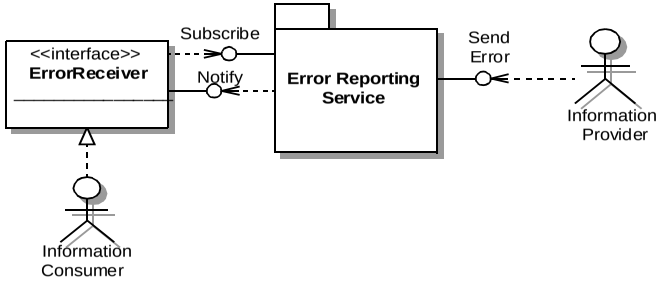


Figure 10-11 Error Reporting Service interfaces

An Information Provider can send the error message to the ERS via the Send Error interface. This interface can be used by any application which wants to report an error. In order to receive the error messages an Information Consumer has to implement the ErrorReceiver interface and construct the criteria which defines what kind of messages it wants to receive. This criteria has to passed to the ERS via the Subscribe interface.

10.3.3.3 Online Histogramming Service

The Online Histogramming Service (OHS) allows applications to exchange histograms. The OHS is very similar to the Information Service. The difference is that the information which is transported from the Information Providers to the Information Receivers has pre-defined format. Figure 10-12 shows interfaces provided by the Online Histogramming Service.

The OHS sub-package will provide also a human user interface in a form of an application. This application is called Histogram Display and can be used by the TDAQ operator to display available histograms.

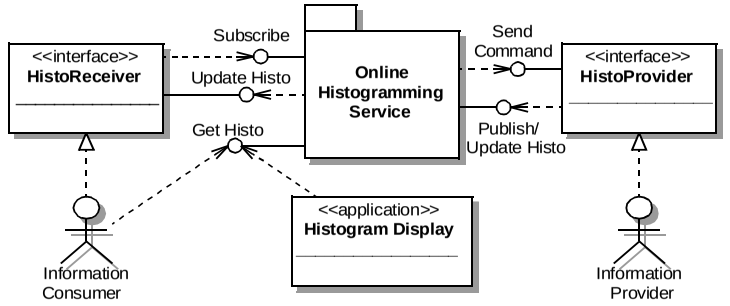


Figure 10-12 Online Histogramming Service interfaces

10.3.3.4 Event Monitoring Service

The Event Monitoring Service (EMS) is responsible for transportation of physical events or event fragments sampled from well-defined points in the data flow chain to the software applications which can analyse them in order to monitor the state of the data acquisition and the quality of physics data of the experiment. Figure 10-13 shows main interfaces provided by the Event Monitoring Service.

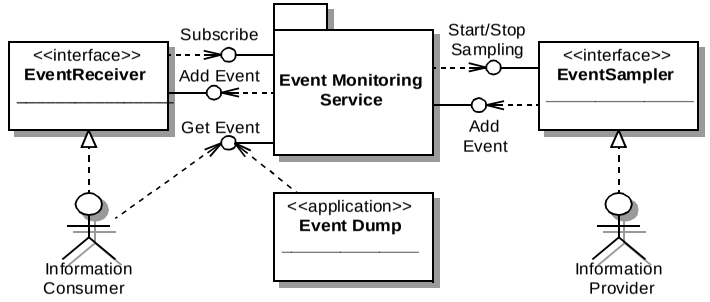


Figure 10-13 Event Monitoring Service interfaces

The application which is able to sample events from a certain point of the data flow has to implement the Event Sampler interface. When the Information Consumer requests the samples of events from that point, the EMS system ask the Information Provider via the Start Sampling interface to start sampling process. The Information Provider samples events and provides them to the EMS via the Add Event interface. When there are no more Information Consumers interesting in event samples from that point of the data flow chain, the EMS system ask the Information Provider via the Stop Sampling interface to stop sampling process.

There are also two types of interfaces for the Information Consumer. One is a simple Get interface which allows consumer to ask event samples one by one when they become necessary. This

interface will be used for example by the Event Dump application that implements a human user interface to the EMS system. A second interface is based on the subscription model. Using it the Information Consumer can ask the EMS system to supply the samples of event as soon as they are sampled by the Information Provider. This interface is more suitable for the monitoring tasks which need to monitor events for a long time in order to accumulate the necessary statistics.

10.3.4 Application of Information Sharing Services to the TDAQ sub-systems

Usage of the information services by the other TDAQ systems, concentrating on differences with general use.

Should be provided by TDAQ systems

This sub-section should only exist if the information is not already covered in Chapter 7, "Monitoring".

10.3.5 Prototype evaluation

The prototype implementations have been done for all the Information Sharing services. These prototypes are aiming to proof the feasibility of the chosen design and implementation technology for the final TDAQ system, and to be used for the ATLAS test beams. This chapter contains description of the services implementation along with their performance and scalability test results.

10.3.5.1 Description of the Current Implementation

The Online Software provides prototype [10-4] implementations for all the Information Sharing services. Each service is implemented as a separate software package with both C++ and Java interfaces. All the services are partitionable in a sense that it is possible to have several instances of each service running concurrently and fully independently in different TDAQ partitions.

The Information Sharing services implementation is based on the Common Object Request Broker Architecture (CORBA) defined by the Object Management Group (OMG). CORBA is a vendor-independent specification for an architecture and infrastructure that computer applications use to work together over networks. The most important features of the CORBA are: object oriented communication, inter-operability between different programming languages and different operating systems, object location transparency.

10.3.5.2 Performance and scalability of current implementation

The most exhaustive tests have been done for the Information Service which provides the most general facility for the information sharing. The other services are implemented on the same technology and will offer the same level of performance and scalability as the IS.

The test bed for the IS test consists from 216 dual-pentium PCs with processor frequency from 600 to 1000 MHz. [10-3] The IS has been set up on one dedicated machine. Another 200 machines have been used to run from 1 to 5 Information Providers on them. Each Information Provider publishes one information object at the start and then updating it once per second. The

last 15 machines were used to run 1, 5, 10 or 15 Information Consumers which subscribe for all the information in the IS. Whenever an Information Provider changes the information, this new information was transported to all the Information Consumers.

The time for transporting information from one Information Provider to all the subscribed Information Consumers have been measured. Figure 10-14 shows the average of the measured time as a function of the number of Information Providers working concurrently.

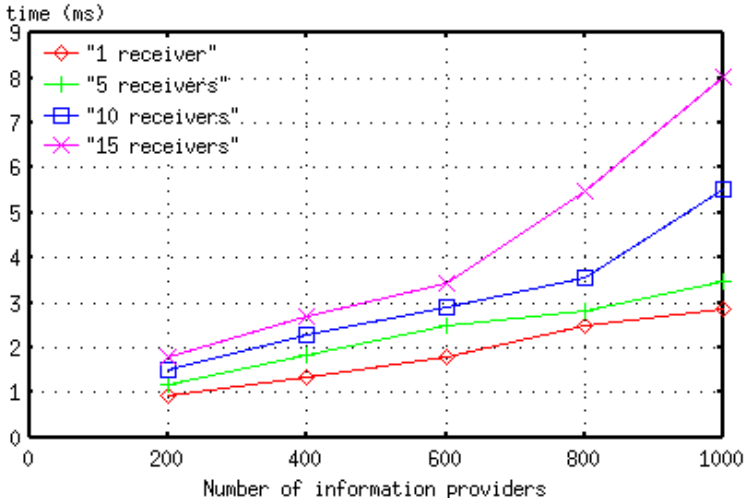


Figure 10-14 Time spent to transport one information object from one Information Provider to a number of Information Consumers vs. the number of concurrent Information Providers.

10.4 References

10-1 Online Software Requirements
http://atlas-onlsw.web.cern.ch/atlas-onlsw/documents/documents_page.htm

10-2 Online Software Architecture
http://atlas-onlsw.web.cern.ch/atlas-onlsw/documents/documents_page.htm

10-3 Test Report of Large Scale and Performance tests, January 2003, in preparation

10-4 Summary document used as input to the ATLAS TPR document
Altas DAQ-1 technical notes
Conference Papers
http://atlas-onlsw.web.cern.ch/atlas-onlsw/documents/documents_page.htm

10-5 Notes on technology evaluation - to be written

10-6 References to external documens on used or evaluated technology

11 DCS

The chapters of Part 2 should contain the major components as identified by the architecture.

Details should be provided on design, implementation and supporting measurements. For each component describe: the purpose/function/scope of the component, the performance requirements of the component, the architecture of the component, a proposed implementation, and performance and validation measurements.

11.1 Introduction

The principle task of DCS is to enable the coherent and safe operation of the ATLAS detector. Safety aspects are treated by DCS only at the least severe level. All actions initiated by the operator and all errors, warnings and alarms concerning the hardware of the detector are handled by the DCS. Concerning the operation of the experiment, an intense interaction with the DAQ system is of prime importance.

To be done...

+ This introduction has to be expanded and wording be reviewed.

+ Description of what is contained in this chapter

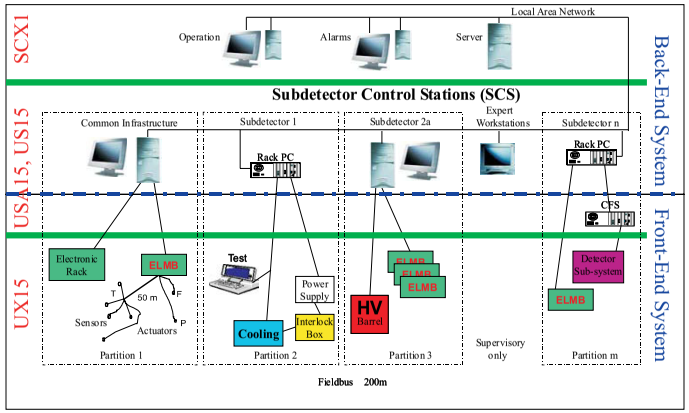
11.2 Organization of the DCS

The architecture of the DCS and the technologies used for its implementation are strongly constrained by environmental and functional reasons. The DCS consists of a distributed Back-End (BE) system running on PCs, which will be implemented with a Supervisory Control And Data Acquisition system (SCADA), and of the different Front-End (FE) systems.

The DCS can be partitioned into vertical slices as shown in figure XXX (ref to the picture below). Such a partition can be operated completely independent from other slices of the DCS and offers full SCADA functionality to its users. A vertical slice controls a subsystems of the ATLAS detector, where a subsystem is defined as an arbitrary part of the detector, e.g. the high voltage system of a subdetector or the subdetector itself.

The DCS FE instrumentation consists of a wide variety of equipment, from simple front-end elements like sensors and actuators, up to complex computer systems that are connected to the SCADA stations by means of standard fieldbuses. A SCADA run-time database contains records of all equipment where the data values are stored.

The equipment of the DCS will be geographically distributed in three areas: the main control room at the surface of the installations, the underground electronics rooms USA15 and the detector's cavern, UX15. The SCADA component will be distributed over the two first locations while the front-end equipment will be placed in USA15, US15 and UX15 as shown in figure XXX.



XXX This figure has to be changed since LCS appears instead of SCSs. XXX

XXX The following has to be summarized...some parts can be moved to the Other FE equipment, i.e. non ELMB stuff. XXX

UX15:

The Front-End (FE) electronics in UX15 (see figure 1.3) is exposed to radiation and to a strong magnetic field. The instrumentation in the cavern must be radiation-hard or tolerant to levels of 1–10^5 Gy per year in the muon subdetector and inner tracker, respectively. In the following, only the DCS FE equipment located outside of the calorimeters in ATLAS where the dose rate is of the order of 1 Gy / year will be addressed. In addition, depending on the location, a magnetic field of up 1.5 T has to be tolerated.

The equipment at this level consists of controllers, which connect to the hardware, either as separate modules or as microprocessors incorporated in the front-end electronics. Field instrumentation like sensors and actuators will be of various types and it will be tributary to the requirements for the detector hardware.

This equipment is distributed over the whole volume of the detector with cable distances up to 200 m. The distribution underground is governed by two conflicting constraints. Because of the radiation level, the magnetic field and the inaccessibility at UX15 during beam time, the equipment should be located in USA15. However, complexity, cost and technical difficulties of cabling suggest condensing the data in UX15 and transferring only the results to USA15.

Hardware interlocks of components will be implemented wherever needed. This is the case, for example, for automatic switch off of the front-end electronics of the pixel detector in case of problems of its cooling system, or for automatic ramp-down of high voltages in presence of over-currents. The operation of interlocks must be ensured even in the case of power failure and therefore most of the interlock systems have to be fed by Uninterruptable Power Supplies. Remote sensing and actuator equipment at the detector level and in the electronics crates and ancillary equipment, such as safety and general electricity, will be connected directly to one of the proposed standard buses.

USA15 and US15:

++++Here we have to mention that the US15 underground electronics cavern and the USA15 are similar and they will hold similar equipment. The only difference is that US15 is not accessible during operation of the accelerator due to the remaining radiation levels.

The equipment in the detector's cavern will be interfaced, via fieldbuses, to FE computer equipment located in the underground electronics room USA15, which is accessible during operation, and which consists of:

Workstations, foreseen for subdetector experts for the supervision of individual partitions, mainly during commissioning and maintenance periods. Dedicated stations that control the equipment running real-time operating systems, usually distributed around the installation. It is foreseen to use a dedicated control computer for each detector. In this context, a controller station is not necessarily a single computer but can be clustered if a high channel count or the heterogeneity of the equipment leads to this requirement. These systems are called Subdetector Control Stations (SCS) in figure *XXX Figure needs to be changed XXX* and they will run the SCADA software collecting data from the front-end devices in their partition. The SCS allows to run a partition either independently in standalone mode or integrated as part of the whole detector.

Depending on the complexity of the subdetectors, it is envisaged to introduce a further grouping at the level of the controller stations. Complex Front-end Systems (CFS), which normally do not run SCADA, are dedicated to specific tasks. The CFS are normally connected to SCS over a dedicated Local Area Network (LAN). CFS can also be placed in UX15 if they support the hostile environment.

SCX1:

The equipment of this layer will be installed in the main control room in building SCX1 at the surface of the installations. This area will always be accessible to the personnel. The equipment will consists of general-purpose workstations, which will be linked to the control layer through a LAN providing TCP/IP communication.

The workstations will retrieve information from the SCS of the different subsystems underneath and can be used to interact with them by means of commands or messages. The system will only provide a limited set of macroscopic actions to generate the sequence of operations necessary to bring the experiment as a whole to a giving working mode. In addition the system will monitor the operation of the sub-systems, generate alarms and provide the high level interlock logic where necessary.

11.3 Front-End System

The ionizing radiation and strong magnetic field limit the types of technologies that may be used. *Amongst other fieldbuses,* CAN bus has been chosen as the standard field bus for this area as this can operate in a strong magnetic field. The ionizing radiation in the cavern is of the order of 1Gy per year outside the calorimeter. The "ALTAS Policy on Radiation Tolerant Devices" [ref] has been formed to give the ATLAS sub-system groups specific rules concerning testing and qualification of radiation tolerant electronics. Three different radiation types have to be studied for full qualification, simulated radiation level (SRL) for the Total Ionizing Dose (TID), Non-Ionizing Energy Loss (NIEL) and Single Event Effects (SEE). The simulation results depend upon

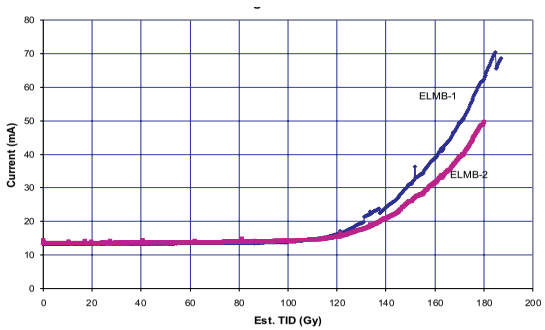
location in the cavern and these values are used with safety factors for qualification of the electronics.

Justification of CAN: Reliability - Built-in CAN error checking, Determinism - errors indicated but no acknowledge, Coverage - used for highly distributed systems, Robustness - designed for harsh environments and has high noise immunity, Galvanic Isolation - essential for long buses, Operation in Magnetic Field - does not use magnetic sensitive components, Low Power Dissipation - necessary for remote powering, Openness - is not proprietary and therefore no license fee.

11.3.1 Embedded Local Monitor Board

Add that no commercial solution exists which fulfils the requirements outlined above and that is why the ELMB has been developed...

The ELMB contains 64 analog input channels each of 16-bit accuracy. As well as the analog inputs, there are 8 digital input lines, 8 digital output lines and 8 configurable (either input or output) digital lines. Other interfaces are available such as a serial port allowing JTAG or other protocols to be implemented.
The standard software that is pre-loaded into the ELMB allows for communication over a CAN field bus using the higher level protocol CANopen. The standard functionality gives 'plug and play' usage for the analog inputs and digital inputs and outputs.
The graph below shows the digital current increase due to TID for three ELMBs.



XXX This picture has to change. Curve for ELMB_3 has to be removed. The different types of microprocessors will not be mentioned here XXX

No destructive SEE has been seen. No effect has been observed for NIEL. The ELMB has been tested in a magnetic field and no adverse effect has been observed. The performance, in terms of accuracy and stability, have also been proven to be sufficient for all applications in ATLAS.

To be summarized...
A motherboard is available that provides standard connectors for the analog and digital inputs and outputs for the ELMB, as well as a standard connector for the CAN bus. Sockets for adapters, used to scale the analog inputs to the required range, are provided allowing standard sensors to be used. For non-standard sensors, adapters may be manufactured to a given specification.
An interlock box has been designed and implemented and provides a hardware interlock with a

set point hard wired using a resistor of pre-calculated value. The device is a stand alone module, though status information can be obtained. The status values may be acquired using the digital inputs of the ELMB.
There is a DAC prototype available for use with the ELMB. This is a module containing 16 analog output channels, each of 12-bit resolution, where up to four modules may be connected to a single ELMB (giving a total of 64 analog output channels).

XXX The following has to be reduced. Only ELMB functionality should be described. Mention that the ELMB has been to consume low current such that it can be powered remotely via the bus. Mention the node supervision mechanism as well...Then it has to be moved up to the ELMB functionality description XXX

CANbus Topology
The CAN bus topology used is specific to the needs of each subdetector, where cost, geographical location, logical grouping, speed and reliability are factors effecting the choices made.

Bus Powering
The ELMB nodes are powered from the control room via the bus as the power supplies cannot be placed in the cavern. It must be possible to switch the digital power for the bus to allow for recovery of SEE by a hard (see Bus and Node Supervision). A large diameter cable is used from the control room to the cavern (to reduce voltage drop) and patch panels in the cavern allow for smaller (more flexible) cabling to be used over the detector. The smaller cable will be more susceptible to voltage drop and therefore the voltage at each node must be verified to be correct. This is more a question for the dynamic effect where if all nodes on the bus draw current at the same time, powering problems may occur.

Bus and Node Supervision
The ELMB uses a monitoring protocol to indicate the node is functioning correctly. If a failure is detected for an ELMB, a command requesting a soft reset of the node is sent (only to the ELMB that is not responding). It is possible that the error on the ELMB is within the communication, in which case a hard reset must be performed. A hard reset involves switching power for the digital part off and on again and is performed for the bus (and not a specific node). The time taken for this operation, allowing all ELMBs on the bus to reach operational state once more, must not exceed 10 seconds. If a hard reset does not clear the fault for an ELMB, no other operation is performed, the unit will be signalled as faulty and would be replaced when access is possible.

11.3.2 Other standard FE equipment

No explicit mention to OPC should be done at this stage...

Commercial high voltage and low voltage power supplies will be used in the experiment. The companies who produce these commercial power supplies also supply software (such as OPC servers that interface between their own communication protocol and the OPC standard) used to allow control of the power supplies. The CAN bus is often used for connection to the devices with various, often proprietary, higher level protocols implemented. Other standards include serial interfaces (various communication protocols) and GPIB.

Programmable Logic Controllers (PLC) compliant with the relevant recommendation given in [21]

Whenever convenient, like in case of large number of field instrumentation channels to be controlled, VME-based controllers may be used.

11.4 The Back-End System

Terms like partitioning or vertical slice should be removed from this section...

The functionality of the BE system is two-fold: It acquires the data from the front-end equipment and it offers supervisory control functions, such as data processing, presenting, storing and archiving. This enables the handling of commands, messages and alarms.

The BE system will be hierarchically organized to map the natural partitioning of the experiment into subdetectors, systems and subsystem. The BE hierarchy allows for the dynamic splitting of the experiment into independent partitions, which can be operated in stand-alone or integrated modes. The coordination of the different partitions is performed by means of commands and messages. The command flow is downwards, whereas the message exchange take place in either direction the within the slice. No horizontal communication is foreseen between different slices or amongst the components of an slice.

In order to provide the required functionality the BE of the DCS will be logically organized in three levels as shown in figure XXX. The actions on the operator time-scale are performed at the upper level, while the RT operations are performed at the lower level.

Global Control Stations

The overall control of the detector will be performed by the uppermost level of the BE system, which consists of the Global Control Stations. These stations are envisaged to provided high level monitoring and control of all subdetectors and technical infrastructure. The full control of the detector is provided at only lower levels in the hierarchy. At this level, different services will be provided like the DCS Information Service to handle the communication with the external systems, namely the LHC accelerator, the CERN infrastructure and the Detector Safety System, or web and database services. Information for these subsystems will be used to build the overall status of the experiment. Bidirectional data exchange between the DCS and the TDAQ system will also be managed at this level. No command exchange between the TDAQ and the DCS is foreseen at this level.

Subdetector Control Stations

The Subdetector Control Stations are placed at the intermediate of the BE hierarchy. There will be one SCS per subdetector and an additional SCS to handle the monitoring of the common infrastructure in ATLAS called Common Infrastructure Controls (CIC). The later will be interfaced with the DCS Information service in the layer above. All actions on a given subdetector provided at the Global Control Stations are also provided at this level. In addition, the SCSs allow for the full and stand-alone local operation of the subdetector by means of dedicated graphical interfaces. The SCSs also handle the communication with the services of the layer above. It is foreseen to have a direct connection from the SCSs to the DCS Information service to provide the different SCSs with the status of the external system, namely the LHC accelerator, the Detector Safety system, CERN services and the ATLAS magnet, as well as with the eviromental parameters of the common infrastructure. The SCS handle the co-ordination of all subsystems underlying in the layer below and are the responsible for the validation commands issued by the operator from the global control stations in the layer above or directly from the TDAQ run control. If low level control is required by the issued actions, e.g. ramp up high voltage, these commands can be propagated to the subsystems in the layer below for their execution. The overall status of the subdetector is assembled (collated???) and pass on to the TDAQ system via the DAQ-DCS communication software, which is described in section XXX, and to the control stations in the layer above.

Subsystem Control Stations

The bottommost level of the BE hierarchy is constituted by the Subsystem Control Stations, which handle the low level monitoring and control of the different systems and services of the detector. The organization of this level for a given detector could be performed attending to either geographical or functional criteria. In the former the arrangement follows the natural partitioning of the detector in sections, subsection, etc. whereas in the second approach, the organization is decided as a function of the different services of the subdetectors, e.g. cooling, high-voltage, etc. This level of the hierarchy is directly interfaced to the FE system. Besides the read-out and control of the equipment, it also performs calculations and fine calibration of the raw data from the FE and comparison of the values with preconfigured thresholds for the alarm handling. The station placed at this level will executed the commands propagated from the SCSs in the layer above although they can also execute predefined automatic actions if required.

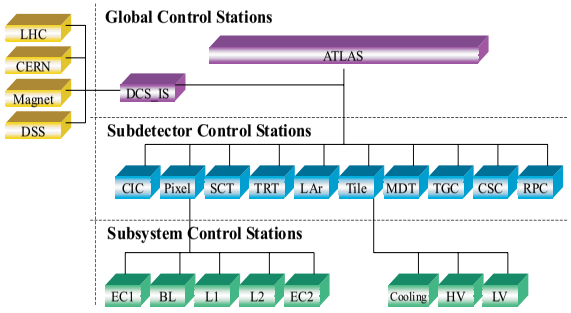
In addition, data and alarm archiving and logging of incidents and commands will be provided at each of these levels. Remote access to a well-defined set of actions to be performed by the two upper levels of the BE hierarchy will also be provided subject to proper access authorization.

Finite State Machine

XXX Is this the right place to talk about the operation of the DCS as a FSM? If yes, we have to think of how to link it with the sections above, i.e. with the description of the functionality of the three layers.XXX

The operation of the different subdetectors will be performed by means of Finite State Machines (FSM), which will handle the states and transitions of the different parts of the DCS. It is envisaged to have a FSM per subdetector. The states of these FSM will be assembled from the status of the different parts or services of the detector, which are determined by the status of the FE equipment, and from the status of the different environmental parameters monitored by the CIC station. As it will be described in chapter XXX (ref to chapter on operation 13 XXX), these states may be conditioned by the status of the external systems interfaced via the DCS_IS, e.g. the state of the LHC accelerator. *XXX We have to make sure that the DCS_IS is known at this moment XXX.*

The global operation of the BE system will be performed by a single FSM whose states will be built from the states of the different subdetectors' FSM, previously configured, and the status of the external systems. Any transition issued at this level will be propagated to the underlying subdetectors' FSM included in the running mode of the experiment.



The picture will be moved up. In addition, some modifications are required

11.4.1 SCADA

The BE system of the ATLAS DCS will be implemented using a Supervisory Control And Data Acquisition (SCADA) product. SCADA systems [37] are commercial software packages normally used for the supervision of industrial installations. They gather information from the hardware, process the data and present them to the operator. Even though SCADA products are not tailored to LHC experiment applications, many of them have a flexible and distributed architecture and, because of their openness, are able to fulfil the demanding requirements of the ATLAS DCS.

Besides basic functionality like the Human Machine Interface (HMI), alarm handling, archiving, trending or access control, SCADA products also provide a set of interfaces to hardware, e.g. CERN recommended fieldbuses and PLCs, and software, e.g. Application Program Interface (API) to communicate with external applications, or connectivity to external databases via the Open or Java DataBase Connectivity (OBDC and JDBC respectively) protocols.

SCADA products constitute a standard framework to develop the applications leading to a homogeneous DCS. Its usage saves development effort reducing the work for the subdetector teams. In addition, they follow the evolution of the market, protecting against changes of technology like operating system or processor platforms.

11.4.2 PVSS

A major evaluation exercise of SCADA products [38] was performed at CERN in the frame of the Joint COntrols Project (JCOP), which concluded with the selection of the PVSS-II, from the austrian company ETM, to be used for the implementation of the BE systems of the four LHC experiments.

PVSS is a device-oriented product where process variables that logically belong together are combined in hierarchically structured data-points. Device-oriented products adopt many properties from object-oriented programming languages like inheritance and instantiation. These features facilitate the partitioning and scalability of the application. The properties and functions of each data point can be parametrized by means of several attributes, e.g. alarm handling, periphery address, etc.

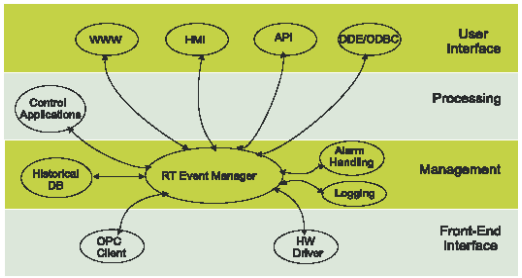
PVSS provides the interfaces to connect to external databases or systems, like the DAQ system or LHC accelerator, and the capability to extend the functionality of the product to interface custom applications or equipment (e.g. availability of driver development toolkit).

It is conceived as distributed systems. The single tasks are performed by special program modules called managers. The communication among them takes place according to the client-server principle, using the TCP/IP protocol. The internal communication mechanism of the product is entirely event-driven. Values and alarms must be notified on change with the possibility to define a dead-band or window range. This characteristic makes PVSS specially appropriate for detector control since, systems which poll data values and status at fixed intervals, present too big an overhead and have too long reaction times resulting in lack of performance.

The managers can be distributed over different platforms, and the communication between them is internally handled by PVSS-II. This has been one of the crucial points in the selection of this product since the DAQ system of the ATLAS experiment is been developed entirely under Linux, where as the DCS will widely use Windows.

PVSS allows to split the supervisory software into small application communicating over the network and it is imposed by the distribution of the DCS equipment in different locations in ATLAS.

The PVSS architecture is centralized around the RT Event Manager which handles the communication with all other modules. The different components can be arranged in functional layers as shown in figure 3.12.



The Front-end Interface layer provides the communication with the hardware by means of dedicated drivers or communication standards such as OLE for Process Control (OPC). It also provides drivers for standard fieldbuses, like Profibus and PLC. Neither CANbus nor VME are supported. The front-end layer also performs data processing to reduce the data volume from the equipment.

The management layer is responsible for the handling of bi-directional transmission of the messages between the different managers, as well as for managing RT data, archiving, alarm handling and administration of the user privileges.

The Processing layer contains control applications which are written within the SCADA system using its own programming language. They can be of two types: the first implements procedures dedicated to monitoring and controlling the detector; the second consists of specialized programs which extend the SCADA functionality, such as Finite State Machine (FSM) or additional reporting features.

The user interface layer takes care of external interactions. PVSS-II provides a powerful Human–Machine Interface (HMI) easily customizable by means of graphical objects that can be linked to the different process variables in the application. It includes a WWW server for remote access to the application. External programs communicate with the Management layer via the API, allowing external applications to subscribe to any RT and historical data and alarms.

11.4.3 PVSS Framework

To be slightly changed...

Although PVSS-II will be used as the basis of the LHC experiment controls, this has been found not to be sufficient to develop an homogeneous and coherent system. An engineering framework on top of PVSS-II is being developed. It is composed of a set of guidelines, tools and components which will be provided to developers of sub-systems in order to:

Reduce to a minimum the work to be performed by subdetectors teams by re-using standard components rather than duplication.

Design the system architecture in terms of distribution of functionality and network distribution.

Define guidelines for development, alarm handling, control access and partitioning, to facilitate the development of specific components coherently in view of its integration in the final, complete system.

Improve performance reliability and robustness.

Cover the lack of functionality of PVSS-II like interface to CANbus or VME, integration of FSM.

The ELMB framework component has been developed using SCADA tools, as a framework to facilitate the usability of the ELMB node to the ATLAS users but also to achieve homogeneity of the SCADA software in ATLAS. This package creates all infrastructure needed to work with the ELMB and it also comprises a “top-down” configuration tool, i.e. an utility to create all non-SCADA components necessary to operate the ELMB.

- *The framework also provides panels for configuration and run time of the project.*
- *Mention that it also contains a top-down configuration tool.*
- *Easy to interface to the conditions database since the ELMB structure is unique.*

11.4.4 Global PVSS based services

All DCS systems from the global level to the local control stations will need some commonly used services. These applications will be implemented once and may be used at all levels.

Data and alarm displays

There will be a standard set of display panels to show information in a consistent way. The information shown will contain data (read from the front-end electronics) and alarms (taken from all systems, including external systems such as DSS).

Web services

For standard information (such as data and possibly alarms) a web interface will be developed showing generic information. The information will be shown in a generic form and not be subdetector specific. No operations will be possible for security reasons.

Histograming interface

Something about the interface to other display tools used in ATLAS

Logging

All actions carried out, whether by an operator or an automatic process, are logged. A system will be available that will allow the logs to be examined.

11.5 Integration FE-BE

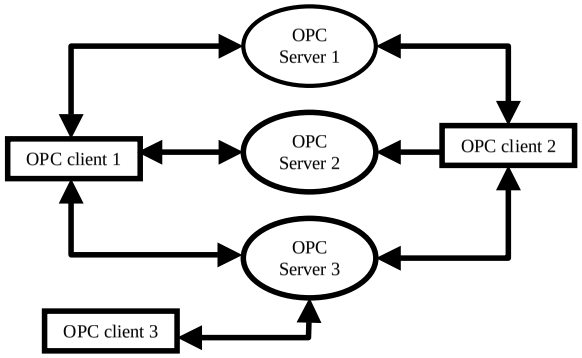
The PVSS-II product will be used as SCADA system for the implementation of the supervisor layer of the ATLAS / DCS. There are several interfaces which allows to connect PVSS-II based systems to hardware.

- Dedicated drivers for PVSS-II; PVSS-II has the drivers for modbus devices, PROFIBUS and some others. It also contains the API to develop drivers by users.
- PVSS-OPC client; OPC is a wide used industrial standard. All commercial Low and High voltage systems are supplied with the OPC servers.
- DIM software, which is a communication system for distributed and multi-platform environments. DIM provides a network transparent inter-process communication layer developed at CERN.

The OPC due to the wide spread usage and the big support from industrial has been chosen as main interface from the SCADA to hardware devices. In turn the ELMB will be widely used in the implementation of the subdetector front-end system. To connect the ELMB to SCADA the OPC CANopen server has been develop. Others possibilities will also be used in suitable cases.

11.5.1 OLE for Process Control

The main purpose of this standard is to provide the standard mechanism for communicating to numerous data sources. The OPC is based on the Microsoft Windows technology. The specification of this standard describes the OPC Objects and their interfaces implemented by OPC server. The architecture and specification of the interface was designed to facilitate clients interfacing to remote server. An OPC client can connect to more then one OPC Server, in turn an OPC Server can serve several OPC clients (Fig 1). All OPC objects, consequently, are accessed through interfaces. Any client sees only the interfaces.



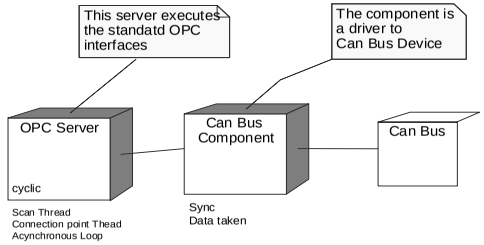
11.5.2 CANopen

CANopen is a high level protocol for the CAN-bus communication. This protocol is widespread as well. CANopen standardizes the types of CAN-bus messages (objects) and defines the sense of them. It allows to use the same software in order to manage of CAN nodes of different types and from different manufacturers.

On the market there are a lot of the CANopen servers. But all of them are tailored to their specific hardware interface cards. This OPC serves provides the CANopen functionality required by the ELMB.

11.5.3 OPC CANopen server

The OPC CANopen server works as the CANopen master of the bus handling network management task, node configuration and transmitting data to the OPC client.The OPC CANopen Server consists of two main parts (Fig ?).



The General structure of OPC CANopen server. *Labels must be standard CAN, CANopen*

- The main part is an OPC server itself. It implements all the OPC interfaces and main loops. Any application interacts with this part through interfaces.

  The CANopen OPC server transmit data to a client only on change, which results in a substantial reduction of data traffic.

- The second part “Can Bus component” is hardware dependent. It interacts with a CAN bus driver and controls CANopen devices.

  It was developed as a COM component. This approach allows changing hardware board rather easy. In such a case only hardware dependent component should be reprogrammed and, in turn, the main part should not be re-compiled even. Also It can recovery the state of CANopen node in case fault due to power cut for example.

Several busses with up to 127 nodes each, in accordance with CANopen protocol, can be operated by the OPC CANopen server. The system topology in terms networks and nodes per bus is modelled in start up time in the address space of the OPC CANopen server from a configuration file.

11.6 Read-out chain

The ATLAS DCS Vertical Slice is defined as the full read-out chain, which ranges from the I/O point (sensor or actuator) up to the operator interface comprising all the elements described above: ELMB, CANopen OPC Server and PVSS-II. The vertical slice also comprises PVSS-II framework panels to manage the configuration and the settings and the status of the bus.

PVSS-II models the system topology in term of CANbus, ELMB and sensor in the internal database by data-points. These data-points are connected to the item in the OPC server to send the appropriate CANopen message to the bus. In turn, when an ELMB sends a CANopen to the bus, the OPC server will decode it, set the respective item in its address space and then transmit the information to a data-point in PVSS. The OPC CANopen server can convert the raw data to physical units. The SCADA application will carry out the fine calibration, trend and archive the value. The main function of each element of read-out chain describes below:

ELMB

- Digitize data according ADC settings. The main task of a ELMB is to convert and analog data to digital form.
- Self ADC calibration. In power up time an ELMB executes the internal ADC calibration.
- Send and received digital data. An ELMB has the input and output digital ports in order to exchange status and control information.
- Send analog data in microvolt or counts. The ELMB can convert counts to microvolt based on the ADC settings.
- Send analog data on changed or on closing window limits. In order to reduce bus traffic and increases the system performance an ELMB can send data only when the value differs from pervious one on predefined value. The second possibility is that the data will be sent when the value crosses the predefined limits.

SCADA system:

- Global control of whole DCS system. The SCADA is a base of developing the Global and Local Detector Control Stations and represent the information to an operator.
- Archive data.
- Visualize and trendling data.
- Fine calibration. There are cases when the calibration claims the information from different issues or data have to correct due to time drifting.
- Network supervision. The reliability and robustness of network and fieldbuses should be supported.

OPC CANopen server:

- Control CAN buses and CANopen node including the recovery procedures after power cut.
- Send data to the OPC client on change.
- Convert to the physical units and the sensor calibration.

The readout chain constitutes the basis for several control applications of ATLAS subdetectors. In most cases, the ELMBs will be exposed to radiation, and therefore will be subject to radiation

effects, which will have to be handled throughout the readout chain, e.g. SEE can be cured by issuing a hard reset of the modules from PVSS as described in section XXX. In order to understand the behavior of the readout chain in a real environment, two complementary tests were performed and are described in the following section.

11.6.1 ELMB Full Branch

To investigate the performance and the scalability of the DCS readout chain to the size required by ATLAS, a full vertical slice (or full branch) consisting of 6 CANbuses having 32 ELMBs each was assembled.

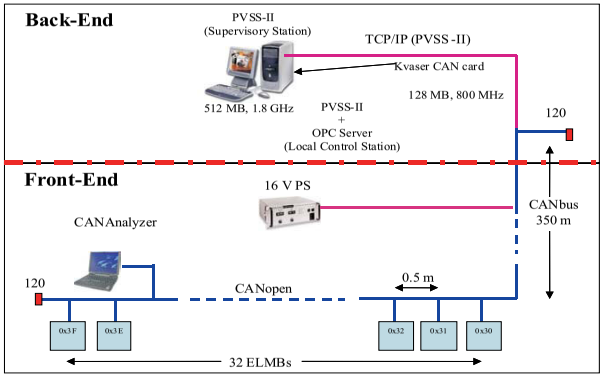
The aim of this test was to study the behavior of the system with these characteristics to discover settings required in order to achieve the optimal results and to establish limits of the readout chain. These limits define the granularity of the system in terms of number of ELMB per CANbus and the number of buses per PVSS system. In particular, the following was to be inspected and investigated:

- Remote powering of the ELMB nodes via the bus. The radiation levels in the detector cavern impose that the power supplies will have to be placed in the underground electronics rooms US15 and USA15. Therefore, the power for the nodes will have to be fed remotely via the CANbus with distances up to 150 m. (*XXX In the ELMB section it must be mentioned that the nodes consume very low current, i.e. they were designed to consume low current for this purpose XXX*)
- Bus loading, which determines the number of nodes per bus. The data traffic on the bus has to be uniformly distributed over time in order to keep the bus load low under normal operation. In ATLAS the bus occupancy will be kept below 60% in order to cope with a higher loads which may arise in case of problems like power cuts. In these cases, an avalanche of channel information which must be handled by the system.
- Optimization of the work balance amongst the different processing elements in the readout chain. The functions to be performed by the ELMB, CANopen OPC server and PVSS are homogeneously distributed to ensure equal load of each of these components and to avoid bottle-necks.
- Optimization of the system performance by tuning of different software settings such as update rates for OPC and the readout rate.
- Determination of the overall performance of the systems, which defines the number of CANbuses with these characteristics per PVSS system, and that will strongly condition the topology of the different subsystems.

The setup employed in the test, shown in figure 1. A system of 6 CANbuses was operated from PVSS-II using the CANopen OPC server and a Kvaser CAN interface. The bus lengths were 350 m in all cases, in order to fulfil the ATLAS requirements with a broad margin. Up to 32 ELMB were connected at the end of each CANbus. The total number of channels in this system was: 12288 analog inputs, 3072 digital outputs, and 1536 digital inputs. In it important to note that the amount of channels in the set up described here, is of the order of magnitude of some large applications in ATLAS.

The other end of the CANbuses was connected to a single PC running the SCADA software. All nodes in a bus were powered from a single power supply connected at the computer's end of the CANbus. All messages on the buses were read into PVSS-II for archiving to the local database. In addition, the network traffic was also logged using a CAN analyzer, for off-line com-

parison between the analyzer and PVSS-II database contents. This allowed to determine whether all CAN messages sent to the bus, have been properly read into PVSS-II.



During the test, the readout rate was increase in order to push the system to the limit. When big bursts of data arrive at PVSS-II very rapidly, the different messages are internally buffered, first by the event manager and then by the data base manager until they are sent to the archiving manager. Two different situations can be distinguished:

- *Steady run*, where all messages sent by the ELMBs to the bus are stored to the PVSS-II database and, in addition, the CPU memory usage remains constant, i.e. no buffering is performed at the PVSS-II or OPC level.
- The so-called *avalanche run*, where the fastest possible read-out rate is estimated for a a short period, typically a few minutes. Under these circumstances, although all messages are archived to the PVSS database, the ELMB data flow is so high, that messages cannot be treated in real time and are buffered at the SCADA level therefore, leading to an increase of the memory usage. It is important to note that although long term operation under these conditions would not be possible, this situation can occur, and must be handled by the system, in case of major problems of the equipment monitored, e.g. power cut, where the read-out system will have to cope with large bursts of messages for a short interval.

The monitoring of the bus load with a CAN analyzer showed a bus load of about 65% in normal operational conditions. This result indicates that the number of ELMB per bus at the bus speed of the test, 125 kbaud, must not exceed 32, if all analog input channels are used.

A readout rate of at least 30 s is required for a steady run in a system of 6 buses with 32 ELMB nodes each (12288 analog input channels). Under these conditions, the fastest readout rate in avalanche mode is limited to 8 s for approximately 3 minutes. The examination of the CPU load showed that in all cases, the results presented are only limited by the CPU work load due to the PVSS-II managers as a consequence of the buffering of the CAN messages. These results indicate that the operation of the readout is strongly constrained by the performance of PVSS-II. Therefore, this plot can also be understood as an estimate of the time required by PVSS-II to handle different data volumes.

11.6.2 Long term operation in radiation

Is this title correct?

The long-term operation of the full readout chain was tested with a number of ELMBs in a radiation environment similar to that expected in the ATLAS cavern, though with a much greater dose rate. This environment allows the different error recovery procedures implemented at the different levels of the readout chain, which are required due to radiation effects to be tested. A CAN bus of greater than 100 m was connected to a PC running the CANopen OPC server and PVSS-II. The test ran for more than two months, which was equivalent to more than 300 years at the expected ATLAS dose rate in terms of TID. The CAN controller in the ELMB ensures that messages are sent correctly to the bus, and will take any necessary action if errors are detected. Bit flips were seen during the test at the ELMB using special test software, and these are also handled at the ELMB level. The OPC server ensures all ELMBs on a bus are kept in the operational state, monitoring the messages on the bus in case of power glitches. At the highest level, PVSS scripts were utilized to monitor the current consumption for the bus (where increase in current is an indication of latch-up or damage from long term TID) and to reset ELMBs if communication has been lost. Through this script, the power supply for the bus was controlled allowing for hard resets to be performed. No user intervention was necessary during the time of the test.

11.7 Applications

Covers: Common Infrastructure controls (racks, cooling, radiation monitoring, etc.)

The ALTAS DCS system has an 'extra' subdetector. This is the Common Infrastructure Controls (CIC) subdetector which is used to monitor and control all systems(?) that are common to other subdetectors. The monitoring and control of these systems is not exclusive to a given subdetector, therefore the need for an overall control station was identified. The CIC will be a full vertical slice of the DCS (including both BE and FE) and will be capable of stand-alone operation. The systems under the supervision of the CIC are:

- *Rack control*
- *Cooling*
- *Radiation Monitoring*
- *Gas????*

Information obtained from these systems is required for the other subdetectors. There will be communication between the CIC and the other SCSs for this purpose. For example, a subdetector may wish to switch off high voltage if the cooling is not functioning correctly. This means that software interlocks of the different subdetectors rely on information provided by the CIC.

The LHC rack project [ref: www.cern.ch/ESS/crateProject/] uses standard connection protocols (CAN or WorldFIP with OPC) to allow control of the sub-racks in the experiment area.

The LHC experiment Gas Control System (LHC GCS) aims at providing LHC experiments with a unique control system (supervision and process control layers) for the 23 gas systems of the four LHC experiments. The EP division Gas Working Group has been set up to investigate the feasibility of developing common gas systems for the many sub-detectors that compose the LHC experiments. The GWG mandated the IT-CO group to design and develop the corresponding control systems. The system uses a PLC with standard interface software.

Muon alignment system, LAr purity control system, CS calibration source of the Tilecal, etc.

Subdetector specific equipment is used where the needs of the subdetector are too specific for commercial hardware to fulfill the requirements. Although the hardware and software of these systems may be non-standard, the interface between these systems and the detector control system must be a recognized standard. These systems often perform complex calculations, though only a sub-set of these results are passed to the SCADA system. The interface is released using such standards as OPC.

11.8 Connection to DAQ

To be reduced...

In order to grant a coherent functioning of both DCS and the physical data triggering and acquisition the following communication functionality is to be provided [11.6_1, 11.6_2]:

- Trigger and DAQ applications should be capable to access DCS data (detector, accelerator and environment parameters) that are necessary for both on-line evaluation of run conditions and of-line event reconstruction.
- DAQ data important for the detector control (e.g., like run type and status) should be available for DCS.
- DAQ (in particular, the DAQ shift operator) should be informed about critical events within the detector control system.
- DAQ should be capable to issue commands for DCS in order to synchronize the states of DAQ and DCS as well as to perform specific operation on detector when they are foreseen for a certain circumstances and / or run state. The command execution results should be then delivered back to the application having the command issued.

11.8.1 Context of Communication Subsystem

In accordance of the concept of TDAQ partitioning [11.6_3] the communication functionality required should be provided for each needing it TDAQ partition independently of others.

The TDAQ Online software package (see Ch.10) provides a series of services for Trigger / DAQ inter-application communications of the content declared above for DCS communication. They are the Information Service (IS) allowing to share the run time information (10.3.3.1), Error Reporting Service (ERS) providing distribution of application messages (10.3.3.2) and the Run Control package (see 10.1.3.1 and 13.2) running a finite state machine to represent and control / synchronize the states of TDAQ subsystems' belonging to a partition. These services / subsystems will be used as the DAQ-side connection points for the communication with DCS.

The PVSS II product (11.4.1.2.3) defined as the base of developing the detector controls has no facilities to communicate with the external application in an active manner. Therefore, the DAQ - DCS Communication subsystem (DDC) has to be implemented as an active interface capable to communicate with both the DAQ connection services defined above and a PVSS system. The tool for the latter is provided by a powerful application program interface (API) of PVSS II allowing full direct network access to the PVSS application runtime database.

In order to provide better performance and reliability and in accordance of DAQ-side connection points the DDC subsystem is to be implemented as three independent applications with the following functionality:

- Bi-directional exchange of data like parameters and status values;
- Transmission of DCS messages, like alarms, to DAQ;
- Synchronization DCS with TDAQ run control and providing ability for DAQ to issue commands on DCS (with feedback).

The context diagrams of these applications are shown in fig. 11.6.1.

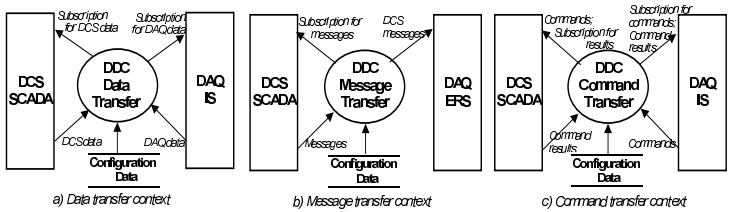


Fig.11.6.1 DAQ - DCS communication system's context diagram

The Configuration Data on the fig 11.6.1 is to define the TDAQ partition that the application is to work at, list of data / messages / commands to handle, etc. This information should be defined in the TDAQ configuration database. The applications configure themselves while starting. Each of the applications then subscribes in the source side for the data, messages or commands having been defined by the configuration and transfers that information to the partner system when arises / changes.

11.8.2 Communication Software (Interface DCS - Trigger/DAQ)

The section describes the DAQ - DCS communication software. The prototype of the DDC package has been tried in the test beam experiments of 2002 - 2003 and demonstrated satisfactory and reliable capability of working.

11.8.2.1 Data Transfer Facility (DDC-DT)

The data exchange in both directions is to be implemented via the Information Service of DAQ Online software. The application keeps the data elements (parameters of the systems) declared in the DDC-DT configuration being the same in both PVSS application and the Information service. This is done on the base of the subscription mechanism available for both sides. The list of data to be transferred in both of directions is the content of the DDC-DT configuration. The collaboration diagram of this use case is drawn in fig.11.6.2. Figure 11.6.3 shows the use case of a single read DCS data on request of a TDAQ application that is also to be provided by DDC-DT.

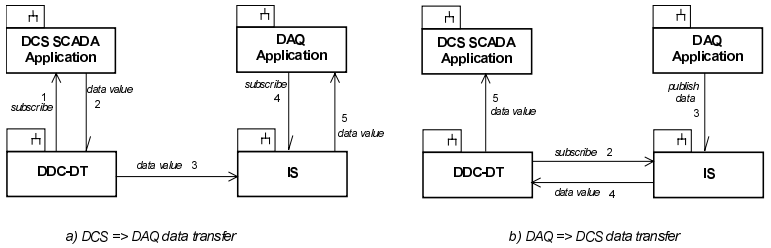


Figure 11.6.2 DDC-DT collaboration diagrams (update data on change)

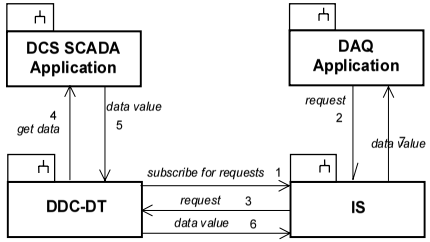


Figure 11.6.3 DDC-DT collaboration diagram (get data on request)

In figures above DDC-DT is the application, which always issues the first message either for DCS (PVSS) or for IS. It is assumed, therefore, that two systems to communicate are already working when DDC-DT starts. However, the situation when a system is not yet ready (as well as stop / restart of any of them during the run) is handled also properly with issuing an error message and waiting the readiness.

The DDC-DT application is to be implemented as a PVSS API manager [11.6_4], which integrates the application program interface of DAQ information service to set and get data to / from there.

11.8.2.2 Message Transfer Facility (DDC-MT)

The DCS message transferring to DAQ is to be implemented via the Error Reporting System of DAQ. The collaboration diagram of the application is drawn in fig.11.6.4.

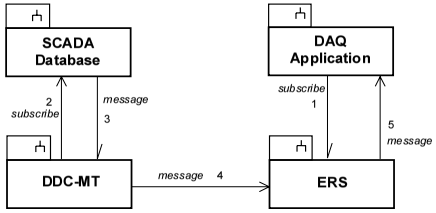


Figure 11.6.2 DDC-MT collaboration diagram.

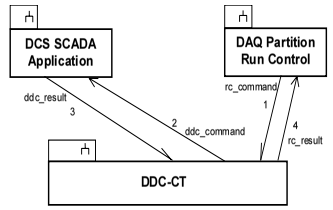
The DDC-MT configuration defines the list of DCS alarms and text variables to be delivered for ERS.

In fig.11.6.4 DDC-MT is the subsystem, which always issues the first message (subscription) for DCS and, while a DCS message has came, sends it to the ERS server of a corresponding TDAQ partition. It is assumed, therefore, that typically both DCS' PVSS application and ERS are already working when DDC-MT starts. If however PVSS system is not running, DDC-MT issues a message about the connection error and waits the readiness of PVSS application. If ERS server is not running when DDC-MT has started a console error message is to be issued. The DCS messages that have been issued earlier than ERS server starts will not be delivered for DAQ. A TDAQ application will receive, surely, only the messages coming to the ERS server after that application has subscribed for them.

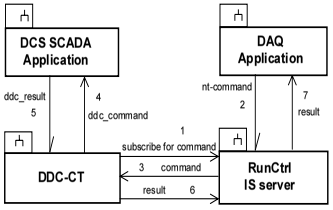
The DDC-MT application is to be implemented as a PVSS API manager [11.6_4], that integrates the application program interface of DAQ error reporting service to distribute messages.

11.8.2.3 Command Transfer Facility (DDC-CT)

The DDC-CT subsystem is implemented as a dedicated run controller (RC) to be included as a leaf into a DAQ partition run control tree (see 13.2). The DDC-CT run controller, like any other run controller, is capable to execute standard commands causing its transitions as defined by the finite state machine. The collaboration diagram of this use case is drawn in fig.11.6.5a.



a) Handling run control command



b) Handling non-transition commands

Fig. 11.6.5 DDC-CT collaboration diagram

By DAQ Run Control we mean the part of the run control tree of a DAQ partition above the DDC-CT controller (parent tree).

The content of a ddc_command and its execution is the responsibility of DCS PVSS application. Mapping of the run control transitions onto certain set of commands on DCS is to be done according to DDC-CT configuration. That configuration defines the list of commands available for issuing for a certain PVSS system as well as the correspondence between the ddc_command set and the standard set of the run control commands (rc_command on fig.11.6.5a).

Another way of operating by DCS from the DAQ side is to be implemented as so-called “non-transition” commands (nt-commands). The latter term emphasizes that those commands do not cause any finite state machine transition, though being executed by the same DDC-CT application. An nt-command may be issued either by the parent run controller or by any other TDAQ application. The collaboration diagram for the use case of non-transition commands is presented in fig.11.6.5b.

It is assumed that normally the DCS application has been started earlier than all the others mentioned in fig.11.6.5. If a command has been issued while DCS is not yet running, an error response code will be returned for the command sender.

The DDC-CT application is to be implemented as a PVSS API manager [11.6_4], who inherits the rc_interface class of TDAQ run controllers and integrates the application program interface of DAQ information service to receive non-transition commands.

11.9 External Systems

To be summarized...

The term External Systems designates systems having their own control system with which the DCS has to interact. We can distinguish two main external systems: the CERN technical infrastructure and the LHC accelerator. The former consists of a number of subsystems like cooling and ventilation, electricity distribution, radiation monitoring, etc. The Detector Safety System (DSS) and the Magnet Control System are also considered part of the technical infrastructure. All these external systems must be concurrent into the general DCS. Although these systems are designed to react in case of problems, early indications of their status must be notified to the DCS since they may have consequences onto the detector and automatic corrective actions, driven by the DCS, may be required. The DCS will reflect also the states of all these systems and, in many cases, will act as their user interface.

The connection will support bidirectional information exchange and, in some cases, also the sending and receiving of commands. This interface will be unique for the 4 LHC experiments and it will be developed in the framework of the JCOP.

11.9.1 Technical Services

XXX This section can be moved to the Applications since these are not really external systems XXX

The technical services around the ATLAS detector include cryogenics and conventional cooling, ventilation, gas system, electricity, radiation monitoring, low and high voltage power supplies. The DCS will also have access to the control and status of infrastructure including AC mains, air conditioning etc. These services will monitor the environment to guarantee the safety of the personnel and equipment and will enable the different subdetectors of the experiment to function within their required operating conditions. Some of the systems will need feedback from the subdetector to operate. This is the case for the gas system and cooling, where part of their equipment consist of external stand-alone PLC or commercial I/O modules, whereas some information come from the detectors themselves. Therefore, slow closed-loops maybe needed between the DCS and this type of systems.

11.9.2 Environmental Infrastructure

Environmental parameters including humidity and atmospheric pressure in the cavern and at surface, the composition of the air in cavern (O2 levels etc.) will be monitored by the overall DCS and made available to the detectors. The temperature, for example, ranges from 4.5 K for the super-conducting magnets, to 88 K for the liquid argon calorimeters, to 253 K for the pixel and silicon parts of the inner detector to 293 K for the TRT, tile calorimeter and muon chambers. The state of these systems and early indications of problems must be presented to the operator. The DCS will handle, present and log this data. Automatic corrective actions must be taken by the DCS if required. Moreover, after the temporary stop of one of these systems, the DCS has to prepare the detector for the restart.

Radiation monitoring is an area where information from many sources will be used. The subdetectors themselves are sensitive radiation probes, but also dedicated sensors and information from the monitoring of the environment will be used.

11.9.3 Detector Safety System

As previously mentioned, the DCS is not responsible for the security of the personal nor for the ultimate safety of the equipment. The former is responsibility of the LHC-wide hazard detection systems, which will alert the fire brigade in case of severe problems such as gas leaks or fire, whereas the latter has to be guaranteed by hardware interlocks and stand-alone PLC and it is responsibility of the Detector Safety System. Although the information exchange between the DCS and the DSS must be bi-directional, actions must go only in one direction. The DCS must not disturb the operation of the safety system. However, warnings about problems detected by the safety system must be notified to the DCS in order to take corrective actions or to shut down the problematic part of the detector. Control access will also be handled by the CERN services and it will be needed at the DCS side.

11.9.4 Magnet system

The magnet system, described in the first chapter, represents one third of the total budget of the ATLAS detector. Although due to its critical requirements [10] and complication, a dedicated PLC-based control system will be implemented and the operator will not need direct control, a detailed online status and knowledge of all important parameters of the magnets, is essential for the operation of the detector and for the subsequent physics analysis. This dedicated system will supervise and control the cryogenics, the cooling system, the power supplies and the instrumentation of the magnet.

The toroid coil system has a 21 kA power supply and is equipped with control systems for fast and slow energy dumps. The central solenoid is energized by an 8 kA power supply. An adequate and proven quench protection system has been designed to safely dissipate the stored energies without overheating the coil windings.

The central solenoid is cooled by a refrigerator. In addition, the barrel toroid and the end-cap toroid, have cold helium pumps to guarantee appropriate cooling by a forced helium flow at 4.5 K. The cooling power is supplied by a central refrigeration plant located in the side cavern and the services are distributed among the four magnets.

A fieldbus will connect all instrumentation to the main control centre in the USA15 cavern. The magnet supervisory control system could be implemented using some standard tools of the general DCS, thus SCADA and general-purpose I/O modules. This would facilitate the integration of this subsystem within the overall DCS.

11.9.5 LHC

An robust interface between the experiment and the accelerator must be provided. Instantaneous beam parameters like the different types of background, beam position, individual bunch luminosities, observed in the detector must be transferred from the experiment to the accelerator for consequent tuning of the beam.

The experiment will also give all information on its status such as status of its magnets, in particular, the solenoid which acts directly on the beams, status of sensitive equipment like high voltage on the sub-detectors and other status signals as well as global status signals such as the operation state of the detector, setting up, etc. The DCS has to make sure that the detector is in an appropriate state (e.g. voltage settings) before LHC is allowed to inject particles.

ATLAS may need the possibility to request actions like a fast beam dump should the backgrounds become dangerous for the subdetectors or injection inhibit. This important feature has to be implemented by a fast interlock system.

On the other hand, machine parameters like status signals for setting up, shut-down, controlled access, stable beams, beam cleaning must be transferred from the accelerator to the experiment. The machine should also provide information on the beam like emittance, focusing parameters, energy, number of particles per bunch, a horizontal and vertical profile, needed for offline physics analysis. Information on the vacuum conditions in the vicinity of and in the experimental straight section and position of the collimator are also of interest to the experiment.

The LHC has dedicated instrumentation for the comprehensive measurement of all these parameters. The subset of operational parameters of the accelerator, relevant to the operation of the detector or to the subsequent physics analyses have to be delivered to the DCS and must be logged.

Although the exchange of many of these parameters is only needed during data-taking, a subset of this information, like the integrated radiation doses in the different parts of the detector measured by the DCS, has to be known to the LHC at all times. Therefore, this communication is required regardless the state of ATLAS. This is one main reason why this communication will be handled by the DCS on the ATLAS side and not by the DAQ system.

Similar functionality is currently required to interface the CERN Super Proton Synchrotron (SPS) during testbeam activities and commissioning of the LHC experiments. The solution currently adopted is presented in chapter 5.

More work is required concerning the interaction of accelerator and the experiment during operation which will have to be addressed in the coming years. All information exchange should be done with the same mechanism as used for the communication with the other external systems.

The CERN Services, LHC and Detector Safety System will be interfaced by means of the Data Interchange protocol to be provided by JCOP.

11.10 References

11-1 H.J. Burckhart et al., “Vertical Slice of the ATLAS Detector Control System”, submitted to 7th Workshop on Electronics for LHC Experiments, September 2001, Stockholm (Sweden).

11-2 F. Varela Rodriguez et al., “ELMB Full Branch Test: Behaviour and Performance”, ATLAS DCS Internal Working Note 13, October 2001.

11-3 F. Varela Rodriguez. “The Detector Control System of the ATLAS experiment: An application to the calibration of the modules of the Tile Hadron Calorimeter”, PhD. Thesis, CERN-THESIS-2002-035, April 2002.

11-4 http://www.kvaser.com

11-5 V. Filimonov, “Description of the CANopen OPC server v2.5”, http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DCS/ELMB/DOC/OPCCOUserGuide.pdf

11-6 References from Connection to DAQ:

11-7 H.Burckhart, M.Caprini, R.Jones “Connection DCS - DAQ in ATLAS”, ATLAS DCS IWN8, Nov 1999,

11-8 http://atlasinfo.cern.ch/ATLAS/GROUPS/DAQTRIG/DCS/dcs_daq_0.6.pdf.

11-9 R.Hart, V.Khomoutnikov “ATLAS DAQ - DCS Communication Software. User Requirements Document”, Nov 2000,

11-10 http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DCS/DDC/ddc_urd.pdf

11-11 <TDAQ Partitioning document> Probably, made already earlier at the document

11-12 <Reference to PVSS> Probably, made already earlier at the document

11-13

12 Interfaces

12.1 External to TDAQ

12.1.1 LHC machine

12.1.2 Detectors

12.1.3 Off-line

12.2 Internal to TDAQ

12.2.1 LVL1

12.2.2 ...

12.3 References

12-1

12-2

13 Experiment control

This chapter has been positioned after all other chapters in this part because it relies on information presented in the previous chapters.

+This chapter will describe how the different systems and building blocks of the overall controls architecture presented in chapter 5 and described in chapters 10 and 11 are used to provide the overall controls functionality required by the experiment in the following scenarios:

- *Physics data-taking,*
- *Calibrations, where only DAQ or DCS are involved or where both systems are required.*
- *Operations outside a run.*

+It is assumed that these scenarios have been introduced in chapter 3.

+The chapter will contain a description of the functionality required by the different systems, namely DF, HLT and the detectors.

+The operation of the Online Software System State machine and the DCS as Finite State Machines and their synchronisation will be addressed .

+The overall control system will provide the flexibility required to operate the subdetectors both in stand-alone mode and in an integrated mode for concurrent data taking. Scenarios will be presented.

+The overall coordination of the systems above and the LHC accelerator for Physics data-taking will be described.

+For Physics data taking the TDAQ control will act as the master while the DCS will act as a slave.

+The DCS has to operate continuously with no interruption.

+Special emphasis will be placed on the connection DAQ-DCS in the different scenarii mentioned above, where the connection points and the flow of data, messages and commands between both systems will be described.

+Some use cases on error handling in the different scenarii will be sketched.

13.1 Introduction

This chapter brings all the control elements together to show the overall control strategy and describe the mechanisms involved. The concepts presented here will have already been introduced in Chapter 5, "Architecture".

13.2 Control coordination

Explain the three different finite state machines present in the system, namely online, DCS, and machine, and their synchronisation.

13.3 Sub-system control

The information presented here might already be presented elsewhere in which case this section is not needed.

online software concepts

detector control

HLT farm supervision

DF control

13.4 Control scenarios

Control scenarios of:

- different types of calibration.
- physics run.
- operation outside a run.

13.5 References

Part 3

System Performance

14 Physics selection and HLT performance

14.1 Introduction

Recall the strategy (as in Section 4) and the inclusive approach (more details later on non-inclusive selections).

Explain the use of selection algorithms at different levels and the selection sequence

Highlight the use of updated detector geometry (also in start-up phase, *i.e.* staged, implementation) and (wherever possible!) the use of realistic data and communication schemes, the use of fully simulated data with proper pile-up

2x10^{33} $cm^{-2}s^{-}1$ (we need a FM variable here!) and L = 1.0×10^{34} $cm^{-2}s^{-1}$ different approaches

Find a clever way to explain how we will do bricolage when we cannot use the full-fledged schema to get byte stream, decode, apply HLT algorithms, derive features, take decision.

Don't do like CMS (because we can) and do not explain all the things that we already said in Physics TDR and HLT TP, but only reference them.

14.2 Common tools for selection

Describe the tools (algorithms) used at the different levels, with a focus on the LVL2 detector reconstruction (e.g. Calorimeter clustering, ID tracking, etc).

Highlight the approach of the Algorithm Task Force, the use of common tools for different selections. Do not forget that EF is "inherited" from off-line and explain how much of the full analysis chain is retained here.

Link also with description of PESA-SW, Steering, Data Access, etc.

14.3 Signatures, rates and efficiencies

Derive from Trigger Menus (of Section 4) list of representative physics signatures (*à la TP*).

14.3.1 e/gamma

Emphasis on this selection: most of explanations will be here.

14.3.2 Muon selection

Differences wrt TP, low-p_T signatures (see later for B), barrel approach, end-cap ?

14.3.3 Tau/jets/$E_T$miss

Highlight major discovery channels, for taus probably start using bricolage

14.3.4 b-tagging

Define on-line strategy for this, explain why we think we need it, discuss implications for jets thresholds (and hence rates)

14.3.5 B-physics

Agree on ATLAS policy. Explain strategy, start with di-muons selection, fill-in with other low-p_T signatures with decreasing instantaneous luminosity (some of this probably already in Section 4)

For each of the above, go through the list of signatures and derive numbers for rates and efficiencies (both HLT wrt LVL1 and between LVL2 and EF steps)

See also recent Saul's comments to include test-bed results in performance evaluation

14.4 Event rates and size to off-line

Define present ideas about data compression and reduction, zero suppression for LAr (and TRT?): this might be probably be elsewhere as well. Differences between zeros at the EF and loss-less data compression in the ROSes.

Global table on rates for initial and high luminosity, implication for off-line reconstruction (costing, later)

14.5 Start-up scenario

Should be here? Picture a global approach on how we are going to handle, at the selection level, the first year of running, assuming a certain machine scenario. It is probably very appealing for LHCC

14.6 References

14-1 *ATLAS detector and physics performance technical design report*, CERN–LHCC/99–14/15 (1999)

15 Overall system performance and validation

15.1 Introduction

- Definition of validation of rate capability, its context and scope.

- Summary of validation process

15.2 Integrated Prototypes

Description of the prototypes:

- HLT/PESA prototype
- integrated 10% system

15.2.1 System performance of event selection

NOTE: This section will (necessarily) be completed at a later time. The work described here is ongoing, and in some cases, not even started. The following sub-sections may be re-shuffled or modified to accommodate the evolving tests.

The High Level Trigger will select and classify events based on software largely developed in the offline environment. This approach minimizes duplication of effort and ensures consistency between the offline and the online event selections. However, given the strict performance requirements of a real-time online environment, it is essential to evaluate the performance of the HLT event selection software ("PESA software") in a realistic trigger environment.

The resource utilization characteristics of the PESA software are an important input to the models that predict overall system size and cost. For this reason, a prototyping program was developed to perform dedicated system performance measurements of the PESA software in a testbed environement. The following sections summarize the outcome of this measurement program.

Here need to be more specific as to what testbeds we are referring to. Differentiate with 10% testbed. Stress: limited scope to PESA sw and dataflow and HLT components necessary to run it.

15.2.1.1 Measurement and validation strategy

Here we need to describe the approach taken in the testbed work: First perform functional integration and then performance measurements. Refer to backup documents (to come).

Address robustness requirements (runs more frequently in online that in offline)?

Short description of testbeds used:

- LVL2
- EF
- integrated LVL2-EF

15.2.1.2 Event selection at LVL2

Describe software components used in event selection:

- PESA Steering Controller
- Steering, COnfiguration. Trigger Menus
- Trigger Algorithms: T2Calo, SiTree/IDSCAN
- Data unpacking: BS converters for LAr,Tile,Si/pixels

Describe hardware components in testbed:

- L2PU
- ROS emulator
- L2SV

Here describe system performance results for various components in different configurations, e.g., PSC overhead, framework overhead, algorithm usage of CPU resources, number of threads, etc. Only a few results shown in a table. The rest will be in a backup document.

Also give a sense of what sort of optimization can still be done in the software/strategy so that perfrormance can be brought to an acceptable level.

15.2.1.3 Event selection at the Event Filter

Same approach as LVL2 above.

The Event Filter will select and classify events using reconstruction algorithms developed by the offline community.

Include results (TABLE).

Comparison with stand-alone offline measurements

15.2.1.4 Testing of HLT

Here will try to treat LVL2/EF as one unit. Show successful integration of LVL2 and EF in a testbed. Briefly talk about benefits of LVL2-seeded Event filter and the use of the pROS.

15.2.2 The 10% prototype

Description of the integrated 10% system

15.2.2.1 Laboratory setup

- machines, networks, OS platform(s), hardware emulators (if any)

- refer to architecture and components chapters for details

15.2.2.2 Description of the measurements

- scope of the measurement (what parameter(s) of Chapter 2, "Parameters" are we testing)

- parameter space covered

15.2.2.3 Results

- prototype results

- comparison with required performance

15.3 Functional tests and testbeam

During prototyping phases, often the performance of a system is put in foreground with respect to its stability and maintainability. Functional user requirements have in this phase of development a lower priority than the achievement of the performance requirements. This is to some extend true also for the TDAQ system, which has focussed its efforts in the area of trigger rates, speed of data acquisition, etc. Nevertheless we have decided to also stress the global functionality of the TDAQ system, by carrying out a series of functional tests and exposing the system to non expert users at the ATLAS test beam sites.

Three different aspects of the functionality have been covered:

- a)Dynamic system configuration
- b)Stability in cycling through TDAQ finite states
- c)Operational monitoring and system recovery in case of errors

All these aspects have first been tested in dedicated laboratory setups and then verified in a "real" environment, during test beam data taking.

- a)A TDAQ system has to be easily reconfigurable in order to accommodate the substitution of hardware, the change of trigger conditions, etc. This means that on one side all the tools to keep the configuration parameters in a database have to be developed and on the other side that the Run Control, DataFlow and Trigger software has to be designed to be dynamically reconfigurable.

To verify the flexibility of our system the following tests have been carried out:

- -substitution of a data taking machine
- - exclusion and reinsertion of a Run Control branch
- -change of communication protocol between the ROS and the L2/EF (= change of Data Collection protocol)

- -change of run parameters.

More detailed test description and measurement results to be included here.

- b)When performing a series of measurements with different configuration options, the TDAQ system must be capable of cycling through its finite states stably. This functional requirement has been checked via automated scripts cycling repeatedly through the finite state machine.

More detailed test description and measurement results to be included here.

- c)In a distributed system such as the TDAQ it is important to constantly monitor the operation of the system. Furthermore, the fault tolerance is a fundamental aspect of its functionality. In this area several improvements are still to be achieved, but we decided to carry out a series of tests in order to assess the present performance of the system in case of errors. In particular we tried to test the fault tolerance of the system in the presence of a fatal error which prevents on or more data taking computers to continue their working.
  - Verification that all applications provide regular information on their status
  - -Failure of a SFO
  - -Failure of a EF subfarm (distributor or collector)
  - -Failure of a EF processing task
  - -Failure of a SFI
  - -Failure of a L2PU
  - -Failure of a DFM
  - -Failure of a L2SV
  - -Failure of the RoI builder
  - -Failure of a ROS
  - -Failure of a ROBIN
  - -Failure of a ROL
  -
  - -failure of online sw servers (is, mrs, ipc, ….)

More detailed test description and measurement results to be included here.

The results of the various tests will determine the summary and conclusions of this section. It is premature to indicate them now.

15.4 Model analysis of mechanism and avoidance of message loss

The availability of network connections and switches with sufficient bandwidth and of a sufficient amount of computing resources in the DAQ and HLT systems does not guarantee a satisfactory system performance. The reason is that congestion in switches may lead to message loss, if not enough buffer space is available. For gaining insight in how likely message loss is and

how it can be avoided a simplified model of the candidate architecture is studied in this section. Choices for the model parameters have been made on the basis of the parameter values presented in Chapter 2, "Parameters".

Simplified model properties:

- Ethernet network technology
- 800 dual ROBIns with one Fast Ethernet connection, each ROBIn outputting 1.5 kByte event fragments (i.e. 1 Ethernet frame per event fragment) via one network connection
- 2 kHz Event Building rate, 0.5 kHz LVL2 RoI rate per ROBIn (total bandwidth required for data: 5 kHz times 1.5 kByte = 7.5 MByte/s, about 63% of the max. bandwidth of the network link
- “Edge switches”: 48 Fast Ethernet ports connected to 48 dual ROBIns, 4 Gigabit uplinks, three links to EB switch, 1 link to LVL2 switch, fragment size on an uplink to EB switch is 144 kByte (96 frames) @ 2/3 kHz, fragment size on an uplink to the LVL2 switch is 6 kByte @ 12 kHz (assuming that on average 4 ROBIns output data for a single RoI),
- Total (rounded) number of uplinks: 4 * 800 / 48 = 68, 51 to EB switch (per link: 96 MByte/s) and 17 to LVL2 switch (per link 72 MByte/s),
- The LVL2 switch is a Gigabit Ethernet switch with 17 input and 17 output ports, the EB switch consists of three switches with 17 input and 17 output ports each,
- 102 L2PUs, input per processor 17 * 72 / 102 = 12 MByte/s, corresponding to 2000 fragments of 6 kByte per second, 6 LVL2 processors connected to 1 up link via switch,
- 102 SFIs, input and output 51 * 96 / 102 = 48 MByte/s. With an event size of 2.4 MByte the event rate per SFI is 20 Hz, 10 EF processors connected via switch to one SFI, input rate per EF processor: 4.8 MByte/s, event rate = 2 Hz,
- The switches are assumed to have infinite internal bandwidth and only output buffers, frames are lost if no slot is available in the output buffer. In total 160 slots are available in each output buffer.

If on all input ports of an“edge” switch, connected to the ROBIns, fragments arrive for the same destination at the same time and the output buffer is empty, fragment loss does not occur as there are more than 96 slots available for storage of the frames. Now all fragments for LVL2 have to go to a single LVL2 up link. If it is assumed that all ROBIns send their LVL2 date at the same time, then it takes 48 * 3 / 125 = 1.15 ms to output all data (raw transfer speed of Gigabit Ethernet assumed to be 125 MByte/s). So if the instantaneous output rate of LVL2 fragments per ROBIn is limited to 0.87 kHz no congestion causing message loss should occur. For the uplinks to the EB switch the same type of reasoning can be applied: per link the instantaneous rate of each ROBIn should be limited to 0.87 kHz (i.e. the maximum event building rate would be 3 * 0.87 kHz = 2.68 kHz). The maximum required bandwidth per Fast Ethernet link would be 10.66 MByte/s = 85.3% of the available 12.5 MByte/s

For the LVL2 switch 17 input links could deliver 17 frames in the same time interval for the same destination. At the end of the time interval still 16 frames will be buffered, as only one of the frames can be output and as all links have the same bandwidth. Although unlikely for the LVL2 trigger, for each input link 96 frames for the same destination could be sent immediately after each other via a single link. Therefore, after sending out the next frame 17 new frames could have been received and stored, etc... .This build-up of buffered frames will lead to a buffer overrun and therefore to discarding of frames. It is therefore necessary to steer the traffic such

that this situation will not or is unlikely to occur. A strategy which seems obvious is to assign successive events to L2PUs receiving data via different links from the LVL2 switch. However, this strategy in combination with the variability in algorithm execution times and the probabilistic nature of the data request patterns could have an undesirable impact on the load balancing in the LVL2 processor system. Therefore another measure is necessary, which consists of requiring that the number of outstanding requests is always smaller than or equal to a certain maximum. In the simplified model discussed here there are 6 L2PUs sharing a link from the LVL2 switch. With a maximum switch output buffer capacity of 160 frames the maximum would be about 25 outstanding requests per L2PU. However, the requests generated by the L2PUs also can cause buffer overflows in the central LVL2 switch. Although unlikely, all L2PUs could be sending their maximum of 25 outstanding requests to the same uplink at the same time. This will certainly result in message loss and can be prevented by either limiting the maximum number of outstanding requests for each uplink and for each L2PU to 1 or by limiting the instantaneous output rate per L2PU so that the amount of requests queued per uplink is always less than 160. If the size of a request message is assumed to be 64 Bytes and the bandwidth of an uplink is 125 MByte/s (Gigabit Ethernet) a request rate of 20 MHz per uplink can be supported. So if the instantaneous request rate per L2PU can be limited to 20 kHz the problem loss of request messages can be avoided. However, there is a caveat: all requests arriving in an "edge switch" could in principle go to the same destination, which is connected via a Fast Ethernet link. To be completely safe the 20 kHz has to be reduced to a maximum instantaneous rate less than 2 kHz per destination (a dual ROBIn). Less than 2 kHz, because also requests from the SFIs and deletes have to go through the same links between "edge switches" and ROBIns. Here it is assumed that the maximum instantaneous rate is1.5 kHz. So a L2PU could be required to output requests with an interval of 50 microseconds at minimum, per destination (dual ROBIn) the interval should be about 600 microseconds at minimum to be completely safe.

For the EB switches the situation is somewhat different from the LVL2 switch, as there is only one data request phase, and as the data request pattern is fixed. Therefore assigning successive events to SFIs receiving data from different links of the EB switches can be expected to help with avoiding congestion. The data request pattern generated by a SFI can also be used to control the traffic patterns. The most straightforward approach seems to consist again of limiting the maximum number of outstanding requests. A limit of 80 is suggested by the model used in this section, in the same way as for the L2PUs (in the model used there are two SFIs sharing a single Gigabit Ethernet link of one of the central EB switches). As a request is shorter than an event fragment and Gigabit Ethernet is bi-directional with equal bandwidth for both directions it can be expected that after a startup phase requests and data fragments will be transferred with about the same frequency. Multi-casting of requests to the ROBIns is not feasible due to the maximum of 80 outstanding requests (addressing groups smaller than 80 ROBins in a single multi-cast may be a possibility but it has to be taken into account that a too large number of simultaneously active multi-casts can also result in message loss). The requests generated by the SFIs also can cause buffer overflows in the central EB switches. The maximum instantaneous request rate per SFI should be 2 million / 34 = about 60 kHz (there are three EB switches, each with 34 SFIs and each connecting to 17 uplinks). With an event building rate of 2 kHz the rate of events per SFI would be about 20 Hz, so the average request rate is 20 times 1600 = 32 kHz, i.e. lower than the maximum of 60 kHz. As with the L2PUs, there will be again a problem when all requests go to the same dual ROBIn. For the requests from the L2PUs we have already reserved 75% of the capacity of the links. For three uplinks (to the EB switches) per "edge switch" therefore the instantaneous request rate of 34 SFIs (and for deletes, these are ignored for this discussion) per destination should not be more than can be transferred with a bandwidth of about 8% of the capacity of a Fast Ethernet link. This corresponds to a request rate of 0.08 * 12.5 / 64 MHz = 16 kHz, i.e. about 0.5 kHz per SFI. So the interval between successive requests per destination

should be larger than 2 ms. If requests are always sent out in the same order and with a maximum rate of 60 kHz the maximum instantaneous rate for a given destination is 2 times 60,000 / 800 = 150 Hz (dual ROBIns and assuming two requests per dual ROBIn), well below the maximum of 500 Hz (but the two requests for each dual ROBIn must not be sent within 2 ms).

Delete commands, not taken into account so far, may be multi-casted to the ROBIns when this type of traffic does not lead to responses from the ROBIns: the delete commands will not cause congestion, as they are fanned out, but responses sent out at the same time definitely will lead to buffer overflows as they all will be sent to the same destination.

Conclusion: congestion in the switches of the candidate architecture can be avoided by limiting the maximum instantaneous fragment output rates of the ROBIns and by limiting the maximum number of outstanding requests in the LVL2 processors and in the SFIs. Values for these maxima need to be determined from the switch configuration, the switch properties and average frame rates in the system, in the same way as done in this section for the simplified model. However, it is not clear that limiting the instantaneous rates is technically possible. Intelligent event assignment schemes may be possible and could prove to be attractive.

15.5 Computer model

- Discrete event simulation
- Object oriented model of system, most objects represent hardware, software or data items
- Two tools: at2sim, based on Ptolemy, and Simdaq, dedicated C++ program
- Testbed models and models of full system.
- For the full system model the same LVL1 trigger menus as for paper model are used to generate an appropriate number and type of RoIs for each event. As in the paper model, the eta and phi coordinates of the RoIs are chosen at random from the possible eta, phi coordinates (as defined by the LVL1 trigger). The mapping of the detector on the ROBIns is the same as for the paper model. Average message rates and volumes and total CPU power utilized as obtained from the paper and the computer model of the full system therefore should be equal within the statistical errors.
- Component models described in detail in back-up document

15.5.1 Result of testbed model

LVL2 Subsystem test, EF subsystem test, Minimal DataFlow test, larger setups ..

Type of results: model and experimental results for throughput, maximum message rate, latency ...

15.5.2 Results of extrapolation of testbed model and identification of problem areas

Full model

Type of results: latency, queuing in system, effectiveness of limiting output rates of ROBIns and of number of outstanding requests in L2PUs and SFIs, effect of different strategies for event assignment to L2PUs and to SFIs.

15.6 Title?

15.6.1 Technology tracking up to LHC turn-on

15.6.1.1 Network technology

15.6.1.2 Processors

15.6.2 Survey of non-ATLAS solutions

(a reality-check on ATLAS approach?)

15.6.3 Implication of staging scenarios

Re-interpretation of performance numbers for staging scenarios

15.6.4 Areas of concern

15.7 Conclusions

15.8 References

15-1

15-2

Part 4

Organisation and Plan

16 Quality Assurance and Development Process

16.1 Quality Assurance in TDAQ

Quality assurance during the production of hardware and software systems is provided for with the adoption of a development framework for DAQ components. The development framework consists of distinct development phases. At the end of each phase a set of deliverables is provided. This framework is complemented by guidelines, checklists and standards, internal reviews, templates, development and testing tools and coding standards. Those are being adopted as common working practice and help for error removal and error prevention in the system.

A TDAQ wide body, the Connect Forum [16-1] assists in coordinating development process activities and quality assurance methodologies across Atlas TDAQ/DCS. It also provides advice, especially via the recommendations and information made available through these Web pages which reflect the dynamic nature of the activity.

A common approach to the development via the use of rules, in-house standards and document templates helps in building a project culture. Those rules as well as the development phases themselves are not enforced but rather ment to be a help for developers. Emphasis on the various phases will vary and evolve with the life of the project. During event production for example, the emphasis will be put on maintenance and regular automized validation testing

A powerful release management system and a convenient working environment provide the necessary technical working basis.

16.2 The Development Process

The software development process (SDP) in Atlas TDAQ provides the structure and the sequence of activities required for development. A basic framework is provided to guide developers through the steps needed during the development of a component or a system. Continual review and modification of the SDP provides it with the flexibility to adapt to the evolution of the components and systems.

Many of the recommended approaches in the SDP are also applicable to the development of hardware components or sub-systems involving both software and hardware. The SDP consists of the following phases as shown in Figures 16-1: Brainstorming, Requirements, Architecture and Design, Implementation, Testing, Maintenance, complemented by reviews. Emphasis on the phases will evolve within time.

16.2.1 Inspection and Review

Written material including documents and code are subjected to a process of inspection and review at each step from Requirements to Implementation, in the SDP. Inspection is essentially a quality improvement process used to detect defects testing. The inspection process in the Atlas TDAQ project is based on Tom Gilb's Software Inspection method [16-2]. An important feature

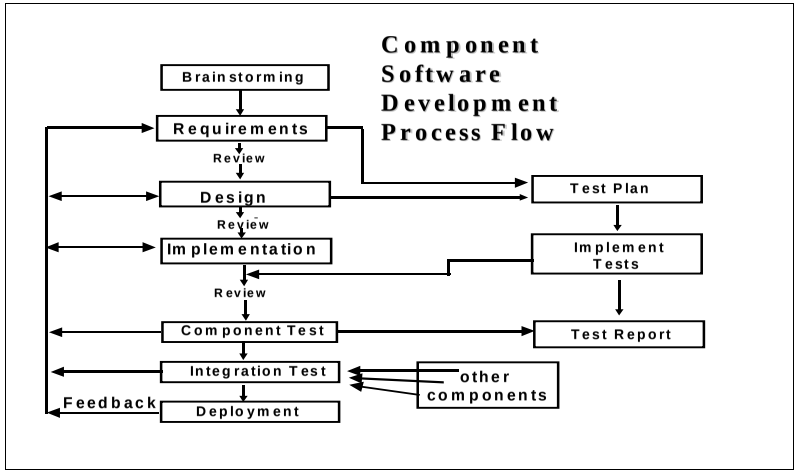


Table 16-1 Phases and flow of the Software Development Process

of the inspection procedure is its flexibility, allowing it to evolve as needs change during the lifetime of the project.

Overall responsibility for an inspection is taken by an inspection leader who appoints an inspection team consisting of the document author and three to five inspectors. The core of the inspection process is the checking phase where the inspectors read the document in detail, comparing it against source documents and lists of rules and standards. Defects are logged in a table, where a defect is defined as a violation of any of the standards. Emphasis is placed on finding major defects which could seriously compromise the final product. The defects are discussed at a logging meeting and their acceptance or rejection is recorded in an inspection issue log. The document author edits the document according to the log making an explanatory note if an issue is rejected. Feedback is also obtained on how the inspection procedure itself may be improved.

The principal aim of inspection is to detect and correct major defects in a product. An additional benefit is the possibility to prevent defects in future products by learning from the defects found during inspection procedures. Inspection also provides on-the-job education to people new to a project and generally improves the project's working culture.

A number of web pages have been produced which provide supporting material for inspections such as instructions for inspectors and log file templates[16-3].

16.2.2 Experience

The Software Development Process provides a disciplined approach to producing, testing and maintaining the various software systems required by the ATLAS TDAQ project. It helps to ensure the production of high quality software which meets the requirements within a predictable schedule.

However, one of the key differences in adopting the SDP in an HEP as opposed to industrial environment is that its application cannot be enforced. Furthermore, the use of such a process may appear too rigid to physicists not accustomed to working in a strong management framework. Nonetheless, the working culture can be changed by increasing awareness of the benefits of the SDP through training, for example involving new group members in inspections, and ensuring that the SDP itself is sufficiently flexible to evolve with the changing needs of an HEP experiment. This approach is working. The SDP as outlined in this section has already been adopted by a number of the sub-systems in the ATLAS TDAQ project with positive outcomes [refs].

16.2.3 The Development Phases

16.2.3.1 Requirements

The Requirements phase for a particular sub-system or component consists of gathering the requirements and then documenting them. Several documents have been produced to aid and control these activities, based on the early experience of some of the sub-systems. The whole process of working group setup, requirements collection, feedback & review is described [16-4]. Another document [16-5] sets out the principles governing the requirements gathering and documentation processes, stressing the importance of, for example, documentation, evolutionary development, communication, and collective ownership of the requirements specification.

The actual process of establishing the requirements for a sub-system or component is aided by a collection of "hints" [16-6], and reinforced by a set of 22 rules [16-7] for the requirements document itself, for which a template [16-8] has been provided in each of the supported documentation formats.

16.2.3.2 Architecture and Design

The Architectural Analysis and Design Phase of the SDP follows the Requirements phase and takes as its starting points the User Requirements & Use Cases together with accompanying documents. This phase has sometimes been referred to as "high-level system design". A system's architecture is the highest level concept of that system in its environment. It refers to the organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces. A design presents a model which is an abstraction of the system to be designed. The step from a real world system to abstraction is analysis. A 'Howto' note [16-9] has been produced describing the overall process.

For this phase, we are largely following the approach of the Rational Unified Process (RUP), which contains descriptions of concepts, artifacts, guidelines, examples and templates. In particular, we have highlighted the RUP descriptions of architectural analysis and design concepts [16-10] and guidelines for producing software architecture and design documents[16-11].

We have adapted the RUP template for architecture and design documents by including explanations and making it available in supported formats[16-12]. The recommended notation is the Unified Modelling Language (UML), and the design is presented in the template as a set of UML-style views. We have also prepared recipes for producing appropriate diagrams and incorporating them into documents.

16.2.3.3 Implementation

The Implementation Phase of the SDP is largely concerned with writing and checking code. At the end of the implementation phase a Software Inspection is performed.

ATLAS C++ coding conventions[16-13] are being applied to newly written code and being introduced for existing code still in evolution. In the case of Java we await the outcome of the Atlas investigation of coding conventions. DCS will follow the coding standards provided by the JCOP Framework for PVSS [16-14].

Guidelines [16-15] have been provided for multi-user multi-platform scripting, as well as many explanations and examples in unix-scripting[16-16].

Experience has been gathered with a number of software tools and recommendations have been made in the areas of design and documentation [16-17], code checking[16-18], and source code management[16-19].

16.2.3.4 Component Testing and Integration Testing

Testing occurs during the entire life-time of a component, group of components or entire system. Referring to figure [SDP], the initial test plan is written during the requirements and design phases of the component, so as not to be biassed by the implementation. Since testing is likely to be an iterative process the test plan is written with re-use in mind. Once implementation is complete and the code passes checking tools the component undergoes unit testing to verify its functionality. Compatibility with other components is verified with integration tests. Several types of tests can be envisaged for both individual components and groups of components. These include functionality, scalability, performance, fault tolerance and regression tests.

A test report is written once each test is complete. To aid the testing procedure, templates [16-20] are provided for both the test plan and test report in each of the supported documentation formats. More detailed descriptions of the types of test, hints on testing and recommended testing tools are also provided [16-21]. Testing is repeated at many points during the life-time of a component for example at each new release of the component software or after a period of inactivity (system shutdown). Automatic testing and diagnostic procedures to verify the component before use greatly improve efficiency.

16.2.3.5 Maintenance

As with testing, maintenance occurs during the entire life-time of a component. Several types of maintenance can be envisaged. Corrective maintenance involves the fixing of bugs. Adaptive maintenance involves alterations to the software in order to support changes in the technical environment such as different operating systems. Preventative maintenance entails the restructuring and rewriting of code for future ease of maintenance. Maintenance is closely coupled to regression testing which should occur each time a maintenance action has been completed to verify that the detected problems have been fixed and new defects have not been introduced. Significant changes to the functionality of the component such as the addition of large numbers of new requirements should involve a full re-iteration of the SDP cycle.

16.2.4 The Development Environment

Regular releases of the sub-system software to be used in test beam operation, system integration and and large scale tests is being complemented by nightly builds and automated tests to ensure early problem finding of newly developed or enhanced products. The use of a source code management system and of the standard release building tool CMT [16-19] allows for the building of common releases of the TDAQ system. These releases are available for the platforms used in Atlas TDAQ which are currently a number of Linux versions and for some sub-systems LynxOS and SunOS. Build policies of different sub-system like the use of compiler versions and platforms are coordinated.

Development tools like design tools, memory leak checking tools, automatic document production tools and code checking tools are vital elements of the development environment.

16.3 Quality Assurance During Deployment

16.3.1 Quality Assurance of operations during data taking times

The quality of the DAQ system must be assured when it is in use during the setup and installation phase of the Atlas data acquisition together with the detectors. Correct and smooth data taking shall be aimed for during calibration and physics event production.

Quality assurance is achieved by prevention, monitoring and fault tolerance.

- prevention: this includes training, appropriate documentation, a well defined software development process, proper management of computing infrastructure (computer farms, readout electronics and networks), tracing of hardware and software changes, regular testing of components.
- monitoring: special tasks to monitor proper functioning of equipment and data integrity. These may run as special processes or be part of the TDAQ applications. Anomalies are reported, analysed by human / artificial intelligence and appropriate recovery action is inititated. This may include running special diagnostic code, replacement of faulty equipment, rebooting of processors, restarting of applications, re-establishing network connections, re-configuration to continue with a possibly reduced system. Incomplete or corrupted data should be marked in the event data stream and possibly recorded in the conditions database. Physics monitoring may lead to a change of run with different trigger conditions and event selection algorithms.

fault tolerance: built into the system from the start and using an efficient error reporting, analysis and recovery system this provides the basis (cf. chap.6 for details). Some redundancy to reduce possible single point of failures is foreseen where affordable (cf. chap. 6).

During the life of the experiment small or major pieces of hardware or software will need to be replaced with more modern technology ones. The component structure with the well defined functionality of each component and well defined interfaces allowing for black-box testing according to those functionality specifications will allow to incorporate smoothly new parts into a running system, in particular also when staging of the system is required.

16.4 References

16-1 http://atlas-connect-forum.web.cern.ch/Atlas-connect-forum/

16-2 reference to Tom Gilb's inspection method

16-3 TDAQ inspection web pages

16-4 Practical Steps towards an Atlas TDAQ Requirements document.

16-5 Requirements gathering and documentation "principles".

16-6 Hints on how to establish requirements.

16-7 Requirements Document Rules ATLAS DAQ 'in-house' rules for Requirements documents. ID: ATD-R-R1.

16-8 Requirements Document Template for Systems and Components, Software and Hardware.

16-9 How-to for Design

16-10 RUP URL for concepts.

16-11 RUP URL for guidelines.

16-12 Template for Software Architecture Document.

16-13 ATLAS C++ Coding Standard Specifiation The coding conventions.

16-14 JCOP Framework for PVSS.

16-15 Multi-user multi-platform scripting guidelines.

16-16 Unix-scripting examples and explanations.

16-17 Doxygen, Visual Thought, Together, DOC++, Source Navigator

16-18 RuleChecker

16-19 CVS, SRT, CMT

16-20 reference to templates on web page

16-21 reference to testing web page

open points:

- HW inventory and information logging

17 Costing

17.1 Initial system

17.2 Final system

17.3 Deferral plan

17.4 References

17-1

17-2

18 Organization and resources

Should the geographical, racks, power supplies, and cooling issues be addresses in this chapter or in the system component ones?

18.1 ...

18.2 References

18-1

18-2

19 Work-plan

Post TDR.

19.1 Schedule

19.2 Commissioning

19.2.1 TDAQ

19.2.2 Tools for detectors

19.3 References

19-1

19-2

ATLAS
High-Level Triggers, DAQ and DCS
Technical Design Report
30 June 2003

This document has been prepared with Release 5.5 of the Adobe FrameMaker® Technical Publishing System using the Technical Design Report template prepared by Mario Ruggier of the Information and Programming Techniques Group, ECP Division, CERN, according to requirements from the ATLAS collaboration.

To facilitate multiple author editing and electronic distribution of documents, only widely available fonts have been used. The principal ones are:

Running text:	Palatino 10.5 point on 13 point line spacing
Chapter headings:	Helvetica Bold 18 point
2nd, 3rd and 4th level headings:	Helvetica Bold 14, 12 and 10 point respectively
Figure and table captions:	Helvetica 9 point