ALEPH 87 - 47
TPC 87 - 6

TPP SOFTWARE OUTLINE

S. R. Amendolia, F. Fidecaro, S. Galeotti, A. Lusiani, B.Lofstedt,
P.S. Marrocchesi, D. Passuello.

1. Hardware configuration

The TPC read-out includes 72 TPPs (ref. [1],[2],[3]).
Each one of the 36 TPC sectors consists of a maximum of 24 TPD modules (housed in 3 FastBus crates) and is controlled by two TPPs : one for wires and one for pads read-out.
The single TPP hardware [4] consists of (hardware compatible options in brackets) :

- a Motorola 68020 microprocessor + floating point co-processor MC68881
- 512 kbytes of EPROM (pin compatible up to 8 Mbytes)
- 512 kbytes of fast RAM memory (pin compatible up to 2 Mbytes)
- a FastBus master port to the TPD crates
- a FastBus slave port on the cable segment to the TPC Event Builder
- an interface to the Cheapernet local area network
- an interface to the Trigger Supervisor's signals
- two serial ports for possible terminal and host line connections

No disk storage is available on the TPP .

The on line computers are VAXes configured as a VAX cluster. These have appropriate memory and peripherals. A local Ethernet provides connections for terminals, to the CERN backbone network and to the TPC Cheapernet.

2. TPP activities

The TPC read out relies on a set of TPP modules to coordinate the acquisition of data in a well ordered way. The high processing capabilities of the TPPs allow them to perform a number of tasks, which are briefly outlined below.

2.1 TPP activities during data taking

The TPPs have to read out and format the TPC data present in the digitizers at each event.
In addition, they have to analyze these data to monitor the detector's behaviour.

The **data taking** activity is started at the start of run and terminates at the end of run, in the framework of a partition of Aleph.
In principle all TPPs in a given partition should execute the same data taking sequence.
However, the operations performed during data taking can be different according to the type of run (e^+e^- collisions, laser calibrations, cosmic ray trigger, ...).
The possibility to accomodate local variations to the data acquisition scheme has to be foreseen from the very beginning, since this feature may occasionally be required in order to cope with noisy channels, inefficient areas in the chamber, etc.

Monitoring the detector's behaviour mainly consists of accumulating distributions of directly measured or derived quantities. One or more samples of data taken in homogeneous conditions are used for this statistical analysis. There are also cases when a single event can provide evidence for detector's faults, for instance when one checks the consistency of the data coming from the digitizers or from the formatting process.
In normal conditions, monitoring processes running simultaneously in all TPPs can detect the most common faults. However, detailed diagnosis in a bad sector may require the use of specialized programs to be run without influencing the work of the TPPs of other sectors. Specifically, basic and specialized programs have to be restarted many times to verify that a fault has been eliminated.

Reports on the data taking activity and on the results of the monitoring processes have to be stored for future treatement and for reference in some mass store, provided by the computers of the data acquisition system.

2.2 TPP tasks outside data taking

One or more TPPs can be operated in a stand alone way, that is **outside** an Aleph partition :
the TPPs implement the facility of dealing with local dedicated trigger input, internally performing some of the synchronization functions usually made by the Local Trigger Supervisor and exchanging data with control processes running on the VAX.
This would happen when running in or debugging a TPC sector or when the electronics calibrations are performed.

The process of **debugging a faulty sector** requires that a person runs several TPP programs that verify the detector. During the course of repair the use of available services on the VAX may be required like the interrogation of the Aleph data base (if for instance hardware characteristics of each module are stored) or the application of graphical methods to ease the diagnostic process. There are also operational difficulties: when replacing modules in a Fastbus crate, one could have to turn off the power supplied to the TPP, thus losing data in the TPP's memory.

The **electronics calibration** in each sector is executed by a special TPP program which is downloaded only when a calibration run is going to take place.
It consists of adjusting the fast ADC gains and pedestals to provide an uniform electronics response to the pulses from the sector's wires and pads. The latter are simulated by a calibrated pulsing system. The results of the calibration procedure are for each channel the settings of the four DACs , the measured noise and the threshold setting. These results are stored in the mass store of the on line computers to be used as reference, for fast reload and for channel history.
The calibration run is started from the VAX by the **start of run process** and the sequence of calibrations in the different sectors is coordinated by the VAX.

2.3 Exceptions handling and error messages

Elements of the TPC read-out have to report several kinds of events. The latter can be seen as messages sent to appropriate handlers (i.e.: to specialized routines built in the software and to people on shift).
These events can be trapped in different places : the first one being the operating system of the TPPs which traps fatal and non fatal errors in the programs that are running at this level. Depending on what happens, this kind of event can have far reaching consequences: it may, for instance, require the run to be stopped if the TPP data acquisition program crashes, or be a simple warning to the author of the monitoring process that his code has no protection against division by zero.
Other events that should be signalled are the completion of an assigned sampling of events, or the warning from a monitoring process that data are not as good as they should be.

The number of objects involved makes the task of handling these circumstances difficult, the problem being the amount of information passed to the person on shift or stored for later analysis by an expert.

The handling of messages and exceptions by the read-out system determines the way the TPC read out appears to the person on shift and is the main source of information available to the experts for " trouble shooting " . Therefore a careful design is important to ensure good conditions of data taking during the Aleph shifts.

2.4 Control of the TPPs

The complexity introduced by the use of such a large number of TPPs requires appropriate tools .

Redundancy in the possibility to " **reboot** " the processor from EPROM is ensured by 3 independent access paths : the Cheapernet, the Fastbus and a front panel RESET input.

The large storage capacity of the non-volatile memory allows the TPP to have powerful self-test capabilities.

These are triggered at 'power on' or when a 'reset' command is issued from the VAX. The result of the self test is made available to the VAX via FastBus or Cheapernet. In addition, an hexadecimal display on the TPP front panel provides a well visible status code.

The possibility to access the memory mapped space of the TPP from the VAX, allows specialized programs to further check the functionality of the module and to test its data acquisition capabilities by simulating , in stand alone mode, a whole read out sequence from the relevant TPD modules.

The result of such tests may well end up in the decision to replace the faulty module.

More elaborate diagnostic tools are then available separately in a dedicated setup to repair the module.

During data acquisition, the **synchronization** and **trigger control** of the TPP modules is performed , in hardware , by the Trigger Supervisor. However, the TPP is responsible for the correct allocation of the 4 front end buffers, as well as of the two TPP output buffers containing the formatted data .

The TPP performances during the data acquisition can be monitored by including the relevant control information into the TPP formatted data, on an event by event basis.

' **Spying** ' the TPC sector's data acquisition is then performed by one of the monitoring processes running in the TPP which conveys to the VAX information relative to the status of the data acquisition in that sector (e.g : buffers occupancy, event size, frequency of time-outs , ...) . The consistency of the formatted data is also checked and reported.

The 'active' control of the TPPs during data acquisition is mainly achieved by means of ' commands ' issued from the VAX (see 4.1 and 4.2) : they can affect the state of any single process running in the TPP, as well as that of the module as a whole (the 'reset' command is an example) .

2.5 TPPs and the VAX

The multiprocessor configuration of the TPC read out with one large host leads to several constraints in the design of the TPP software.
As an example, data that can be updated by several processors should be centralized, or sensible data should not reside in the TPP memory, which is less protected than the VAX one.
Non time critical operation like command interpretation and validation should also be performed by the TPC VAX.
On the other hand, the data acquisition is performed by the TPP interface with the Fastbus (the VAX has no direct access to the front end modules) therefore **local monitoring** of each sector can be performed directly by the two TPPs that control that sector, **in parallel** with all the remaining sectors and taking advantage of the full processing power of all the TPPs.
Information specific to the global TPC event (or at least that part of the event belonging to a partition) can be monitored at sub-detector level (for instance, by the VAX) or at partition level (for instance, by the Event Builder) .
The TPC VAX has enough computing power and disk capacity to support several activities that would otherwise take place in the TPPs.

It supports the human interface between the TPC read out and the people on shift .
This may include the recognition of valid " **commands** " directed to all the TPPs belonging to a given environment (for instance, the start of run command to all TPPs inside one partition).
In addition, parameters relative to one task running in one (or more) TPP(s) may be changed by the user on the VAX.
This may require the management of several windows on the VAX and the definition of a " communication protocol " for the exchange of commands and parameters (see 4.1)

Usually no good facility is available in the TPPs to display the collected information. This has to be done by the TPC VAX.
The analysis of the monitoring results should be automated as much as possible. This requires, on the VAX , facilities for recognition of abnormal distributions, comparision with reference histograms, and semiautomatic feed back to the TPC read out system.

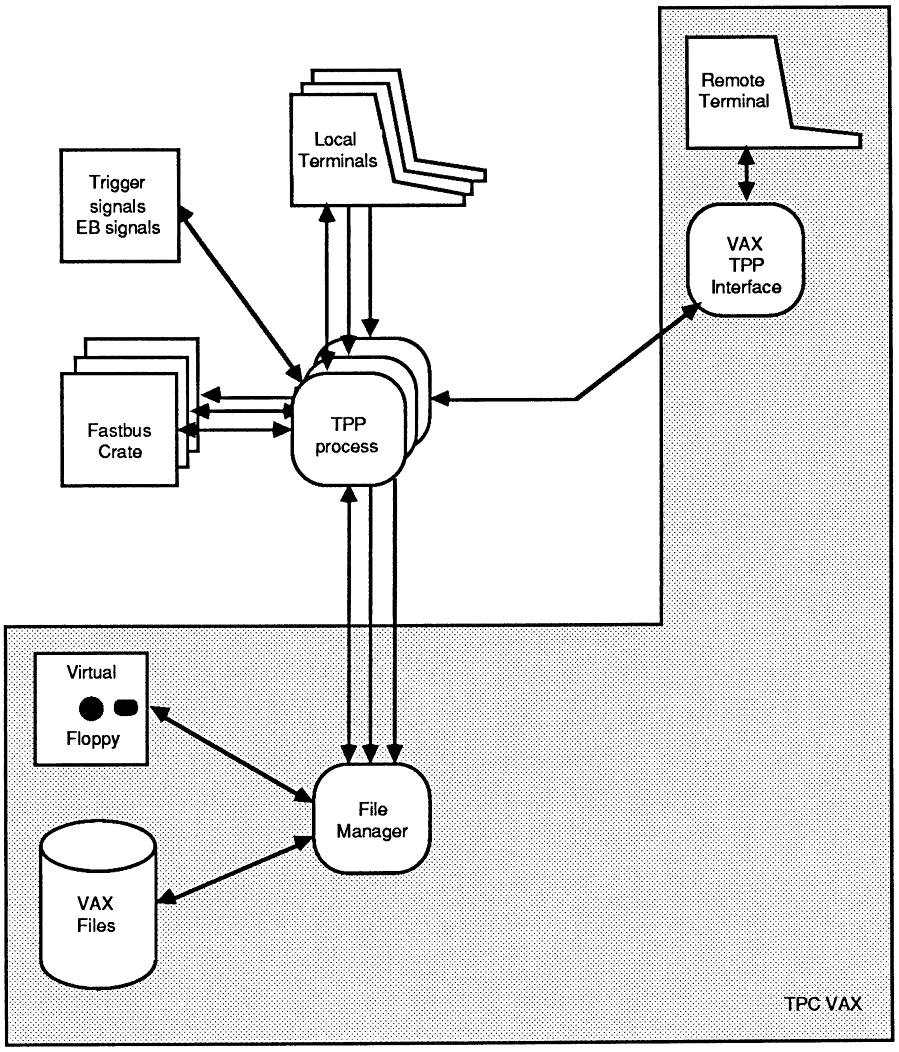
3. Examples of uses of the TPP and detailed software needs

We will now describe the various situations the TPC read out system should be able to cope with.

The simplest one is **standalone sector debugging** by a user from a terminal.
In order of increasing complexity, the running of the **data acquisition** program follows together with some monitoring programs in the framework of a partion of Aleph.
The **start of run** procedure and the **communication** between the user and several processes during a run are then analyzed .

3.1 Standalone TPP

In this situation, the sector where the TPP resides is outside an Aleph partition. The TPP acts independently from the data acquisition, under the control of a user connected to it via a terminal. The situation is illustrated below :



The terminal can be either connected directly to the serial port on the TPP or be any VAX terminal . In the latter case one needs a **remote console** process on the VAX that selects a particular TPP, checking its availability, and ensures communication between the terminal and the TPP via the network interface .
One could also imagine to have a window on a workstation instead of a terminal.
When running in **standalone mode** the TPP itself should be able to display menus of monitoring tasks to the user allowing for a full interactive session in order to debug the hardware quickly .
By loading into the TPP memory an appropriate user interface layer capable of driving the user's terminal, one should be able to decouple the work on the TPP from the VAX . This may turn out to be extremely useful during the period of assembly of the TPC above ground, when different teams will be able to **work in parallel** on different TPC sectors, without additional load on the VAX CPU .
Along the same line, it would be useful to have some **histogramming and graphics tools** running directly in the TPP to present monitoring information in graphical form .

As far as **data storage** is concerned, since the TPP has no direct disk storage, one uses the communication software to provide two kind of files :

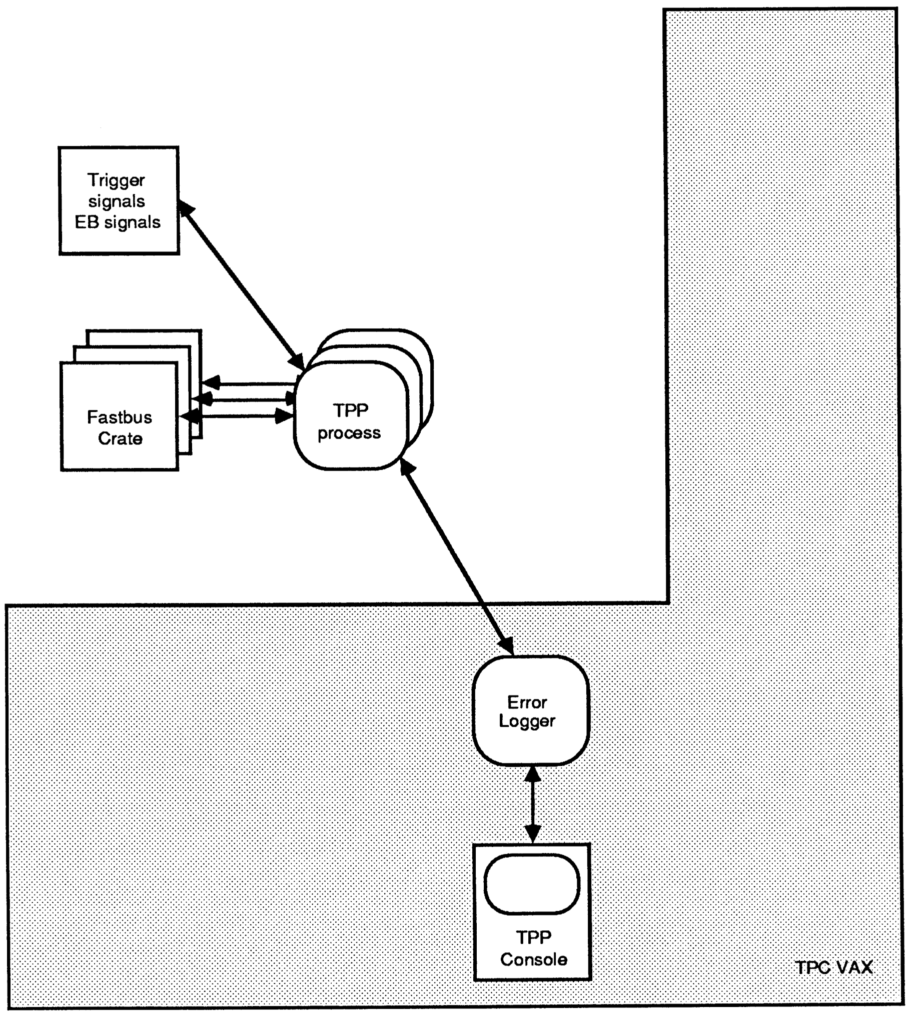
- **native files** that are only relevant to the TPP and its operating system (source files, binary images, ...) .
- **data files** that are to be shared with the TPC VAX (calibration files, setup files, ...) .

Access to the native files is made via TPP operating system commands issued from the TPP console or by OPEN, READ / WRITE, CLOSE procedures from the TPP tasks.
Access to the VAX data files is provided by the **file manager** as well as resource management (e.g.: validation of write access from a TPP into a VAX file) .

3.2 Running the data acquisition

The **data acquisition program** runs in a TPP that belongs to a partition. It is the only process in the TPP that should access the Fastbus and the trigger signal registers.
In principle no interaction with the VAX is needed, because the process reacts to the trigger signals and follows the communication protocol with the Event Builder.
However, one has to consider the case where the data acquisition program finds some anomalies and wants to report them. This is the case of a message that has to be sent to the VAX.

A more critical case is when some trap, either fatal or non fatal is detected by the TPP operating system. Information of such an event has to be conveyed to the user : it is an exception to the normal running of the program. The latter situation is illustrated below.



The difference between **message** and **exception** is that a message is issued under control of a process, while the exception is not.
Messages will also convey urgent monitoring information. One has then to consider the fact that many TPPs run together.

As in the standalone case, each TPP has access to files, Fastbus and trigger signals, but now any process running in it must be able to connect to a **system console** to log exceptions via a **message and exception logger.**
In addition to message logging, the latter process should eleborate the available message information to produce warnings and early diagnosis to the people on shift .

3.3 Starting a run

In order to start a run the TPP has to perform several operations:

- execute a **startup procedure** (if required)
- perform **calibrations** (if required)
- start the **data acquisition program** (the program itself sends the appropriate signals to the trigger supervisor when ready) .
- begin the **monitoring** activities.

These steps imply the loading and running of several processes in the TPP.
Let us consider a few examples appropriate to the TPC environment :

(i) <u>The startup procedure</u>

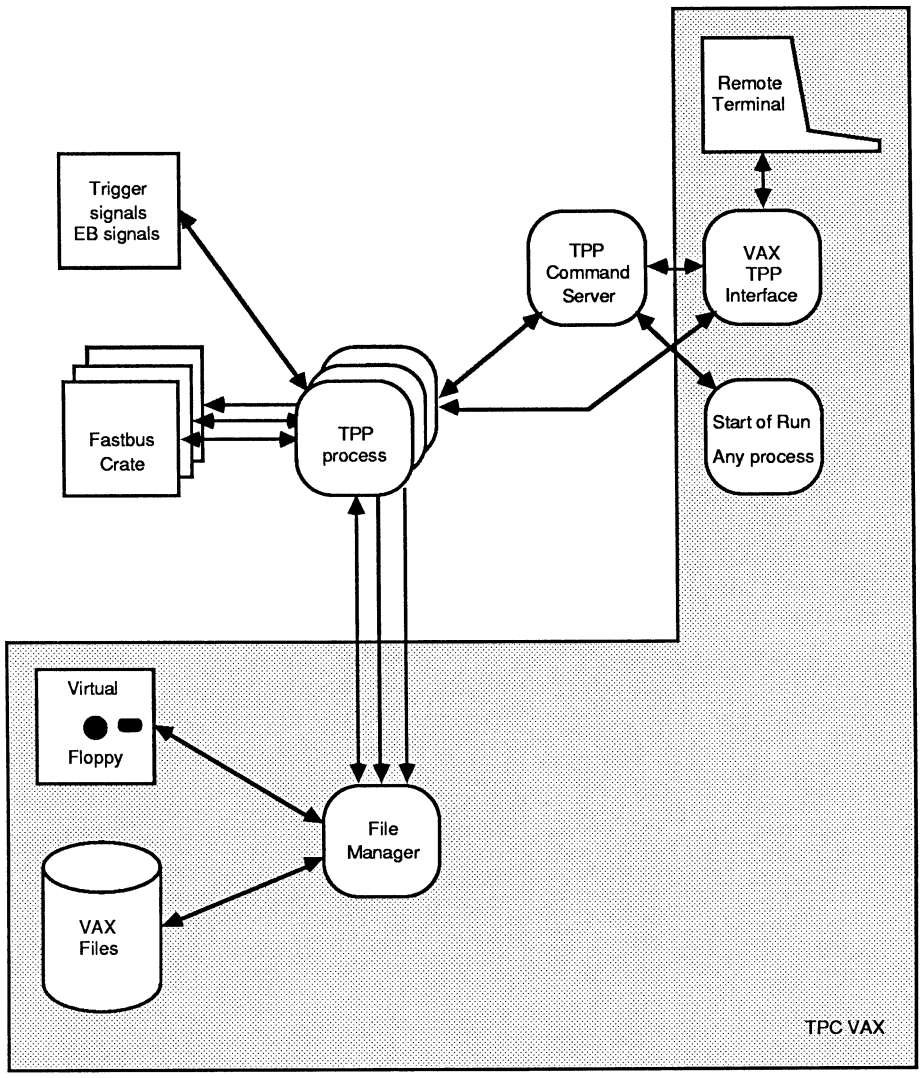
The **startup procedure** consists of a number of steps, each step being executed only if required.
The TPC read-out initialization includes a FastBus initialization phase which may consist of :

- a partition initialization
- TPPs initialization
- TPDs initialization

Each phase involves a number of FastBus actions to determine and set the status of the relevant hardware.
The first phase may involve the necessary operations to configure the partition (e.g.: load the SI table, initialize the EB , etc...).
The second one may consist of a series of operations to ' check ' the funcionalities of the TPPs in the system or to ' reboot ' them.
The third phase may consist of a scan of the front-end crates to find out which TPD modules are responding and reload them if necessary .
The latter phase cannot be carried out by the VAX directly, but needs the TPPs to access the front-end modules . This implies the loading of a special ' setup ' program in the TPPs .
The configurations required in each TPP are sometimes different, because of special monitoring needs like check of dying channels, noisy sectors, etc.

(ii) <u>The start of a run</u>

The **start of run** is achieved by starting the data acquisition task and the default monitoring tasks in all TPPs belonging to the partition .
The commands to **start, stop** or **pause** a run are issued by the **VAX start of run process** .
The communication interface on the VAX should be able to send such commands to a **command server** task resident in each TPP (see 4.1) .
The latter task controls the run activities within the TPP by means of system directives and intertask communication provided by the operating system running in the TPP .
The command server is always resident in memory as well as the system kernel .
The configuration at the start of a run in a partition looks as follows :



(iii) The data acquisition program

Only a brief description will follow : more details are to be found in [1] and [5] .
A finite state machine implementation of the DAQ program is depicted in Appendix A.1 using the Petri Net approach [7] .

The program is interrupt driven by Level 2 Yes triggers .
The trigger interrupt routine has to allocate one front-end memory bank of the TPDs for the next trigger to come. This is done by mantaining a circular buffer of pointers to the 4 front-end memory banks .
Allocation is performed by the TPP by writing into the TPDs CSR space via FastBus .
The TPP **busy** signal to the Trigger Supervisor is released, unless all 4 banks are allocated .
One bank is de-allocated when read-out from such a bank has been completed .

The read-out from one TPD bank proceeds according to [5] .
First, the **hitlist** relative to the bank has to be built up by the TPDs .
This operation is started by the TPP on all the TPDs at the same time .
The **hitlist generation** takes a known amount of time, so that the TPP starts a hitlist time-out (using its internal programmable interval timer) in order to receive an interrupt when the hitlist is ready (no such an interrupt can be caused by the TPD directly) .
In the meanwhile, the TPP may resume processing of the previous event that has been fully read-out and resides entirely into the TPP memory .
As soon as one hitlist is available, read-out can be started from the relevant TPD memory bank into one of the 2 TPP output buffers .
Obviously, this operation cannot be started if the TPP output buffer has not yet been released by the Event Builder, which is in charge to read-out the TPP event information .
The detailed protocol between TPP and EB is described in [3] .

Once the read-out from one of the TPD front-end buffers has been completed, the TPP :

- releases the front-end buffer
- starts a hitlist generation on the next TPD bank to be serviced
- may resume processing of the previous event (e.g.: pad cluster finding [6] for the pad TPP , wire reduction for the wire TPP)

At end of processing, the TPP can initiate the read-out of the next bank to be serviced according to the aforementioned constraints (one hitlist must be ready, one output buffer must be available).

3.4 Running monitoring tasks

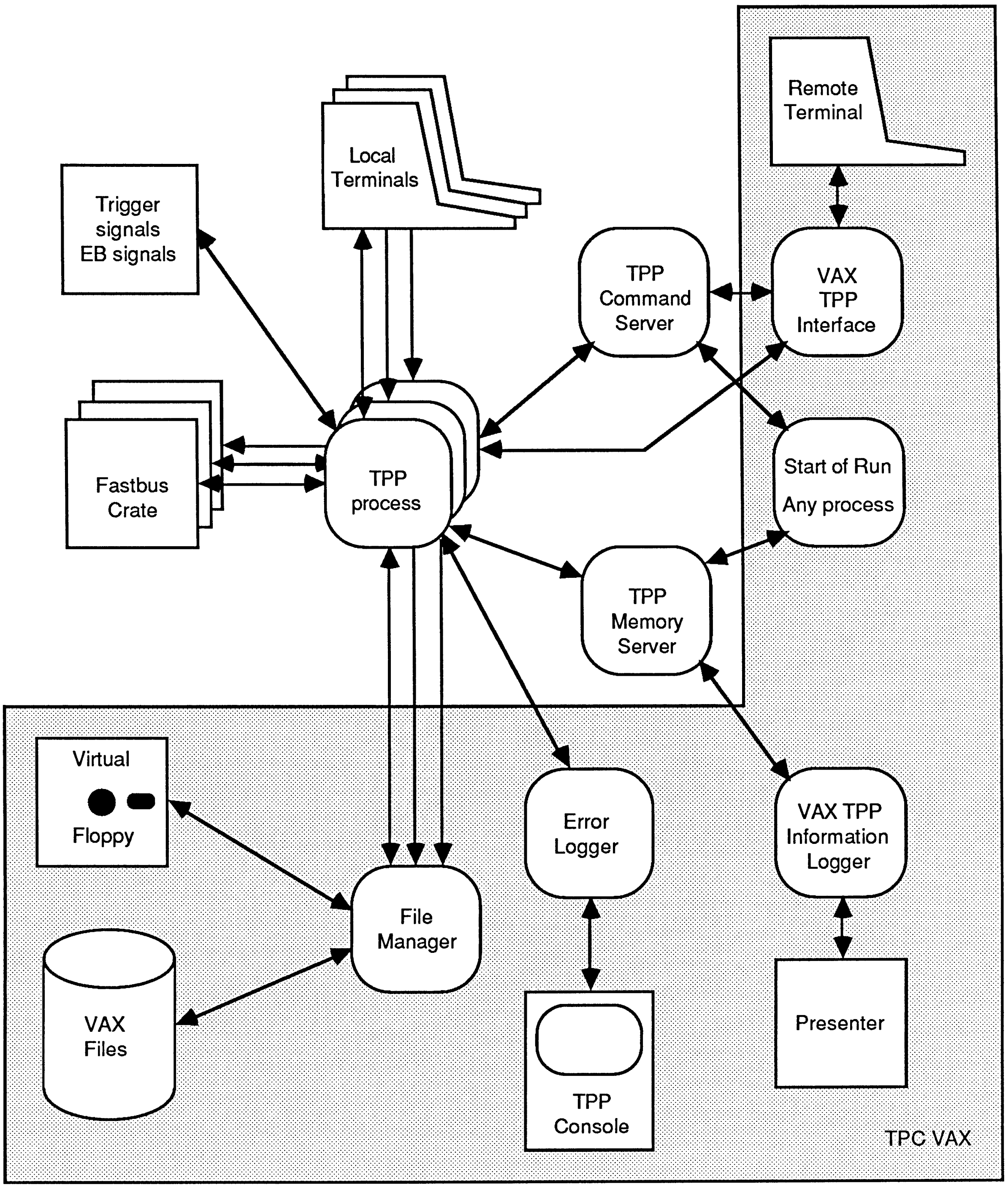
We shall first analyze the simpler situation where a ' standard ' set of monitoring programs is started for each TPP, each program with its usual parameters.
Then, the TPP **command server** (see 4.1) starts the monitoring activity according to the default task configuration .
At any time, one should be able to stop any TPP task from the VAX by issuing a command to the command server .

The actual situation of the monitoring is made available to the VAX by a **memory server** (see 4.2). This program knows from the other tasks where the relevant **data memory banks** are . It can therefore serve requests from the VAX and send out the contents of these banks.

Responses should be filtered, to present only significant information to the user .
The selection should reflect the way the TPC is working. This means that one should be able to select according to the current partition configuration : a whole endplate or the whole TPC or sector by sector.
Filtering the responses means , for instance, that if only **one** TPP out of 72 gives something different from the others, one should see only 2 messages (one for that TPP and one for the 71 remaining).

So far, we have been analyzing a 'standard' task configuration consisting of a command server, a memory server, data acquisition program , default monitoring tasks .
However, the configurations required in each TPP are sometimes different , because of special monitoring needs due to local problems in a sector of the TPC .
This may require the presence of one or more special monitoring tasks in the relevant TPP(s) .
In this case, the user 's approach should be to **select** a set of programs, in one or more TPPs , **check** that they can understand a common command (like start with defaults) and then **give** that command to them.

A **command server** in each TPP should be able to determine whether the task can start or not, and get the list of accepted commands from the task.
In the next paragraph, the latter situation is analyzed in more detail .
The configuration of processes running on the TPP(s) and on the VAX is sketched below :


Remote Terminal
Local Terminals
Trigger signals EB signals
TPP Command Server
VAX TPP Interface
Fastbus Crate
TPP process
Start of Run
Any process
TPP Memory Server
Virtual
Floppy
Error Logger
VAX TPP Information Logger
File Manager
VAX Files
TPP Console
Presenter
TPC VAX

3.5 The states of a TPP process

The command to load a new task into the memory of one TPP is issued from the VAX and passed to the TPP **command server** .
The latter invokes the appropriate system directives to load the task (under control of the operating system memory management) making use of the network file services .
One would then like to start the task without stopping the run and be able to change the task parameteres interactively from the VAX .
Direct communication cannot be allowed , since there are too many TPPs around .
Alternatively, one can use the command server to access the data area of the task and to update its parameters .
When the task is first started, it executes its initialization phase which may consist of the creation of a task's **data module** whose access is shared by all tasks in the system .
The data module may contain all of the task's parameters and a description of the task's menus.
This information can be made available to the VAX at any time via the **memory server** .
Making use of the latter information , the user interface of the VAX should be able to build and present the task's menus.
The change of a task's parameter from the VAX by a user would then result into a command be issued by the VAX to the command server of the TPP to change the parameter in the task's data module .
A simple protocol to pass the relevant information can easily be defined .
The flexibility of the above system , may look eccessive at first sight and questions may arise about the amount of storage needed on the VAX to manage task's menu information and on the expected load on the network .
A few considerations follow :

- **flexibility** to handle a frequently changing situation (i.e.: the possibility to modify the task's source code, to reload a new version of the task into memory and to change the task's parameters frequently via a comfortable user's interface) is well known to be strongly required during " trouble shooting " in any large detector .

- the ' default ' monitoring tasks running in all the TPPs of the partition are handled by a **single set** of menus on the VAX (commands and parameters are changed for all the TPPs) . Only ' special ' tasks loaded in the relevant TPPs will require their own set of menus.

- the information relative to the task's menus (which is stored in the task's data module) can be used directly by an appropriate user's interface running in the TPP to display menus and task's monitoring information whenever the TPP is run in **standalone mode**. This can be extremely useful, since the number of partitions that can be run simultaneously is limited in practice by the available hardware (total number of EB and TS modules available) .

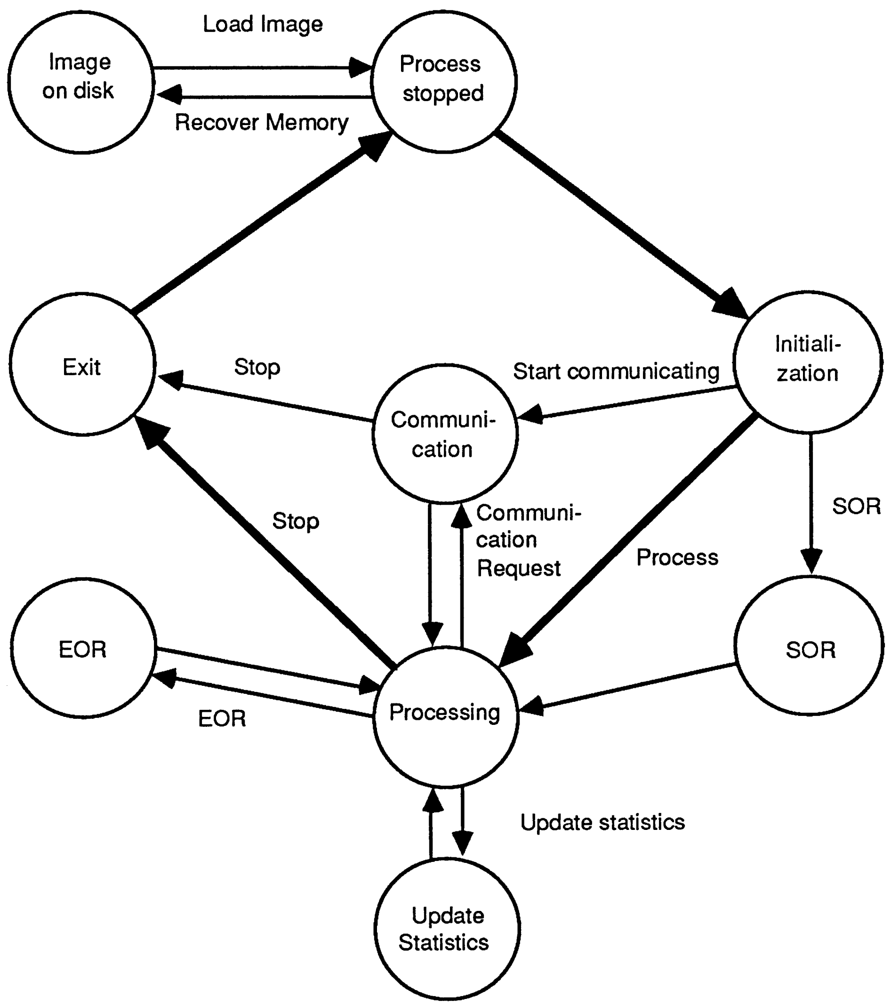
The above ideas about the interplay between the command server and the TPP tasks have several implications :

- first of all, the monitoring programs in the TPPs should have a well defined structure to be able to respond to the command server (see next picture describing task's states)

- then, these programs should have only well established ways of communication with the outside world : through responses to the command server and through messages. These responses have to be somewhat codified so that filtering can occur easily (see par. 4.1) .

The processes running in the TPP can only be in well defined states. The initialization, event processing or dialog with a user are some of these states.
The complete set of such states defines how and when a TPP process is allowed to communicate with outside and a uniform set of commands can be defined that should be obeyed to by all processes, no matter what these processes do.
Start event processing, perform end of run action are typical commands that should be common to all processes running in a TPP (see 4.1) .

The states of a process in the TPP are described by the transition diagram on next page .



The data streams of a TPP process are the following:

- User Input/Output (Terminal input/output by a user)
- Disk Input/Output
- Exception report by the TPP operating system
- Message report by the TPP process

These streams are allocated according to the following table:

Streams	Communication state	Others
Terminal I/O	To the user terminal	Not allowed
Disk I/O	To the file manager	To the file manager
Exception report	To the user terminal	To the TPP console
Message report	To the user terminal	To the TPP console

4. The software run by the TPP

The TPPs run the data acquisition software to read out and format the TPDs of the corresponding sector. Concurrently monitoring processes analyze a sample of events and report their results to the TPC VAX.
The framework where application programs should run consists mainly of :

- a **real time kernel** with an adequate scheme of interrupt handling and an appropriate execution scheduling that can assign priorities to the different processes inside the TPP.
- a **network interface** that ensures task to task communication and file services between a TPP and the TPC VAX using as a physical media both the **FastBus** and the **Cheapernet.**

The communication package should include :

- VAX **file handling**
- routines for **message logging**

Additional pieces of software are then required to get the proper environment :

- **file access** across FastBus and the Cheapernet for the local files
- a **remote terminal** across Cheapernet

In addition, one needs a few utilities that should run together with the programs:

- a **command server** that performs the necessary checks before starting tasks
- a **memory server** that can report the contents of booked memory banks

In the following the latter two utilities will be specified in more detail .

4.1 The Command Server

The operation of the command server should be interrupt driven for efficiency .
For such a reason, the TPP implements (in addition to the Cheapernet interrupt driven interface) an additional hardware interrupt for the FastBus **write** access to the memory mapped space of the TPP from the cable segment .
Upon receival of a 'command' from the VAX (either through the network or via FastBus) a **command interpreter** should get control to decode the VAX command .
The **command format** should contain information relative to the target task(s) and parameters relative to the command .
As explained in 3.5 , a set of commands can be defined that should be common to all processes running in the TPP .
The following commands may be included :

- Load task image in memory
- Start of Run
- Start / Resume processing
- Start communicating
- Update / Reset statistics
- End of Run
- Stop
- Unload task (recovering memory space)

On the VAX, the user interface to the 'standard' commands directed to the TPP tasks can be implemented as a single set of menus (i.e.: **the same** for all TPPs) .
However, the situation is different for those tasks that are **specific** to one (or more) TPP(s) and which may require a menu (o menus) of their own .
In this case, a VAX command results in an update of the task's menu(s) parameter space in the TPP memory .
The command format refers directly to the menu structure of the task whose description is stored in the task's data module, a copy of which exists on the VAX .
All necessary calculations to evaluate the offset of the parameter(s) relative to the start of the task's data bank (consistently with the menu description in memory) are carried out by the command server .
The task running in the TPP may respond to the command server directives using system and

user defined codes so that filtering by the command server can occur easily .
As an example, a positive response from all the monitoring tasks in a TPP after starting their sampling activity will be received by the command server and a **single** status message will be produced for the VAX .

The execution of directives and commands of the TPP operating system from the VAX via the command server should also be possible, providing the functionality of a remote system console, without actually logging in from a terminal.
The status of the current tasks in the system should also be available .

4.2 The memory server

Although this piece of software can be specified as a separate module, this does not imply that the memory server should be an independent task (i.e.: it may be embedded into the command server) .
The monitoring information is kept in **data banks** (in the following, a **BOS** data structure will be assumed). This is the smallest unit of monitoring information that can be referenced to from the VAX. In other words only complete banks are shipped from the TPP to the VAX. These BOS banks are updated by the monitoring processes running in a TPP. They usually contain histograms, scatter plots, or simply arrays of numbers like progressive averages or hitmaps.
In principle they can be **read** at any time, and the monitoring process is not aware of this spying action. In order to avoid clashes, however, a validation of the bank contents is performed by the monitoring process.

The memory server is in charge of shipping this monitoring information to the TPC VAX.
It obeys to VAX requests, reading these banks and sending them over the network. For this purpose, the memory server maintains a **directory** of all banks that have been declared by the monitoring processes. These banks can be anywhere in the TPP memory, although most of the time they will be part of the memory space of the declaring task.

4.2.1 Classification of BOS banks produced by the monitoring tasks

The BOS banks should include the following information:

- unique identification within a monitoring process
- TPC sector and process they refer to

- kind (histogram, scatter plot, array of numbers, mixed information)
- last time they were updated

It should be possible to replace one or more of these specifications with a " * " , requiring thus all occurencies of a bank (wildcarding). These definitions should follow the conventions adopted in the TPC Presenter.

4.2.2 Memory server commands

The following memory server commands should be available to a TPP process:

- declare a bank
- cancel a bank declaration

From the VAX side, the following commands could be sent to the memory server :

- **send** one or more banks
- **list** declared banks (also by task and by kind : like histograms, scatter plots, statistics banks, ...)
- **receive** a bank (i.e. : **write access** by the VAX into a TPP bank . This requires some care, since the task is not aware of it)

The memory server should keep track of the active tasks and cancel the data banks declarations of the stopped processes.

5. The software run by the TPC VAX

This software should include :

- a **start of run** process to coordinate the system startup and the start of run in all TPPs
- a **filemanager** for VAX and TPP files
- a **message and exception logger**
- an **information logger** to collect the monitoring information

Other utilities are also necessary like the **remote terminal** program on the VAX .

5.1 The message and Exception Logger

Messages and exceptions are issued direclty from each task running in the TPPs to the VAX, using the interrupt message facilities provided by the communication software .
Local filtering in the TPP only occurs for task responses to the command server .
Filtering on the VAX occurs at the level of the VAX Message and Exception Logger which has to be fully integrated with the VAX DAQ message system .

5.2 The Information Logger

It is an interface between the **VAX presenter** and the TPPs.
It issues requests for data memory banks to the memory servers in the appropriate TPPs .
It may handle the management of the directory of all the histograms (and in general of any data bank) produced by the TPPs .
As an example, it may request information about the current status of the tasks running in any TPP from the command server and use such information for system monitoring and status display .

6. The software development environment for the TPP software

As a minimal harware configuration, one Fastbus crate with one TPP and a few TPDs with all the connections to the VAX via Cheapernet and to the trigger supervisor has to be reserved for the software development of the TPP.

6.1 The software development environment for the TPP system software

Although most of the **system software** resident in the TPP will be bought or written by support groups, some development is necessary to adapt this software to the specific hardware used. This development will continue during the TPC assembly above ground.
Optimized code for the data acquisition program will be written in **assembly language**. Higher level languages like **C**, **Pascal** or **Fortran 77** will be used for system programming wherever appropriate .

6.2 The software development environment for the TPP monitoring processes

A monitoring process in the TPP runs in a well defined environment : there is a source of events, an output of results that are usually produced once a certain amount of statistics is accumulated. Errors have to be reported by a message system. Interaction with the user is infrequent as monitoring parameters are rarely changed.

This environment can be easily simulated in the TPC VAX where these monitoring processes have to be developed. By following some coding conventions, the monitoring processes developed in the TPC VAX will be transported to the TPP with only well defined modifications. The language will therefore be the same as the one used by TPC people to write their monitoring programs on the VAX, namely **FORTRAN 77** with some extentions, although optimized code will be written in assembly language.
A specialized **package for histogramming** is required providing full optimization for processors of the 68000 family.

References

[1] S. R. Amendolia et. al. , Time Projection Processor, TPC Note 84-56 , July 1984

[2] S. R. Amendolia et. al. , Requirements for the Aleph Data Acquisition Design - Data Flow, Aleph Note 115, Jan 1984

[3] S. R. Amendolia et. al. , Aleph Data Acquisition System (Hardware functional specifications) DATACQ Note 85-21 , Nov 1985

[4] S. R. Amendolia et. al. , The MC68020-based FastBus read-out processor for the Aleph Time Projection Chamber, IEEE Trans. on Nucl. Science, NS-34 , n. 1, 128 (1987)

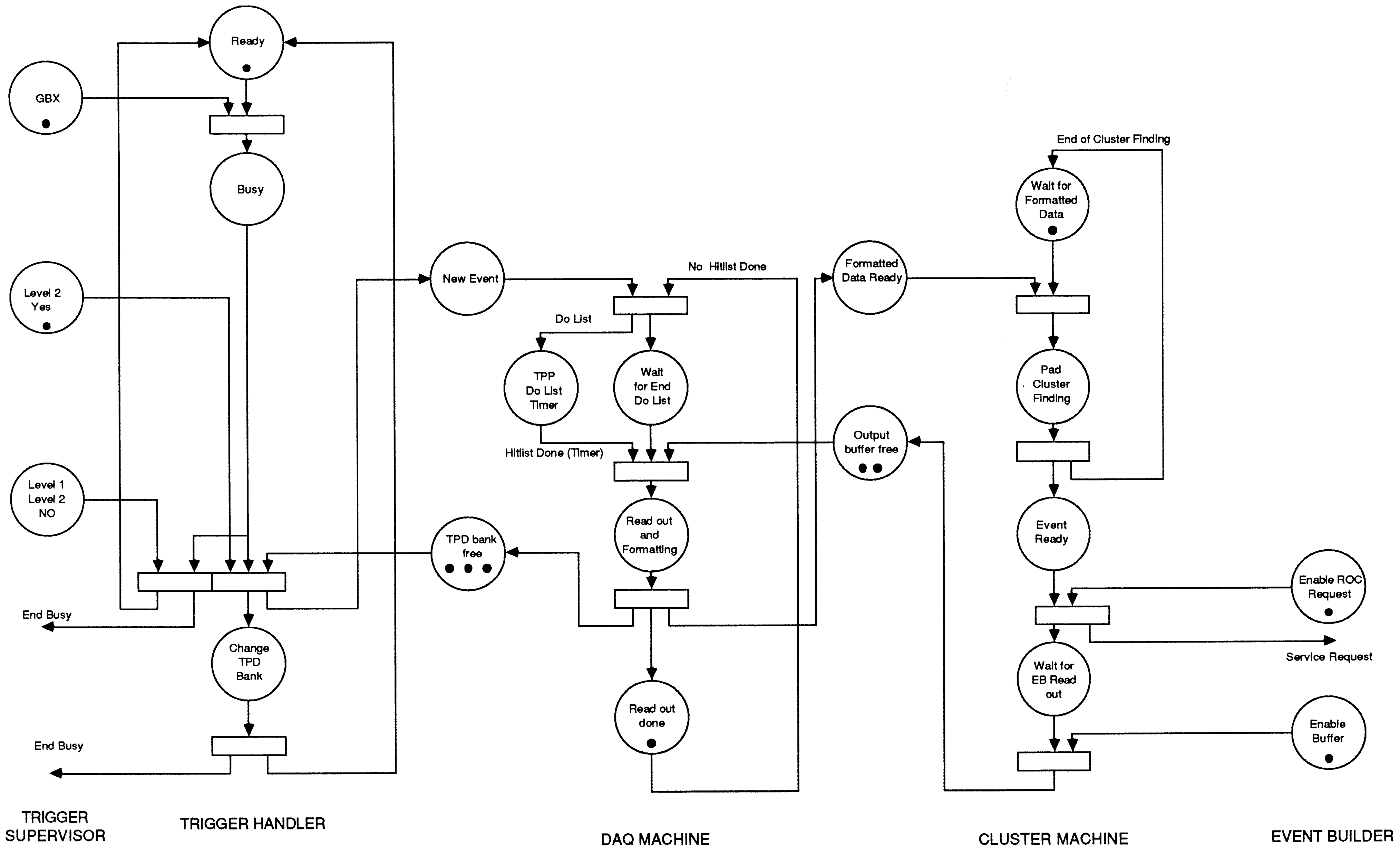
[5] B. Lofstedt et. al., The Time Projection Digitizer, Functional Specifications

[6] P. S.Marrocchesi, Towards an on-line identification of pad clusters in the TPC, Aleph Note 159 (1986)

[7] J. L. Peterson , " Petri Nets " , Computing Surveys, vol 9, n. 3 , Sept. 1977

Appendix A.1

- Petri Net for the TPP Data Acquisition Program



Petri Net for the
DAQ Program
13-may-1987
FF-PSM-SRA