EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN–ECP 94–5
22 April 1994

THE GENERAL SURVEILLANCE SYSTEM (GSS)

Jean-Pierre Puget, Edo Sbrissa, Laurence Vinot
CERN, Geneva, Switzerland

Abstract

In an environment which simultaneously contains a mixture of inflammable gas, air, and electronic equipment, it is of the utmost importance to watch carefully for any conditions that could lead to an accident. In the case of the four LEP experiments, the required system has to be of high performance: work in real time, handle a large number of parameters, handle a large number of rules, be open to the outside world, provide reliability, security, and stability for the application.

The General Surveillance System (GSS) application is based on Equipment Control Assemblies (ECAs) (data acquisition units), on a 0^+ expert system and on some processes. This document explains how the GSS application is organized, and details each part of it.

Edited by the Desktop Publishing Service, CERN

CONTENTS

INTRODUCTION

The General Surveillance System (GSS) must provide safety for persons and equipment in each of the four LEP experiments. The GSS application consists of a complete safety system adapted to each experiment.

The GSS receives information from sensors used to monitor the environment. The GSS has some means of actions that allow to react according to the severity of the encountered problem.

The purpose of this document is to describe the current state of the GSS in order to have a basis to propose a solution for a re-engineering of the system.

The GSS can be divided into two parts: the **hardware part** for the acquisition, the transmission of information from/to sensors and equipment, and a first-level processing (see Section 1), and the **software part** that processes information and provides a lot of facilities to allow users and external systems to interact with it (see Section 2).

Sections 3 and 4 give some information on the **run-time expert system** and its **generation**. In Section 5, we describe the **data acquisition system** including its current new developments. The last section of this document contains a synthesis of the **weak and strong points of GSS**. This part will probably be a starting point for the re-engineering proposal.

Definition:
In the document, the **GSS central** refers to the set of processes described in the software part of GSS.

1. HARDWARE DESCRIPTION

1.1 Computers and networks

Software processing of all GSS information is done on DEC VAX machines running VMS. Both the DELPHI and ALEPH experiments have a VAX4200. The L3 and OPAL experiments still have μVAXIIs. It is planned to upgrade to VAX3100s on both L3 and OPAL. Communication between the GSS VAX and the front-end system Equipment Control Assembly (ECA), formerly called UPS, was made through Utinet. Utinet, a medium speed LAN developed in-house, is currently being replaced by Ethernet.

Communication with the slow control systems of the experiments is done through Ethernet.

Communication with the LEP Central Alarm Server (CAS) is done through a gateway (router) between Ethernet and the LEP token ring.

1.2 General communications architecture

Communication between the different security systems is realized through Ethernet and the token ring (Fig. 1).

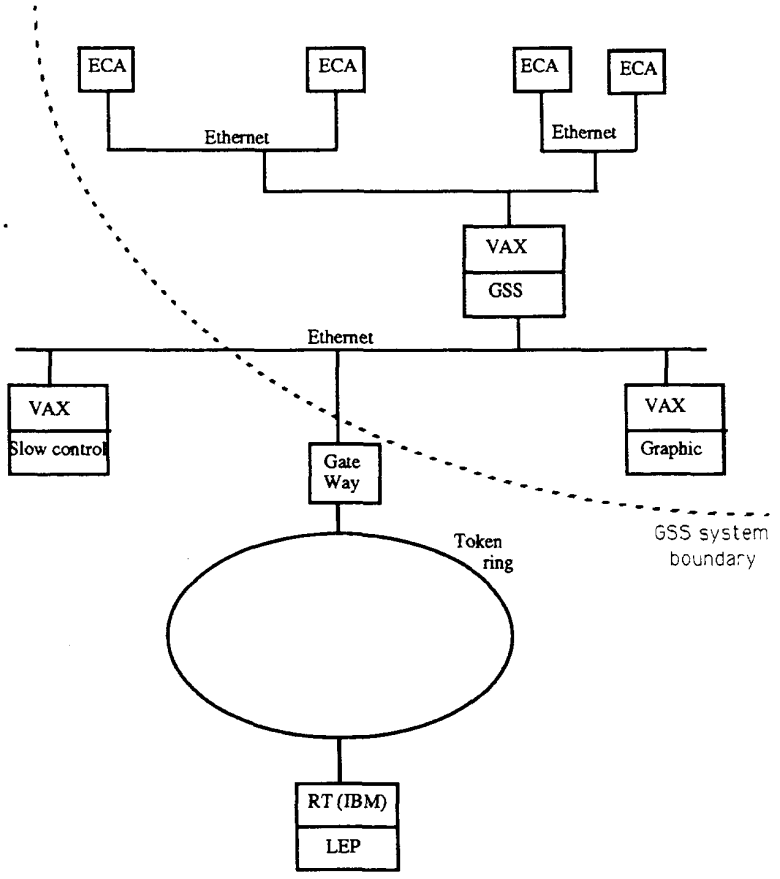

Fig. 1: Communication between the different security systems

1.3 Equipment control assembly

Equipment Control Assemblies (ECAs) are based on a VME bus and controlled by a processor of the 680X0 family running the OS-9 operating system. They read sensors (input) and drive equipment (output) such as valves (water, gas), heavy relays, fans, etc.

The ECA surveys the sensors and when a problem is detected it sends a message to the software part of GSS, that will initiate alarm processing (Fig. 2).

The ECAs can also execute local emergency procedures ('reflex actions') to turn off power supplies, cut the high voltage, etc. These procedures reside in the ROM of each ECA, but can also be downloaded. ECAs also execute commands received from the GSS central.

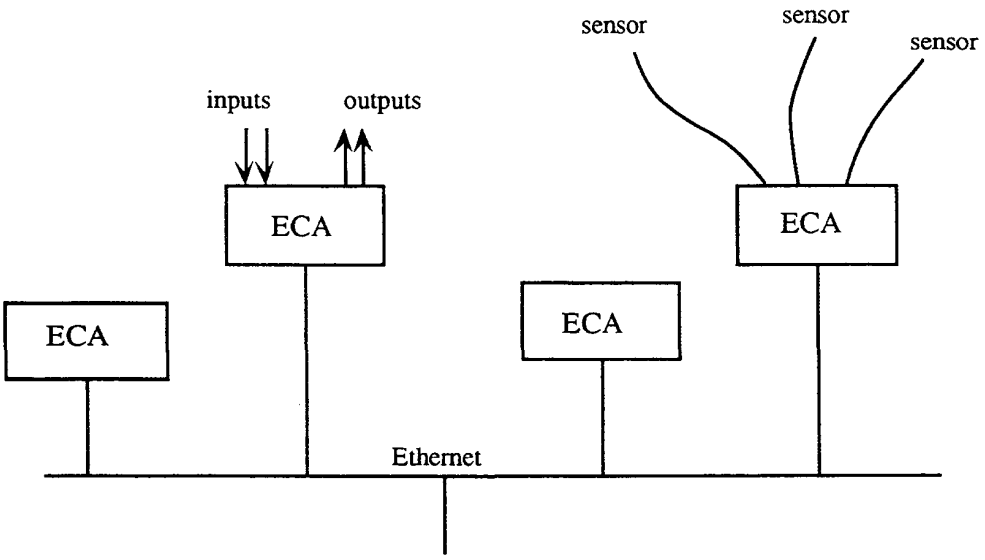

Fig. 2: ECAs and their communications

1.3.1 Hardware

The ECAs are part of the safety system of the experimental zones (Fig. 3). They have been designed to meet TIS and INB specifications. These ECAs are connected to the electrical power network and in case of power loss they will run for two hours in the backup power supply mode. The ECAs are designed to guarantee the safety network N3 in the event of one or more systems failure.

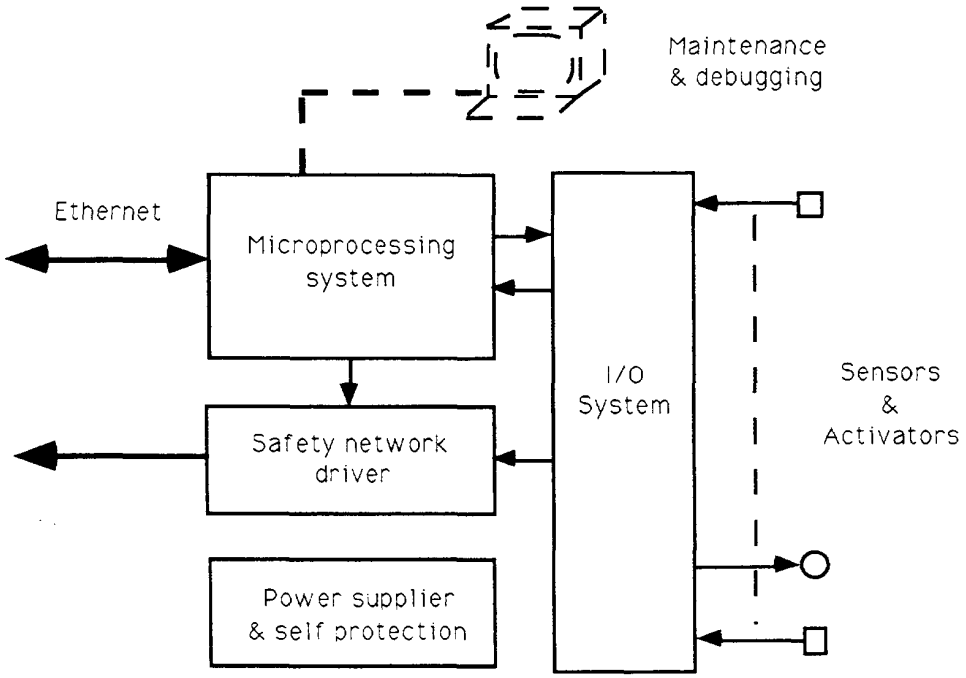

Fig. 3: Schema of the hardware block

1.3.1.1 Microprocessing system

The microprocessing system is based on a VME 6U containing the Themis TSVME 131-4FPA processor and an interface circuit which is used to access I/O equipment and the safety network N3.

1.3.1.2 Input/Output system

The I/O system can be configured according to the functions the ECA will perform. It can contain up to 30 input or output modules. Each of these modules has its own address and must use the I/O bus access protocol to communicate with the processor. Input modules have been designed to permit sensors to take direct action on the safety network [1].

1.3.1.3 Safety network driver

The safety network driver allows sensors connected to the ECA to have direct access to four normally closed safety loops: GAS, FIRE, WATER, GEN. There is also a SYS loop that carries safety network status information (Inhibit or Anomaly) [2].

1.3.1.4 Power supplies

At the heart of the power supply section is a modular DC–DC converter which is used by many European telecommunications services. These modules are mounted in parallel in order to double the necessary power. A self-surveillance system monitors the output voltages and temperatures of each module and will send an alarm signal in the event of failure or malfunction.

1.3.2 Software

The entire software in the ECA has been written in the C language under OS-9 version 2.3. Some routines dedicated to comput the interruption signals have been written in the 68K assembly language.

Figure 4 shows the general organization of the software part of the ECAs. The functioning of each part is detailed in the following paragraphs.

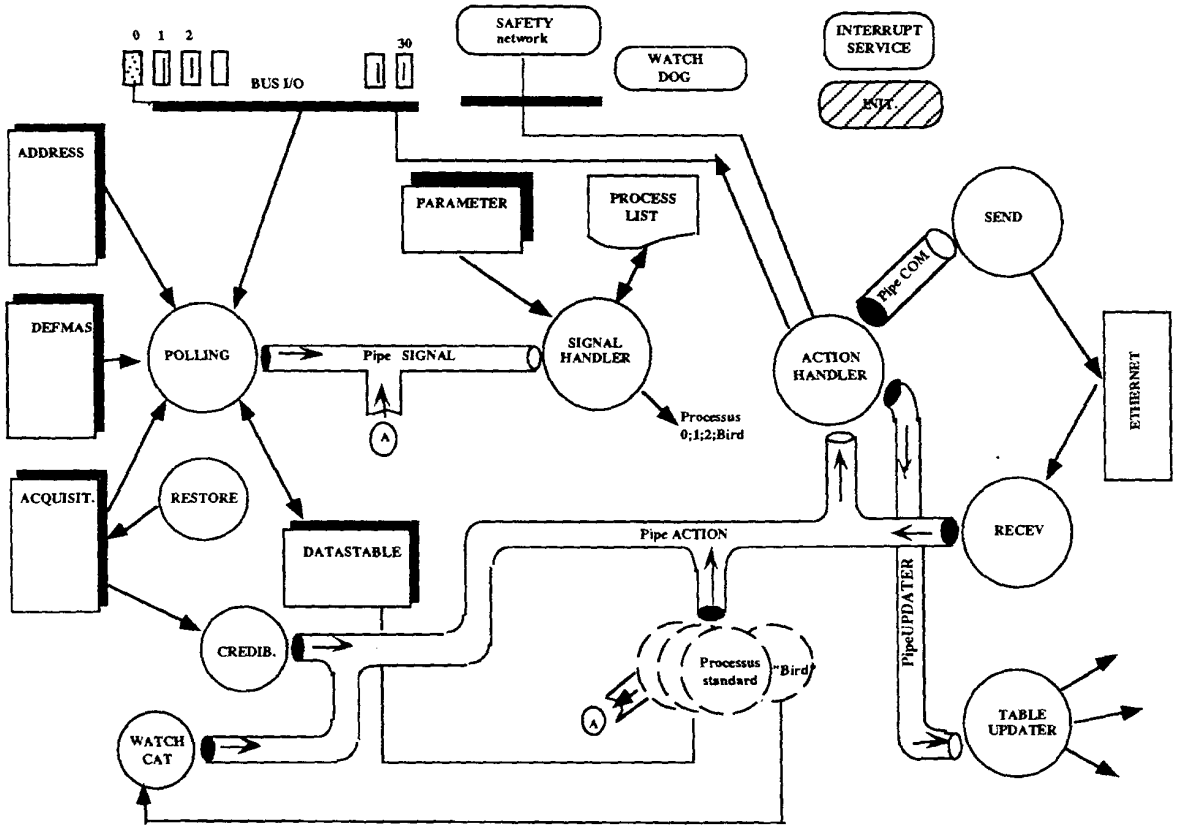

Fig. 4: Internal structure of the software part of ECAs

1.3.2.1 Role of the ECAs

The main role of the ECAs is:

1) Get information from:
   - the input/output bus (I/O bus)
   - the GSS central
   - from the self-surveillance systems of the ECA
2) Execute pre-programmed 'reflex' actions corresponding to the acquisition.
3) Inform the GSS central of new acquisitions and actions performed by the ECA.
4) Allow the GSS central to change dynamically the content of tables and to perform actions.

Since each experimental site contains many multifunctioned ECAs (10 to 15), it is difficult to write specific ECA software. Therefore, by using a general software with particular parameters, we achieve specialization while maintaining the flexibility of each data acquisition unit.

To program the 'reflex' actions (role no. 2), a perfect knowledge of all parameters describing a situation and their realistic and probable combinations is required. But because of the complexity of the installations and the wide number of parameters needed to define a real situation, it is not trivial.

We describe three very simple 'elementary situations' that are defined by a few parameters and are represented in the software by *standard processes*. By combining several standard processes run simultaneously or sequentially, it is easy to simulate complex situations.

1.3.2.2 Standard processes

Process of type 0

A change in state of the source signal S0 starts the process which is represented by the transition from state E0 to state E1 (Fig. 5).

The action foreseen on the state E1 is taken, and if a time-out T1 is specified, the process sleeps for T1 seconds. At this point there are two possibilities: the time-out is finished and action A1 is executed, or the signal S0 resets to its initial value and action A0 is taken. In both cases the process is completed.

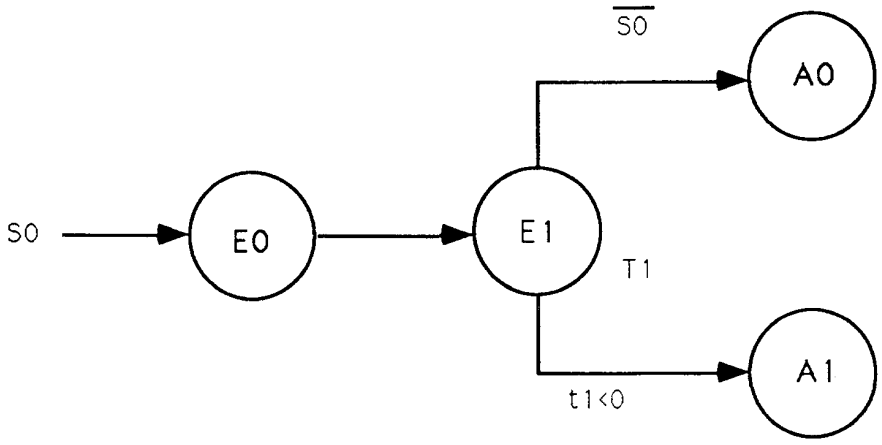

Fig. 5: Process of type 0

Remarks:

1) The toggle of S0 stops the current process and initiates a new one.
2) The foreseen actions and time-out are linked to the variation of the source signal S0 and stored on the Parameter-Table.

Process type 1

The change of S0 (Fig. 6) starts the process as mentioned above. S1 and S0 are essential in describing an elementary situation where S1 represents a piece of information closely linked to S0. At the start of the process, S1 is evaluated and the process continues in either state E1 or E2. The action foreseen on the chosen state is taken and its time-out is started. Every change of S1 will toggle the process between E1 and E2. The time difference between states will be compensated for and the foreseen action of E1 or E2 will be taken once only.

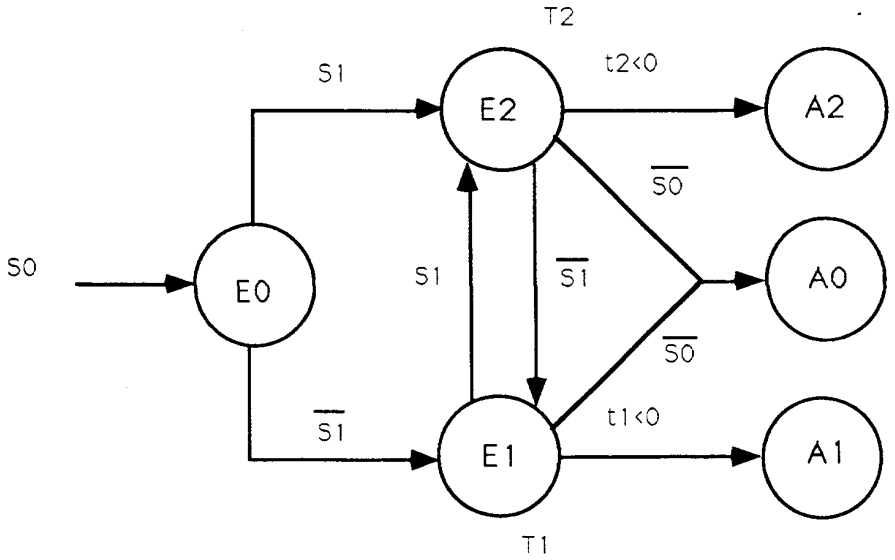

Fig. 6: Process type 1

Process type 2

The change of S0 generates a process. The information in S1 and S2 is closely linked to that of S0. The description of processes 0 and 1 can be applied to process type 2 (Fig. 7).

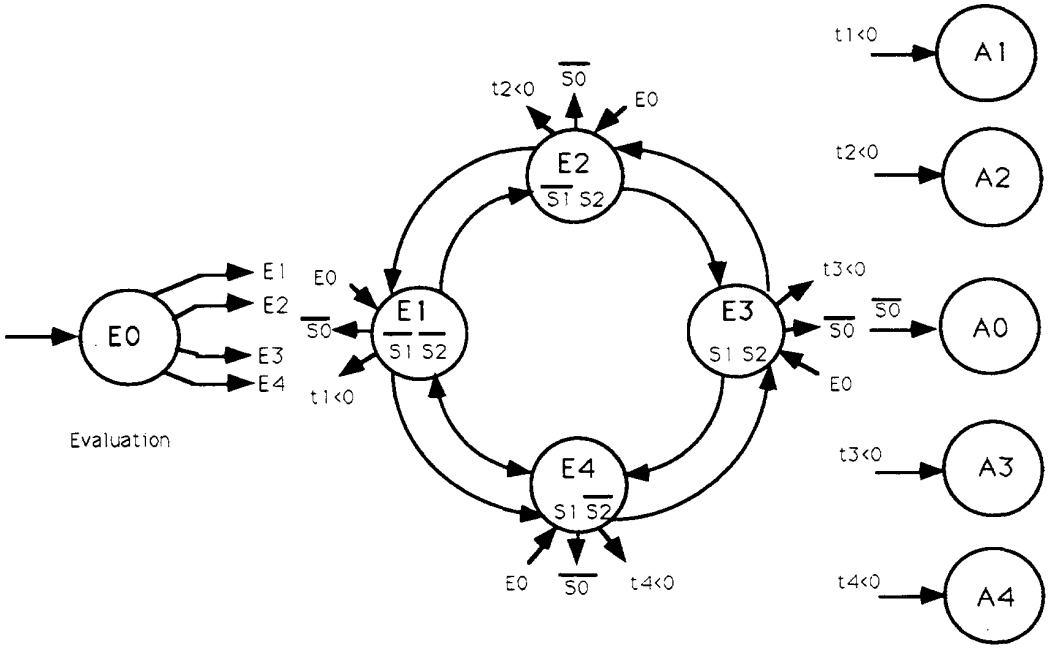

Fig. 7: Process type 2

Time compensation

In each state E_i there is a time-out value T_i which is used to set a counter t_i. Every period of the system clock decrements all counters to zero. At the change from E_i to E_j, the time difference is known and is labelled t_{ir}.

Example:

E_a and E_b are the states of a process where the toggle is controlled by S1 (Fig. 8).

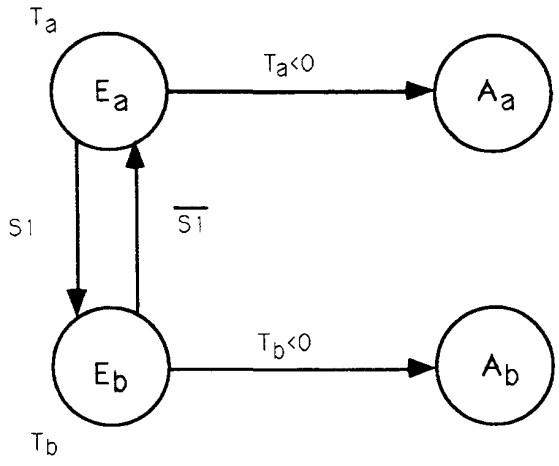

Fig. 8: Example of time compensation

- Transition of **S1**: change of E_a versus E_b

if	$T_b - (T_a - t_{ar}) > 0$	then	$t_b = T_b - T_a + t_{ar}$
	≤ 0	then	action A_b

- Transition of **S1**: change of E_b versus E_a

if	$T_a - (T_b - t_{br}) > 0$	then	$t_a = T_a - T_b + t_{br}$
	≤ 0	then	action A_a

1.3.3 Structure of the software

The software is stored on EPROM memory and its structure, as shown in Fig. 5, composed of processes, data modules, pipes etc. is loaded into RAM at the start-up of the ECA.

1.3.4 Data module

The EPROM 'Param & Config' contains all values of every table described below. This EPROM is a simple structureless list of values that specify the configuration of the I/O system and its reactions; it is used at the PowerUp to initialize every I/O module.

1.3.4.1 Address_Table

This table contains 31 structures of variables used by 'Polling'

Type	1 byte	Specifies the type of I/O module
		00 Logical Input
		01 Analog Input
		02 Single Output
		03 Multiple Output
		08 Virtual Input
		FF Slot free
Inh_R	1 byte	Inhibit of real module
Inh_V	1 byte	Inhibit of virtual module
Val	1 byte	Value of the virtual module

1.3.4.2 Defmas_Table

This table is used by 'Polling' and 'UpdaterTable'. It contains information for the Logical Input modules.

Default	1 byte	Default value
Mask	1 byte	Mask value

1.3.4.3 Acquisition_Table

This table is used by 'Polling', 'UpdaterTable', 'Credib' and 'Restore'. It contains 31 structures of 2 arrays.

Tolerance	8 bytes	Maximum number of oscillations tolerated for one input
Credib	8 bytes	Oscillation counters

1.3.4.4 Parameter_Table

This table contains all foreseen actions executed by an ECA in accordance with its acquisitions. It is composed of the 496 structures described below.

No.	1 byte	Type of standard process to start
S0	1 byte	Index of source signal
S1	1 byte	Index of first signal closely linked
S2	1 byte	Index of second signal closely linked
E1	4 bytes	Action to intermediate state
E2	4 bytes	"
E3	4 bytes	"
E4	4 bytes	"
T1	4 bytes	Time-out to intermediate state
T2	4 bytes	"
T3	4 bytes	"
T4	4 bytes	"
A0	4 bytes	Action to exit state
A1	4 bytes	"
A2	4 bytes	"
A3	4 bytes	"
A4	4 bytes	"

T1 to T4 are numbers of time ticks and coded as 'unsigned long integer'.

The four bytes of the actions, intermediate or exit, are coded as an internal message:

C	Code of message
N	Position number of the module on the I/O system
F	Function executed in this module
D	Data associated with the function

1.3.4.5 Datastable_Table

This table contains the 31 values produced by the acquisition task. The values issued from the Logical Input module are filtered by 'polling'.

1.3.5 Processes

1.3.5.1 Polling

To complete an acquisition of available information, the 'polling' process of the ECA reads the I/O bus ten times per second. To do this, the process needs the type and the position of each module, and parameters associated with its functioning. This information is stored in ADDRESS.

To be processed, signals read on the I/O bus must satisfy the following two conditions:

- For ten seconds, its state must not change more than the maximum number of times stored in ACQUISITION.
- It must be stable during three consecutive acquisitions.

If a signal can be processed by using its index number (obtained according to its place on the I/O bus) it is stored in the DATASTABLE containing the current states of the I/O bus. The index number plus the new logical state of the signal is then transmitted to SIGNAL_HANDLER.

1.3.5.2 Signal_Handler

The management of signals and their index number can be divided into two parts:

First, the PROCESS_LIST is consulted. Each process present in that list and waiting for the index number corresponding to the acquisition is woken up. It allows the process to perform new actions.

Second, for each signal a new standard process is created. The type of this process and information related to its functioning come from PARAMETER. Each new standard process and its associated information is stored in PROCESS_LIST and has an ID number.

1.3.5.3 Action_Handler

ACTION_HANDLER is the hub of the ECA software. It is the largest but also the simplest program of the ECAs. It handles messages received from other processes and executes corresponding commands. These messages are sent by standard processes and by the process ECACOM that manages the ECA–GSS central communication (see Section 2.3.3.)

Some of the actions associated with the messages are intended for the I/O bus. Therefore, ACTION_HANDLER contains access functions to the I/O bus (read/write). Other actions transmit messages or data to GSS central through ECACOM, or ask ECA to execute programs from its ROM.

1.3.5.4 *Standard processes*

Standard processes have been detailed in the previous paragraphs. They can access DATASTABLE to see the state of entries. Actions requested by standard processes are transmitted to ACTION_HANDLER.

1.3.5.5 *Receive*

The RECEIVE process receives TCP/IP packets, checks their checksum and the coherence of their message (packets coming from the GSS central contain only one message) before passing it on to the process ACTION_HANDLER.

1.3.5.6 *Send*

SEND is the companion process to RECEIVE. The SEND process receives messages from ACTION_HANDLER, and combines them into TCP/IP data packets. These packets contain the message to be transmitted and a set of bytes useful for the GSS central to check the message integrity. Communication from ECA to GSS central is made through Ethernet, using the TCP/IP protocol.

1.3.5.7 *Table Updater*

The GSS central can update the following tables:

- PARAMETER
- ADDRESS (partially)
- DEFMAS
- ACQUISITION

These tables are shared between several processes. Their access is handled using semaphores.

1.3.5.8 *Credib and Restore*

These two processes are like filters. Credib is used by polling at the end of an I/O acquisition cycle. Each time an entry toggles, the Credib counter associated with this entry is decremented. When its value is null, the entry will be considered as oscillated and will not be part of the acquisition. The Restore process is active every 10 seconds. It sets up each entry variation counter at a predefined value contained in ACQUISITION.

1.3.5.9 *Self-surveillance programs*

There are three processes used to monitor the ECA software. They check if the essential processes (Polling, Signal_Handler, Action_Handler, Table_Updater, Send and Receive) are running correctly.

Every 20 seconds, the WATCH_CAT process sends a message to ACTION_HANDLER and waits for a wake-up signal (a time-out is associated with it). ACTION_HANDLER interprets this message and writes to a virtual module. The next polling leads to the acquisition of this data, which creates the BIRD process that wakes up WATCH_CAT. This check sequence allows the testing of every essential function of the ECA. Then WATCH_DOG (which is a hardware mechanism directly linked to the 'reset' entry of the VME processor) can be regenerated.

1.3.6 Messages

Messages can be of three types:

- Messages from ECA to GSS central
- Messages from GSS central to ECA
- Internal ECA messages

They are described in Appendix 1.

1.3.7 Start-up sequence

At the start of an ECA, the INIT program builds the complete structure of the ECA software using data from the EPROM. It sets up the tables (data module) with default values. All processes are put in a WakeUp signal waiting state except Action_Handler, Send, Receive and Updater_Table which are activated.

The 'PowerUp' message is transmitted to the GSS central and the ECA waits, with a time-out, for data update.

As soon as the 'EndOfUpdate' message is received, or after a time-out, the 'Startup' message is sent to the GSS central. Therefore, the GSS central knows whether the ECA start-up has been completed or not.

Then the ECA makes its first data acquisition, and all the processes run as described previously .

2. SOFTWARE DESCRIPTION

The GSS software (GSS central) is a set of dedicated processes communicating together. The complete description of each of the processes can be found in Ref. [3].

2.1 Notations - definitions

Definition:

An **event** is a piece of information sent by an ECA (either an alarm, or a 'back-to-normal state').

Remark:

Everything described in this paragraph applies to **one** experiment. There is **one** GSS system running on each of the four LEP experiments.

2.2 General organization of the GSS processes

The general organization of the GSS processes is shown in Fig. 9

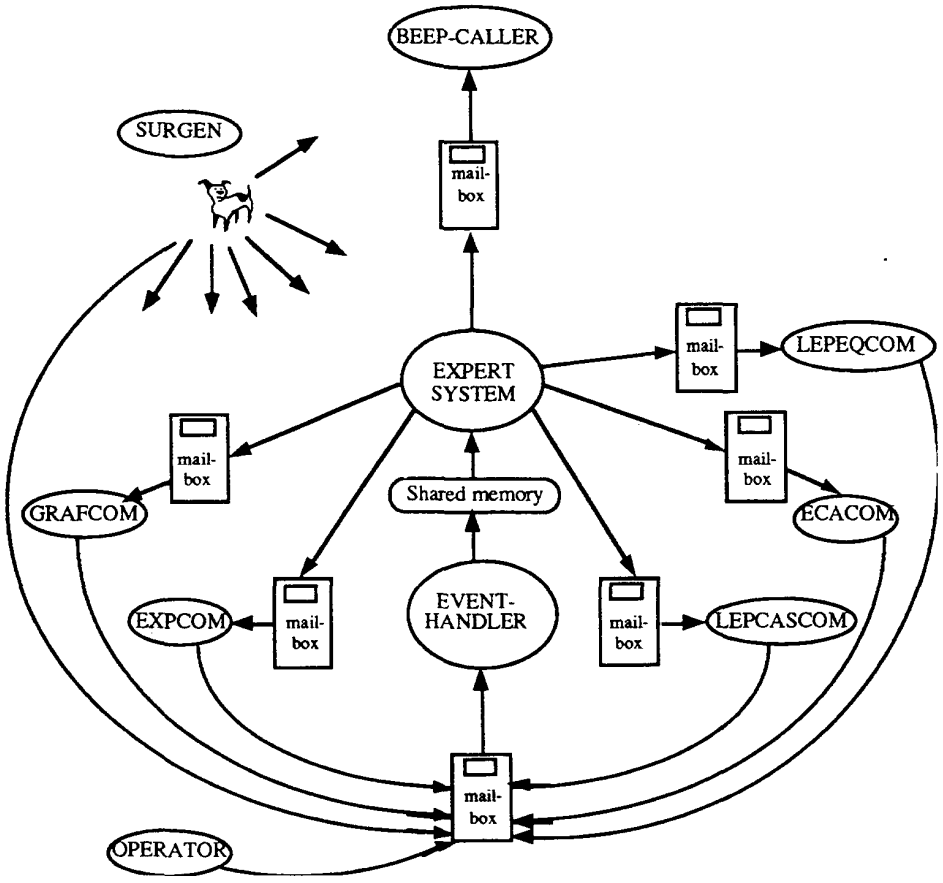

Fig. 9: Organization of the GSS system processes

These processes can be divided into five categories:

- **5 communication processes:** communicate with the different remote security systems and the graphics station.
- **2 main processes**: the event-handler and the expert system. The event-handler provides data to the expert system.
- **2 watchdog** processes named SURGEN and SURGEN-LOG.
- **1 interface** package 'OPERATOR' for operator access. It allows the system to be controlled remotely via a modem from an ANSI terminal.
- The **graphical interface.**

All GSS supervisory processes run on the same computer. They communicate through mailboxes with the exception of the event handler and the expert system which communicate through shared memory.

2.3 Communication processes

2.3.1 Communication with the LEP Central Alarm Server (CAS)

The LEPCASCOM process is the same on all four experiments (Fig. 10).

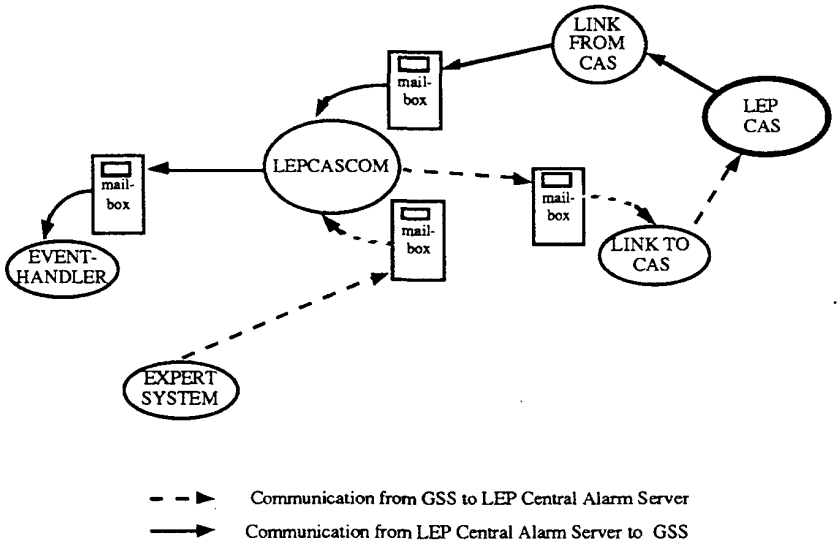

Fig. 10: Communication between GSS and LEP CAS

The LEPCASCOM process receives information from the expert system and transmits it to the LEP Central Alarm Server (CAS). It also receives information from the LEP CAS and transmits it to the expert system through the event handler. It makes periodically a back-up of the information sent by the LEP CAS and tests the proper functioning of the communication.

Communication between LEPCASCOM and the LEP CAS is done through a client process LINK-TO-CAS and a server process LINK-FROM-CAS. They use RPCs on the CAS side and mailboxes on the GSS side.

2.3.2 *Communication with slow controls*

The **EXPCOM** process receives information from the expert system and transmits it to the experiment's slow-control system (Fig. 11). It also receives information from slow control and transmits it to the expert system through the event handler. Communications are made through mailboxes.

EXPCOM is specific to each experiment since each slow-control system has its own communications protocol. It differs mainly in the data format.

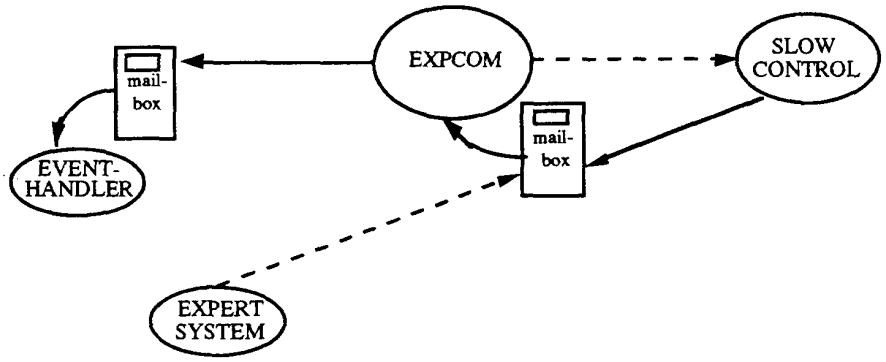

Fig. 11: Communication between GSS and slow controls

2.3.3 *Communication with the ECA*

The ECACOM process is the same on all four experiments (Fig. 12). It receives information from the ECAs and transmits it to the expert system through the event handler. Information sent by the ECA is stored into an Ethernet buffer which is emptied by ECACOM. ECACOM also receives information from the expert system and transmits it to the ECAs.

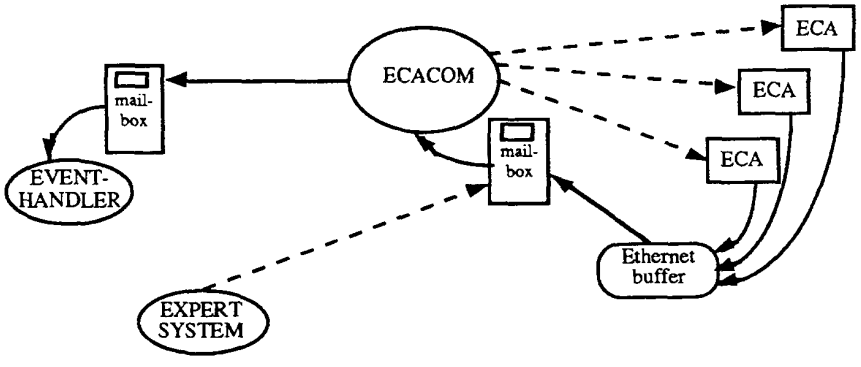

Fig. 12: Communication between GSS and ECAs

ECACOM executes OPERATOR requests and commands issued by the expert system, and can kill processes in the ECA. It also makes back-ups of the ECA, tests the communication with it, filters bouncing and logs errors.

2.3.4 Communication with the graphics station

The GRAFCOM process is the same on all four experiments (Fig. 13). It receives information from the expert system and transmits it to the graphics station. GRAFCOM also receives information from the graphics station, and transmits it to the expert system through the event handler. Operators can interactively perform commands such as turning on an electrical panel or closing a valve.

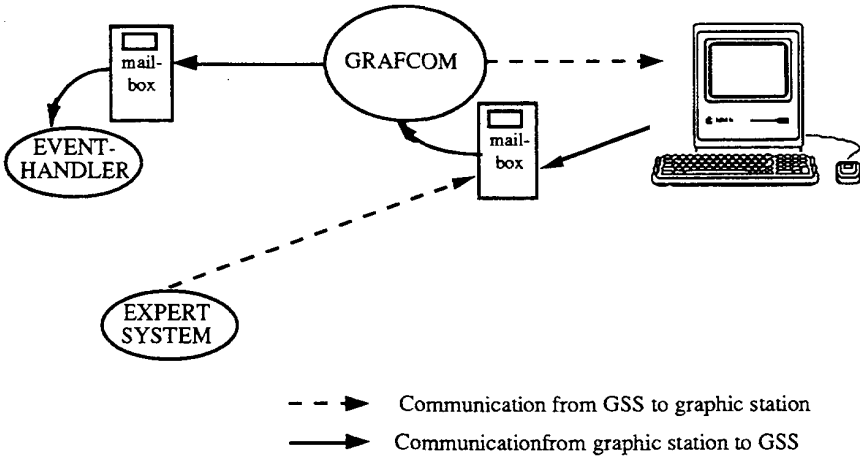

Fig. 13: Communication with the GSS graphics station

2.3.5 Communication with the paging system

The GSS is able to contact on-call people through CERN's paging system, commonly called beep. When the **BEEP-CALLER** process (Fig. 14) receives a request from the expert system, it calls the specified persons through a modem. Since search calls through CERN's paging system are not entirely reliable, BEEP-CALLER has a recall facility. The BEEP-CALLER process is the same on all four experiments.

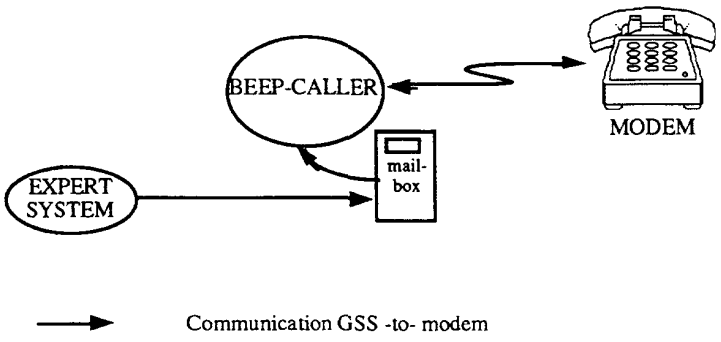

Fig. 14: Communication with the paging system

2.4 Main processes

2.4.1 The event handler

The event handler process is the same on all four experiments.

The event handler collects events from the different communication processes and updates the memory shared with the expert system (Fig. 15). The shared memory contains values; it is the fact base of the expert system. Events not relevant to the expert system are filtered out.

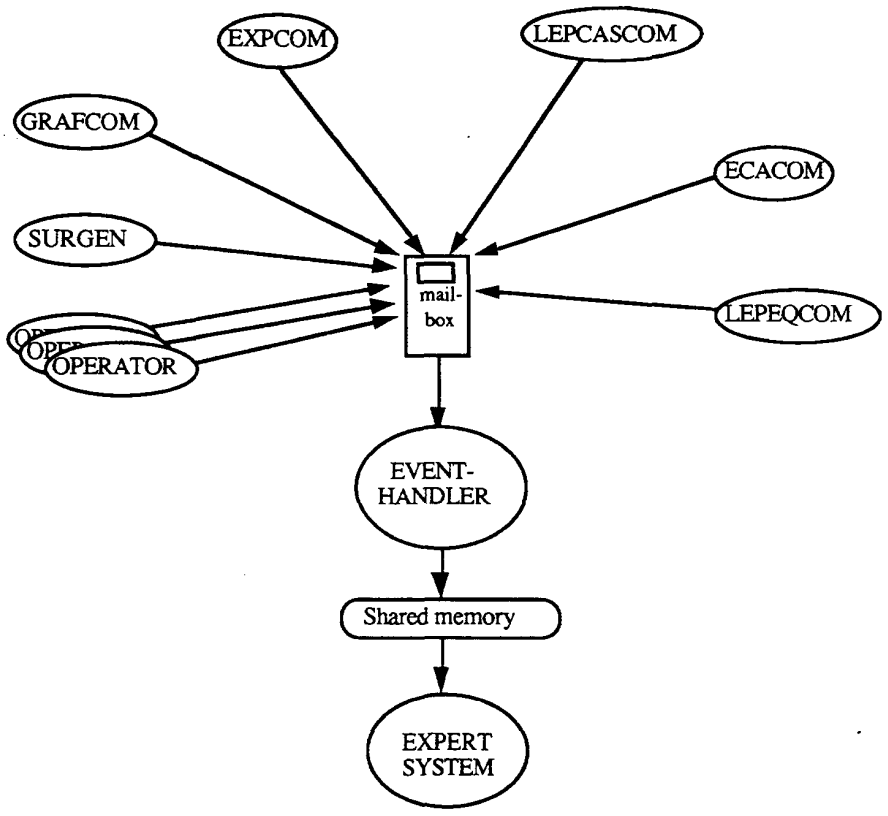

Fig. 15: Relations between the event handler and other processes

The event handler manages the timers of the expert system and executes some of the actions requested by it.

2.4.2 The expert system

The event handler communicates through shared memory with the expert system (instead of a mailbox). This memory is actually the fact base of the expert system. The expert system sends requests to the five communicating processes via mailboxes (Fig. 16).

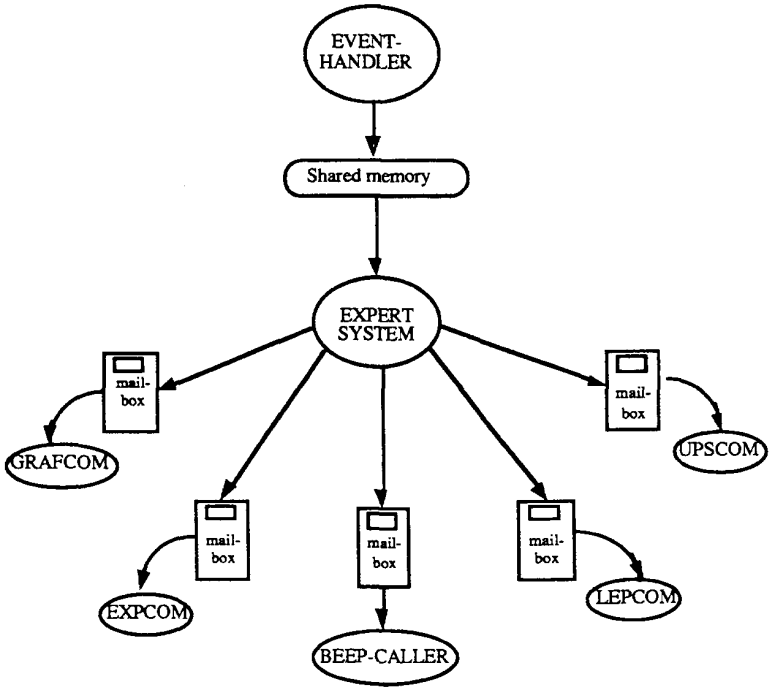

Fig. 16: Relations between the expert system and other processes

2.5 Watchdog processes

2.5.1 The SURGEN process

SURGEN surveys all other processes. In particular it ensures that all processes are running and, if needed, restarts a missing process automatically. It also checks if there is enough space left on disks and that the workspace of the processes is sufficient.

Problems encountered are reported to the expert system through the event handler (Fig. 17).

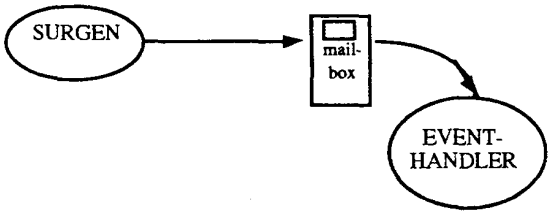

Fig. 17: The SURGEN-to-event-handler communication

2.5.2 The Surgen-Log process

Surgen-Log is a DCL command that surveys the processes and informs maintenance operators of problems. Surgen-Log does the following:

- Surveys the SURGEN process and restarts it if it is not running.
- Restarts any other GSS process if its state is abnormal.

Each time a GSS maintenance operator makes a login, he can see a list of information messages corresponding to the problems detected by Surgen-Log. This list of messages also includes information about the state of ECAs (ECA01 IS NOT RESPONDING), and information about the state of the communication between the GSS central and the graphics station.

2.6 The operator's access package

This package provides a general text-based interface for accessing the GSS system running on VT100-type terminals. Some operations can be done by anybody, others require a user name and a password. The operator package provides the following facilities:

- Display the list of active warnings and alarms
- Display the list of active anomalies
- Display the list of logged alarms
- Read electronic mail
- Authorize inhibition of sensors by giving a time interval
- Display the list of inhibition authorizations
- Mask a sensor for a specified period
- Display the list of masked sensors
- Initiate a test of the GSS system. This functionality is useful, for example, for testing the communication with the graphics, the modem used by BEEP_CALLER, the OPERATOR package, etc.
- Interact with the ECA: display and modify the content of ECA tables, change the value of a virtual entry in an ECA, execute an action in a command module, read the status of a module of an ECA, read the analog stack, kill an OS-9 process, start an ECA test program, mask an entry in an ECA, inhibit an ECA module, etc.
- Interact with the GSS processes: put the processes in debug or log mode, add, delete, or modify user names/passwords of the operator package, read the status of the ECA, ECACOM, etc.

2.7 The graphical interface

The graphical interface is used to display events coming from GSS central through GRAFCOM (Fig. 18). It is also used to transmit information entered by operators on the graphical screen. All the processes of the graphical interface are the same on all four experiments. Only the data change (files containing image descriptions, and initialization files).

The graphical interface is made up of:

- 1 main process named GSSUPDGRAF: it receives information and after having processed it, sends it to other processes.

- 3 processes managing the graphics screen (GSS_GRAPHICS, GSS_RD_MSG, GSS_BEEP_GRAF): they display on the graphics screen the place and alarm level of an event (by highlighting and colouring a graphical object), and associated messages.
- 3 dynamic processes: at the operator's request, GSS_GPX_HELP displays a help screen, GSS_PROCED displays a procedure, and GSS_GPX_CMD transmits a command to the GSS central (GRAFCOM).
- 1 watchdog process GSS_SURGEN: it surveys all previous processes.

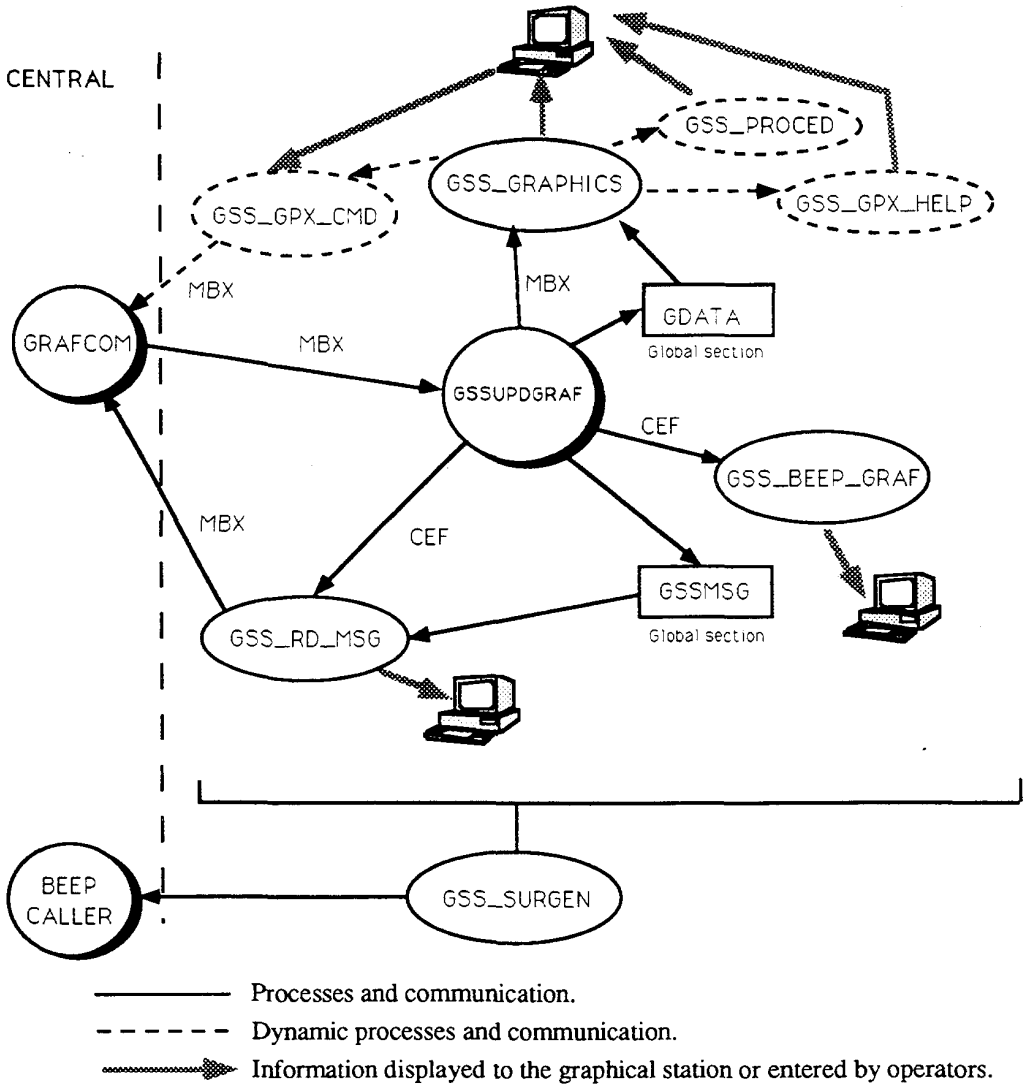

Fig. 18: Inter-processes communication of the graphical interface

2.7.1 The main process: GSSUPDGRAF

The GSSUPDGRAF process receives information from GRAFCOM. Communication between GRAFCOM and GSSUPDGRAF is made through mailboxes. GSSUPDGRAF updates both shared memories GDATA (for images) and GSSMSG (for messages). It also updates the file containing the last 1000

messages (gss_message.msg) and the file containing the active messages (gss_active_alarm.msg). Then it wakes up the processes that are concerned with the information (through a common-event flag or mailboxes).

GSSUPDGRAF also requests a back-up from the GSS central every 15 minutes.

2.7.2 The GSS_GRAPHICS process

The role of this process is to display interactively the view requested by the operator (the list of available views is proposed on a board on the graphics screen). Each element of the requested view is displayed in the colour corresponding to its alarm level (the alarm level is stored in GDATA):

green:	normal state
white:	inhibit
blue:	anomaly
orange:	warning
red:	alarm.

Communication between GSS_GRAPHICS and GSSUPDGRAF is made through mailboxes (for new events).

If the operator requests to see a procedure, GSS_GRAPHICS starts the GSS_PROCED process. If the operator asks for more information by clicking on a graphical object, it starts the GSS_GPX_HELP process. On the operator's request, GSS_GRAPHICS also displays the list of persons physically present inside an experiment. This list is stored in a file updated every 15 minutes (by the LEPEQULACC process).

2.7.3 The GSS_RD_MSG process

The GSS_RD_MSG process displays as a priority the messages corresponding to alarms and warnings, and other messages only upon request (inhibit and anomalies). For this it uses the gss_active_alarm.msg file. If requested, it also displays the last 1000 messages (by reading the GSSMSG shared memory). GSS_RD_MSG allows the operator (Slimos for example) to acknowledge an event by clicking on the message corresponding to it, and by giving a username and a password. The acknowledgement is directly transmitted to the GSS central (GRAFCOM) through a mailbox.

GSSUPDGRAF wakes up GSS_RD_MSG using a common-event flag.

2.7.4 The GSS_BEEP_GRAF process

As soon as a new alarm is received, the GSS_BEEP_GRAF process displays a small red and white flashing window and sends out a beep that can be deactivated by any operator. It also displays another white and black flashing

window each time a new event is received. The blinking of this window can be deactivated by any operator.

2.7.5 The GSS_PROCED process

The GSS_PROCED process is dynamically started by GSS_GRAPHICS upon the operator's request. It displays the procedure corresponding to the element on which the operator has clicked (in addition, since a graphical object can correspond to several alarm levels, the operator must give the desired alarm level). The procedure contains the list of actions that will be (have been) executed by the expert system (see Section 5).

2.7.6 The GSS_GPX_HELP process

The GSS_GPX_HELP process is dynamically started by GSS_GRAPHICS upon the operator's request. It displays more information on the object on which the operator has clicked.

2.7.7 The GSS_GPX_CMD process

The GSS_GPX_CMD process is dynamically started by GSS_GRAPHICS upon the operator's request. It allows the operator to execute remotely (through the graphic) a command. For example, the operator can close a valve, cut off an electrical panel, etc. A username and a password are required.

2.7.8 The \GSS_SURGEN process

GSS_SURGEN surveys the state and the presence of all the graphical processes. In case of detection of a problem, GSS_SURGEN sends a message to BEEP_CALLER (asking it to call the software maintenance beep). It also restarts non-running processes.

2.7.9 VSGLEP: a special graphics station

Contrary to the graphics stations of the experiments that are dedicated to their experiment, VSGLEP is a graphics station (situated in Bldg. 530) that allows one to visualize the graphics of the four experiments. Operators cannot send commands from VSGLEP. The text of the processes is written with the same philosophy, but there are some differences in the code.

VSGLEP can be used to replace a broken graphics station in one of the experiments. Software permutation is easy.

3. HOW WE OBTAIN A RUN-TIME EXPERT SYSTEM

We use the expert system Genesia I, which is very efficient at run time owing to the way the rules are defined. They are explicit expressions without variables to avoid time-consuming condition testing. As a consequence, the number of rules is large.

Writing something like 20 000 rules (in DELPHI) is not very easy, safe or exciting. Moreover, such a rule base is not very legible. Some of the rules are repetitive since they belong to the same family of alarms and because there are no variables. For that reason the run-time system is automatically generated from a more legible higher-level expert system. In this way we keep execution speed and gain legibility.

3.1 Principles and advantages of an automatic generation

The requirements of the GSS system are well fulfilled by the automatic generation of rules. There is a huge amount of data of the same type and there are general security rules. This allows one to use a high-level language with variables (Genesia II) to generate low-level security rules without variables. One general security rule written in Genesia II generates as many sets of low-level rules as there are configurations of sensors corresponding to the security rule.

With this solution, we benefit from both the rapidity of Genesia I-TR for the real-time, and the rich vocabulary of Genesia II to design generative rules. Moreover, it frees the designer from all repetitive tasks.

This methodology is very useful for processing such huge knowledge bases as ours.

System maintenance is easier:

- Some writing errors can be avoided, mistakes are easily recoverable (one change in the generating rule modifies the corresponding set of generated rules).
- If a security procedure changes, only the generating system must be modified.
- If the type or the description of a sensor changes, only the database must be modified.
- Each modification of either the generating system or the database implies an off-line generation.
- Unique source: there is one fact base for each experiment. There is only one generating rule base for the four experiments.

Maintenance of such a system is much easier and more flexible than maintenance of four different and independent systems.

3.2 Schema of the automatic generation

The description of the sensors is stored in an Oracle database. This database contains the complete description of all sensors: low-level information (cabling information), and high-level information (description, location, type, alarm level, etc.). It also contains information on what must be transmitted to other systems (LEP, slow-control, graphics display).

The INTERFACE program transforms Oracle data into a fact base. The operational rules obtained by automatic generation are then compiled off-line before being installed on the experiment's computer (Fig. 19).

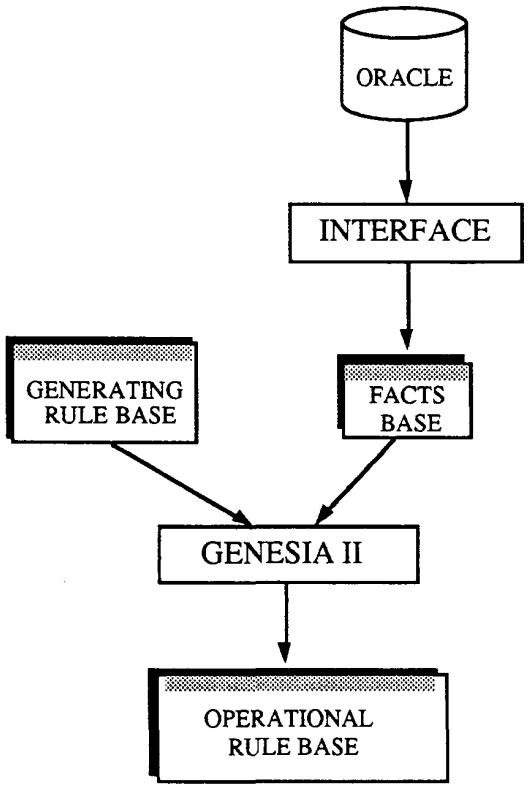

Fig. 19: Schema of the automatic generation

3.3 Compilation and installation of the operational rule base

The compilation of the operational rule base creates an object file (.obj) and an address file (.adr). These two files plus several other files (obtained by running a command file) must be copied to the experiment's computer. The running system is then stopped and the new one can be started.

3.4 Usage of the expert system in the four experiments

Table 1 contains the parameters of the four expert systems on 25 October 1993.

Table 1

	ALEPH	DELPHI	L3	OPAL
Generating expert system				
Number of generative rules	500 RULES			
Number of facts	9 600	27 600	10 260	11 860
Operational expert systems				
Number of operational rules generated	4 800	18 800	5 600	6 850
Number of lines generated	35 800	147 300	42 300	52 450
Number of objects in fact the base	1 800	6 670	1 980	2 560

3.5 What goes on behind the automatic generation?

In the previous paragraphs, we explained the main principles of the automatic generation. In addition, a lot of files needed for the operational functioning are generated. Figure 20 shows the complete generation chain.

The user can generate by himself a new version of any experiment by running five commands. Directions for using these commands can be obtained by typing the command 'SOS' at any moment in the generation account. All the files useful for the target system are generated. The installation of a new system on a target machine is also very simple: the user makes a connection to the required target machine and runs the command, '@install'. He must then stop and start all GSS processes.

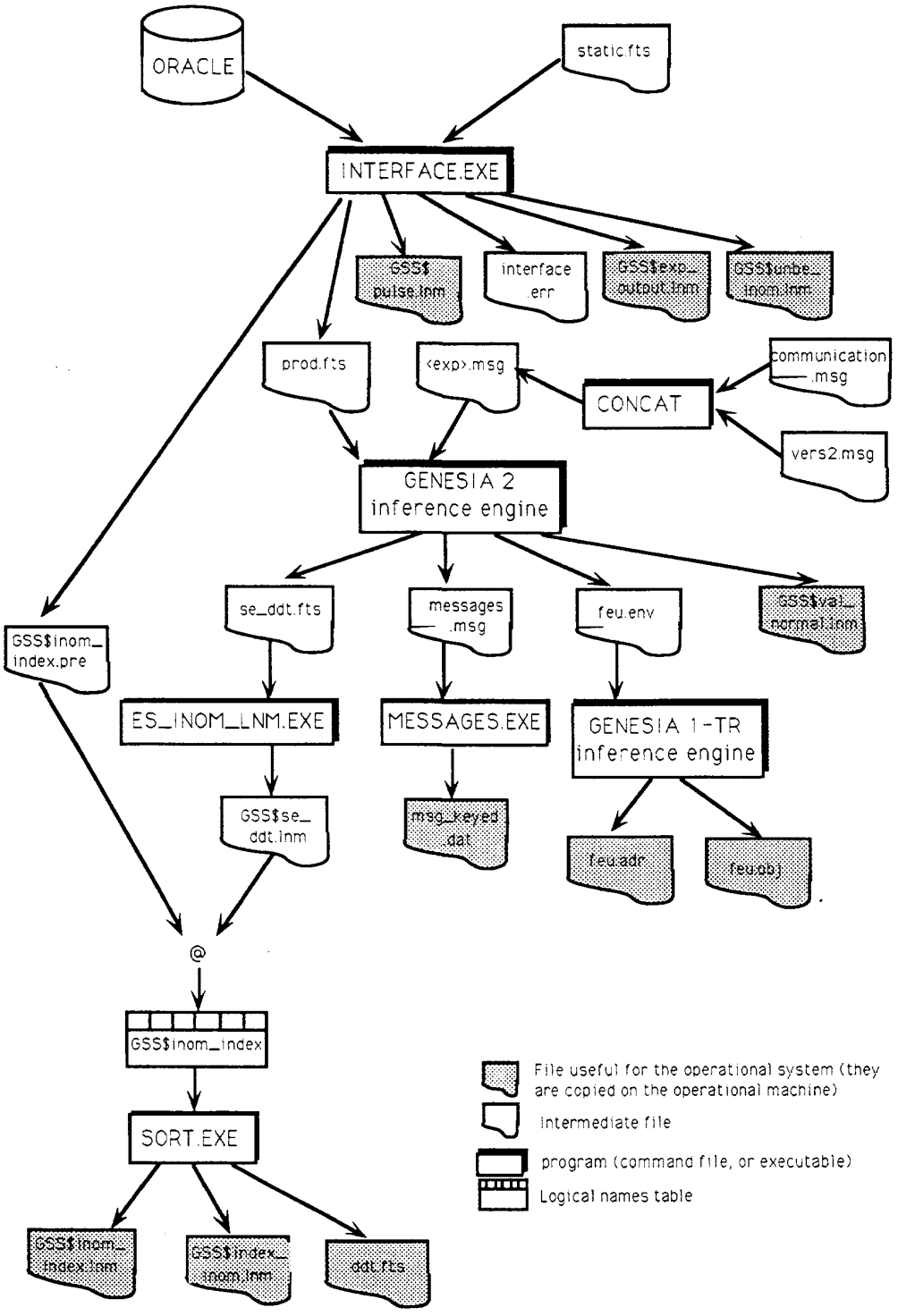

Fig. 20: Files used and generated during the generation stage

4. THE RUN-TIME EXPERT SYSTEM

The run-time expert system processes alarms in real time. Therefore, it must be reliable and very fast. However, rapidity of a system implies constraints:

- It is better to have a large number of variables than a large amount of instanciations: this phase of the inference engine takes a lot of time during execution.
- Compilation rather than interpretation of the rule base allows one to reduce execution time.

The run-time expert system is written in propositional logic with Genesia I-TR (there is no variable instanciation). The rule base is then compiled before being executed.

4.1 Rules

The format is:

```
RULE  name_of_the_rule
IF    <condition>
THEN  <action>
```

4.1.1 The condition part

Example:

```
RULE      CUT_OFF_ELECTRICITY_AND_GAS
IF        [ FIRE_ALARM_P1 = '1'
          AND  FIRE_ACKNOWLEDGEMENT_P1 = 'I' ]
          OR   [  GAS_ALARM_P1  = '1'
          AND  GAS_ACKNOWLEDGEMENT_P1 = 'I' ]
THEN      CUT_OFF_GAS = 'F'
          CUT_OFF_ELECTRICITY = 'F'
```

4.1.2 The action part

A **consequent** can be:

- An entity followed by = and a character in quotation marks ('1' or 'C' for example).
- A $ followed by the name of an external procedure and its parameters in parentheses.

```
$ DISPLAY ( RESULT, 'OK', 12, Z )
```

External procedures are very useful to send or receive results from outside the expert system. External procedures must be written by the expert system developer. They are linked with the inference engine.

4.2 Inferences

The compiled rule base is read only once at the start-up of the expert system.

The run-time expert system is an infinite loop (Fig. 21). The loop continues (a new inference begins) if the shared memory has been modified (if there is at least one new event). If not, it waits for a new event. When a new inference begins, the shared memory is read: the inference engine takes a 'picture' of the shared memory. This picture is the fact base for the current inference.

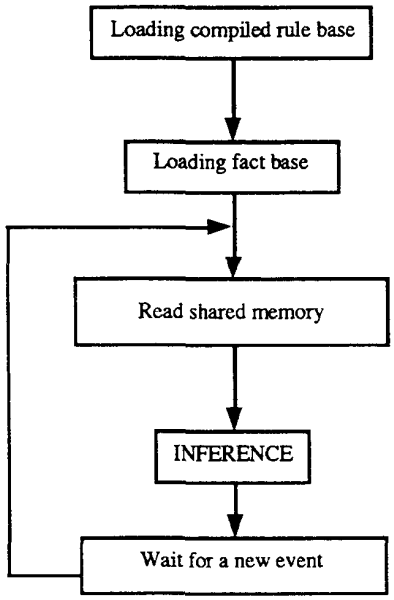

Fig. 21: Logic of the expert system

4.3 The four operational rule bases

Operational rules of each experiment can be divided into five categories:

- Rules concerning the GSS processing of each alarm.
- Rules that send events to the graphics display.
- Rules sending events to the slow-control system (the type of events depends on the experiment).
- Rules that send events to the LEP CAS.
- Rules sending events to ECA.

4.3.1 Rules for GSS processing of alarms

There are three different kinds of alarms: single digital alarms, composite alarms, and single analog alarms. These rules can be separated into three sets:

– Rules corresponding to the GSS procedures

Alarms can be filtered or not. If an alarm is not filtered, the only condition for the execution of the GSS procedure is the arrival of the alarm. If the alarm is

filtered, the execution of the GSS procedure depends on the value of several alarms.

– Rules transforming information into alarms

These rules update the value of the composite alarms (their value cannot be set by hardware), and rules that transform analog values into digital ones.

– Rules making diagnoses

These rules diagnose ventilation problems. Using information about the state of the two ventilation systems (extraction and pulsation), and depending on the existence of fire or gas alarms, the expert system detects abnormal situations. If a problem is detected, the expert system will provide information about the diagnosis process. It will also provide information about what should be the normal status of the ventilation system under the same circumstances.

4.3.2 Rules sending events to the graphics system

Depending on the set of alarms that corresponds to an object in the graphics system, the expert system asks the graphics system to colour this object and/or to display a message. This set of rules is divided into three parts:

- Rules that initialize the graphics system.
- Rules that detect which changes will be made on the graphics screen (messages or plots).
- Rules that send new colours for plots and new messages.

4.3.3 Rules sending events to the slow control

Depending on what the slow control wants to receive (described in the database), the expert system sends a message when necessary.

4.3.4 Rules sending events to the LEP Central Alarm Server

Depending on what the LEP Central Alarm Server wants to receive (described in the database), the expert system sends a message when necessary.

4.3.5 Rules requesting actions from the ECA

These rules request actions from the ECA on equipment, and verify if the action has been executed. If not, they diagnose the kind of problem (no answer from the ECA, no answer from the equipment, no communication, time-out, etc.).

The decision to perform an action is taken by the other rules of the expert system (mainly the rules that correspond to the GSS procedures).

4.3.6 Relations between rules

Some rules can depend on others: a consequent of a rule can be tested in the premise of another one. For example, rules that govern the GSS processing can decide on actions to be executed by the ECA. Other rules (see Section 3.3.5) ask execution from ECA, and diagnose the possible problems of execution.

4.4 Size of the different parts of rules

Table 2 shows the number of lines and rules of the expert system of each experiment, and the percentage it represents on 26 November 1993.

Table 2

		ALEPH		DELPHI		L3		OPAL	
Rules that send events to slow control	lines	0		20 796	14.5%	4 157	10.2%	0	
	rules	0		2 821		588		0	
Rules that send events to LEP Central Alarm Server	lines	2 451	7.4%	2 906	2.2%	2 185	5.6%	2 014	4.2%
	rules	384		456		342		315	
Rules that ask actions from ECA and diagnose execution	lines	1 482	3.5%	12 801	7.6%	2 658	5.4%	7 362	12.1%
	rules	140		1 218		252		700	
Rules that send events to the graphics system	lines	20 383	56.7%	67 464	45.8%	20 531	48.7%	28 404	55%
	rules	2 704		8 600		2 737		3 815	
Rules for the GSS processing of alarms	lines	11 480	32.4%	43 367	29.9%	12 776	30.1%	14 682	28.7%
	rules	1 568		5 696		1 677		2 011	
Total	lines	35 781		147 330		42 303		52 459	
	rules	4 796		18 791		5 596		6 841	

5. THE DATA ENTRY SYSTEM

By 'data entry system' we mean a tool providing facilities to enter and update all the information in the Oracle databases.

5.1 The current data entry system

5.1.1 Description of parameters

The information describing the parameters managed by the GSS are stored in an Oracle database. A basic tool using SQL forms is currently used in the VAX development machine to enter and update information in the database.

5.1.2 Description of the procedures applied to parameters

All parameters managed by the GSS are classified according to the type of problem they are describing (gas, water leak, fire, smoke, etc.) and their alarm level.

This classification is used by the GSS to determine how they should be processed. The set of actions to be performed by the GSS in order to process the parameters are described in the safety procedures provided on paper support by the people responsible for the safety in the experiments. These procedures are transformed into rules and introduced manually into the system by a knowledge engineer.

5.2 The future AGATHE data entry system

The current system leads to two major problems:

- The SQL form tool used to populate and update the database is not strong enough to maintain consistency of the database. For example, the description of a parameter can be partial, there is no control on the content of some fields, there is no consistency between content of the four databases (we would like to have an homogeneity in the description of similar parameters).
- Procedures are stored on paper. If there is a new procedure or a modification of one of those procedures, an expert must modify manually the generating expert system.

To avoid these problems, a data entry tool (AGATHE) has been designed and partially implemented.

5.2.1 *Description of parameters and procedures*

The aim is to integrate both parameter description and procedure description in the same database, and to access it using the same tool AGATHE. Specifications of AGATHE can be found in Ref. [4]. AGATHE is currently being implemented using the 4D relational database on Macintosh.

AGATHE will provide user-friendly interfaces, syntax and consistency checks and a unified way of accessing the four databases.

5.2.2 *The generating expert system*

In order to minimize the role of the expert and since the description of the procedures will be stored in Oracle, the generating expert system must be rewritten. Instead of having a generating expert system containing the details of the procedures, we will have a general generating expert system able to operate on the content of the procedures in the database.

Therefore, if a procedure changes, the modification must be done through AGATHE in the database, and the generating expert system must not be modified.

The generating expert system will be modified only if, for example, a new kind of action is created in the procedures. The probability of that is very low, and the maintenance will be much easier. Users will be able to add or modify procedures without requiring the intervention of the expert.

The new generating expert system has already been written, but must be tested. The main difference from the previous generating system is that it generates temporal rules. Therefore, the temporal version of Genesia I-TR will be run on the experimental computers. This version has previously been tested with success, but it has never run in practice.

6. STRENGTHS AND WEAKNESSES OF THE PRESENT GSS SYSTEM

GSS was designed in the 1980s. Therefore, it is based on technologies that will or could become obsolete. Meanwhile, some concepts and facilities are still relevant, and should be kept if it is decided to upgrade GSS. In this section we try to list the strengths and weaknesses of GSS, considering the developer's point of view as well as that of the user.

6.1 Strong points

- The same processes run on the four experiments (except the process EXPCOM communicating with the slow controls of the experiments). This implies a **unique** source of modifications in the development machine, and a centralized version handling.
- The expert system is **automatically generated**. Persons without software experience can generate and install new versions of the operational expert systems whenever they need to.
- The industrial expert system used (Genesia) is a very open software. It can easily be connected to other systems such as acquisition systems, databases, man–machine interfaces, graphics packages, etc. This product is supported by AIIT, an English company.
- As explained in the previous two points, the **genericity** of the system is very strong. It implies a **software maintenance** cost divided by four (compared with four independent systems). It also implies a minimal cost if a GSS system had to be installed on a new site or a new experiment.
- All the facilities provided by GSS to end-users must be kept. It is important, for example, to keep the same philosophy, and exactly the same layouts for the graphics package.
- Some facilities provided to operators through the operator's access package are very easy to use: read details of active alarms, authorize inhibition, mask sensors, interact with the ECAs (display, modify, and mask the content of the ECAs' tables), and interact with GSS processes. All these functions can be accessed from any VT100-type terminal.
- The possibility to **perform remote actions** on equipment by clicking on an object on the screen must be kept. Although it is currently not possible to access all the equipment, it may be possible in the future (everything is ready for this on the GSS side).

6.2 Weak points

- The **graphical package** has been written with **DICE**, a home-made software tool. It runs on VAX machines using **UIS**. Unfortunately, UIS will no longer be supported by DEC. DICE was a satisfactory tool at the beginning of GSS, but now, compared with other existing tools, it seems to be rigid and difficult to use by inexperienced persons.
- Most of the **communication processes** use **UPI** (user program interface), a library of task-to-task communication and menu-handling modules designed by the ALEPH experiment team. Unfortunately, the ALEPH upgrades to UPI have not been followed and therefore there is no more support. Moreover, it is not completely reliable. Some random problems appear sometimes in the behaviour of the UPI procedure calls. Therefore, some of the GSS processes that use UPI contain lines of code that overcome (but do not solve) these problems.
- The **operator's access package** is also written using UPI. It is very frequently used by operators, but is not user-friendly.
- The **expert system** requires the intervention of an expert each time there is a modification to the procedures (see Section 5). Although it is a generic system that minimizes the number of interventions (none of the modifications possible in the database require the intervention of the expert), it should be more generic especially for procedure changes.
- The entire GSS system runs on **VAX** machines running VMS. This is not an advantage since UNIX is becoming more and more important at CERN and elsewhere. Moreover, UNIX is an open system as opposed to VMS which is a proprietary system. UNIX may be preferable since it is cheaper, a lot of commercial software runs on it, and it is really becoming a standard.
- The diversity of **communication protocols** used is also a weak point. Communications use RPC and DECNET, Ethernet buffers, mailboxes, shared memory, etc.
- The **upgrade of OS-9** in ECAs is difficult. An upgrade from OS-9 2.3 to 2.4 implies the modification of all the EPROMSs.
- The size of the memory of the graphical stations is not sufficient, it should be increased.
- The parameters used by the ECA software are stored on a PC machine and are edited with MS-Access. Therefore, there is no link between the GSS Oracle database and this editor tool. This part should be provided by the AGATHE data entry system that could create or modify the Parameter_Table according to the GSS database.

Acknowledgements

We thank W. von Rüden for suggesting to us to write this document and the ECP-LI group for helping with technical questions.

References

[1] C. Dechelette, E. Sbrissa and J.-M. Schmitt, Unite d'acquisition et processing satellite - interconnexion d'entrée-sortie, GSS note 5, EF-LI/076N/ES/lp, 1986.

[2] M. Brolli, C. Dechelette, E. Dutruel, E. Sbrissa and J.-M. Schmitt, Le réseau de sécurité de GSS, GSS note 3, EF-LI/031N/ES/lp, 1985.

[3] F. Chevrier, GSS documentation, Working document, January 1993.

[4] L. Vinot, AGATHE, a GSS acquisition tool, Working document, February 1993.

APPENDIX A

MESSAGES SENT AND RECEIVED BY THE ECA

Messages can be of three types:

Messages from ECA to GSS central
Messages from GSS central to ECA
Internal ECA messages

They are defined in Table A.1. The 'T' time has not been implemented.

Table A.1

ECA–GSS Central

Code	Description	Format
00	Status changed	CODE;T;B;E;ID T = Time YY MM DD HH MM SS 6 bytes ID = Ident. number of process 4 bytes
02	End of process	CODE;T;N;B;E;ID
20	Credibility	CODE;T;N;E
30	Power status	CODE;T;NOVRAM_OLD 41 bytes
60	Messages	CODE;T;TYPE;+... TYPE : 00 - Wrong input N;B 10 bytes 01 - Wrong module N;B 10 bytes 02 - Action I/O executed N;F;D 11 bytes 03 - Status of PGM test N;PGM;STAT 11 bytes PGM = Program number STAT = OK;00 noOK: error number 04 - Spurious interrupt 8 bytes 05 - Wrong code UPS 242 bytes 06 - Wrong code Central 8 bytes 07 - Checksum error 8 bytes 08 - Wrote internal input 8 bytes 09 - Safety loop opened NBR 9 bytes 0A - Ask action from Central 11 bytes
70	Startup	CODE;NOVRAM_NEW 16 bytes

GSS Central–ECA

Code	Description	Format
80	Write virtual input	CODE;N;B;VALUE
81	ACKnowledge	CODE;N;B;VALUE VALUE = 00 or 01
82	Action I/O	CODE;N;F;D
83	ACK	CODE;N;F;D
84	Write module type	CODE;N;TYPE (not yet implemented)
85	Module type changed	CODE;N;TYPE (not yet implemented)
86	Close safety loop	CODE;NBR
87	Loop closed	CODE;NBR
88	Open safety loop	CODE;NBR
89	Loop opened	CODE;NBR
90	Read stack analog module	CODE;N:B;ID
91	Answer value	CODE;N;BYTE;..bytes.. (128 bytes maxi)
92	Kill process	CODE;N;B:ID
93	Process killed	CODE;N;B;ID (ID = 0: process does not exist)

GSS Central-ECA (cont.)

Code	Description	Format
A0	Read element of array	CODE;N_T;N_E
A1	Answer	CODE;N_T;N_E;VALUE N_E = Element number N;B
A2	Read array	CODE;N_T
A3	Answer	CODE;N_T;VALUE
A4	Write element of array	CODE;N_T;N_E;VALUE
A5	ACK	CODE;N_T;N_E;VALUE
A6	Write array	CODE;N_T;VALUE
A7	ACK	CODE;N_T;VALUE
B0	Mask an input	CODE;N;B;MASK
B1	Input masked	CODE;N;B;MASK Unmask = 1 Mask = 0
C0	Inhibit a module	CODE;N;VALUE Bit 0: real module Bit 1: virtual module value = 0: non inhibit 1: inhibit
C1	ACK	CODE;N
C2	Set time	CODE;T (not yet implemented)
C3	Date & time	CODE;T (not yet implemented)
D0	Get status	CODE
D1	Status	CODE;STATUS
D2	Start PGM test	CODE;N;PGM (not yet implemented)
D3	PGM test started	CODE;N;PGM (not yet implemented)
D4	Read NOVRAM_OLD	CODE
D5	Answer	CODE;+ 35 bytes
D6	Read NOVRAM_NEW	CODE
D7	Answer	CODE;+ 10 bytes
D8	Write NOVRAM_NEW	CODE;+ 10 bytes
D9	ACK	
E0	Write EEPROM	CODE;Protoc. S1 S9 (not yet implemented)
E1	ACK	CODE;Status S1 S9 (not yet implemented)
F0	End of update	CODE;

List of array

N_T	Array name	Elem/array (byte)	Perm	Table name
00	Virtual	1 / 30	R/W	ADDRESS
10	Mask	1 / 30	R/W	DEFMAS
20	Default	1 / 30	R/W	DEFMAS
30	Datastable	1 / 30	R	DATASTABLE
40	Tolerance	1 / 240	R/W	ACQUISITION
50	Parameter	112 / —	R/W	PARAMETER
60	Equipment	1 / 30	R	ADDRESS
70	Inhibit	1 / 30	R	ADDRESS

Internal messages

Code	Description	Format
10	Action I/O bus	CODE;N;F;D N = Module number F = Function D = Data
12	Action Central expected	CODE;N;B;E E = Edge
40	Write internal input	CODE;NBR;B;E Virtual input for chaining
50	Open safety network	CODE;NBR;0;0 NBR = Loop number 01: FEU 02: GAZ 04: EAU 08: GEN 10: SYS

Code	Description	Format
F1	Update aborted	CODE
FC	Watch_cat message	CODE;+ FACE FE
FD	Get mailbox index	CODE
FF	Empty code	CODE;FF;FF;FF