CERN 84-16
Proton Synchrotron Division
20 December 1984

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

SYSTEM SOFTWARE OF THE CERN PROTON SYNCHROTRON CONTROL SYSTEM

B.E. Carpenter, R. Cailliau, G. Cuisinier and W. Remmer

GENEVA
1984

RN — Service d'Information scientifique — RD/655 - 2200 - Décembre 1984

ABSTRACT

The PS complex consists of 10 different interconnected accelerators or storage rings, mainly controlled by the same distributed system of NORD-10 and ND-100 minicomputers. After a brief outline of the hardware, this report gives a detailed description of the system software, which is based on the SINTRAN III operating system. It describes the general layout of the software, the network, CAMAC access, programming languages, program development, and microprocessor support. It concludes with reviews of performance, documentation, organization and methods, and future prospects.

FOREWORD

This report describes the system software of the control system for the CERN 28 GeV Proton Synchrotron (PS) and other machines of the PS complex, as at mid-1984.

It is intended as a general report, as a reference document for programmers and system supervisors, and as a background document for end-users. An intending user should also obtain a complete copy of 'COOKBOOK' (see Section 12).

The present report supersedes the following CERN internal notes and incorporates their material as appropriate:

PS/CCI/Note 78-8 Rev.
PS/CO/Note 80-21
PS/CCI/Note 78-10 Rev.
PS/CO/Note 80-1

CONTENTS

1. THE CERN PS CONTROL SYSTEM

The PS complex consists of the following equipment:

Linac I	Old linear accelerator
Linac II	New linear accelerator
PS	28 GeV Proton Synchrotron
PSB	Proton Synchrotron Booster
AA	Antiproton Accumulator
LEAR	Low-Energy Antiproton Ring
TT	Transfer lines
LIL	LEP*) Injector Linacs (under construction)
EPA	Electron-Positron Accumulator (under construction)
ACOL	Antiproton Collector (planned).

This complexity, with the associated multiplicity of beam types and beam destinations, creates an equally complex controls problem. The cycle times of the accelerators are of the order of 1 s. The 'supercycle', comprising several individual cycles feeding several destinations, is of the order of 10 s. However, periods of up to 24 h must be taken into account for accumulator operation sequences.

It became apparent in the early 1970's that the old, only partly computerized, control system was inadequate for the complexity of the machines. A Divisional decision was made in favour of the PDP-11 as the standard minicomputer, and several disjoint controls projects (new Linac, ejections and beam transport, PSB measurements) based on PDP-11s were implemented in the mid-1970's. At the same time, studies began for an integrated control system for the PS complex, and in 1976 the Controls Group tabled its design study proposing a distributed control system based on PDP-11s [1]. The Management of CERN approved the idea of a new control system and funded the project, but subject to the use of European Norsk Data (ND) computers and to the re-use of components of the Super Proton Synchrotron (SPS) Control System [2] where possible [3]. The Controls Group then spent somewhat more than a year familiarizing themselves with ND hardware and software, and on a pilot project using a precise miniature copy of the SPS system. The detailed design of the system described below thus started in late 1977; the system went on-line to the AA in the summer of 1980, to the PSB in October 1980, and progressively to the PS and TT up until early 1984. (The Linacs remained on their dedicated PDP-11s and LSI-11s, and LEAR was added to this latter system in 1983).

The PS Control System was designed at the time when minicomputers with 16-bit addressing and not more than 0.5 mega instructions per second (Mips) speed were the norm. Neither 32-bit super-minicomputers in the 2 Mips class nor microprocessors with traditional minicomputer performance were available. In this context, it was normal to opt for a 'democratic' distributed system of minicomputers, with no a priori hierarchy or centralization.

A fundamental request of the Operations Group was for general-purpose consoles, as at the SPS, capable of working from minute to minute on any part of the PS complex. Linked with this was a request for multiple parallel functions on the consoles, specifically for simultaneous interactive access to several independent displays, analog and video signal multiplexing, and alarms. These requirements dictated the decision to have an independent and powerful minicomputer for each console.

The limited geographical extent (about 1 km) of the PS complex, and its neat division into logical entities, led to the decision to dedicate a front-end minicomputer to each such entity or process. These are normally referred to as process computers or front-end computers (FECs), and are named after the process that each controls (PSB, CPS, etc.). They access the process by Serial CAMAC, in some cases via microcomputers configured as CAMAC auxiliary crate controllers (ACCs).

An outline of the topology is given in Fig. 1, and full details are given elsewhere [4]. In addition to the set of virtually identical console computers, and the set of FECs, three additional computers with central roles should be mentioned. Firstly, there is the Main Control Room (MCR) computer, configured as a console computer, but dedicated to shared resources of the Control Room such as printers. Secondly, there is the TREES computer, configured as a FEC, and responsible for administration of the various logical tree structures through which the operators access the processes. Thirdly, there is a large program development computer.

Within this topology, a multi-layered modular system of applications software has been constructed to implement the required supervisory control system (for example, Daneels [5]). Starting nearest to the process CAMAC hardware, the various layers are as follows:

HRT	Hard Real-time Tasks (in ACC or FEC, tightly tied to real-time events)
IM	Interface Modules (hide CAMAC details)
EM	Equipment Modules (hide equipment details)
CVM	Composite Variable Modules (handle coupled equipments)

*) Large Electron-Positron Colliding Beam Machine.

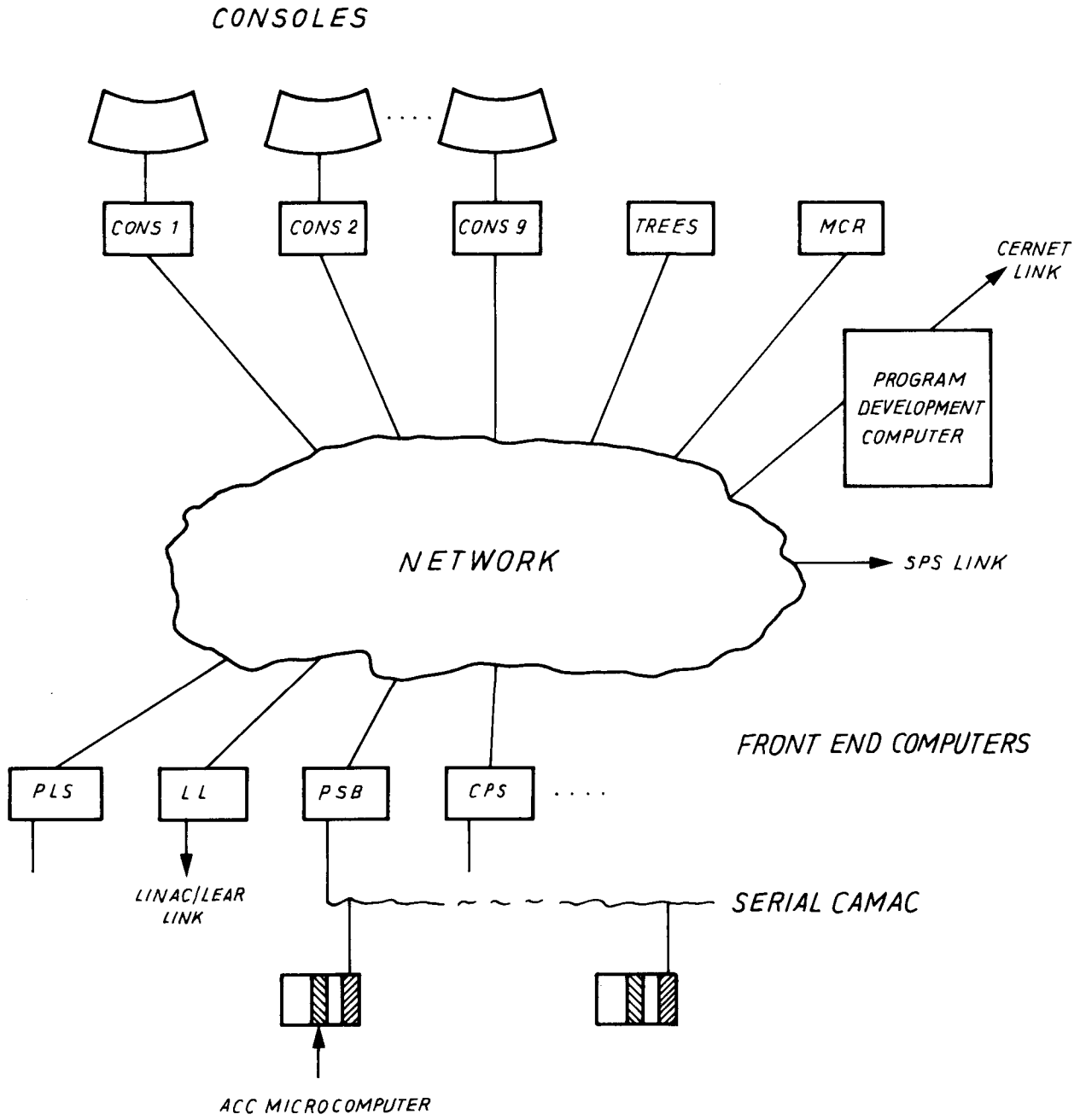

Fig. 1 Computer topology of control system

{	NODAL	Interactive Programs (local interaction with FEC)
{	PCP	Process Control Program (handle complete control sequences)
	OM	Operator Modules (handle operator interaction with console).

These layers are each composed of similar modules confined to one computer. They are supported and supplemented by general quasi-system software (nevertheless written by the applications teams) including:

	MIP	Main Interactive Program (kernel running OMs)
	TIP	Tree Interactive Program (manages tree for selecting OMs)
}	SIP	Analog Signal Interactive Program
}	VIP	Video signal Interactive Program
}	LIP	Alarm Interactive Program
}	--	Knob software (connects knobs to equipment)
}	--	Display pipeline software (routes data from equipment to displays).

(Manage other trees)

This general application software is distributed across all computers.

It should be noted that the IMs, EMs, and CVMs are *passive* modules, i.e. subroutines, whereas all other modules are *active* programs. Active programs (other than HRT programs) are collectively referred to as soft real-time (SRT) programs, since they all have similar characteristics despite their varied functions.

An important subsystem is the Program Line Sequencer (PLS), consisting of a dedicated computer and serial transmission system distributing 256 bits of beam sequencing information in real time [6].

Thus the total control system consists of about 1000 software modules (written by at least 80 different application programmers), distributed across about 20 minicomputers and 100 microprocessors, the whole representing more than 100 man-years of programming effort. Through 7 essentially identical consoles, the operation team controls about 10000 machine parameters, most with one or more set-points and acquired values, a status word, and activations (control bits). In addition, perhaps 100 technicians, engineers, and physicists use the system to some extent for tests and maintenance. The targeted over-all availability, after accounting for all sources of failure in the control system, is 99%.

2. GOALS AND OUTLINE OF SYSTEM SOFTWARE

2.1 Goals

The system software of the control system has as its primary goal the provision of the infrastructure for all forms of applications software. To a close approximation, this means that the system software must take account of the real-time constraints of the control system, but should contain no specific dependencies on the nature of particle accelerators or accumulators. This is not an academic distinction: only by this means can one be sure that the system software is general enough to cope with totally unpredictable future demands. (A prime example: when the system design started in 1977, we only had to consider pulsed accelerators. Yet only a handful of library routines were added to cope with the AA storage ring, which in fact was the first machine on-line).

The second major goal of the system software team was to provide software development facilities for applications programs. The importance of this in a 100 man-year project can hardly be understated.

2.2 Principles

A number of general principles were consciously adopted throughout the project.

i) Existing software was used wherever appropriate, the ultimate reason for this being to save manpower. Active participation in the NORD Computer Users Society (NOCUS), and frequent CERN-wide contacts, were essential adjuncts to this policy. However, it was in several instances more appropriate to develop new system software that would be better adapted to the whole project than what was then available.

ii) It was crucial to support a range of users from 'casual' (e.g. CAMAC technicians, or machine physicists) to 'professional' (i.e. full-time applications programmers). This is not to imply a valued judgement of their respective merits, but simply to state a range of needs. Some software in an accelerator control system can be temporary and does not need to be especially well designed, coded, or documented; other software may live ten years and must conform to the highest standards of software engineering. The system had to cater for the entire range.

iii) The system software had to find the right compromises between speed, reliability (protection), and facilities, for a given set of computers and systems programmers.

iv) The project history was such that the users of the system software were not able to state their requirements until it was too late, i.e. after implementation had begun. Thus, the system team had to keep ahead of the users' requirements, but on minimal input. This was achieved mainly by documenting *before* implementing, and forcing discussion at that stage.

v) In a similar vein, the system software team has to continuously look ahead to assess future requirements and the impact of future technology.

vi) Wherever possible, good software engineering has been applied. Such techniques are not the main point of this report, but their importance should not be neglected.

2.3 Constraints

The major constraints on the system software should be briefly stated.

The choice of NORD-10 and ND-100 minicomputers was made by the CERN Management, and TMS9900-based CAMAC microcomputers were chosen as the best available (and were a CERN standard at the time).

The cycle time of the PS complex being potentially as small as 650 ms, there was no alternative to the decision that all time-dependent applications software must be compiled rather than interpreted.

The PS complex being in 24-hour production most of the year, we were forced to treat the planned shutdowns as unmovable deadlines (despite serious personnel shortages).

2.4 Outline of implementation

Subject to these principles and constraints, a large set of system software facilities was installed or implemented by the system team. In summary:

- support for interactive or autonomous, compiled or interpreted applications programs or modules of various types;
- remote execution and file transfer on network;
- flexible file system;
- language mixing;
- microprocessor facilities;
- debugging aids;
- development aids, standards, documentation, education;
- libraries, utilities, operating system extensions, and so on.

Major options in the implementation were:

- the manufacturer's operating system SINTRAN III/VS chosen for its facilities and support;
- the CERN/SPS network chosen for its speed;
- the CERN/SPS NODAL interpreter chosen for its convenience;
- the manufacturer's systems language, NORD-PL, and TMS9900 assembler, chosen for systems or hardware-oriented work;
- Pascal and P^+ implemented for the sake of reliability and maintainability of applications programs.

It should be noted that Pascal is rather unsuited to large sets of interdependent software modules. An important development was the introduction of P^+, a Pascal derivative philosophically akin to MESA and Ada, providing syntactically controlled separate compilation and other powerful features. A 2000-line program in P^+ has been known to run first time.

The remainder of this report describes these various aspects (although not in the same order). More technical material is covered in the Appendices.

3. COMPUTERS AND OPERATING SYSTEM

3.1 The NORD-10 and ND-100 computers

The NORD-10 computer was introduced by Norsk Data A.S., Oslo, in 1973, as successor to the much simpler NORD-1. The NORD-10/S, almost identical but allowing cache memory, was manufactured from 1977 until superseded by the ND-100 in 1980. These machines are strictly forwards-compatible for applications programmers (i.e. a NORD-10 binary program will run unmodified on an ND-100). The ND-100 has a few extra instructions and a different range of peripherals, which means that system software is not in all cases either forwards or backwards compatible.

The basic architecture of this family of computers is an accumulator-based, single-address central processing unit (CPU) with 16-bit data and addresses [7-9]. Addressing is word-based, thus doubling the virtual memory size compared with 16-bit byte-addressed architectures such as the PDP-11. The accumulator (A-register) is supplemented by several other special-purpose registers visible to the programmer, including in particular the B-register (base register, normally used as the stack frame pointer). There is no stack pointer, but this is a minor disadvantage compared to the advantage of having a stack frame pointer.

In addition to all necessary instructions for handling 16-bit two's-complement integers, the family has (unlike the PDP-11 family) uniform 48-bit floating-point instructions. (More recently, 32-bit floating point has become the preferred option, but the SPS and PS Control Systems are tied to the 48-bit format).

Hidden from the applications programmer, the architecture has 15 hardware priority levels, each with a full set of registers to reduce level-switching time. It also has a sophisticated memory management system, allowing a distinction between instruction and data space in virtual memory and embodying 4 sets of 64 page index registers (PIT 0 to PIT 3) used for mapping 16-bit virtual addresses onto 18 or 24-bit physical addresses. Individual pages may be protected, and the latest ND-100 models incorporate 16 sets of page index registers, but neither of these features is fully supported by the current software.There is also a 'ring' protection system, allowing privileged instructions.

A priority interrupt scheme (including internal interrupts such as those from the memory management system) and dedicated input/output (I/O) instructions accessing an internal or external I/O bus complete the architecture.

An ND-100 may serve as the front-end processor to an ND-500 CPU; the ND-500 is a powerful 32-bit superminicomputer introduced in 1980, comparable to but faster than a VAX-11/780 [10].

3.2 The SINTRAN III operating system: general description

The operating system for all ND computers in the control system (with one exception, see Section 5) is SINTRAN III, developed and supported by ND. Prior to 1974, ND offered two operating systems, designed for the NORD-1 but

usable on the NORD-10: a real-time system called SINTRAN II and a time-sharing system known as TSS. With the arrival of the NORD-10 and consequent availability of memory management hardware, a new 'multi-mode' system intended for both real-time and time-sharing, and known as SINTRAN III, was in a pre-release phase during 1974–75. Those with knowledge of SINTRAN II and TSS understand that some of the less explicable features of SINTRAN III are in fact either conceptual leftovers of the earlier systems, or compatible facilities for software portability. (Examples: the rather primitive set of 'real-time' monitor calls are directly derived from SINTRAN II; there are some undocumented file system monitor calls which are identical to TSS calls).

A number of usable releases of SINTRAN III appeared in 1976–77, although ND has now curiously re-baptised these as 'pre-releases'. Since 1978 there have been one or two releases per year; the I release is thus the ninth official one, but in reality the twentieth or so.

The best general description of SINTRAN III is unfortunately in an obsolete manual [11]. It consists of the following major components:

- resident kernel (handles interrupts, multiprogramming, timeslicing, system calls, and memory management),
- resident I/O drivers,
- file system,
- command processor,
- batch processor,
- 'real-time' loader.

The following sections describe these aspects as they stand in 1984, but considerable internal changes are expected in future releases.

3.2.1 Resident kernel

i) Interrupts, if internal, are routed directly to the appropriate part of the kernel (clock handler, memory management, error handling, system call handling). If external, they are routed via appropriate tables and data structures to the device driver code. In most cases, interrupts are treated with the CPU running on the corresponding hardware level; thus latencies for relatively low-priority interrupts (e.g. CAMAC) can be of the order of several hundred microseconds.

There is no convenient provision for user-written interrupt service routines. Such routines, if required, must be patched into the system kernel at a system-dependent absolute address, and activated by patching the system's interrupt identification tables.

A special facility related to interrupts is the 'direct task'. Some of the CPU hardware priority levels are normally unused. Memory resident user code may, for special applications, run on such levels as direct tasks. They are activated either by an interrupt from a user device or by executing a suitable privileged instruction.

ii) Multiprogramming is based on the concept of a 'real-time program description' (RT description). An RT description is a data item (known as a Task Control Block in most operating systems), which describes the characteristics and status of a task (known as an 'RT program'). The major information stored in an RT description can be summarized as follows:

- status bits,
- priority (0–255, 0 is lowest priority),
- scheduling information,
- start address,
- initial segment numbers (see memory management),
- saved registers when task is inactive,
- current segment numbers (see memory management),
- links in various system lists.

Any change to the state of any RT description, and any decision to switch execution from one RT program to another, must be carried out by code running on hardware level 3 (the 'monitor level'), which is used only for this purpose. The monitor is always activated by code running on some other hardware level linking a data item into a system list called the 'monitor queue' and then activating level 3 by a suitable privileged instruction. This ensures that the total set of RT descriptions always remains in a consistent state.

The monitor works as a preemptive priority scheduler. Among the significant events causing the monitor to search the execution queue for the highest priority program to run are:

- system calls by any program,
- start and end of transfer in device drivers,
- real-time clock events,
- memory management disc transfers.

No system calls for dynamic task creation or installation are provided. The maximum number of user RT programs is defined at system generation and cannot be exceeded; a relatively small number (a few tens) is normal to avoid excessive space requirements. Within this constraint, user RT programs may be created and destroyed only from a time-sharing terminal (using the 'real-time loader', see below).

iii) Timeslicing is performed by a dedicated high priority RT program built into the system nucleus. It adjusts the priorities of the programs to be timesliced on a periodic basis. Timeslicing applies only to so-called 'background' programs, i.e. those attached to each terminal, remote terminal, or batch processor. A user's other real-time programs are outside the control of the timeslicer.

iv) System calls are known as 'monitor calls', because they are intercepted and in some cases executed by the level 3 monitor. In other cases, where the monitor call may take too long to execute, the monitor switches control to level 1 for the actual elaboration of the system call. In this case, a higher priority RT program may in theory pre-empt the CPU during treatment of the system call.

The system calls may be divided into three sets: those available to all programs, those available only to background (timesliced) programs, and those available only to RT programs. This differentation between background and real-time facilities is an enormous inconvenience.

v) Memory management is based on the concept of 'segments'. A segment is a contiguous zone of virtual memory somewhere in the range from address 0 to address 65535. By software convention, segments are numbered from 0 to 255. Segments are of two types: 'demand' and 'non-demand'. Demand segments are divided into 1 Kword pages (up to a maximum of 64), of which between none and all may be in memory at a given instant. The page index table(s) in use by the currently running RT program are set up by the memory management software so as to point only to the resident pages of whichever segment(s) the program is currently using. The current segments are indicated in the RT description. If the program generates an effective address within a resident page, the memory management hardware translates the virtual address to a physical address. If not, a 'page fault' interrupt is generated and control passes to the memory management software running on level 3. This will initiate the necessary disc transfer(s) to swap in the required page, during which time other programs may run.

Non-demand segments are swapped in and out of memory in their entirety. In a real-time system with many context switches between tasks, this leads to excessive disc activity. However, non-demand segments may be permanently fixed in memory, which is useful for very time-critical tasks.

Any demand-page segment may be attached to a program as a 're-entrant' segment, with the characteristic that pages which are never modified exist in one copy, but modified pages exist in one copy per program. This is useful for shared libraries.

It should be stressed that the concept of a segment (an automatically swapped piece of virtual memory) is quite orthogonal to the concept of an RT program (an independent task or process). At a given instant, an RT program may have not more than two current segments plus a third 're-entrant' (shared library) segment. However, these segments may be logically replaced via system calls; this is analogous to overlays in a traditional system. One segment may be simultaneously used by many RT programs and in general the user is responsible for re-entrancy protection.

3.2.2 Input-output drivers

These are based on the concept of a 'datafield'. This is a record containing standard items (such as an indication of which RT program has reserved the device) and device-dependent items. It is used as re-entrant work space by the driver (thus one driver can serve many identical devices) and for communication between interrupt service routines and driver code running at monitor level (3) or RT program level (1). Pseudo-datafields are used for various other purposes, including queuing requests for the monitor.

Drivers are not loadable. They must either be included with the system nucleus when it is generated by ND, or they must be subsequently patched in at a system-dependent absolute address. Similarly, user datafields must be patched in, and the system's logical unit number tables must be patched to point to such datafields. There is no documentation and no uniform procedure for writing device drivers.

3.2.3 Resident memory

This is the absolute memory area, starting at address zero, which contains all of the above. Up to the first 36 Kwords are normally mapped one-to-one by PIT0 (page index table 0), which is reserved for the system. The remaining part of resident memory (up to absolute address 65535) is only accessible while the memory management system is disabled ('paging off mode'). The resident code is divided between these two regions on a pragmatic basis.

3.2.4 File system

This is a set of demand segments containing re-entrant code, which is called via the monitor when any RT program issues a file system call. This code runs on PIT 0, similar to the resident system (to which it needs access),

and thus the file system segments start above virtual address 36 Kwords.

It implements a single-level directory on each disc, with up to 256 users having up to 256 files. Files can be contiguous, or 'indexed' (expandable) and can be accessed in byte-sequential or random-record modes. There is no record management system, and all types of disc are treated identically.

3.2.5 Command processor

This is the user interface to the 'background' system. After logging in via a local or remote terminal, the user interacts with the command processor to call background programs (or to access the real-time system). The command processor is implemented, similarly to the file system, as re-entrant code which runs on PIT 0 for access to the system kernel. Each terminal also possesses a 'system segment' of a few Kwords, used as work space by the command processor. (In fact, part of the system segment is also used by the file system. For this reason, there is a single resident pseudo-system segment used during file system operations for all RT programs, which are therefore executed one at a time by use of a semaphore.)

The command processor presents an extremely convenient user interface (English-like commands, abbreviation possible and consistent, parameter prompting, defaults, etc.). On the other hand, it in no sense provides a full job control language. It provides a simple facility for processing command files ('mode' files).

In addition to the system segment, each terminal owns a 64 Kword segment in which the user may load and execute programs (known as background programs, restricted to one at a time and to one segment). This may be viewed as a virtual ND-100 dedicated to the terminal user (but privileged instructions are forbidden). In larger systems, the segment may be 128 Kwords long, allowing the running of programs with disjoint instruction and data spaces ('two-bank programs'). The background segment is normally mapped by PIT 2 (PIT 1 from the I release); a second PIT is used in two-bank mode.

3.2.6 Batch processor

This is in fact an extension of the command processor allowing jobs to be run independently of any real terminal and at slightly lower priority. In general, the facilities are identical to those for the normal command processor, with a few annoying differences.

3.2.7 'Real-time' loader

This is misnamed, and should be called the 'segment loader'. Unlike the conventional loader used for background programs, it is capable of creating and destroying segments and of loading code into any segment (including parts of the system itself). Like the file system and command processor, it is itself re-entrant code on a PIT 0 segment; it may only be called by privileged background users, to whom it appears as an extension of the command processor.

The default page index table for user segments is PIT 1. PIT 0 may be used for very special cases; PIT 2 is reserved for background. PIT 3 was formerly reserved for direct tasks (which are loaded into segments and then fixed in memory), but from the I release of SINTRAN III it is used by the system.

3.3 System generation and patching

The resident part of SINTRAN III and some of the standard virtual memory segments must be configured according to the hardware and software options required. This process, known as system generation, is normally carried out by ND on request by the customer, and thus tends to be a rare event. Certain parts of the system are invariant; these are kept in absolute binary format. The configurable parts are kept in the form of assembler source code containing many conditional assembly marks. A system generation thus consists of a complex batch job which re-assembles what is necessary, combines the result with the invariant parts, and produces absolute binary output. The result of this process is a set of floppy discs and a set of system listings. The floppies contain:

- new system as multiple binary files,
- stand-alone program to dump binary to main disc,
- system symbol tables,
- system initializer,
- patch file.

The user can then dump the new system to his main disc, run the initializer, and run the patch file. The latter installs corrections which have been detected since the official release of whatever version of the system is involved.

It should be noted that (most of) the system is stored in two copies on disc, the safety copy and the working copy. If the working copy becomes corrupt or suspect, it may be overwritten by the safety copy. This procedure is known as a cold start or a HENT (an abbreviated Norwegian word). A HENT has the side-effect of destroying all virtual memory segments loaded by the user, so it is essential that command files to reload all segments be constantly up to date.

System patches may in general be installed either on the safety copy or the working copy. In either case, the patch must be made with all absolute addresses corresponding to the system generation in question, leading to administrative problems if many different systems have to be maintained. In any case, system patches can be made only in octal or assembly language.

Unfortunately, in our application, it has proved impossible to avoid extensive local patching of SINTRAN III. These patches have essentially three justifications:

i) drivers for local devices (data links, consoles),
ii) minor functional changes,
iii) bugs which ND have not corrected.

It should be pointed out that the maintenance effort for system patches is enormously greater, in proportion, than for any other type of software, so such patches should be consciously kept to a minimum. For this reason, class ii)—functional changes—requires some explanation. SINTRAN III has several defects when viewed as a real-time operating system and most of our functional changes are concerned with this, although one or two are of a cosmetic nature. An example of such a defect is that the same bit is used in the RT description status word for two distinct purposes: to remember that the RT program should be restarted from the beginning when it finishes, and to remember that its next request to enter a wait state should be ignored. Several patches are necessary because of this feature.

A standard technique has been adopted for installing these local patches. Source files (assembler or structured assembler) are kept on the program development computer. They contain conditional assembly marks to allow for different target systems (consoles, FECs, etc.) and for different versions of SINTRAN III. For each target and version, the assembler source files, the appropriate system symbol tables, and *ad hoc* command files, are stored on two (or more) floppy discs. By means of special versions of the assembler, these floppies are used to install the patches on the target system, modifying the safety copy of the system. Thus the local patches remain even after a cold start.

Not only do the addresses of all patches depend on the system generation, but also the details of the code depend on the system version. Thus the installation of any new release of SINTRAN III requires a detailed review, possible modification, and exhaustive test of all local patches. This is a fastidious task and means that several months' work is required to prepare a new SINTRAN release for installation in the on-line computers [12].

In addition to successive releases (A to I at the time of writing), and to minor options, there are three significantly different 'flavours' of SINTRAN III: VS, VSE, and VSX. VS, which is frozen at the H release, may be used on NORD-10 or ND-100 CPUs. VSE is necessary to exploit 24-bit physical addressing on the ND-100, and is incompatible with the NORD-10. VSX further exploits certain extra instructions of the ND-100/CX for increased speed. At PS, the FECs are frozen on the E release of SINTRAN III/VS owing to technical restrictions (Section 4) and to insufficient disc space to upgrade to the H release. On other computers we follow new releases as far as is practical. To reduce the number of system generations currently supported to a tolerable level, we have insisted on compatible hardware configurations (in particular: all FECs are today NORD-10 [S] with Hawk discs; all consoles are ND-100 with Phoenix discs).

3.4 Experience with SINTRAN III

We have applied SINTRAN III in two rather different domains:

i) A large central program development system, supporting the other ND computers and cross-supporting microprocessors:

   At the time of writing, this system has:
   220 registered user names,
   32 terminals,
   300 Mbyte disc,
   4 Mbyte memory, etc.

ii) About 20 real-time systems, each connected to a few hundred external hardware modules (in CAMAC):

   These systems each have:
   256 to 1024 Kbyte memory,
   10 to 30 Mbyte disc,
   about 200 compiled software modules (NPL, P^+, Pascal),
   about 200 interpreted software modules (NODAL),
   about 30 RT descriptions (some allocated dynamically),
   about 100 user segments.

The consequence of such a heavy real-time structure is that the system recovery mode files used after a HENT cold start (or after a lesser disaster) are numerous and complex: on a typical computer, 20 or 30 mode files are

involved and a HENT takes at least two hours to run. Since our computers are normally required to run 24 hours per day, seven days per week, it is clear that a HENT is nothing less than an operational disaster. Sound and systematic operating procedures (backups, logging of software modifications, and so on) are therefore essential to reduce the number of HENTs to an absolute minimum. This involves a total of more than twenty people (programmers, technicians, and part-time system supervisors) and is thus a substantial, and unpredictable, sink for manpower.

Our experience can be summarized as a series of positive and negative points, where in almost all cases each positive point has a corresponding negative one, and vice versa.

i) The fact that SINTRAN is a multi-mode system was a great benefit since our users (not all professional computer scientists) had only to learn one operating system and file system interface; this saving can be counted in man-years on a big project. On the other hand, the fact that the background and real time systems within SINTRAN have different (but overlapping) sets of facilities was a source of confusion and frustration. In particular, it means that many software tests can only be carried out live, in the real time mode, since background testing (with much better protection) can only be partial. Even with two or three off-line computers for real time testing, we still often install faulty software on-line.

ii) Demand-paged virtual memory was an essential component of our system design; it freed us from the traditional calculations of memory size normally needed in a real-time system. However, the restriction to two main segments per program at one time, and the global restriction to 255 segments, are very severe handicaps for system design. In particular, if a main program (on segment X, say) calls an overlay (on segment Y), and the overlay wants to call a second overlay (on segment Z), it is impossible for the code on segment Z to receive parameters of which some reside on X and some on Y. Even in a 16-bit computer, one could do better than this.

iii) The command interface of SINTRAN III is excellent (many people prefer it to UNIX).

iv) Error and exception handling, and debug tools, are inconsistent, incomplete, or non-existent.

v) The file protection scheme is well adapted to our environment, except that there is essentially no file protection for real time. This is a constant nuisance due to mutual interference between real time programs which both accidentally use the same file number, and accounts for many of our operational mysteries. A multi-level file directory structure would have been useful.

vi) The file-oriented nature of all I/O is a valuable simplification. However, there are many inconsistencies and serious missing features (such as timeouts, an asynchronous event system, and reliable internal devices). XMSG, the ND message system, came too late for our application.

vii) It is convenient (but not essential) that background programs are often portable between different SINTRAN installations. This is a result of there being rather few system options (except peripherals); the corresponding, and serious, disadvantage is the monolithic nature of SINTRAN, and the enormous difficulties of configuring big systems and adding user device drivers which this causes.

viii) The relatively easy contact with ND's internal specialists is a bonus. This easy-going approach has its negative side: the traditionally informal quality control and documentation.

ix) SINTRAN can be very fast (direct tasks). It can also be very messy for system programmers! The system documentation is incomplete and out of date, and the only solution is to read the listings—which are under-commented, are badly laid out on the page, and employ needlessly obscure coding tricks.

3.5 Assessment of SINTRAN III

In 1977, a detailed comparison was made of the two real-time operating systems available for the NORD-10, namely SYNTRON II [13-19] and SINTRAN III. The result of this comparison was an unambiguous preference for SINTRAN III [20]. Our practical experience has given no reason to regret this choice and the comparison will not be repeated here.

A more general basis for an assessment of SINTRAN III may be found by comparing it with UNIX as a program development system, and with RSX-11M as a real-time system (i.e. postulating the use of DEC rather than ND computers).

Indeed, it is scarcely conceivable that we could have run the necessary program development service on a PDP-11 with any system other than UNIX, and it is to the credit of SINTRAN III that it can be used successfully in a ND-500 configuration supporting 32 terminals, 2 network connections, 2 printers, 6 disc units, 4 floppy disc units, a magnetic tape drive, and two CPUs. There are more than 200 user accounts and at least 40 active users. The average period between crashes (unintentional stop for whatever reason) is about 2 weeks, although intentional but unplanned stops occur somewhat more often (e.g. the system is reloaded to clear a software blockage).

Table 1 is a subjective comparison of SINTRAN III as a program development system with PWB/UNIX [21], the most suitable version of UNIX available a few years ago.

A preliminary conclusion from this comparison would be a preference for UNIX. However, this must be counterbalanced by the enormous advantage of having the *same* operating system for the on-line machines. Nevertheless, the availability of an efficient UNIX implementation on ND hardware would be of great interest, especially if coexistent with a SINTRAN system on the same machine.

Table 1

Subjective comparison of SINTRAN III and UNIX

Aspect	SINTRAN III	UNIX
File naming	Directory/user/name/type/version	Hierarchical
File structure	Byte stream	Byte stream
Space allocation	Good	Weak
Command interface	English commands, full prompting and abbreviation	Very terse, no prompting
Command language	None	Shell: powerful but obscure
Utilities	Good but difficult to combine	Excellent, easy to combine by shell and pipe
Batch facilities	Conventional	Unconventional (parallel processing in shell)
'Help' facilities, documentation	Weak	OK
Source code control system	None	Exists
Text processing	Good	Good
User community	Small	Large
Reliability	Average	Good

A rather different set of aspects must be considered to compare SINTRAN III with RSX-11M [22]. Table 2 is a brief and subjective comparison.

Table 2

Subjective comparison of SINTRAN III and RSX-11/M

Aspect	SINTRAN III	RSX-11/M
Multi-mode system	Yes	No
Memory management	Paging virtual memory	Roll in/roll out
Loader/builder	OK	Good
Dynamic tasking	No - Manual installation only	Dynamic installation
RT system calls	Incomplete	OK
Event system	No	Yes
Context switch	~ 1 ms	~ 2 ms
Shared libraries	Yes (segments)	Limited
Driver facilities	Poor	OK
Documentation	Poor	OK

One may conclude that the choice between the two systems is quite finely balanced. The larger memory space of the ND architecture, and its superior memory management software, compensate for SINTRAN III's intrinsic weakness as a real-time system.

4. GENERAL SOFTWARE LAYOUT

This section is concerned with the general system software layout which provides the infrastructure for applications. This layout was designed in view of the control system topology and applications software structures described in Section 2, and within the constraints of real-time programming in SINTRAN III.

4.1 User requirements

4.1.1 Definitions

i) Hard real time

This refers to process- or equipment-oriented computing processes which must, in general, be executed in a time frame of less than one accelerator cycle (minimum 650 ms). (Abbreviation: HRT.)

ii) Soft real time

This refers to computing processes which are desired to be executed in the same time scale as a human operator's reactions. This includes genuinely interactive programs, and others which are not intimately linked to the accelerator hardware. (Abbreviation: SRT. The abbreviation RT includes SRT and HRT.)

iii) Process ensemble

This refers to a set of process-oriented software which, because of interface topology must all be loaded together in one virtual memory segment of one process computer or FEC.

iv) Background

This refers to all other computing processes whether or not accessing the network (e.g. file copying, printing a process log).

v) Program development

This refers to all computing activity concerned with program preparation and off-line testing. Such activity is restricted to the program development computer.

4.1.2 Functional requirements

SRT programs are written in NODAL or in a compiled language and execute in the console or process computers. Such programs must be able to access process-oriented software locally or in remote computers, display drivers in the local or in remote computers, and global data.

Process-oriented software modules are of two types, passive subroutines (EMs and CVMs) or active (HRT) programs. The choice for a particular case depends on the equipment concerned and the real-time constraints applicable. In both cases, a software interface must be provided for communication with SRT programs. A serial CAMAC driver is provided by the system for use by process software.

Similar remarks apply to the console equipment drivers for displays, etc. However, it is necessary both for the console interface and the process interface that program/program interaction is not limited to the NODAL interface. To achieve real-time display of computed results, for example, direct communication between compiled software and the display driver is essential. Console devices are accessed by conventional SINTRAN III I/O.

Another user requirement is for file transfer on the network. This is unavoidable for sending a log to a remote printer, for loading programs from the program development computer, and for accessing a NODAL file remotely.

An operational software framework for microprocessors embedded in CAMAC auxiliary crate controllers (ACCs) is also required. Basically this provides a very simple communications protocol between an RT program acting as host and the ACC software, and a framework for loading and running interrupt-driven applications routines in the ACC. On the ACC side, these facilities are provided by a small resident monitor; on the NORD-10 side, by a small set of subroutines using the serial CAMAC driver.

4.2 Constraints

The principal constraint (which made it impossible to use the SPS software layout unmodified) is the accelerator cycle time (500 ms if one considers the Linac). This makes the general use of an interpreter and unrestricted file transfer on the network unacceptable.

The network topology implies that all process computers are specialized rather than general-purpose, indicating that the 'equipment end' of applications programs (known as 'process control modules') are not distributed programs, and can be compiled (for speed or structural reasons) instead of being purely or partially interpreted.

4.3 Implementation of network software organization

The computers in the network are all NORD-10s or ND-100s running the SINTRAN III operating system. Additional system software is provided in two layers:

i) system layer, identical in all computers (except the program development system), providing the network software, languages, and general-purpose subroutines;

ii) local layer, varying from computer to computer, providing for example the CAMAC driver and ACC access routines in the process computers or console facilities in the console computers.

The system layer is maintained by the System Software Section, and the local layer by the System Software or Console Section, if necessary in collaboration with a system supervisor for the individual computer.

4.3.1 Process manipulation package

Owing to the lack of dynamic task facilities in SINTRAN III, a sophisticated package was developed to provide the necessary functions. Compiled SRT or HRT processes, previously loaded onto a virtual memory segment, may be named, started, connected to interrupts, stopped, etc., independently of the finite number of RT descriptions allowed by SINTRAN III. A programmer using the system for non-trivial applications must become familiar with this package [23].

4.3.2 Global variables

This is a zone of global variables accessible from SRT programs.

These globals are intended to be used for indirect (mail box) communication between various independent programs, e.g. for control of the operations tree. Library routines are provided to access them.

A dynamic implementation of the global variables was chosen because a precompiled implementation would have had insufficient run-time flexibility. Access time for a global therefore includes a list-searching operation, as well as semaphore reservation and release.

4.3.3 Process ensembles

HRT programs are used to provide active process-oriented software, including microprocessor host processes, where this is necessary.

An HRT program, at execution time, is restricted to one computer. It has no access to the network, although it may be started remotely. It may communicate with other programs only through data buffers.

Also, each process ensemble must include an external interface, i.e. a set of procedures allowing it to be manipulated by SRT programs. Calls on this interface will be executed directly if they originate in the local computer and via a server program provided by the system if they are of remote origin. This interface is in fact in normal cases an equipment module.

In the maximum case, a process ensemble consists of
- one or more HRT programs*,
- data buffers*,
- data tables*,
- EMs and CVMs (with single-equipment and array-call entry points).

The items marked * may be located in an ACC (microprocessor).

4.3.4 Soft real time programs

NODAL or compiled SRT programs execute basically in one computer and may use network and system facilities for the following purposes:
- to manipulate local or remote global variables;
- to communicate with the interaction interface of any local or remote process ensemble (i.e. to call EMs);
- to call library routines locally or remotely;
- to queue requests for background processing;
- to load remote NODAL files.

4.3.5 Background activities

These run at low priority, using system and network facilities for three main purposes:
- to copy files between computers (using the network);
- to process files and print logs (using the SINTRAN III batch system; MCR computer only);
- to preload RT programs (using SINTRAN III batch or background terminal).

4.3.6 Library subroutines

Re-entrant library subroutines are either 'overlaid' and available only to SRT programs, or in special cases 'resident' and available to all RT programs. 'Overlaid' subroutines will be loaded on virtual memory segments which may be accessed by any SRT program; the overlaying is transparent to the high-level language programmer and may or may not imply a disc swap. In general, overlaid subroutines will be compatible with both interpretive and compiled calls [according to the Interpreter-Compiler Compatible Interface (ICCI), see Appendix 1]. Furthermore, process ensemble segments and ICCI subroutine segments are identical from the system's viewpoint, except that process ensembles may contain HRT programs.

'Resident' routines will be very few in number and loaded in a memory-resident common area of restricted size. They are essentially limited to data-link, CAMAC and ACC access.

In addition, there are 'linkable' library routines of which a new copy must be included in each segment using them. In some cases, because of language-dependency, a trivial linkable routine may be used to interface to a swapped or resident routine. This will be transparent to the programmer except at the link-load stage.

In a NODAL program, an ICCI routine in a remote computer may be called using the standard NODAL commands EXEC or IMEX. In a compiled program, an ICCI procedure in a remote computer may be called using the form

computer_identifier & procedure_identifier (parameter_list);

This is known as a 'remote procedure call'.

Figure 2 shows the layout in a typical process computer (FEC). The consoles are similar, but without process ensembles or Serial CAMAC, and with extra device drivers and other facilities. Figure 3 shows the layout including the network.

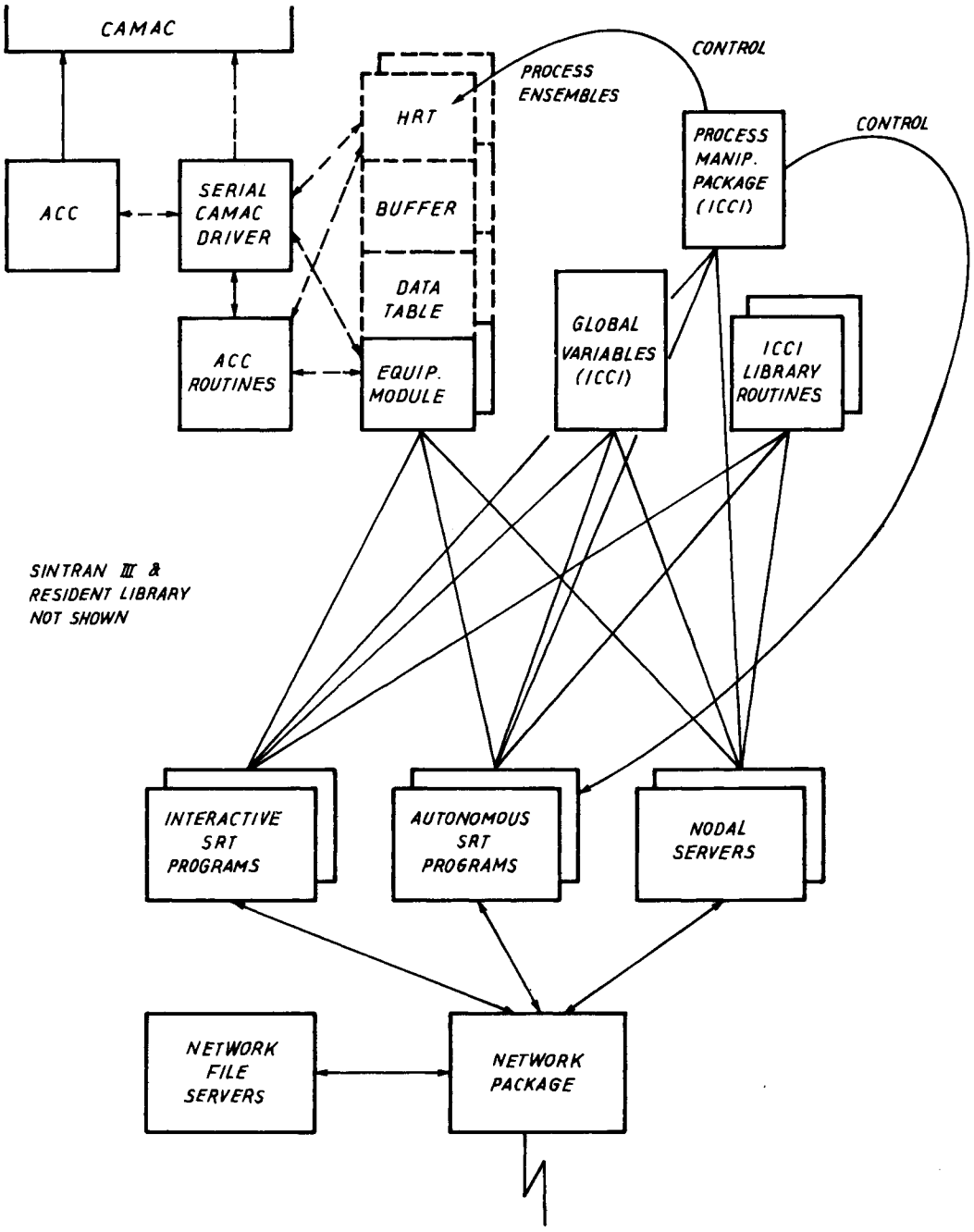

Fig. 2 Software layout in FEC

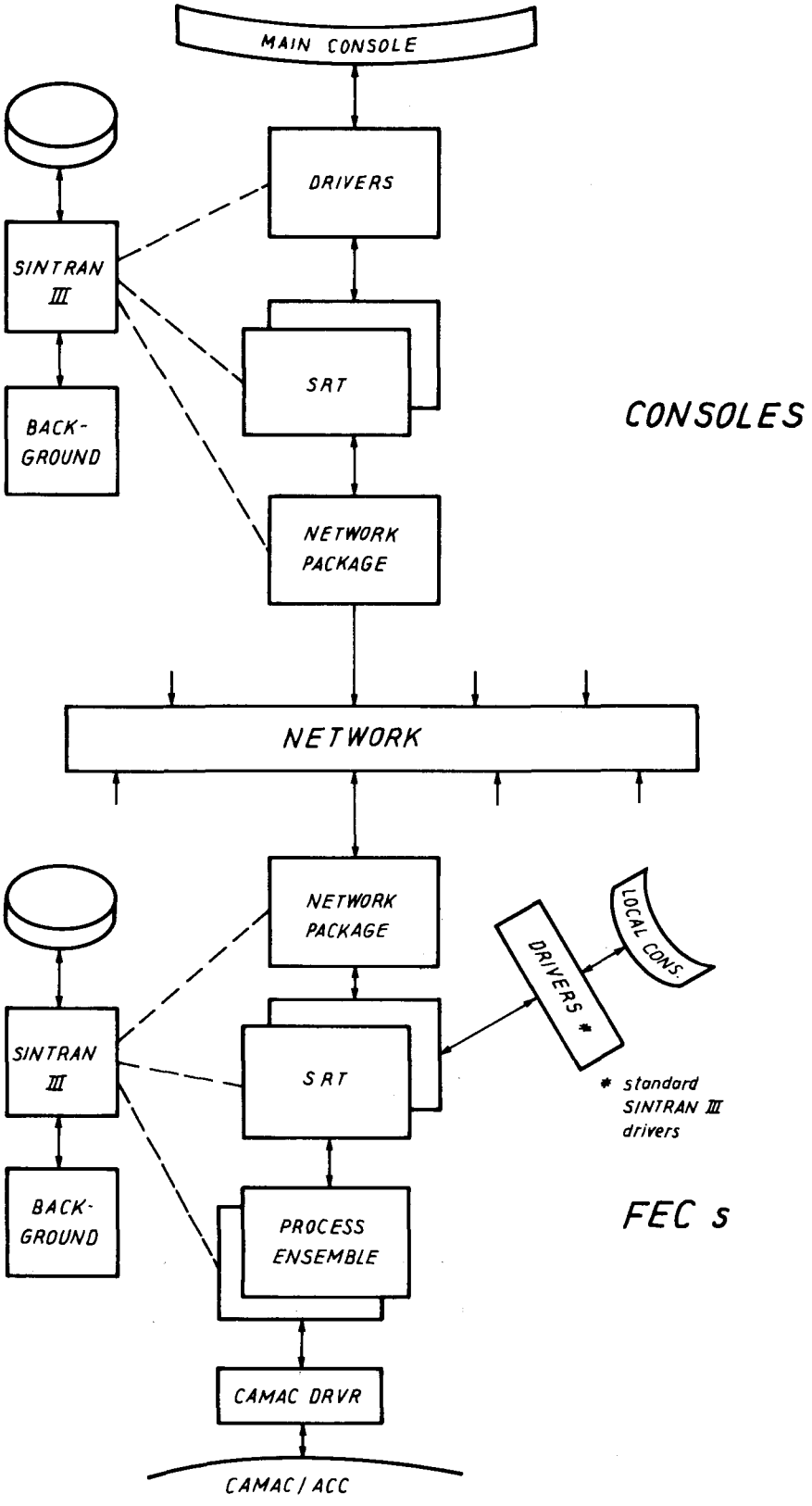

Fig. 3 Layout including network

4.3.7 Some implementation details

i) NODAL requirements

Each computer requires a small number of interactive SRT NODALs (general purpose in the process computers, with special characteristics in the consoles) and a small number of NODAL servers (see below). The NODALs are able to call all ICCI routines including equipment modules.

ii) NODAL servers

Autonomous NODALs to service remote requests (NODAL IMEX, EXEC, and remote procedure calls) are included. The data link package builds a queue of remote requests and feeds them to the autonomous NODALs, which are never disactivated but spend most of their time in a wait state.

One autonomous NODAL is provided to execute NODAL source files without connection to any interactive terminal; this is the 'file-driven NODAL'.

iii) RT description pool

A pool of RT descriptions (the SINTRAN III resource used as a context for an active RT program) is provided for the activation of autonomous compiled programs by the process manipulation package.

iv) Network file servers

Specialized server programs are provided both for high-priority file transfer (the NODAL OLD statement) and for low-priority file transmission. These programs are completely invisible to the user.

v) Rapid file access

In certain FECs, where rapid access to NODAL files is important, the resident library contains a system-dependent rapid open routine (ROPEN).

vi) Data-link package

The data-link package consists of: i) routines, data, and buffers in the resident library; ii) routines, drivers, and programs in a resident virtual memory segment; iii) additions to the memory-resident part of SINTRAN III. All this is invisible to the user, hidden by the high-level network access facilities (IMEX, EXEC, remote procedure call, file transmission).

vii) Software priorities

High: data link package
high-priority file transmission servers
HRT programs
data-link NODAL servers
interactive SRT
autonomous SRT and file-driven NODAL
SINTRAN III background

Low: low priority file transmission servers

Lowest: test level

All programs must follow multiprogramming etiquette, i.e. release the CPU whenever possible, specifically during all waits.

viii) Use of virtual address space by HRT and SRT

A given HRT or SRT context cannot have more than 64 Kwords of virtual address space at a given moment, although monitor calls may be used to overlay virtual address segments (which may or may not cause a disc swap according to circumstances).

Also, a given context can instantaneously be mapped at most onto:

- two non-overlapping segments of virtual memory,
- a real-time common (RT common) area of predefined size,
- a 're-entrant' segment, which is read-only, but which is copied one page (1 Kword) at a time onto the corresponding page of one of the other two segments the first time each page is written to after starting the program concerned. Pages which are only read (or executed) are not duplicated. This allows shared code and non-shared data to be loaded on the same re-entrant segment.

An RT program can call a subroutine on another segment by using the MEXIT monitor call, as long as the new segment does not overlap the segment containing the call. It is possible to load code into RT common, thus providing the effect of an extra library segment shared by all RT programs. To exploit these facilities, the utilization of virtual address space shown in Fig. 4 is employed.

With this arrangement, SRT programs may call the resident library or ICCI procedures [via MEXIT]. The called procedures will have direct access to the SRT stack. Procedures called from both compiled and NODAL SRT, or called remotely, must be ICCI.

HRT programs may call the resident library; each HRT program has a small internal stack to which the library will have access.

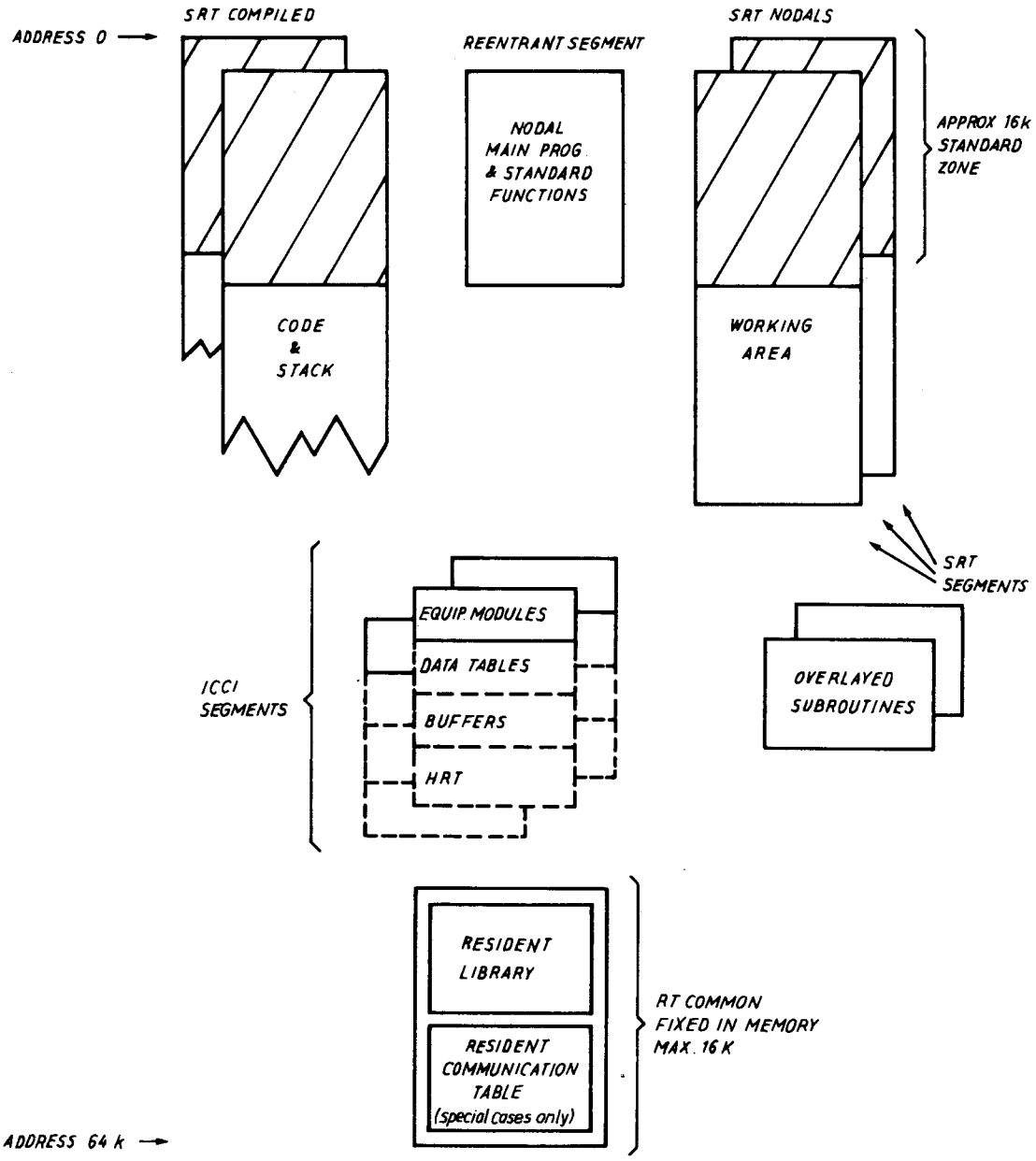

Fig. 4 Virtual address utilization

Equipment modules or other ICCI procedures may call the RT library, especially the CAMAC driver or ACC routines.

HRT programs may call neither procedures in SRT address space nor ICCI procedures. ICCI subroutines may call routines in the standard (re-entrant) zone of SRT address space, but this is intended only for the writing of specialized NODAL functions which need to access the kernel of NODAL. ICCI to ICCI calls are possible but slow.

Within one computer, certain 'magic' addresses must be fixed for this layout. These are:

MLAD: main program load address, i.e. start of code and stack for compiled SRT and start of working area for NODAL;

OLAD: overlay load address, i.e. start of ICCI segments, thus end of main program segment and NODAL working area.

See Appendix 2 for more details.

Note: For special applications, there is nothing to prevent a program being loaded independently of this layout (i.e. anywhere between address zero and the beginning of RT common) but such a program cannot easily communicate with standard software. Nevertheless it may be treated as an HRT program by the process manipulation package.

ix) Serial CAMAC driver

The serial CAMAC driver has to be fast, and must handle direct-program-control random and single addressing, and simulated and true direct memory access (DMA).

To get the speed, the technique used is to include the driver as an entry point in the RT library, which runs privileged and so is able to execute direct IOX instructions. The stack is used to provide re-entrant work space, and where resource reservation is needed standard SINTRAN III semaphores are used with an overhead of 1 to 2 ms; for speed, indivisible CAMAC operations are protected by interrupt masking.

Because true DMA may only be done to physical addresses less than 64 Kwords in the NORD-10 [S], the driver uses DMA buffers fixed in memory bank 0, and allocates them to users as required.

5. NETWORK

5.1 General description

The SPS network, actually developed jointly by SPS and the company TITN during 1974-76, was chosen for the PS system in competition with HDLC and with CERNET, on grounds of performance [24]. The best general description of the network is in Altaber [25].

The network is a packet-switching system based on high speed (600 kHz) serial links capable of working over several kilometres with repeaters. The topology is that of multiple stars connected by their centres (although the PS network is in fact a single star). The centre of the star is a NORD-10 dedicated to packet switching and functioning as a 'transparent black box' for the rest of the system. It is known as the message handling computer (MHC).

The high-level services offered to the user are as follows:

- remote procedure call (RPC) (compiled languages),
- remote interpretive execution (NODAL),
- file transmission,
- remote file load (NODAL),
- remote file save (NODAL, limited),
- display pipeline.

Remote file access (i.e. remote read and write) and program-to-program communication are *not* provided.

The remainder of this section covers the various high-level services in more detail, and then discusses the various layers of the implementation.

5.2 High-level services

5.2.1 Remote procedure call

An RPC is a procedure call issued by a program in one computer, but executed in another, with automatic passage of parameters in both directions through the network. The availability of such a facility is a great simplification for the writing of applications programs using the network and for this reason RPC is available in the PS Control System. At the time of writing, it is only implemented for the NORD-10 and ND-100 minicomputers.

An RPC facility is available with suitable syntax in the supported high-level languages, and by use of suitable macros in structured assembler. It is inevitably subject to certain restrictions, some of which are fundamental and some of which are due to implementation. An RPC is actually encoded at run time in the same way as a simple remote interpretation request, and so the server program which executes an RPC is in fact an activation of the NODAL interpreter. The restrictions are thus as follows:

i) Remote procedures must conform to the ICCI convention.
ii) A pointer cannot be passed as a parameter (fundamental restriction, since addresses in one computer cannot be meaningful in another).
iii) Variant parameters not allowed in general (except for one special case, they are not supported by NODAL).
iv) Optionally repeated parameters not allowed (not supported by NODAL).
v) Maximum 8 parameters (NODAL restriction).
vi) Total size of parameters not more than about 900 bytes (network message size is limited to 1 Kbyte).
vii) Procedure name unambiguous in first 6 characters (NODAL restriction).
viii) Execution time limited to 2 elapsed seconds (network congestion control restriction).

A more complete description of the RPC protocol is in Carpenter [26].

5.2.2 Remote interpretive execution

The remote execution facility of the NODAL language is used for reasons similar to those for RPC, i.e. to simplify the design and coding of distributed applications programs. Either a single line of NODAL or a set of lines plus variables, may be sent for interpretation in a remote computer by use of the immediate execute (IMEX) or execute (EXEC) commands respectively. The IMEX, like RPC, causes the calling program to wait for completion; the normal EXEC allows the calling program to continue in parallel with the remote activity, the user being obliged to WAIT explicitly for completion. Theoretically, this allows parallel remote execution, but this possibility has been little used at PS.

The restrictions applicable to remote execution, apart from those intrinsic to the NODAL language, are:

i) Total size of IMEX not more than about 100 bytes, total size of EXEC not more than about 950 bytes.
ii) Execution time limited to 2 elapsed seconds.

5.2.3 File transmission

The transmission of a complete file from disc to disc through the network may be requested by calling the TRANS routine. A low-priority server is activated in each computer concerned, so that file transmission takes place as a background activity. No handshakes take place between the two servers, and there is no congestion control so the reliability of the protocol is poor. Since the servers run autonomously, there is no error reporting to the program or terminal calling TRANS.

It should be noted that these defects arose from certain fundamental characteristics of lower levels of the network which are described later. Utility programs (TRANSMIT and FILE-DISTRIBUTION) have been written to compensate as far as possible for this.

5.2.4 Remote file load and save

Complete files of NODAL code and variables may be read (LOAD, etc.) across the network in an interactive or autonomous way. A high-priority server is activated in the remote computer for this purpose. Although there are no handshakes between the client program and the server, this facility is rather reliable since the source (a disc) is considerably slower than the destination (a memory buffer), so network congestion is extremely unlikely.

The inverse operation, sending a NODAL file to a remote disc (SAVE) depends on a non-handshake transmission to a high-priority server. Although limited to 1024 bytes to reduce the chances of congestion, it suffers from the same disadvantages as the low-priority file transmission facility and is not recommended. It is in any case available only towards the FECs, since the corresponding lower level protocol is used for other purposes towards the consoles.

5.2.5 Display pipeline

A basic requirement of the control system is to route data from the process, via microcomputers and the FECs, onto various display screens attached to the console computers. A console may require data from several FECs, a FEC may have to supply several consoles, and each FEC may have to collect data from many CAMAC crates.

The applications software to multiplex and de-multiplex these data streams in the FECs and consoles is supported by a 'pipe' protocol which is in fact a trivial adaptation of the remote file save protocol just described. In the consoles, the relevant file server is replaced by a display pipeline server (considered to be an applications program). In the FECs, an access routine PIPE is provided which routes a message of user data to the pipeline server in the appropriate console. To avoid congestion, the pipeline server discards messages which it cannot process in time. For a more complete description, see Mérard [27].

5.3 Overview of implementation

This section describes the network roughly in terms of the ISO Open Systems Interconnection Reference Model [28], although the correspondence between this model and the actual implementation is rather loose.

The *applications layer* (number 7) is of course the body of at least 1000 applications programs using the services described above.

The *presentation layer* (6) is represented by

i) a run-time library routine REM which mediates RPCs for all compiled languages;
ii) the EXEC, IMEX, REMIT and WAIT commands of NODAL, the corresponding routines in the interpreter (with the same names), and the routines REXEC and EXIM which mediate NODAL's use as a remote execution server;
iii) the TRANS procedure and its associated disc-read server (TRANR) and disc-write server (TRANW). TRANS is an ICCI routine loaded on a library segment; the servers are resident RT programs loaded on the dedicated data-link segment, running at relatively low priority;
iv) the file-handling commands of NODAL (OLD, RUN, SAVE, etc); the corresponding routines in the interpreter; and the associated disc-read server (READL) and disc-write server (WRITL): these are resident RT programs loaded on the data-link segment, running at relatively high priority;
v) the access routine PIPE of the display pipeline, embedded in the standard relocatable library.

The *session layer* (5), the *transport layer* (4), and the *network layer* (3) are difficult to identify precisely in the implementation. Nevertheless, the following distinctions can be made:

Access routines: There are four principal access routines, resident in RT common, and in principle callable by any privileged (ring 2) RT program. The following descriptions are abstract: for historical reasons, these routines do *not* follow the P^+ calling convention.

LOPEN (open-type, computer, usercode, filename, LUN)

This 'opens' a connection between the user program and the network software by setting up a control block (whose logical unit number is returned in 'LUN'). Additional actions depend on the 'open-type' as follows:

0: Remote NODAL read (OLD, etc). Read request packet is sent to 'computer' indicating 'filename'.
1: Remote read (TRANS). Like type 0.
2: Remote NODAL write (SAVE)/PIPE request. Write request packet is sent to 'computer' indicating 'filename'.
4: IMEX request. First and only packet of IMEX datagram is prepared.
5: EXEC request. First packet of EXEC datagram is prepared.
6: Read remote job (used by IMEX/EXEC servers).
7: Send TRANS request. A packet is sent to 'computer', which is the destination for the required file transmission. The packet must be in a special format.

The 'filename' parameter is indeed a file name for types 0, 1, and 2. For types 4 and 5 it is a special code called 'the NODAL type' (5 for IMEX and 3 for EXEC). For type 6, it is the control-block LUN which is to be used (which is set up before a server is started). For type 7, it is the full 64-word TRANS request packet.

LCLOS (LUN)

This 'closes' the connection between the user program and the network software. In the case of an 'open for write' (types 2, 4, 5) it causes any partly-filled packet to be sent. In the case of 'open for read' (types 0, 1, 6) it causes waiting or subsequent incoming packets to be discarded. For type 6, in addition, it re-opens the connection for write to allow the reply to be sent (REMIT). Similarly, for types 4 and 5 it re-opens the connection for read, to allow the reply to be received. There is no 'close' for a type 7 'open'.

LSTIN (LUN, buffer, byte-count)
LSTOU (LUN, buffer, byte-count)

These (pronounced 'link string input' and 'link string output') respectively read/write a string of bytes from/to the network. A datagram may be received/sent by several consecutive calls to the appropriate routine.

Note. The various file servers do not read datagrams via LSTIN. Instead they delink arriving packets directly from a queue (one queue for each server). The IMEX and EXEC servers use LSTIN, but also call a special initialization routine INRTD when they wish to wait for work. This routine delinks arriving datagrams and sets up the control-block LUN for LOPEN (called by the server after returning from INRTD).

The above access routines call a number of *primitive routines,* also resident in RT common, such as MSEND which, given a packet, inserts it in the driver's output queue (or, exceptionally, triggers the DMA hardware directly if the queue is empty).

Below these primitives is the *transport station* which has three responsibilities: buffer pool management, link status management, treatment of all incoming packets. The transport station runs as a SINTRAN III direct task, on hardware level 8, with a dedicated hardware level and Page Index Table (PIT). Thus it can simultaneously 'see' its own segment, the resident kernel of SINTRAN, and RT common.

i) Buffer pool management is simple and classical: 64-word packet buffers are delinked from a 'release queue' and linked into a 'free queue' whenever the transport station is activated.
ii) Link status management involves processing requests to logically open or close input from the network or output to the network. Closure involves complex activity to flush ongoing transactions.
iii) Incoming packets fall into three categories: surveillance packets (immediately echoed); network service packets (link status changes, used to update tables); data packets. The latter are inserted into the appropriate queue, according to whether they are intended for a server ('unexpected messages') or to satisfy an LSTIN call ('expected messages').

The *Data Link Layer (2)* of the ISO model is represented by interrupt drivers on levels 11, 12, and 13, which handle the line protocol, error interrupts, and retries. They also mediate a simple form of flow control: a computer which is running short of buffers sends a link control bit "suspend long message transfer', which inhibits the sending of new multi-packet datagrams by the neighbouring node. Unfortunately, this indication is not broadcast in the network, so a source which is saturating a given destination is not inhibited until it has also saturated all intermediate nodes.

The *Physical Layer (1)* is of course the TITN hardware and interconnecting cables. This is outside the scope of this report.

5.4 Particular aspects of the implementation

This section describes various particular aspects of the implementation which may be of general interest.

5.4.1 Flow control

As mentioned, flow control is limited to the 'suspend long message' indication at the data link control level. This indication means simply that the computer issuing it has very few free packet buffers and cannot accept any further multi-packet messages. Unfortunately, single-packet messages (e.g. IMEX calls from NODAL, file read request, etc.) may still be sent. This leads to a real possibility of the buffer pool becoming completely empty in an adverse combination of circumstances. If this happens, the line protocol in general cannot recover and it is normally necessary to restart the computer.

If a computer maintains the 'suspend long message' indication indefinitely, a supervisory program in the MHC node will time out after a few seconds, and will close and re-open the link. If this sequence occurs several times, the MHC definitively closes the link and manual intervention is required. This error condition [CLOSE INPUT xx (17) where xx identifies the computer] has become infamous, and has become known as 'the Copenhagen effect' since one of the authors was forced to fly back from that city on 16 October 1980 on its account. Its most frequent causes are a saturated computer, or a dead EXEC server (killed by SINTRAN III after a fatal software error).

5.4.2 Message length

The restriction on message length to 1024 bytes (including overhead) is closely related to flow control: the 'suspend long message' indication is normally sent when the buffer pool contains space for at most one 1024-byte datagram plus a small margin. This must not be allowed to represent more than (say) one third of the total buffer pool, to minimize the incidence of the Copenhagen effect.

5.4.3 Timeouts

They also interact with flow control. An EXEC or RPC is allowed 3 seconds of real time (not CPU time) for completion; an IMEX is allowed 2 seconds. After this, execution of the server is aborted and all packets are returned to the buffer pool (this is done by a timeout supervisory program called PROTO). If these timeouts were not imposed, catastrophic flow control failures would occur continually owing to the build-up of long queues for the servers.

There is also a 10 s timeout associated with LSTIN, to deblock programs awaiting lost replies. No indication except a timeout ('REMITTED DATA LOST') is returned when an EXEC, IMEX or RPC is aborted, or when an attempt is made to read a non-existent or protected file.

5.4.4 Memory layout

The memory layout of the network software within SINTRAN III is rather special in order to minimize lost time, at the expense of reduced protection [29].

The access routines (LOPEN, etc.), routines needed by the whole network package, and data structures needed by the whole package, are memory-resident in RT common [they are in fact loaded into RT common by the system start-up program (STTUP]. The transport station (direct task) and the interrupt drivers reside in a segment [known as the data-link segment—(DLSEG)], which is fixed in memory at a given physical address. A page-index table (P1T3 up to and including the H-version of SINTRAN) is dedicated to use by the network package and permanently maps

a) the resident kernel of SINTRAN III;
b) the data-link segment;
c) RT common.

In versions of the network package prior to the H-version of SINTRAN, the DLSEG and RT common were located in memory bank 0 such that their virtual and physical addresses were equal. The packet buffer pool was included in RT common.

From the H-version of SINTRAN onwards, the packet buffer pool is included in the DLSEG, which is loaded in the same memory bank as used for SINTRAN's DMA buffer pool. (This is in fact only possible in NORD-100 computers whose external bus controller allows DMA transfers outside bank 0. It is also required by more recent SINTRAN versions, whose resident kernel is too big to leave space for DMA buffers in bank 0. This is the main reason why the PS NORD-10 computers are 'frozen' on the E version of SINTRAN.) For convenience, the DLSEG's virtual address is equal to its physical address within its bank, but this is not a fundamental requirement. RT common is located arbitrarily in memory.

It should be noted that with the packet buffers included in the DLSEG, they can be accessed from an RT program only by use of the alternate page table mechanism of the ND-100 architecture as supported by SINTRAN (ALTON monitor call).

The result of this sophisticated memory layout is that packets may be transported from one RT program to another via the network in about 7 ms, as opposed to about 5 ms in the original SPS implementation. This is a very small overhead compared with the benefits of a protected virtual memory system as provided by SINTRAN.

5.4.5 Generation of the network software

For historical reasons, this takes the unfortunate form of a complex absolute assembly job, resulting in two highly interdependent binary files, one containing the image of the DLSEG, the other the image of the relevant parts of RT common. These binary files, furthermore, embed references to absolute addresses in the SINTRAN kernel, and therefore depend on particular SINTRAN system generations and patch files. To ease the evident problems posed by this situation, a small toolkit in the form of a set of job-control macros (PERFORM MACROS) has been implemented [30].

5.4.6 MHC computer

This has three roles: packet switching, network control, and system clock/calendar. It runs the SPS SYNTRON operating system, supporting a simple disc file system and the NODAL interpreter in addition to the network functions. Packet switching is performed by high-priority RT programs 'on top of' a transport station and drivers very similar to those described above. Network control consists of watchdog programs (including generation of surveillance and service packets) and associated NODAL functions and programs for operator interaction. The system clock/calendar, formerly based on hardware and software from the European Molecular Biology Laboratory (EMBL) at Heidelberg is now derived from SPS (if the link to their network is open), or from the battery clocks of the console ND-100 computers, whenever the MHC is reloaded. It is used to update the system clock in other computers on the network whenever they are restarted.

5.5 Assessment of the network

The network is such a key element of the whole control system that its defects are more obvious to the user than its main advantage: it works, within its known limitations, with good reliability and high speed. The following comments should be interpreted in this context.

5.5.1 Comments on the design

i) Topology

The restriction to interconnected stars leads to efficient and robust table-driven routing and address mapping. Its obvious disadvantage is the absence of redundancy. However, at least in PS experience, the reliability of the node hardware and software is such that this disadvantage is acceptable (mean time between interruptions of MHC service is at least a month, and interruptions longer than a few minutes are extremely rare).

ii) Addressing

Eight-bit addresses are only marginally wide enough. In a CERN-wide context, at least 16-bit addresses should be used.

iii) Packet size

Statistics show that 128 bytes is a good choice of packet size for control applications. On the other hand, an extensible packet header is necessary since 8 bytes is too restrictive. Of course, this could be done by eating into the packet body for header extensions.

iv) Message integrity

Since packets stay in order, there is theoretically no possibility of a message arriving apparently correct but in fact incomplete. Unfortunately, however, this can in practice happen owing to hardware corruption of a message header; this is not just a theoretical possibility and (at PS) has resulted, for example, in corrupted files. It can be argued that a packet numbering scheme or equivalent is necessary for secure transmission of multi-packet datagrams.

v) Flow control

One can live without end-to-end flow control, and small buffer pools, in a predictable world. Because much of the traffic in an accelerator control system is linked to (and limited by) the accelerator cycle, point-to-point flow control is sufficient as long as the work load generated per cycle does not exceed the limiting throughput, and as long as the buffer pool in the node is big enough to hold one cycle's messages.

At PS, we have had some flow control problems, manifested as data links closing owing to flow-control timeouts. Often these closures are due to service programs in a satellite computer failing for one reason or another (not necessarily a software bug), but in some cases they are caused by a computer (normally a console) which cannot process the incoming traffic. With accelerator cycle times of the order of 1 s (in contrast to 10 s at SPS) this is a real problem. End-to-end flow control is desirable for certain types of datagrams, i.e. those sent unexpectedly to service programs.

vi) Datagrams and services

At PS, a real need has emerged for program to program communication (not necessarily by virtual circuit), virtual terminal services, generalized file access, etc., in general with less stringent time constraints than for the

'traditional' SPS services. Possibilities for extension in these directions should be provided on top of the basic datagram services, most elegantly by providing an 'escape' header (see comment on packet size). This seems in fact to be possible without reducing the efficiency of the basic services, which of course is crucial in a real-time control system.

5.5.2 Comments on the implementation

As described above, the implementation is in layers (drivers, transport station, user interface, service programs). For reasons of efficiency, the transport station is implemented on a dedicated hardware level, and accesses the system nucleus by physical address. Similarly, the drivers are written as operating system components. The user interface and service programs share data structures and subroutines with the transport station (by shared virtual address space in SINTRAN III).

This philosophy largely avoids the overhead of operating system calls in the process of moving data between user space and the link. This is more significant in obtaining high performance than the restrictive protocol design since it produces a saving counted in milliseconds rather than microseconds.

The price of this implementation philosophy was low-level language coding and relatively unprotected software. This leads to a shake-down period of up to several years to eliminate all significant bugs.

The published implementation documentation is not up to a satisfactory standard (neither at SPS nor at PS). Internal documentation of the software has slowly and painfully improved over the years.

6. CAMAC ACCESS

CAMAC access from the FECs is a fundamental and critical aspect of the system, but it can be described relatively simply. To each FEC are attached one or more CAMAC crates driven by the external parallel I/O bus of the NORD-10, via a CC-N10 (or JCC-10) crate controller. These crates are referred to as 'parallel' or 'mother' crates. Each parallel crate contains the modules needed to drive one or more Serial CAMAC loops, an optional DMA module, and (in the case of one crate per FEC) an external interrupt module and a PLS receiver. The last two allow software in the FEC to be synchronized with external events and to be conditioned by beam sequencing information.

The associated system software consists of

a) the CAMAC driver itself (named CAMDR) and its tables,
b) a real-time program responding to all CAMAC interrupts (known as 'LAMs' or 'look-at-mes'),
c) a real-time program to detect DMA timeouts,
d) a library allowing connection of programs to Serial LAMs,
e) a monitor call allowing a program to wait for or be connected to external events,
f) a set of relevant NODAL functions.

The process manipulation package (see Section 5.3.1) provides the user's interface to parts (d) and (e), but CAMDR is called directly from applications programs.

6.1 CAMDR and its tables

The speed of CAMAC access is a critical point in the control system. For this reason, CAMDR is memory-resident in RT-Common and runs in privileged mode, like the network access routines (see Section 5.4). It offers the user access to parallel or serial crates, in single-word or block-transfer modes. To gain speed, it directly executes IOX (input-output) instructions, and when necessary disables interrupts in order to prevent multiple access to hardware registers. It never issues SINTRAN III monitor calls unless the CPU must be relinquished during DMA transfers.

Nevertheless, a single Serial CAMAC transaction takes about 400 μs of CPU time despite repeated hand-optimization. This is primarily caused by the clumsy hardware configuration. DMA transactions are worthwhile only for block transfers of at least 20 words, and have not been installed operationally since the time-critical CAMAC traffic consists overwhelmingly of very short transactions.

Several tables are associated with CAMDR in RT common. The most important one is the CAMAC description table (CAMDT), which describes the configuration of the parallel crates (including the positions of the basic modules such as the serial drivers) and, by use of a bit map, defines which serial crates are present in the various loops. CAMDR makes continuous reference to this table when preparing transactions, and the NODAL function INCAM, which initializes the whole serial CAMAC transmission system, also depends on this table.

Other tables associated with CAMDR are:

- a bit map in which power failures in serial crates are indicated for surveillance purposes (POWFA),
- table to record connections of programs to serial LAMs (CONTB),
- a buffer for the most recent PLS telegram (PLSTB),
- table to record synchronizations with external events (PSYTB).

6.2 Other CAMAC software

The real-time programs for CAMAC handling are all loaded on the same virtual memory segment, which for best response time is fixed in memory. They are:

- PCLAM (Process CAMAC LAM handler), which is connected by SINTRAN III commands to all relevant LAMs from the parallel crates: if several LAMs occur together, they will be handled one after the other by PCLAM.
- DMASU (DMA surveillance) is scheduled once per minute to identify and abort blocked DMA transfers. In addition, it will re-initialize any parallel crate which has been inhibited for some reason.
- POWFL. This is an optional very small program to print a 'power failure' message at low priority.

The monitor call for synchronizing program with external events (i.e. combinations of external interrupts and specified PLS conditions) is installed as a 'user' monitor call number 173. (It should be noted that a totally different version, with the same external specification, is installed on the console computers: this masks the radical differences in CAMAC layout between FECs and consoles). The FEC version of MON 173 sets the appropriate indications in the table PSYTB in RT common according to the option used (wait for event, hook program to event, unhook from event). Event processing itself is handled by PCLAM.

MON 173 is highly dependent on the version and generation of SINTRAN III and must be adapted and regenerated whenever SINTRAN is updated. Thus its installation is in effect an extension of the system patching process described in Section 3.

The present (Spring 1984) version relies on RT common in the FECs being in physical memory bank 0, and allocates a CAMAC DMA buffer also in bank 0. This is incompatible with the new memory layout used from SINTRAN H-version onwards (see Section 5.4).

A library SLAMLIB is provided to allow connection and disconnection of programs to serial LAMs via the table CONTB; this library, similarly to MON 173, is mainly used by the process manipulation package.

The suite of NODAL routines includes principally:

CAMDR:	direct access to CAMAC driver,
SCAM:	simplified access to serial CAMAC,
INCAM:	initialize CAMAC,
UNBYP:	'un-bypass' a serial crate,
TRCDT, etc.:	read or write the various CAMAC tables,
ASSIGN:	assign CAMAC LAM to logical unit number.

The general system library, in addition to an interface to CAMDR, contains a function PLSLI which returns the state of a given PLS line.

7. LANGUAGES

From the earliest studies, it was apparent that the PS Control System's manpower cost would be dominated by applications software. For this reason, the choice of languages for applications became a major issue in the system software design, to the extent that the choice of a system programming language was a very subsidiary point. It was concluded that the investment of several man-years in languages would be cost-effective if it reduced applications programming time (including design, debugging and documentation) by even 10%.

It was also concluded from an analysis of data rates and computer speeds that the 'everything in NODAL' philosophy of SPS [2] was physically incapable of coping with PS requirements. It should be emphasized in this connection that the syntax of NODAL [31] precludes full compilation of all NODAL constructs, in particular of certain string-oriented commands very much used in practice.

It was also clear that a wide range of programmers had to be provided for, ranging from equipment specialists programming a few times a year to people trained at postgraduate level in computer science. Quite apart from varying technical requirements, no single programming language could satisfy this varied 'public'.

As a result, a 'triad' philosophy was adopted: low-level language for system software and special cases, the NODAL interpreter for some applications, and P^+, a 'high quality' compiled language, for general applications.

Although this philosophy was adopted in early 1978, following experience with the pilot project using the SPS philosophy, in practice the compiled language P^+ was not fully available until late 1981. In consequence, the EMs were written in low-level language, clearly an undesirable situation but not without advantages of efficiency and flexibility. Another consequence was that the first versions of many applications programs (particularly interactive programs in the consoles) were written in NODAL, reducing both their performance and their engineering quality drastically. At the time of writing, the EMs remain in low-level language, but much of the interactive software has been converted to P^+.

It is not possible at the present point in the life cycle of the control system to assess whether the investment in language development has indeed been cost-effective. What is clear, however, is that it has provided a focus and a

direction for much of the software effort in the project and has been a source of motivation.

The remainder of this section reviews the more technical aspects of each part of the languages 'triad'.

7.1 Low-level languages

In the NORD-10 and ND-100, two low-level languages are available: the MAC macro-assembler (which exists in several variants), and the NORD-PL intermediate-level-language compiler. The NORD-PL compiler generates MAC as its output, and provides escape mechanisms allowing embedded assembly code in NORD-PL source programs: thus a NORD-PL programmer must also have a working knowledge of MAC.

The four variants of MAC are as follows:

MAC: This is an 'in situ' assembler, i.e. it assembles a source file in a single pass into the same memory space as that containing the assember itself. This variant is a historical relic from the days of paper tape but is parent to the following three and explains their peculiar nature.

FMAC: This is an 'image file' assembler, i.e. it assembles a source file into a file considered to be a 64 Kword memory image. Thus it is primarily an absolute assembler. Optionally it can produce binary relocatable format (BRF), but without detecting all errors.

MACM: This is a stand-alone version of MAC, which does not need an operating system and includes drivers for various mass storage devices. It is used to install a SINTRAN system from floppy discs onto a system disc and contains mysterious pseudo-operations for this purpose, including the famous HENT (Section 3.3).

DMAC: This is a debugging version of MAC, capable of acting on SINTRAN III virtual memory segments as FMAC acts on image files. The human interface of DMAC renders it unusable.

The syntax and semantics of MAC are simplistic, the macroprocessor is inadequate, and the assembler as a whole is inconvenient, incomprehensible, and unreliable. It is difficult to imagine a less satisfactory software product, and the authors have certainly not encountered one.

Given this basis, NORD-PL provides a considerable enhancement of low-level programming facilities. It is a language of the PL-11 class [32] providing elementary structured programming constructs without hiding the machine registers. Thus it enhances programming productivity and software quality with minor influence on speed or size of code. It has therefore been a general policy to write all new low-level software in NORD-PL, and very few exceptions to this policy have been made.

The system software components of the software layout were almost exclusively written in NORD-PL (and exceptionally in MAC). The system software imported from SPS, principally the network software and the NODAL interpreter remain in MAC for the most part; however, enhancements to these packages and even some SINTRAN system patches were written in NORD-PL.

When it became clear that a substantial number of applications software modules would be written in NORD-PL, a set of facilities to simplify this was provided. This consists of macros, run-time support routines, and written recommendations, allowing NORD-PL programmers easily to conform to the general software layout and to the P^+ and ICCI subroutine conventions. This arose, historically, before the P^+ code generator existed, so the design of the latter was constrained by the need for compatibility with existing NORD-PL environment (in its turn constrained by NODAL!).

As mentioned above, the DMAC debugging assembler proved to be useless. A replacement DMAC ('debugger made at CERN'), implemented in Pascal [33], has been used to good effect for low-level debugging, but has proved difficult to maintain.

An excellent suite of cross-software provided by DD gives assembly and link-editing facilities for the TMS9900 microprocessor embedded in the CAMAC ACCs.

7.2 NODAL

The NODAL language is defined in Shering [31]. The NODAL interpreter consists of a pure-code re-entrant kernel, which is activated once for each NODAL process (whether interactive or a server). In addition, each process must possess a read-write working area which contains (among other things)

- a global block containing context information,
- a dynamic memory area containing source code and variables,
- a stack.

The code of the re-entrant kernel consists of a large number of relatively small assembler coded routines, calling each other in a recursive manner as interpretation of the source code proceeds. When the interpreter encounters an identifier in the source code, it is sought in various linked lists, in general by sequential search. One of these lists is of particular importance: it contains an element known as a NODAL header describing each 'NODAL function', i.e. compiled functions, such as EMs and other ICCI functions, which may be called from NODAL applications programs. The NODAL header contains the name of the function, a parameter descriptor, and the full address of the entry point (segment number plus 16-bit address).

The version of NODAL originally provided by SPS was embedded in a rather simple way in SINTRAN. The modifications and enhancements made at PS include:

- conversion of NODAL kernel to be a SINTRAN re-entrant segment,
- support of any number of overlayed (ICCI) segments,
- improved file buffering,
- hash table to accelerate access to ICCI functions,
- new interface to network,
- interface to process manipulation package,
- interface to console system,
- new and modified function types and parameter types (see Appendix 1),
- new treatment of EM calls and properties (see Appendix 3).

No functional changes have been made to the language syntax or semantics, and essentially all the basic library routines documented in the NODAL manual have been retained.

It should be understood that the above points, and hundreds of small points not mentioned, have absorbed a considerable, but essentially invisible, software effort. Although NODAL is constructed in a highly modular way, the coding of each module consists of a solid block of assembler code, many instructions per line, with many embedded jumps and few embedded comments. It is a specialized and painstaking task to modify such software.

In practice, NODAL has proved convenient for a wide range of applications, for tests, commissioning and operation, in all cases where software is written by a non-specialist. Problems arise when such a program is required for regular use and is too slow.

Unfortunately, NODAL is unsuitable for full compilation. Nevertheless, situations may arise when it is desirable to develop applications programs interpretively—typically for experimenting with new instrumentation, but later to increase their speed substantially.

There are two possible techniques to achieve this. One is for a specialist programmer to recode the program according to the user's requirements. This expensive technique should be limited to cases where the program in question is accepted as a fully operational applications program.

The other technique is appropriate for cases where the program is only quasi-operational, and can stay in NODAL, but a reasonable speed improvement is needed. A compiler has been developed (partly by a software house and partly at SPS) with which it is possible to compile NODAL defined functions written in a subset of NODAL. The resulting binary code can be loaded with an LDEF command exactly like the original defined function in source code. The NODAL subset excludes all string commands, all I/O commands, computed DO or GOTO, dynamic array sizes, error trapping, etc. The code produced occupies about twice as much memory as the source, and runs about ten times faster.

Thus programs which are possible candidates for partial compilation must be written according to the following simple rules:

i) The main program contains terminal or console handling, equipment access, string handling, and general organizational code.
ii) One or more defined functions loaded with LDEF contain(s) numerical computation—especially array and matrix calculations.

If such a program is later deemed too slow, the defined functions can be compiled off-line and subsequently installed on-line in binary format. In principle, the main program need not be changed.

This facility is available on a few computers at PS where it has proved desirable.

7.3 Compiled high-level languages

Essentially four languages were considered during 1977–78 as serious candidates ;

FORTRAN
CORAL-66
Pascal
P+

7.3.1 FORTRAN

The FORTRAN compiler available for the NORD-10 in 1978 was an extended FORTRAN IV compiler generating relatively efficient machine code. The majority of the software specialists involved in the controls project felt, however, that it was essential to use a language of more modern design with support for data structures and structured programming.

Nevertheless, FORTRAN has been used in a number of cases in the control system and a limited facility exists to install FORTRAN main programs and ICCI routines (in the TEMPX computer). Unfortunately, this facility is incompatible with the newer FORTRAN-77 compiler for the ND-100.

7.3.2 CORAL-66

This language, which provides the limited structured programming facilities of Algol-60, but no support for data structures, was in use in the NORD-10 computers of the ISR control system [34].

CORAL-66 had been used for many successful real-time applications, mainly in Britain, and was a serious candidate for use at PS. In the end, however, it was not considered to have a convincing advantage over the alternatives.

7.3.3 Pascal

The first Pascal implementation on ND computers was in fact done at PS in 1976-77 [35] and by 1978 we had considerable experience of both the advantages and disadvantages of Pascal. The advantages are in the area of structured programming and data structuring; the disadvantages are in the area of separate compilation and in applications connected to the real, rather than the academic, world. Thus, if Pascal is to be used for a 100 man-year real-time project, it can only be in the form of a considerably extended and non-standard Pascal.

Nevertheless, the PS Pascal was upgraded to an acceptable level of efficiency [36] and in a slightly modified form known as RT Pascal was used for a number of early applications programs in the control system. Only during 1983 did it finally become possible to put PS Pascal on a zero-support basis and switch to the official ND compiler.

7.3.4 P^+

Following the publication of the description of Pascal, and observations of its limitations, a number of successor-languages have appeared, mainly oriented towards separate compilation of large software systems. Among these is P^+ [37] designed in 1978 and released for the NORD-10 and ND-100 in 1981. The language is described in the reference cited, and the implementation will be described elsewhere; only a few general remarks are made here.

Since the primary motivation for designing and implementing P^+ was the provision of a high-quality language for applications programs, a consistent design philosophy was adopted:

- syntax designed to be easy to read, not easy to write,
- full syntactic checking of separately compiled modules,
- consistent implementation of abstract data types,
- built-in treatment of character strings,
- powerful support for compile-time parametrization,
- built-in features for distributed computing (RPC),
- built-in features for language mixing (ICCI convention).

The resulting language is more sophisticated than Pascal but less complex than Ada (no tasking, no generic types, no exception handling). It is distinguished from other languages such as MESA, MODULA, and MODULA-2 by its primary orientation towards applications programming rather than systems work, although this is by no means a clear-cut distinction and version B of P^+ will contain additional features with a systems flavour.

P^+ was implemented by R. Cailliau and a changing team of collaborators. The compiler and code generator are written in Pascal, and the run-time library in low-level language. Continuous difficulty was experienced throughout the development of version A owing to the limitations of the 16-bit address space of the NORD-10 architecture and owing to the lack (at that time) of a reliable Pascal implementation from Norsk Data. Finally, a special version of the PS Pascal system was created, in which code could be overlayed in the 16-bit instruction space and data (heap variables) could be paged in the 16-bit data space. Fortunately, version B of P^+ will be implemented without these constraints by use of ND Pascal in a 32-bit ND-500 computer.

The P^+ compiler is a one-pass recursive descent compiler (but using precedence analysis for expressions), generating a two-stream intermediate code, with one instruction stream (I-code) and one data description stream (D-code). The code generator is pipelined, using the D-code to define the stack layout of each procedure before translating the I-code into machine instructions. The 'hypothetical computer' corresponding to the I-code has an accumulator and multiple address spaces (rather than the stack of traditional Pascal hypothetical computers) and distinguishes 'hardware' data types (such as integers) from 'software' types such as strings.

The output of the code generator, for version A, is standard ND BRF, which can be immediately link-edited with the output of NORD-PL/MAC when required.

At the time of writing, at least 150,000 lines of applications software have been successfully written in P^+. Despite the usual teething troubles, the lack of a symbolic debugger, and slow compilation times, the language has become popular with its users and has gained the reputation of allowing one to write programs which may well work first time.

8. LIBRARIES

Subroutine libraries are an unglamorous but important part of any large software system. Several categories of libraries can be identified in the system software layout.

i) Libraries completely specific to a given programming language

These exist either because of particular details of the language, or for historical reasons (e.g. multiple versions of trigonometric routines), or for efficiency (when the time to call a shared library would greatly exceed the execution time of the routine). Although such libraries lead to duplication of maintenance effort, their existence is unfortunately inevitable. They exist for NORD-PL, RT Pascal, NODAL (built-in), and P+ (see below).

ii) Resident shared library

This, a single re-entrant library loaded in the RT common area, as mentioned in Section 4.3, is available to all RT programs. It contains the CAMAC driver, the data link access routines, the ACC (microprocessor) access routines, and miscellaneous routines and data. To avoid problems when a new version of the library is installed, a table of jumps to the various entry points is loaded at a fixed address in RT common; programs use the addresses in this table rather than the true entry points.

iii) P+ library

This is a relocatable library, which must not only be link-edited with P+ code, but is also used with NORD-PL code, and extracts from which are included in the RT Pascal library. It contains not only specific routines for P+, but also a substantial number of routines for general access to the real-time environment from applications programs.

The P+ library is mainly re-entrant, although a few components are non-re-entrant and must be loaded once for each independent process sharing the same virtual memory segment.

iv) Overlay segments

A number of libraries are in the form of virtual memory segments accessed from compiled programs as ICCI overlays and/or from NODAL as ordinary NODAL functions (see Section 7.2). Libraries in this category include

- the process manipulation package,
- the global variable package,
- NODAL functions for CAMAC access,
- NODAL functions too big to fit in the NODAL kernel,
- LIB-FUNS, a general 'catch-all' library.

In all segments containing ICCI routines (callable by compiled code), a jump table is placed at the beginning of the segment to avoid link-editing problems.

Coordination, documentation, and extension of libraries is an endless task whose benefits (the avoidance of duplication of effort among users) are hard to demonstrate. A substantial but hidden effort has been invested in the various libraries mentioned above.

9. PROGRAM DEVELOPMENT, TESTING, DEBUGGING, AND ERROR HANDLING

9.1 Program development

Since the late summer of 1976, there has been continuously available in PS a NORD-10 and ND-100 program-development service (apart from the usual interruptions due to back-ups, faults, maintenance, upgrading, and the Christmas holiday). From a slow 256 Kbyte NORD-10 with a few terminals and 33 Mbytes of disc, it has grown to a twin-processor ND-100 and ND-560 system with 4 Mbytes of main memory, 300 Mbytes of disc, 32 terminal lines, and at least 40 active users. It runs round the clock on a self-service basis, and provides editing, filing, compilation, and documentation facilities (plus limited testing possibilities). In addition to a standard SINTRAN operating system and the standard utility programs from ND, it provides access to numerous utility programs from other ND customers and to those written at PS. A TITN link to the control system is used to transmit object programs to their intended target computers. A CERNET link is used to gain access to the CERN computer centre, in particular to allow an incremental backup package to save all modified source files on the IBM Mass Store every night.

The program development system is not part of the control system and its details will not be described here. Nevertheless, it is the primary working environment for most applications programmers most of the time, and it has absorbed a significant but hidden proportion of system software effort. It is only by qualitatively improving program development facilities that one can significantly increase programmer productivity, and our facilities have only just kept up with the demands placed on them, especially by heavy use of the P+ compiler.

9.2 Testing and debugging

In the main, system software has been initially tested 'the hard way' by low-level debugging (including inspecting the memory of a crashed computer). Indeed it is hard to see how else such software can be debugged. Following such initial tests and debugging, more sophisticated tests (looking for re-entrancy bugs) have been carried out by saturation testing. For example, a network load generator was coded in NODAL to execute various network operations in different computers at short random intervals. This program may be activated in multiple copies and in several computers to test a new version of the network software.

Nevertheless, systematic testing of system software is intrinsically difficult due to its diverse and general nature: the best test-bed is real life.

Two computers named TEST and PROBAC ('process backup') are available for testing applications software. The former contains a sophisticated software package to simulate the environment of various FECs; this was implemented by our colleagues in the Exploitation Section. The PROBAC contains standard FEC software, and is connected to a CAMAC crate which users may reconfigure to suit their requirements. PROBAC is used for testing EMs in detail; TEST may be used to test a complete chain of software from the console down.

As mentioned in Section 7.1, a low-level debugger DMAC is available (and is in practice installed only on off-line computers). A similar debugger ACC-DEBUG is available for the ACC microprocessors, (see Section 10). No symbolic or high-level debugger is available; this is a weak point and often forces programmers to insert WRITE statements for debugging purposes.

9.3 Error handling

This too is a weak point. In a complex system, errors arise in many software modules: in an ideal world, they would all be signalled in a compatible format and reported by a standard mechanism. This has not proved possible in the PS system software, because of the historical diversity of the various software components. Major sources of error indications, and their treatment, are as follows:

9.3.1 SINTRAN III kernel

This detects 'spontaneous' errors (unexpected interrupts, memory or disc parity errors, etc.), and errors in programs such as effective addresses lying outside the current program's virtual memory segments. These errors are automatically printed on the system terminal, and in many cases the program involved is aborted immediately.

Nothing can be done in user programs to react to such errors. A SINTRAN patch has been introduced such that two flag bits are set in a dedicated word of RT common: one bit reports that a serious software error has occurred, and the other indicates a serious hardware error. The alarm system in the main operator consoles displays appropriate messages when these bits are set, since human intervention is normally required to identify and correct the problem.

9.3.2 SINTRAN III file system

When any program calls the file system, directly or indirectly, an error code will be returned in the event of failure.

In NODAL, file system error codes are arbitrarily converted into NODAL error codes (see below). In other languages, they are returned to the user as a completion code.

9.3.3 System libraries

All locally implemented system software libraries adopt the philosophy that all routines return a completion code, either explicitly as a parameter, or implicitly as a global variable. Furthermore, these completion codes are uniquely allocated and do not overlap the SINTRAN file system completion codes.

Thus all completion codes may be passed to a general library routine SWEAR (software error alert report), which is intended to print an appropriate message. At present, only a very simple version of SWEAR is available, giving plain text for file errors and a cryptic message for other completion codes.

9.3.4 Language errors

The run-time systems of NODAL, RT Pascal, and P$^+$ contain their own error-detection mechanism, generating internal error codes. NODAL also has a built-in exception-handling mechanism, which must be used even to treat 'normal' errors such as incorrect file names or network timeouts. In the compiled languages, 'normal' errors are handled via completion codes and require no special mechanisms. Run-time errors are usually catastrophic failures such as 'value out of range' and are usually eliminated during testing. In P$^+$, this is simplified by a trace-back printed after the occurrence of an error, if the appropriate debug files are present.

9.3.5 Other errors

Errors may also be detected by our own software which, similarly to SINTRAN kernel errors, can be attributed to no program. These include

- errors detected within the network software,

- errors in the MHC,
- errors detected during CAMAC LAM handling,
- errors inside an ACC microprocessor.

These errors cause an appropriate print-out and/or the setting of status indicators.

10. MICROPROCESSOR SUPPORT

Many of the CAMAC crates in the control system contain an ACC built around a Texas Instruments TMS9900 microprocessor. This is a 16-bit microprocessor, standardized at CERN some years ago, with a 64 Kbyte address space. In the ACC, 32 Kbytes of the address space correspond to real (RAM) memory, and 32 Kbytes are used to map CAMAC station numbers, sub-addresses, and function codes. When the microprocessor program accesses one of these addresses, the ACC takes control of the CAMAC crate and performs the corresponding CAMAC cycle. In addition, the ACC's 32 Kbyte memory can be accessed from the host minicomputer by dedicated CAMAC functions (set address counter, read/write memory) and the microprocessor can be stopped and started by other dedicated CAMAC functions.

The ACCs were introduced to provide hard real time intelligence at the CAMAC level, principally to implement pulse-to-pulse modulation of control settings. They are also used for data reduction in complex instrumentation systems, as graphics pre-processors, and in stand-alone test systems. A suite of software to suit these applications was implemented and is summarized below [38].

10.1 Cross-software on the program development system

This is a group of programs provided by the Data Handling Division (J. Montuelle) as follows:
Cross-assembler, producing CUFOM output (CERN Universal Format for Object Modules),
Link-editor,
Librarian,
'Pusher', which converts CUFOM to Texas Instruments object format.

These programs are supplemented by CREATE-ACC-IMAGE, which converts Texas Instruments object format into a binary image format, used to achieve faster down-line loading.

10.2 ACC Monitor

This is a simple monitor, in its standard form occupying less than 2 Kbytes. It provides a very limited set of system calls allowing the connection of user tasks to external interrupts, output of debugging messages to a terminal, and elementary communication with the host minicomputer. It also provides a breakpoint mechanism used by the debugger (see below).

The monitor enters one of three modes when the microprocessor is restarted:

i) On-line mode. The monitor starts a pre-designated user task and then idles (until a system call or interrupt requires treatment).
ii) Debug mode. As on-line mode, but monitor polls for debugger requests instead of idling.
iii) Stand-alone mode. Monitor bootstraps an image from floppy disc.

In stand-alone mode, the monitor is in EPROM; in the other two modes it is in RAM and is down-line-loaded from the host minicomputer. However, the monitor is identical in all cases and distinguishes the modes dynamically.

10.3 ACC-DEBUG

As mentioned in Section 9.2, this is a low-level debugger. It is a fully interactive debugger, with a wide range of facilities including breakpoints, single stepping, and disassembly. It is coded mainly in Pascal and runs as a real-time or background program on the host minicomputer, using CAMAC to access the memory of the ACC. It relies on the breakpoint system call of the ACC monitor to detect breakpoints and to implement single stepping. Theoretically this is unfortunate since the monitor is in RAM and may be overwritten by serious bugs, but this is a minor problem in practice.

ACC-DEBUG is a large program representing many months of effort. It is fully described in Krüger [38]. It is clear that without such a tool, an ACC program hidden away in a CAMAC crate would be essentially impossible to debug.

10.4 Communications

A number of routines exist for communication between real-time software in a FEC and the ACCs in the various CAMAC loops. The most important are described below.

PUTBL and GETBL: These respectively put and get a block of binary data, i.e. transfer data between a buffer in the user program in the FEC and a buffer in the ACC's RAM. They are principally used by EMs.

STACC and TRACC: These respectively check a status indicator and trigger a status indicator in the ACC. The ACC monitor provides equivalent system calls, and all these operations are atomic; thus the status indicators may be used to construct safe protocols between applications in the FEC and the ACC.

LDACC: This down-line-loads an ACC from a memory image file, and is used when an ACC must be reloaded for operational reasons.

11. PERFORMANCE

During the design of the control system, the greatest concern about performance was in relation to the two HRT activities necessary to keep the particle beams running: generation of PLS telegrams, and pulse-to-pulse modulation (PPM) of equipment set points. Indeed these two activities are closely related, since the PLS conditions the PPM in real time. After detailed studies by the relevant specialists, it was decided to implement PPM exclusively by means of ACC microprocessors essentially dedicated to this purpose. The PLS telegrams, however, are generated by high-priority software fixed in memory in the PLS front-end computer. These two decisions were justified by elementary throughput calculations which have been borne out in practice. The load on the PLS computer is increased by the generation of dense video displays in real time; this load was in fact underestimated and means that a cache memory is essential. Nevertheless, it can be stated that there are no problems in the performance of the essential HRT software.

The situation is more complex and subtle in the rest of the system. Programs are activated in response to operator interaction on the main consoles or on local terminals; they are swapped to and from disc in an unpredictable way; they compete for computer time and memory space; they interact through the network; their execution time may depend on unpredictable external events or data. The result is that elementary throughput calculations give a highly optimistic view, and that the phenomena of statistical queuing theory [39] must be considered.

11.1 Theoretical CPU load limits

Two external measures of performance are considered: throughput and response time. These are constrained by the resources available (CPU time, memory space, I/O time) and by the statistical distribution of demands on the system. The relationship between the measures and constraints is complex, especially with a cyclic process with some synchronous and some asynchronous work load. However, from the most general results of queuing theory it is known that the relationship between *average* response time and the utilization factor (where 0 = system idle and 1 = maximum theoretical throughput) is of the general form shown in Fig 5:

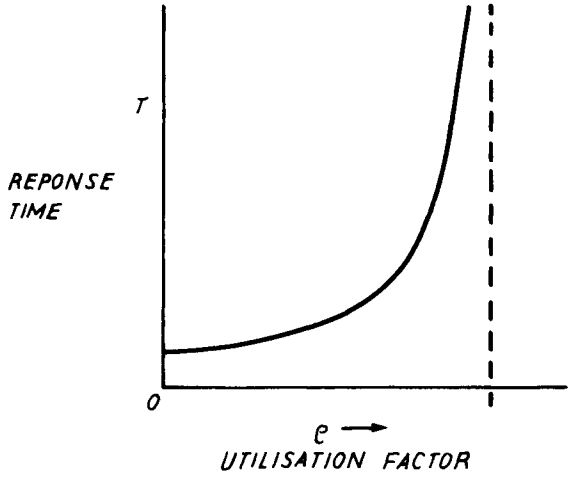

Fig. 5 Response curve

Mathematically, the simplest model of a queuing system gives the result:

$$T/x = 1/(1 - \varrho),$$

where T = average response time, x = average execution time (CPU time, etc.), and ϱ = utilization factor. More complex models give much more complicated results, but all response curves are in practice very similar to that above.

[Note that if the system (CPU) is busy with a fixed load for a fraction b of the time, the above result becomes

$$T/x = 1/(1 - \varrho - b),$$

where T, x, and ϱ now refer to the random load not contained in b].

The consequences of this response time result are very important. In any situation with a random load (typically the case in all our computers) the trade-off between response time and utilization is given by the above curve. The price of increased utilization, beyond a certain point, is a dramatic increase in response time.

To make use of this theory, one must introduce two numerical values:
- the maximum acceptable value of T/x,
- an estimate of the theoretical throughput.

From these, one can readily compute the work load which *on average* can be achieved with acceptable response times, and one can evaluate the cost in response time of exceeding this load.

Once the acceptable load has been calculated, then for a given number of simultaneous applications, an allowable load per application may be calculated. Any application which noticeably exceeds this load will cause performance problems.

In the process computers (FECs), the maximum acceptable value of T/x is taken to be 1.5 (i.e. the average response time is 50% greater than the average execution time). In the console computers and the network, the maximum acceptable value of T/x is taken to be 2 (i.e. the average response time is twice the average execution time). This discrimination reflects the fact that the process computers must be more closely synchronized than the others. On applying the above formula, the consequence is that the utilization factor must not exceed 1/3 for the process computers, or 1/2 for the consoles and network, to maintain acceptable response times.

11.2 Memory utilization

In computer science, 'working set' refers to the total virtual memory requirement of a system, integrated over a sampling interval comparable to the disc swap time. In our case this time is 50–100 ms, comparable also to the critical time slices of the accelerator cycles. If the working set noticeably exceeds the available physical memory, an exponential increase of *all* response times occurs, regardless of theoretical CPU load or priority.

Similarly to queuing theory, the theory of working sets is rather complex [40]. To determine the maximum acceptable ratio of working set to physical memory in a simple-minded way, let us define

w = ratio of working set to physical memory size ($w > 1$),
s = physically possible rate of page swaps,
r = rate of page references (where a page reference is a group of consecutive references to words within the same page).

Then the rate of page faults may be estimated as

$$r(w - 1)$$

and for stability it must be true that

$$s > r(w - 1)$$

or

$$w < (s/r) + 1 .$$

For our swapping discs, s = 20 swaps per second (average disc access = 50 ms). The value of r depends entirely on the structure of the programs and on the CPU utilization. Assuming purely sequential programs (i.e. traditional coding style), 30–50% CPU utilization, 2 μs and 1.5 words per instruction (including data) we estimate

$$r = 300 \text{ references per second}$$

giving $w < 1.07$ in this case. However, with modular and looping programs using a stack for their data the value of r is potentially much lower, say by a factor of 5, giving $w < 1.33$.

We adopt 1.25 as the maximum acceptable ratio of working set to physical memory. A computer needing more than 125% of its physical memory size within a fraction of the accelerator cycle time is likely to experience severe disc thrashing.

11.3 Practical observations

A number of performance analyses were carried out during the period following the control system start-up in 1980. (These were quite independent of the analysis of applications software performance [41]).

Observations of the performance of the network, in particular of the MHC at the central node, were made using tools developed at SPS [42]. These have shown a broadly similar traffic pattern to that observed at SPS, with the differences being easily explicable. The load on the network, which can be analysed down to a resolution of 100 ms, has gradually increased since 1980 as the number of FECs and consoles has increased. To a good approximation, the load is 5% for routine surveillance by the MHC, plus 3% on an average per active console, with peaks of 10% per active console. It can be concluded that the network could handle up to 10 consoles without risk of exceeding 50% of its capacity on an average, peaking at 100% busy.

It is interesting to note that the 10% peaks of activity, normally limited to a single 100 ms sample, correspond to individual synchrotron cycles despite the stochastic nature of the total computing load noted above. These peaks occur approximately ten times more frequently than the corresponding peaks observed in the SPS network.

Individual network transactions have been timed approximately as follows:

Minimum NODAL EXEC	30 ms
Minimum RPC	20 ms + 4 ms per parameter
Single packet end-to-end	7 ms.

These measurements relate to NORD-10s CPUs with cache memory. ND-100 CPUs give reductions of up to 30% in these times.

Observations on the general behaviour of an individual computer have been made using two SINTRAN III tools:

i) RT-PROGRAM-LOG. This allows one to measure, with a sampling interval of about 30 ms and a resolution of ~ 1 s, the CPU utilization, swapping rate, and disc utilization, of a given program and of the system as a whole.

ii) START-HISTOGRAM and related commands. These allow one to trace the position of the P-register (program counter) of a given program in the form of a histogram: thus the regions in which a program spends or wastes most of its time may be identified.

These tools have been used whenever necessary to analyse the performance of computers under heavy load. In general, the results of these measurements are rather specific and are not given here: they have made it possible to quantify working sets and physical memory requirements (so as to keep the swapping rate low) and to justify the purchase of cache memories to accelerate the FECs. They were also a major factor in justifying the conversion of the consoles from NORD-10 to ND-100 processors. The theoretical considerations described above proved useful in interpreting the measured statistics.

One particular statistic of great importance was indeed measured. In the FECs, a large proportion of the CPU time is spent in one program, the Main Display Routine (MDR). This is a general-purpose applications program, activated once per accelerator cycle, which acquires process data (by calling selected EMs), prepares them for display, and sends them through the display pipeline (Section 5.2, also see Mérard [27]). When several consoles are taking data from the same FEC, the MDR may consume more than 50% of the CPU time and have a working set of at least 150 Kbytes (owing to calling many EMs per cycle). Thus, the synchrotron operators' requirement for multiple updated displays on the main consoles dominates the computing resources of the FECs (and of course of the console computers) and essentially defines the necessary size and speed of the computers. The MDR ensures that the FECs in practice far exceed the desirable utilization factor of 1/3, and indeed response time is poor when several consoles are sharing one FEC.

Between 50 and 98% of the calls to CAMDR come from the MDR (with the percentage varying between FECs). A program-counter histogram of the MDR (and of other FEC programs) shows that most individual programs actually spend up to about 50% of their time inside CAMDR. In fact, this peak in the histograms is localized within one very small section of CAMDR which performs a single-word transfer on the Serial CAMAC highway. This code is complex and relatively slow since it has to make several parallel CAMAC transactions in order to realize one Serial transaction plus unavoidable error checking. Thus, a transaction whose 'hardware' time is about 50 μs, in practice takes about 400 μs despite careful optimization of the code. Today, this is the major performance bottleneck in the control system.

A special measurement tool was developed by I. Killner to investigate the statistics of queues within SINTRAN III. Queues are normally based on semaphores (either internal or user-level) and I/O devices all possess a semaphore property for queuing purposes. The tool [known as the Semaphore Monitoring Sytem (SMS)] was based on a patch on the SINTRAN clock interrupt handler. Each 20 ms, SMS could update a histogram in resident memory to indicate the size of the queue for each of five semaphores. Histogram bucket 0 indicated no reservation, bucket 1 indicated that the semaphore was reserved, bucket 2 that it was reserved with one program queued, etc. A slightly modified version of the patch could produce an equivalent histogram for the execution queue (0 = idle loop, 1 = program running, 2 = program running and one program waiting for CPU, etc.).

In addition, a small resident program dumped the histograms to a direct access disc file at 1 s intervals until the file was full. (Empty histograms were not dumped, to save time and space.) The data could then be analysed off-line. The overhead imposed by running SMS was immeasurably small, so was assumed not to affect the statistics.

The SMS was used on the program development computer to confirm suspicions that the main performance problem (in 1981) was that of long queues for disc access—typically the disc was 95% busy with four or five programs waiting for it most of the time. This situation was not significantly improved until the machine was upgraded to a ND-100 and ND-560 with 2 Mbytes of memory in 1983.

The SMS was also used to investigate semaphore queues in the FECs and console computers. It was determined that the semaphores used to protect access to the ACCs against re-entrancy were busy for no more than a few per cent of the time: thus many ACCs could share the same semaphore without affecting performance. Various other semaphores were surveyed without finding indications of excessive queues, except of course for the CPU queue and/or the swapping semaphore in overloaded computers. Finally, the raw samples for the swapping semaphore in a busy console computer were analysed to see whether the cyclic nature of the underlying load was reflected in the variations of the queue size, i.e. whether the swapping rate of the computer was correlated with the synchrotron cycle. No such correlation could be found.

These generally negative results from the SMS led to a positive conclusion. SINTRAN III is, on this evidence, a very effective virtual memory and multiprogramming system which responds to increased load in the manner of a theoretical queuing system, without discontinuities or surprises. By increasing the basic resources of CPU speed or memory, one may increase performance in a predictable (but non-linear) way.

12. DOCUMENTATION

System software documentation always has been, and presumably always will be, a problem. To be complete, the documentation must reflect the complexity and above all the diversity of the software: but complexity and diversity make the documentation very difficult to produce and to use.

The available documentation is in a number of forms:

12.1 Suppliers' documentation

Our software suppliers (ND, other companies and organizations, and other CERN groups) of course supply some form of documentation. A central reference set of 'system copies' of all such manuals is kept up to date, and copies of all useful manuals are available for users. The quality of such documents is largely out of our control, although an important reason for membership of user groups such as NOCUS is to exert pressure for improved documentation. Except for manuals intended for commercial customers, the quality and maintenance of ND manuals is often felt to be below industry standards. In particular, at the time of writing, the internal documentation on SINTRAN III has not been updated for four years!

12.2 User documentation

Since the beginning of the control system project, we have concentrated the bulk of locally written system software documentation in a format known as 'COOKBOOK'. This is in the form of various specialized sections (e.g. 'languages', 'utilities') consisting of individual articles ranging from one to thirty pages according to the topic. Although the articles are independently distributed in loose-leaf form, there is a global index and list of contents. Of course, COOKBOOK is maintained on the computer (by B. Bunaciu). Originally, this was done with the REPORT text formatter developed in PS; now COOKBOOK is being transferred to the ND NOTIS system (whose text formatter is based on REPORT).

A rather special COOKBOOK article is that which documents all system software library routines (as outlined in Section 8). This article applies equally to NODAL, NORD-PL and P$^+$, and the same source file is used to generate the P$^+$ 'definition modules', which allow programmers to import library declarations.

Certain aspects of system software are documented by *ad hoc* notes not included in COOKBOOK owing to their size or restricted applicability. These include the P$^+$ language [37], DMAC [33], and the process manipulation package [23].

12.3 Implementation documentation

We have unfortunately adopted no systematic method for this; each programmer has done what his professional conscience dictated! For the bulk of the locally written system software, either paper documentation or internal documentation in the source code is available and sufficient. Two areas are less than satisfactory: the TITN network software (which is complex and coded in a primitive assembly language with obscure identifiers and limited comments); and the P$^+$ compiler and code generator (which, in version A, are highly complex and also lack adequate internal or external documentation). For TITN, a dossier of diagrams and tables has been prepared, but is permanently out of date due to being in manuscript. For P$^+$, the version B compiler and code generators will be clarified and better documented.

13. ORGANIZATION AND METHODS

As can be deduced from the Acknowledgements, the permanent staff have been outnumbered four to one by short-term members of the system team. Fortunately, almost all of the people involved had one thing in common: a belief in the value of top-down design, structured programming, and other aspects of software engineering. On the other hand, systems programmers are notorious individualists and do not take readily to 'industrial' working methods or pedantic management. For this reason, in general, rather informal working methods have been adopted. A list of the responsibilites of each individual, and a list of all known current action items, are kept up to date. People work towards agreed general targets and deadlines, but without formalized progress reviews, code inspections, structured walkthroughs, etc. Of course, some situations have called for more explicit managerial attention.

In particular, when it became apparent in 1979 that the P$^+$ compiler would be substantially later than hoped for, a careful review of the work remaining was carried out, with several independent estimates being made of the manpower for each job. A critical path diagram of the remaining work was developed to reach a full understanding of the status of the project and to improve its visibility to Management. A similar exercise was performed for the enhancements to PS Pascal which made it suitable for real-time applications.

Indeed, RT Pascal was produced almost precisely in accordance with the critical path diagram (by three people working for about three months in very close collaboration). Version A of P$^+$ was produced rather more slowly than the critical path diagram indicated, initially owing to one item being greatly underestimated (namely enhancements to PS Pascal to support very large programs such as the P$^+$ compiler itself) and consequently owing to changes of personnel. Nevertheless, the diagram and associated documents were vital tools in understanding and stimulating the progress of the project.

Project control for the control system as a whole was centred around a Controls Contact Meeting (CCM), held every one or two weeks and chaired by the project leader B. Kuiper. The CCM was the formal means of communication between the System Software Section and the rest of the project and was attended regularly by B. Carpenter and W. Remmer, and occasionally by R. Cailliau and G. Cuisinier for special topics. While the strategic importance of such a forum should not be neglected, direct informal contacts with users and written proposals were more important for technical questions.

Rather few formal System Software Section meetings were held, mainly during 1978 when the team peaked at eight or nine people. At other times, the team was sufficiently small that it was found more convenient and pleasant to communicate over coffee or lunch. Of course, meetings were organized on technical topics when necessary, but this normally also involved people from outside the system team.

Rather rigid control was imposed on one aspect of the team's work: installation or modification of system software. From the day that one of our FECs first controlled beam (in fact the AA computer in Summer 1980), such activity has been subject to some bureaucracy:

i) A 'system software modification note' is filed centrally;
ii) The new version is installed and tested on an off-line computer (BACKUP or PROBAC or preferably both), and left to run there for some time;
iii) If relevant, it is provisionally installed and tested on a single console computer;
iv) Prior to each accelerator shutdown, a distribution procedure (using floppy disc or the network) is prepared so as to distribute and install all new versions which have passed through the above steps. This procedure is also tested (on off-line computers)!
v) As early as possible in the shutdown, the new versions are distributed, installed and briefly tested. All file updates are logged twice (once in the computer log books, once in a 'Systems software versions book').

In practice, the above bureaucracy can be applied flexibly in the case of extremely urgent bugs, but the off-line test and the multiple loggings must never be omitted. Without these elementary (and very brief) precautions, it is impossible to keep track of the installed software, to guarantee consistency across more than twenty computers, and to identify easily the secondary effects of modifications.

Mild bureaucracy was also introduced to control the handling of suspected bugs reported by users. A system software fault report (SSFR), modelled on that used by Norsk Data, was introduced. Bug reports from users are only guaranteed to be remembered if they are in the form of SSFRs! A central file of unsolved SSFRs is kept, in particular so that non-urgent problems will not be totally forgotten.

Our experience suggests that a team providing basic support for a large system, like a computer manufacturer's software support staff, absolutely must have these two bureaucratic aids: distribution/installation control and formal fault reports. Otherwise they will become psychologically submerged by daily problems and unable to spare any time for interesting development work.

14. FUTURE PROSPECTS

This report has surveyed in greater or lesser detail all major aspects of the system software of the PS Control System as it stands at the end of the formal project funded in 1976 and completed in early 1984. The system went on-line to the AA in Summer 1980, to the PS Booster in October 1980, and progressively to most of the rest of the CPS complex since then. During this project, about 30 staff man-years and 15 supernumerary man-years have been devoted to management, implementation, and support of system software: perhaps half of the total has gone into support, including a substantial support load ever since the program development service was established in 1976. Apart from continuing support responsibilities, what are the future prospects?

The control system will last as long as the PS complex itself, and will be both pushed and pulled by unforeseeable technological developments. The system team has a continuing responsibility to be aware of and to respond to these developments and to continuously update its medium-term strategy accordingly. In this way, the system team can be a motor for future developments in the control system as a whole.

At the time of writing, the major development effort of the System Software Section is directed towards the provision of facilities for the Motorola 68000 16/32 bit microprocessor which will be the engine of a forthcoming Super ACC. Since applications which currently run in the FECs will undoubtedly be off-loaded into the Super ACC, a sophisticated software environment is required. This will include a P^+ code generator, a full version of NODAL, a multi-tasking operating system, and communications software [43].

In the near future, the FEC software layout must be adapted to run on the latest version of SINTRAN in ND-100 Compact computers, to be introduced for the LEP pre-injector and as replacements for ageing NORD-10 machines.

A smaller current development is the installation of a front-end computer known as LL to support a point-to-point link to the LINAC/LEAR control system. This link will provide multiple virtual channels to be used by appropriate gateway software.

A strategic issue will arise during 1985: should the program development computer be converted to UNIX when this system is released for the ND-500? This, and/or other steps to enhance applications programming productivity, are of great importance in the continuing lean manpower situation. In the same connection, error handling, testing, and debugging facilities should be improved.

Another major issue in the coming years will be the enhancement of the facilities provided by the network and (once all the computers are upgraded to ND-100 processors) replacement of the old, expensive TITN hardware by modern technology such as Ethernet. This topic requires a serious strategic and technical study to ensure that today's applications software can continue to run smoothly throughout the changeover period, and to address questions of CERN-wide compatibility.

The speed of serial CAMAC transactions must be improved. Perhaps this will be achieved simply by installing future ultra-fast ND-100 processors; perhaps it will require completely new CAMAC driver hardware and software.

At present, the control system embodies very little classical process control, on-line modelling, etc. The price of computing power is now so low that it would seem inevitable at some stage to install a ND-500 superminicomputer as part of the on-line system for modelling and real-time control applications. This would raise challenging problems of integration into the network.

Nevertheless, present indications are that the NORD-10 and ND-100 instruction set, and the SINTRAN III operating system, will be available and industrially supported for many years to come. The decision taken in 1977 to base the control system's software on commercially available products will allow the system to live on, and to continue developing, into the 1990's.

ACKNOWLEDGEMENTS

The permanent staff members of the system team were as follows, with their most important responsibilities:
B. Bunaciu (Documentation)
R. Cailliau (P^+ system)
B. Carpenter (Section leader, SINTRAN III, network)
G. Cuisinier (NODAL, network)
W. Remmer (CAMAC, RMS68K)

Many other people have spent time in the team as Fellows, Associates, students, etc.:
D. Bates
M. Bertier
J. Field
R. Glitho
C. Greffe
J.P. Jeanneret (network)
I. Killner (P^+)
M. Krüger (ACC software)
Le Van Kiet
G. Maximov
J. McCullough
K. Osen (P^+)
D. Ross (P^+)
L. Stoate
L.K. Thomas (P^+ design)
P. van der Vossen
C. Vicent (network)

It is impossible to acknowledge individually all the contributions of our colleagues in the PS Controls group to the design and, sometimes, implementation of the system software, but at least the following people must be mentioned:
C. Cate-Bettels
C. Granieri
C. Hazlehurst
D. Heagerty
P. Horne
F. Perriollat
G. Quickfall
P. Skarek

Outside the group we have benefited from the knowledge and experience of the Data Handling Division on-line computing (formerly Experimental Physics) team (C. Eck, H. Overås); the Data Handling Division Software team (J. Blake); the Super Proton Synchrotron teams (J. Altaber, G. Shering); the former Intersecting Storage Rings team; the Deutsches Electronen-Synchrotron controls team; European Molecular Biology Laboratory (D. Iversen, R. Herzog); and our software suppliers (Norsk Data, Nittedata, Gex-Informatique).

R.H. Perrott and F. Perriollat have commented usefully on a draft of this report.

Finally, the constant encouragement and support of our group and project leader, B. Kuiper, has been valuable beyond words.

Note: Part of Section 2 appears in different form in the Proceedings of the NOCUS Meeting, Brighton, November 1981. Parts of Section 3 appear in different form in the Proceedings of the NOCUS Meeting, Monte Carlo, November 1982. Material from various CERN internal notes has been incorporated.

Trademarks

PDP, VAX, RSX and DEC are trademarks of Digital Equipment Corporation.
UNIX is a trademark of AT&T Bell Laboratories.
Ada is a trademark of the US Department of Defence.

REFERENCES

Note: References marked with an asterisk are CERN Internal Reports or working documents, of which copies are available on request from the authors at CERN. All current Norsk Data manuals are listed in Ref. [45].

[1] B. Kuiper, Improvement programme for the CPS controls, PS/CCI/Note 76-34 (1976).*
[2] M.C. Crowley-Milling, The design of the control system for the SPS, CERN 75-20 (1975).
[3] B. Kuiper, Improvement programme for the CPS controls (addendum), PS/CCI/Note 76-45 (1976).*
[4] PS CO Group (Reported by B. Carpenter), Computer topology for the PS controls system, PS/CO/Note 81-12 (1981).*
[5] G.P. Benincasa, J. Cupérus, A. Daneels, P. Heymans, J.P. Potier, Ch. Serre and P. Skarek, Design goals and application software layout for the CERN 28 GeV accelerator complex, presented at the 2nd IFAC/IFIP Symposium on Software for Computer Control, Prague, 1979, PS/CO/Note 79-1 (1979) and preprints SOCOCO '79 (General Computing Centre, Czechoslovak Academy of Sciences, Prague), vol. 1, p. XII-I.
[6] P.P. Heymans and B. Kuiper, Concurrent control of interacting accelerators with particle beams of varying format and kind, presented at the EPS Europhysics Conf. on Computers in Accelerator Design and Operation, Berlin, 1983.
[7] NORD-10 Reference Manual, ND-06.001.01 (Norsk Data, Oslo, 1974).
[8] NORD-10/S Reference Manual, ND-06.008.01 (Norsk Data, Oslo, 1977).
[9] ND-100 Reference Manual, ND-06.014.02 (Norsk Data, Oslo, 1979).
[10] ND-500 Reference Manual, ND-05.009.02 (Norsk Data, Oslo, 1980).
[11] SINTRAN III User's Guide, ND-60.050.09 (Norsk Data, Oslo, 1979).
[12] B. Carpenter, Steps in installing a new SINTRAN, PS/CO/Note 84-16 (1984).*
[13] J. Altaber, Basic description of the software operating system for the computer control of the SPS, CERN Lab II-CO/75-1 (1975).*
[14] J. Altaber, SYNTRON II Preliminary User's Guide, CERN Lab II-CO/Int/Comp Note/75-6 (1975)*.
[15] J. Altaber, The file manager for SYNTRON II, SPS-CO/Comp Note/76-15 (1976).*
[16] B. Carpenter, General introduction to SYNTRON II, PS/CCI/Note 77-5 (1977).*
[17] J. Altaber, V. Frammery, C. Gareyte and P. van der Stok, Principles of the operating system for the SPS real-time control network, CERN/SPS/ACC/79-13 (1979).*
[18] SINTRAN II User's Guide, ND-60.014.02 (Norsk Data, Oslo, 1973).
[19] Generating SINTRAN II, ND-60.043.01 (Norsk Data, Oslo, 1973).
[20] D. Bates and B. Carpenter, The choice of operating system for the new PS controls system, PS/CCI/Note 78-15 (1978).*
[21] T.A. Dolotta, R.C. Haight and J.R. Mashley, The programmer's Workbench, Bell Syst. Tech. J. **57**, 2177 (1978).
[22] RSX 11/M System Reference Documentation (Digital Equipment Corporation, Maynard, Mass., 1974).
[23] B. Carpenter, Software process manipulation package for SINTRAN III, PS/CO/Note 80-31 (1980) (revised 1983).*
[24] B. Carpenter, The choice of data links for the new PS controls system, PS/CCI/Note 78-13 (1978).*
[25] J. Altaber, Real-time network for the control of a very large machine, Proc. Computer Network Symposium, Trends and Applications 1976, Gaithersburg, Maryland, 1976, IEEE Publication 76 CH11 43-7C (1976) p. 173.
[26] B. Carpenter and R. Cailliau, Experience with remote procedure calls in a real-time control system, Software Pract. Exper. **14**, 901 (1984).
[27] L. Mérard, T. Pettersson and C. Serre, Concurrent execution of real-time displays, PS/CO/Note 84-02 (1984).*
[28] H. Zimmerman, OSI reference model—the ISO model of architecture for open systems interconnection, IEEE Trans. Comput. **28**, 425 (1980).
[29] B. Carpenter, Revised layout for data link package, etc., PS/CO/WP 83-9 (1983).*
[30] B. Carpenter, New recipes for generating data link package and RT-common, PS/CO/WP 83-94 (1983).*
[31] M.C. Crowley-Milling and G.C. Shering, The NODAL system for the SPS, CERN 78-07 (1978).
[32] R.D. Russell, PL-11: a programming language for the DEC PDP-11 computer, CERN 74-24 (1974).
[33] M. Kruger, R. Cailliau and B. Carpenter, DMAC - a debugging tool for NORD-10/100 computers, PS/CO/Note 80-35 (1980).*
[34] Coral-66 reference manual for NORD 10, Systems Designers Limited, Camberley, England (1977).
[35] D. Bates and R. Cailliau, NS-Pascal User's Guide, PS/CCI/Note 77-3 (1977).*
[36] R. Cailliau and M. Krüger, PS-Pascal User's Guide, PS/CO/Note 79-22 (1979).*
[37] R. Cailliau and B. Carpenter, P$^+$ User's Manual, PS/CO/Note 82-21 (1982) (issued without number in 1978).*
[38] M. Krüger, General information on microprocessor software, PS COOKBOOK (1979), Chapter 9.*
[39] L. Kleinrock, Queuing systems (Wiley, New York, 1975-6) (two volumes).

[40] P.J. Denning, The working set model for program behaviour, Commun. ACM **11**, 323 (1968).
[41] A. Daneels, Ch. Granieri, J. Kenaghan, E. Malandain and C.H. Sicard, Performance measurements of the CPS control system,PS/CO/Note 83-6 (1983).*
[42] J. Altaber and J.P. Jeanneret, Mesures du réseau de contrôle du SPS, CERN-SPS/ACC/78-1 (1978).*
[43] B. Carpenter, System software layout for the MC68000 Super ACC, PS/CO/Note 83-15 (1983).*
[44] O. Barbalat, Structured naming scheme for the machine components of the PS complex, PS/DL/Note 77-3/rev. 4 (1977).*
[45] Norsk Data Documentation Catalogue, ND-40.004.04 (Norsk Data, Oslo, 1983).

STACK AND PARAMETER CONVENTIONS

Note: This Appendix concentrates on general principles. Technical details (in the form of instructions for NORD-PL programmers) are in COOKBOOK 4.3, as are useful diagrams.

The NORD-10 and ND-100 architecture provides the B register and associated addressing modes which allow it to be used conveniently as a stack frame pointer. However, there is no stack pointer (thus no PUSH and POP instructions) and no generally accepted convention for stack manipulation and parameter passing. Norsk Data are now trying to adopt a uniform convention, but this is too recent a development to have had any influence on the PS system. Essentially, we use three different conventions (although they are rather similar):

i) The P^+ convention, used in P^+ code and compatible NORD-PL code.
ii) the NODAL convention, used in the implementation of NODAL and NODAL functions.
iii) The Interpreter-Compiler Compatible Interface (ICCI), a compromise between the above two which allows language mixing.

In *all three conventions,* the stack of a procedure during its execution may be viewed as in Fig. A1.1. The B register points to the stack frame. (In P^+ code, B is actually offset by 127 to extend the range.) The stack frame contains six saved register values, then a pointer to the 'global block' (see below), and then two words marked W7 and W8 (see below). This fixed part of the stack frame is followed by a list of addresses of the actual parameters (never values), followed in turn by local data.

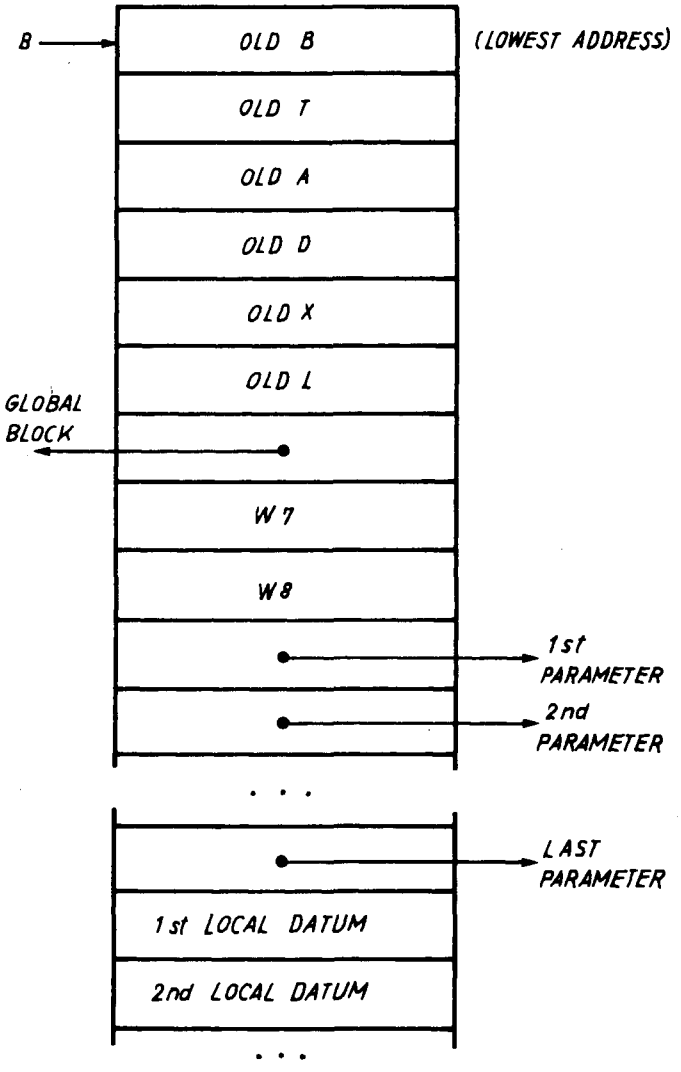

Fig. A1.1 Stack frame

The *global block* is a 256-word context block, in the address space of the process to which it belongs. There is one global block per process, so library routines may use pre-allocated words in the global block as working storage. It contains three sets of data

i) a working area for the P+ library,
ii) a working area for console device drivers,
iii) internal variables for the NODAL interpreter.

Some of NODAL's internal variables are of more general use (including one known as 'the NODAL type', which indicates whether the current process is an interactive NODAL, a server NODAL, a compiled SRT program, an HRT program, etc.).

The words *W7 and W8* (at offsets 7 and 8, respectively) have varying uses. In NODAL, they point to two routines STK and USTK in the kernel of NODAL. These pointers are used only by NORD-PL code which invokes the procedure entry and exit macros in their original (SPS) form. This is *not* the normal method at PS, but is used for historical reasons in some cases.

In the P+ convention, W7 is used as the 'static link'. This is (in a recursive block-structured language) a pointer to the stack frame of the procedure which contains the current procedure at compile time. Such a link is necessary to find the correct versions of variables declared in a recursive program. W8 is the 'maximum stack' pointer, i.e. it contains the address of the last word of stack known to be used by the current procedure.

In *all three conventions,* although the parameter addresses are found in the stack frame, they are not put there by the code which calls the procedure. Before issuing a procedure call, the caller must prepare the parameter address list in any convenient buffer (normally among the caller's local data). At the point of call, regardless of the convention used, the A register must point to this parameter address list (see Fig. A.1.2).

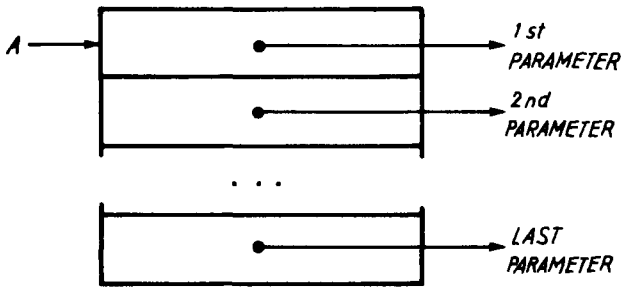

Fig. A1.2 Parameter list

(This convention is indeed derived from an old ND FORTRAN convention).

Whether the call uses the P+ convention, the NODAL convention, or the ICCI convention, the procedure entry sequence includes the creation of a new stack frame as above, implying that the parameter address list (but not of course the parameters) is copied.

i) NODAL convention

The call (preceded by a MEXIT monitor call to overlay the appropriate segment) is just a JPL instruction. The entry macro USENT (SPS version) calls a NODAL kernel routine FSTK, passing as parameters the number of parameters and the total stack space required. FSTK carries out a stack overflow check, propagates the new stack frame at the *lower* address side of the current stack frame, and copies the parameter address list. Thus, although the stack frame is viewed as growing towards high addresses, the NODAL stack grows towards low addresses.

ii) P+ convention

The call is mediated by a library routine 0CPPP (call-procedure-p-plus), which follows the static link if necessary before branching to the procedure entry code. This in turn uses a library routine 0ENPP (enter-p-plus), which carries out a stack overflow check, propagates the new stack frame at the *higher* address side of the current stack frame, copies the parameter address list, and performs other initializations.

NORD-PL programmers use the P+ convention via macros PCALL and PPENT (plus PPRET for the return).

iii) ICCI convention

At the instant of call, the current stack frame, the parameter list, etc., correspond exactly to the NODAL convention. Thus, an ICCI call may *originate* from NODAL or P+, and may call code *running* according to the NODAL or the P+ convention.

If called from NODAL, an ICCI procedure must normally be known to NODAL (via its header) as a 'type 7' or 'type 12' NODAL function. Type 7 is used for equipment modules and type 12 for general procedures. Special routines within NODAL (SYSVR and ASSFN) build the parameter list, overlay the appropriate segment, and execute a JPL instruction.

If called from P^+ (or compatible NORD-PL code via the macro ICCI), the call is mediated by a library routine 0CPIC (call-procedure-ICCI). This builds a pseudo-NODAL stack frame at the *high* end of the free space in the stack, and then calls a library routine XSTOS (X-register segment-to-segment) in the NODAL re-entrant segment. XSTOS overlays the appropriate segment and executes a JPL instruction.

A historical NODAL function (normally of type 12) may be called by either of the above methods and use the macro USENT (SPS version), and then continue to use NODAL internal conventions. Such a function can only be coded in low-level language.

A P^+ procedure declared as ICCI (or compatible NORD-PL code) may be called by either of the above methods. Its entry code is mediated by a library routine 0ENIC (enter-ICCI), which builds a P^+ stack frame at the *low* end of the free stack space, and then continues like 0ENPP.

The foregoing is an *incomplete* description and COOKBOOK 4.3 should be consulted for diagrams and more details. The *importance* of the ICCI convention lies in the following characteristics:

a) P^+ procedures (declared as ICCI) may be called as overlays by SRT programs (in P^+, NORD -PL, NODAL, and RT-Pascal).
b) NORD-PL procedures obeying the rules of the COOKBOOK 4.3 may be linked with P^+, and may be overlaid by SRT programs if they follow the ICCI rules.
c) In some cases, code following the traditional NODAL conventions may also be overlaid by SRT programs.
d) ICCI procedures may be called remotely.
e) ICCI procedures may call other ICCI procedures, even on another segment.

The *restrictions* in the ICCI convention include the following:

- all eight of those for remote procedure calls (Section 5.2),
- all parameters of ICCI-to-ICCI calls must be local variables (in the stack),
- string parameters must be initialized before the call, and must not be passed on again as parameters.

It is worth mentioning that the *NODAL headers,* jump lists, and (in some cases) parameter massaging code needed at the start of a segment of ICCI procedures are normally created by a utility program MAKE-HEADERS (documented in COOKBOOK 4.3). This program can create headers, etc., for normal P^+ or P^+ compatible ICCI procedures, EMs, composite variable modules, and in certain cases for FORTRAN subroutines.

For completeness, the following list shows the NODAL function types which can, *in principle,* be called as ICCI from P^+ (allowing for implicit parameters in most cases). Only types 7 and 12 are recommended.

7* equipment module
8 read/write real function
9 write-only real function
10 read-only real function
11 procedure with implicit flag
12 procedure with no implicit flag
32* read/write string function (generalization of old type 22)
33* write-only string function (generalization of old type 23)
34* read-only string function (generalization of old type 24)

The parameter types supported are:

2 evaluated integer (RO)
6 integer reference (RW)
1 evaluated real (RO)
5 real reference (RW)
8 integer array (RW)
7 real array (RW)
3* evaluated string (RO)
10* string reference (RW)
11* ICCI variant (RW)

The NODAL headers for all types except equipment module are as in the NODAL manual, with the appropriate NODAL type and parameter types inserted. The items marked with an asterisk were implemented or modified at PS. Parameter type 11 applies only to a special variant type for the 'global variable' package.

APPENDIX 2

ADDRESS LAYOUT

As outlined in Section 4.3, the 64 Kword virtual address space is split into four regions for SRT programs:

i) Re-entrant segment (NODAL kernel), approximately 16 Kwords,
ii) Main program segment (code plus data), approximately 16 Kwords,
iii) Overlay segment (ICCI procedures, at most 24 Kwords),
iv) RT common (resident library plus communication table, at least 8 Kwords).

This layout cannot be varied at present, since the ICCI convention (Appendix 1) relies on all SRT programs and ICCI segments being able to execute code in the NODAL kernel. An HRT program may be loaded either as part of an ICCI segment, or on a separate segment replacing regions (i) and (ii), and linked to data buffers in an ICCI segment. In exceptional cases, an HRT program can be loaded without reference to the above layout, but such a program cannot easily communicate with the control system as a whole.

Regions (i) and (ii) normally overlap slightly, for the following reason. In the case that the SRT program is an interactive NODAL, it is necessary that NODAL functions (normally ICCI procedures) can be temporarily installed for testing or other reasons. This is done by a special NODAL command USER-LOAD (x), where x is a SINTRAN segment number. USER-LOAD scans the given segment, extracts all valid and non-duplicated NODAL headers from a linked list starting at the beginning of the segment, and makes a working copy of these headers. This copy is in fact in the form of an extension of NODAL's initial list of headers (Section 7.2). Since a SINTRAN re-entrant segment may of course not be modified, this extension of NODAL's list is placed in the overlap area. The command USER-UNLOAD, or an exit from NODAL, cancels the effect of USER-LOAD.

It is apparent that all main program segments (in a given computer) are constrained to start at the same address (except that for NODAL, this address is preceded by the overlap with the re-entrant segment). This address is named MLAD and is never less than 40000 octal.

In order that any SRT program may call any ICCI overlay, it is also apparent that all ICCI segments should start at a fixed address (in a given computer) and that no SRT program may exceed this address. In special cases, of course, a longer main program could call shorter overlays: but this is not recommended.

The 'magic' address for ICCI overlay segments is named OLAD and is never less than 100001 octal. (The offset of 1 is due to a corresponding offset of -1 in the relocatable load address of the first NODAL header in each ICCI segment.)

The values of MLAD and OLAD may be changed by the system supervisor if a given computer encounters difficulties with the existing layout. Typically this arises because more NODAL headers are added to NODAL's initial list: the re-entrant segment is then too small and MLAD must be increased. OLAD must then be increased by the same or a larger amount. Of course, this increases the high boundary of all ICCI segments, but they must not in turn overlap with RT common. Thus, the system supervisor remains constrained by the 64 Kword address space.

MLAD and OLAD may only correspond to 1 Kword page boundaries, so must be modified in multiples of 2000 octal. They are defined in a command file named MOD-CHOOSE, which allows the system supervisor to choose various options within the software layout. MLAD and OLAD may not be changed without also changing several other symbols in MOD-CHOOSE which interact with them, as indicated by comments in the file. Finally, after changing MOD-CHOOSE, it is necessary to rebuild all segments without exception. This is best achieved by a HENT cold start (with the consequent loss of all data tables). Another address defined in MOD-CHOOSE is DEFAD, allowing control over the NODAL 'defined function area' size.

Note that MLAD and OLAD are used in RT Loader command files, so that these are independent of the address layout. This relies on a locally developed patch to the RT loader (which in its standard form allows only octal address definitions).

The following summarizes the address symbols in MOD-CHOOSE:

MLAD	Main program load address Cannot be less than 40000
OLAD	ICCI overlay load address + 1 Cannot be less than MLAD + 40001
WORK = MLAD + 2000	Start of NODAL working area
STSRT	End of NODAL working area
RTDEX = STSRT + 200	NODAL global block
MSBUF = RTDEX + 200	NODAL disc buffer
DEFAD = MSBUF + 400	Start of NODAL defined function area
DEFIN = OLAD − 2	End of NODAL defined function area

OVLPZ	Size of re-entrant segment overlap (typical 6000)
OVLPS = MLAD − OVLPZ	Start of overlap
NASZ = DEFIN − WORK + 1	Total size of NODAL area
'160 000'	Start of RT common = limit of overlays + 1

APPENDIX 3

SPECIAL FACILITIES FOR EQUIPMENT MODULES, etc.

Equipment modules are the PS development of the SPS 'data modules' [2]. They are re-entrant subroutines, existing in a single non-volatile copy per FEC, comprising the necessary code and a data table containing both quasi-permanent values such as CAMAC addresses and variable values such as set points. One EM controls one type of equipment (e.g. POW controls power supplies), but of course there are many equipments of each type, and each equipment has multiple properties (e.g. set point, reference value, etc.). Some properties have multiple values for the same equipment owing to pulse-to-pulse modulation.

To allow the construction of general-purpose applications programs, independent of any particular type of equipment or any particular computer, a standard calling sequence applies to all EMs. The same calling sequence is also used for CVMs, which are an abstract form of EM which may call several EMs internally. The system software makes no distinction between EMs and CVMs.

Equipment modules are in fact ICCI procedures (see Appendix 1) with two entry points: a single equipment call and an array call. The parameters of the single equipment call are as follows:

1) The VALUE, a read-write parameter.
2) The FLAG, a read-only integer
   + 1 means write into VALUE a datum read from the EM.
   − 1 means write into the EM a datum read from VALUE.
3) The EQUIPMENT NUMBER, a read-only integer selecting a particular piece of external equipment.
4) The PROPERTY, a read-only integer selecting a particular property of the equipment.
5) The PULSE, a read-only integer selecting a particular accelerator pulse type.
6) The COMPLETION CODE, a write-only integer.

Equipment modules are specially signalled to NODAL by a Type 7 header; they may be treated as read-write functions so that parameters 1 and 2 are implicit. Defaults are available for parameters 4, 5, and 6, so the simplest call available is of the form

```
SET X = POW (3)
```

although the defaults are little used in practice.

NODAL also knows the allowed properties by name, so the user can avoid numeric values. P^+ calls must provide all the parameters, but again pre-declared constants are available for the property codes.

All standard EMs have an 'array call' entry point name which is the EM name preceded by the letter A (e.g. APOW). The parameters are identical to those of the single equipment call, except that the VALUE, the EQUIPMENT NUMBER, and the COMPLETION CODE are replaced by corresponding arrays (whose first item indicates the total number of items). Array calls are normal type 12 NODAL functions and normal ICCI procedures.

Truly general-purpose programs must be data-driven. For this reason, there is a special ICCI routine EQM which has the same six parameters as any EM, preceded by an 'equipment module number' (not the same as the 'equipment number'!). Thus EQM may be used to call *any* EM or CVM by number rather than by name. Since a remote call may be executed in a computer selected by number too, completely data-driven programs may access any property of any equipment on any FEC.

EQM thus has the job of dispatching EM calls within one FEC. This is achieved as follows. During initialization, NODAL builds a special table with one entry for each Type 7 header. The table is indexed by EM number and consists of pointers to the relevant headers. By consulting this table, EQM can quickly locate the requested EM if it exists. (Duplicated EM numbers will cause problems, but they are rejected with an error message during initialization). For convenience, EQM is an ICCI routine, but embedded in the NODAL kernel.

Data-driven array calls may be carried out by the related routine AEQM. Since 'array call' EMs are ordinary Type 12 NODAL functions, AEQM *assumes* that a Type 12 header immediately following a type 7 header is the corresponding array call. The EM programmer can exceptionally indicate that this is not the case by using type 100007 (octal) for the non-standard EM.

In the console computers, a more abstract form of EM access has been implemented. Three NODAL functions PUT, GET, and LET are available: they act on 'structured variable names' [44] as SET and TYPE act on normal variables. They proceed as follows:

i) look up name via dictionary in TREES computer (using ICCI routines and data base implemented by J. Cupérus). This gives FEC number, EM number, and equipment number.
ii) access equipment by a remote call to EQM.

Thus the console user can type, say

PUT PX.DHZ05 = 1.03

to set a magnet current.

The NODAL header for a PS EM or CVM is shown in Fig. A3.1 (although MAKE-HEADERS, Appendix 1, creates a more sophisticated version).

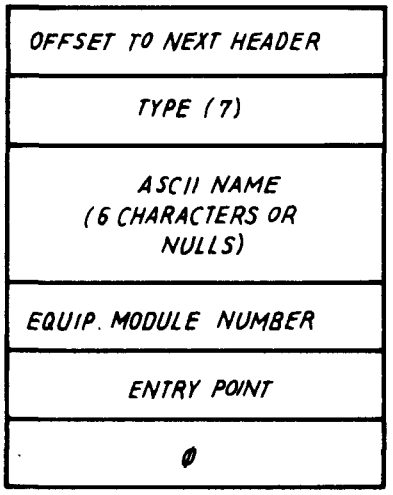

Fig. A3.1 EM header

APPENDIX 4

GENERAL INFORMATION FOR SYSTEM SUPERVISORS

For each operational NORD-10 or ND-100, there is a system supervisor (and possibly a deputy). The current list of system supervisors is issued by the Exploitation Section. The range of responsibilities foreseen for the system supervisor is as follows:

i) (ALWAYS TOP PRIORITY) Ensuring that the disc backups are done properly.
ii) Acting as the first filter for non-trivial problems.
iii) Maintaining certain system files.
iv) Allocating system resources.
v) Installing or helping to install new software.

1. BACKUPS

The backup scheme is defined by the Exploitation Section. User documentation of the backup utility employed on the PRDEV is in the source file (SYSTEM-SOURCES) NPL-HAWK-BACKUP.

The backup discs (one copy each of the fixed and removable working discs) are stored in the fireproof cabinet in the PRDEV computer room.

The backup tapes (older copies each of the fixed and removable discs) are stored in a locked metal cupboard in another building.

It cannot be overemphasized that meticulous adherence to the backup procedures is essential. No shortcuts whatsoever are safe.

2. FIRST FILTER FOR NON-TRIVIAL PROBLEMS

The system supervisor has to decide:

i) Is this a problem I can and should solve?
ii) If not, who can and should solve it?
iii) What is the true priority of this problem?

Only with experience can a system supervisor easily decide these questions. For system software problems, one of the lists cited in Appendix 5 may be used to identify the relevant specialist.

3. MAINTAINING SYSTEM FILES

COOKBOOK 4.2 explains the purpose of most of the basic system files maintained by the system supervisor. To summarize, these are:

(SYSTEM)BAT-LRESTART-SEQ	Initialization batch job
(SYSTEM)MOD-LRT-LOAD	RT loader command file
(SYSTEM)MOD-LOCAL-HENT	Cold start command file
(SYSTEM)MOD-CHOOSE	Chooses various options and defines symbols

Another file is:

(SYSTEM)SYS-GO:NOD

This is a NODAL file which is automatically executed by RT NODAL at completion of BAT-LRESTART-SEQ.

This file is initially dummy, and should be expanded by the system supervisor to carry out all real-time environment initialization (CAMAC initialization, hooking or scheduling programs, etc.). The data link should be opened by the command OPENDL.

Some extra notes on RT loader files are needed:

i) It is *essential* that the RT loader command files are always up to date. If not, after a cold start (HENT), it is impossible to restore all programs to their previous state.
ii) MOD-LRT-LOAD should be limited to general-purpose segments for the specific computer. Applications programs, EMs, etc. should be loaded by a family of files called in a tree structure from MOD-LRT-LOAD, and following the normal file naming standard (COOKBOOK 2.1).
iii) Users other than the system supervisor must *not* be given the SYSTEM password. Applications programmers need the RT password in order to be able to run the command file loading their segment. The command files must be developed by the Applications Programmers in cooperation with the System Supervisor.

4. ALLOCATING SYSTEM RESOURCES

The system resources which must be allocated manually are:

- disc space,
- virtual memory segments,
- semaphores.

In general, the removable disc (directory name MAIN) is completely used by the system.

The 'fixed' disc (but it does go round), with directory name FIXED, is allocated by the system supervisor to *project* users. Personal users (e.g. JEAN-DUPONT) should never be allocated space on disc. The fixed disc will be a default directory.

Virtual memory segments and semaphores are allocated from the ranges marked AP in the lists cited in Appendix 5.

When programs refer to segment numbers or semaphores, they should always do so with a symbolic external reference, e.g. *) 9EXT SEMFK. The system supervisor then defines the symbol's value in the appropriate file (often MOD-CHOOSE) with the RT loader DEFINE-SYMBOL command. This will make it possible to reallocate semaphores and segments without recompiling programs.

5. INSTALLING OR HELPING TO INSTALL NEW SOFTWARE

5.1 System section or console section software

This software will be tested on the BACKUP computer, and/or the PROBAC, and/or one console. Installation on the operational computers will then be by agreement between the system programmer and the individual system supervisors. A 'version book' is kept in the computer room to centralize information on installation of new versions.

5.2 Applications software — ICCI functions, equipment modules

See COOKBOOK 4.3 for general information. The user can test his or her functions as described there (logged in as RT). For permanent installation, choose a segment number (see above), modify the appropriate command file, and update MOD-CHOOSE.

5.3 Applications software — NODAL programs

These are standard :NOD files, stored under any user on the FIXED directory. Only RT-NODAL is available in the operational computers. To call RT-NODAL, log in as any user (e.g. GUEST) and then type RT-NODAL.

To be able to write :NOD files from RT-NODAL, the user concerned must create RT as a friend with RWA access. To be able to create :NOD files from RT-NODAL, RT must be a friend with RWACD access. It is a good idea for the system supervisor to do the appropriate CREATE-FRIEND and SET-FRIEND-ACCESS commands whenever creating a new user.

NODAL files may be STARTed or SCHEDuled as in Carpenter [23], e.g.

```
>START("(PLS-APPLIC)MYPROG", COCO.)
```

The status of running NODAL files may be examined with the NODAL command LISP.

5.4 Applications software — compiled programs

This is complicated! Compiled programs are *NOT* given RT descriptions in the standard way, i.e. *)9RT in NORD-PL. RT descriptions are allocated dynamically by START, etc. Therefore, compiled programs must follow the prescriptions in COOKBOOK. For further information, see Carpenter [23].

APPENDIX 5

IMPORTANT LISTS

Several lists of importance to system supervisors are maintained on the program development computer. Since printed copies of these lists will always be out of date, the versions on disc should be inspected when information is required. All of the files are pure ASCII text and may be inspected with the standard editors.

i) Allocation of virtual memory *segment numbers*.

File (SOFTWARE-LAYOUT)TXT-SEGMENTS : SYMB

ii) Allocation of *semaphore numbers*.

File (SOFTWARE-LAYOUT) TXT-SEMAPHORES : SYMB

iii) Allocation of RT program *priorities*.

File (SOFTWARE-LAYOUT)TXT-PRIORITIES : SYMB

iv) Allocation of space in RT common *communication table*.

File (SOFTWARE-LAYOUT) TXT-COMTB-LAYOUT: SYMB

v) *System files* installed in various computers.

File (SOFTWARE-LAYOUT) TXT-SYS-FILES : SYMB

vi) *System software error codes*
(NORD-PL source file. Also see COOKBOOK 4.10)

File (SYSTEM-SOURCES)NPL-SS-ERRORS : SYMB

vii) *Specialists on various topics*

File (SOFTWARE-LAYOUT) TXT-SPECIALISTS : SYMB