2.0 INTRODUCTION - RUNTIME SYSTEM

Formal description of runtime system routines are given in a SIMULA 67 like language.

The difference of the language used for description, as compared to ordinary SIMULA 67, is as follows:

1. label variables and the denotes operation for labels have been introduced with an obvious meaning. Labels as in SIMULA will be considered to be label constants.

2. The basic symbol exit has been introduced, meaning the place where a procedure was called. exit is a designational expression.

3. Arithmetic and comparison operations on ref expressions have been included with an obvious meaning:

   1. The expressions "ref + integer", "ref - integer", "integer + ref", "integer - ref", treat the reference as an integer, the address designated by the ref quantity.
      The result is an integer.

   2. The assignment

      ref variable :- integer expression;

      causes the value of the integer expression to be stored in the ref variable as an address.

   3. Value relations between ref expressions are defined by adding the integer 0 (zero) to both operands, applying rule 1 above, and use the conventional rules for relations between arithmetic expressions.

4. The notation refto X is used to obtain a reference to a quantity X. It is a reference expression.

5. The notation val X is used to refer to the value of a quantity referred to by X.

6. The notation ref(program) has been used for reference to memory cells (program points).

The term "textual link" ("static link") means a pointer from the driver of a block instance to the driver of the block instance textually enclosing the former. An example is: the textual link of a procedure P is to the driver of the block instance B where the procedure is declared.

The term "dynamic link" means a pointer from the driver of a block instance to the driver of the block instance dynamically enclosing the former. An example is: the dynamic link of a procedure P is to the driver of the block instance B where the procedure is called.

The abbreviation B.I. will be used for "block instance".

2.1 Driver technique

Since a terminated B.I. seldom requires the system information that was necessary prior to the termination, it is natural to try to find a method so that this information may be discarded when the B.I. terminates.

One such method is the driver technique. The system information in the B.I. itself is reduced to a minimum, the rest is stored outside the object in a "notice".

There are two kinds of notices: "drivers" containing system information relevant to any kind of B.I., and "eventnotices" containing sequencing information for an active or suspended process.

All notices are assumed to be of the same size.

2.2 Storage organization

When the driver technique is used, data core storage (assumed to be continuous) is dynamically divided into two distinct parts called POOL 1 and POOL 2. Notices are allocated from POOL 2, all other storage from POOL 1.

Storage is allocated from POOL 1 simply by moving a pointer towards POOL 2. POOL 2 uses a similar technique only it has in addition an available storage list of the last in - first out type. On this list, notices are kept chained when not in use.

The head of this list is a ref (notice) nothead declared in the runtime system.

In the following it is assumed that POOL 1 is from the "lower end" of available storage while POOL 2 is from the "upper end".

Four ref variables declared in the runtime system serve as boundary pointers for the two pools of storage: POOL1FIRST, POOL1LAST, POOL2BOTTOM, POOL2TOP.

Two of these are fixed when execution of the SIMULA program is initiated: POOL1FIRST points to the first word of available storage, POOL2BOTTOM points to the last word of available storage where a notice may start.

The two other ref variables, POOL1LAST and POOL2TOP determine the boundaries of POOL 1 and POOL 2 respectively.

POOL2TOP points to the first word of the uppermost notice (whether available or not). POOL1LAST points to the first word following POOL 1.

A storage collapse condition exists whenever a storage assignment would cause POOL2TOP to become less than or equal to POOL1LAST + 1. (At least storage for one integer must be available for the store collapse between POOL 1 and POOL 2.)

Fig. 1 indicates the usual situation during program execution (proportions of POOL 1 and POOL 2 are distorted).

During program execution, the ref (driver) variable CD ("current driver") in the runtime system will always point to the dynamically innermost driver in the current operating object.

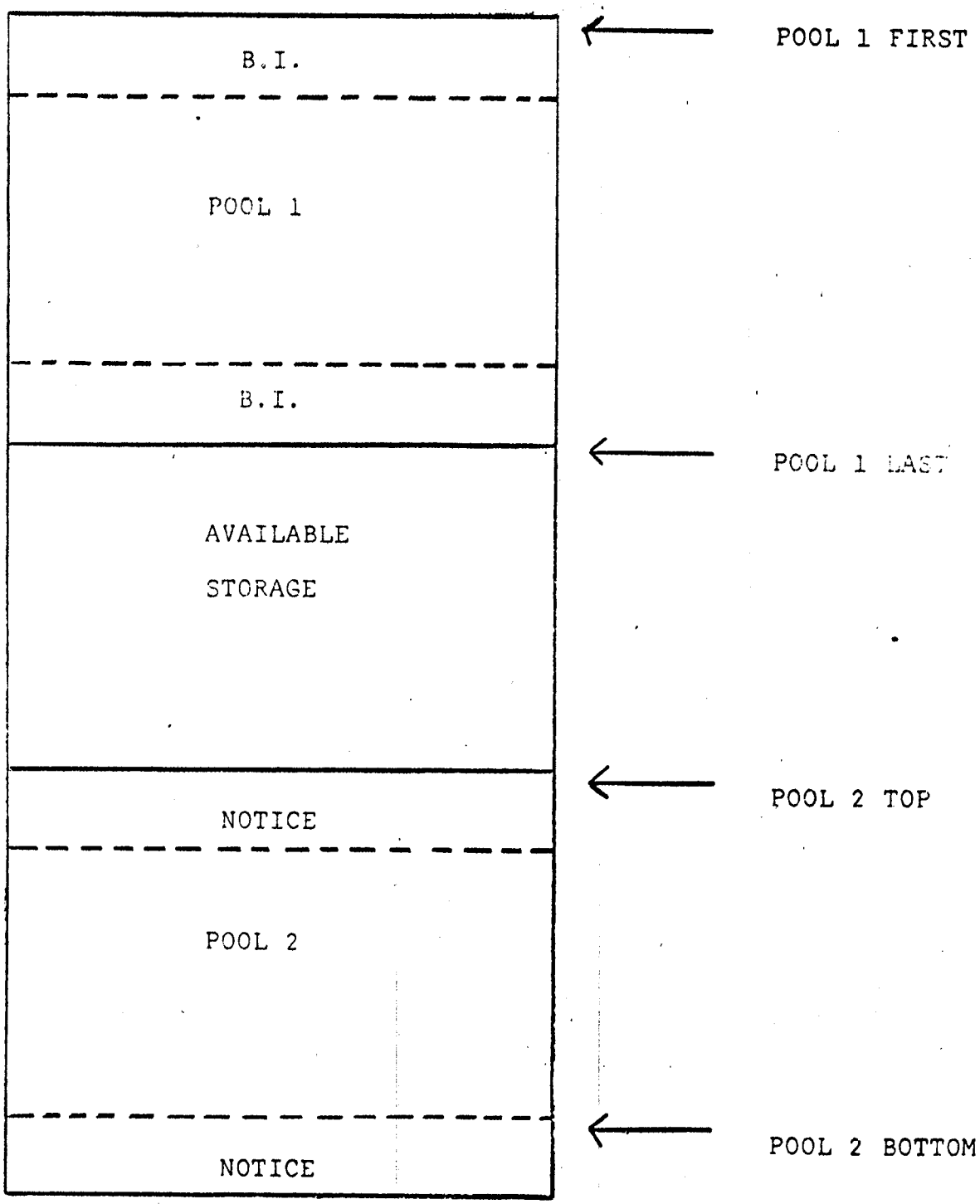

Fig. 1. Core Storage Layout during program execution.

2.3 The classes "object" and "prototype"

A logical unit of information in POOL 1 will be called a "block instance". Examples of data blocks are: instance of a subblock, procedure instance or class instance (without their local arrays), an accumulator stack or an array.

A block instance is an instance of a subclass of the (fictive) class "object". For each family of data blocks (different instances of the same block constitute one family) there exists one compiler generated "prototype" describing that family. An object contains a reference to the corresponding prototype, called the "prototype pointer" (abbreviated PP).

Arrays and accumulator stacks are considered two special families. They have no prototype. The relevant information is contained within the objects.

Special values of the prototype pointer PP are used to indicate arrays and accumulator stacks.

The use of each variable declared in class prototype is described below:

Type	Name	Usage
integer	lg	Total length (including that of data for possible prefixes) of a B.I. of this family. This includes necessary space for description of each array declared within the object.
integer	nvirt	Number of virtuals in B.I.'s of this family.

Type	Name	Usage
integer	nrp	Number of parameters for B.I.'s of this family.
integer	nrl	Number of local pointers in this family.
integer	level	Static block level of the declaration of this family.
integer	vtype	For procedure, type of result.
ref(program)	statements	Address of first statement in body.
ref(program)	inretur	Return address from statement inner; in this class. none if not prototype for a class.
ref(program)	declare	In-line coding to perform declarations local to the object. For a class, or a prefixed block, this in-line coding returns to the run-time system. In all other cases, the in-line coding will continue with the first statement.
integer array	relad	Relative address in data object for parameters and local pointers.
Boolean array	valu	true if and only if value parameter.
ref array	progaddr	Either: program address (for virtual labels and switches) or pointer to procedure prototype (for virtual procedures).

Type	Name	Usage
Boolean	pb	true if and only if this family is a prefixed block.
Boolean	ob	true if and only if this family is a class or a prefixed block.
Boolean	local classes	true if and only if this family has local class declarations.
ref (prototype) array	prefix	Pointers to prototypes for the prefix hierarchy of this family. Prefix (0) is the outermost prefix.
ref (program)	endblk	Address of statement after end of a prefixed block.
integer	plev	Prefix level (i.e. the number of prefixes in the prefix chain of this block).

As mentioned earlier there are two kinds of notices: drivers and eventnotices.

The ref variable "notc" is only used during store collapse to indicate where a notice has been moved. In this case the actual contents of the notice is not used. Thus "notc" may, in actual implementation, occupy the same part of a notice as for instance "obj".

"evp" and "acs" are mutually exclusive, and may thus use the same space in the driver.

In an actual implementation, the pointers to POOL 2 may be made relative by assuming a maximum size of POOL 2. This also applies to pointers from POOL 1 to POOL 2, for instance "MDP".

The use of each variable declared in class notice and its subclasses driver and eventnotice are described below.

Type	Name	To	Usage
ref (object)	obj	POOL1	Points to the B.I. belonging to this notice. The value none is illegal for "obj" except when the driver has not been completed. The store collapse must be able to handle the case "obj == none" even for master drivers.
Boolean	referenced	n.a.	false always except when in store collapse where it is set true if notice is referenceable.
ref (notice)	notc	POOL2	Used solely during store collapse. When a notice has been moved, "notc" in the old notice will indicate where the new notice has been put. Used for update of pointers to POOL 2.
real	time	n.a.	Evtime for the process referenced by obj of this eventnotice.
ref (eventnotice)	BL,LL,RL	POOL 2	See the section on organization of the sequencing set.
ref (program pex		program	Program address description: for procedures, thunks and attached objects: return point (exit). For detached objects and prefixed blocks: program point where object or prefixed block is to resume operation.

Type	Name	To	Usage
ref (driver)	cdrp	POOL 2	Pointer to the driver of the B.I. statically outside the connected B.I. (May be dropped if compiler remembers the block level of the class declaration).
ref (object)	acs	POOL 1	Pointer to accumulator stack which was saved when this driver was created. The value none indicates no accumulator stack.
ref (driver	drp	POOL 2	Pointer to the driver of the B.I. statically outside the B.I. belonging to this driver.
ref (driver)	drex	POOL 2	Pointer to the driver of the B.I. dynamically outside the B.I. belonging to this driver, except for prefixed blocks and detached objects when it is a pointer to the driver of the dynamically innermost B.I.
ref (driver)	drch	POOL 2	Used in store collapse to chain drivers for later processing.
Boolean	con	n.a.	true if and only if the driver is a connector or a thunk driver acting for a connector.
Boolean	rp	n.a.	true if and only if this is the driver of a detached object or a prefixed block.

Boolean	pb	n.a.	true if and only if this is the driver of a prefixed block.
Boolean	dot	n.a.	true if and only if this is the driver of a procedure called by remote referencing (dot-notation).
Boolean	md	n.a.	true if and only if this driver is a master driver.
Boolean	ob		true if and only if this is the driver of an attached or a detached object or a prefixed block.
ref (eventnotice)	evp	POOL 2	Pointer to the eventnotice of the process belonging to this driver. The value is none if "obj" does not point to a suspended or active process.
integer	level	n.a.	Apparent block level of this driver.

Below is a summary of the contents of drivers for different block states. Note that the contents of "obj" and "drp" are not shown since their contents are the same for all drivers.

"drch" and "notc" are also omitted since they are only used during store collapse.

For the Boolean variables, f is used for false and t for true.

	pex	drex	evp	acs	con	pb	rp	md	ob
Subblock	none	equal to drp	none	none	f	f	f	t	f
Procedure or attached object	exit	pointer to driver of dyn.outer block	none	pointer to acc. stk.(3)	f	f	f	t	(7)
Thunk (5)	exit	pointer to driver of block where thunk is called.	none	pointer to acc. stk.(3)	(5)	f	f	f	f
Prefixed block	react. point. (1)	pointer to innermost block (4)	none	none	f	t	t	t	t
Detached class body	react. point (1)	pointer to innermost block (4)	none (2)	none	f	f	t	t	t
Connector (6)	none	pointer to driver of the dyn. outer blk.	none	none	t	f	f	f	f

(1) Conceptually none if operating B.I. (Cfr. Common Base section 8).

(2) Pointer to eventnotice if active or suspended process.

(3) none is a possible value meaning "no accumulator stack".

(4) Disregarded for innermost operating B.I.

(5) "con" and "cdrp" as driver of calling B.I.

(6) "cdrp" is equal to "drp" of the connected B.I.

(7) false for procedure, true for attached B.I.

2.4 Display

For reasons of efficiency, the static chain may be duplicated in a vector of fixed locations (registers or core storage) called a display.

The display may be explicit as a vector or implicit by the static links (drp) and B.I. pointers (obj) in the drivers.

We may, when using driver technique, talk of two different displays:

1. DISPLAY pointing to B.I.'s

2. DDISPLAY pointing to drivers

Obviously it is possible to establish either of these from the information in CD.

If DISPLAY is kept in index registers, the number of possible static levels in a program may be limited. This number should however never be less than 8.

DDISPLAY may be partially present or completely absent in some implementations.

DISPLAY and DDISPLAY must be updated whenever the value of CD is changed, except in the case of exit from a subblock, connection block or a prefixed block.

2.5 Actual prototype implementation

Information in the prototypes is heavily used at runtime. This is most obvious in the store collapse, where the prototype must be scanned once for every referenceable B.I. Even when a B.I. it is not referenceable, the length (lg) of the data universe must be found in the prototype.

Prototype information is also used when a block, prefixed block, procedure or class B.I. is created.

Since, in other cases, the access to prototype information usually would be indirect (through the prototype pointer, PP, in the object), part of the prototype information may be duplicated in the drivers.

The following should be noted about the information kept in the prototype in the formal description:

lg — Required item. Without it, the length of a B.I. could not be found at runtime. To avoid accumulating length of all prefixed at runtime, this length is the total length including data for the prefixes.

ob,pb — Required.

nvirt — Required in one form or another. Virtual descriptors must be located at fixed positions within the prototype relative to PP. Thus, nvirt is required to skip these descriptors when scanning the prototype during store collapse.

nrp — Required for procedures to compare the actual number of parameters to the number given in the procedure declaration. This information is used when a formal or virtual procedure is called.

nrl — Redundant if a special end prototype signal is introduced in the prototype.

level — Required.

statements — Required for classes and prefixed blocks. May be omitted for subblocks and procedures if the compiler generates a jump from end of declarations to first statement.

inretur — Required if the compiler does not generate the end of a subclass as an in-line return following inner; of the prefix.

endblk — Required for prefixed blocks.

declare — Required for procedures, classes and prefixed blocks. May be omitted for subblocks if the declarations immediately follow call on BB.

prefix — Required for classes and prefixed blocks.
Prefix is used to get fast transition from declaration code in the prefix to declaration code in the main part, as well as fast transition from the declaration code in the main part to the statements in the outermost prefix.

relad
kind
type
valu
} Required as part of the individual descriptors.

virtloc
progaddr
} Required as part of virtual descriptors.

local classes — Required for driver deallocation purposes.

type Required for procedures. For a ref procedure it could be a pointer to the prototype of the class qualifying the result.

plev Required.

2.6 Wholesale deallocation

The present section describes a possible technique for wholesale deallocation in SIMULA 67 Common Base program. The formal description of the runtime system does not include a description of the allocation scheme and the store collapse required for this technique.

The natural dividing lines between parts of a SIMULA 67 program are the prefixed blocks. If a wholesale deallocation technique should be used, the fact that no non-local references may refer to anything local, is of great importance. This property is inherent in prefixed blocks, sub-blocks and procedures. In the present approach, only prefixed blocks and sub-blocks will be considered.

The approach described makes wholesale deallocation of data objects possible. Drivers are assumed to be deallocated separately.

POOL 1 is assumed to contain a list of descriptions of the prefixed blocks and sub-blocks in the system at each instant. This is a dynamic description where a declared block may be represented more than once due to its use within an object or due to recursive use of procedure. Since the size of this list varies dynamically, it must be possible to expand and contract this list.

The rest of POOL 1 is divided in as many parts as there are prefixed blocks and sub-blocks in the system at any time.

The compacting garbage collector is called whenever an area overflows into that of another block. Only the area that overflowed is compacted. Possibly, some part of the available area for the block preceding this one in core is stolen.

If area compacting does not yield a sufficiently good result, the entire store is compacted and new areas assigned.

Upon generation of a class B.I., its storage, including that for local arrays, is taken from the area assigned to the block where the declaration of the class is found. The local storage for a procedure may go into that of the block surrounding the call.

If a denotes operation is available for arrays (this is not the case within the Common Base), arrays could be put into a separate area. The array area would be compacted only when the entire store was compacted.

When a prefixed block or sub-block is left, the entire area assigned to that block is made available simply by removing its descriptor from the descriptor table. Drivers may be deallocated by the sequential scan of the area.