ARPA

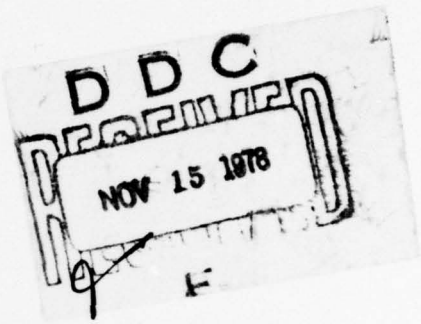
Command Systems Cybernetics

LEVEL II

12

AD A061207

DDC FILE COPY



This document has been approved for public release and sale; its distribution is unlimited.

78 10 19 069

AD A061207

(12) 43p

(12)

Contract Period Covered by Report: (11) 1 June 1978 – 31 Aug 78

(9) Quarterly rept. 1 Jun - 31 Aug 78,

DDC
RECEIVED
NOV 15 1978
F

Quarterly Research and Development Technical Report
(6) Spatial Data Management System

Computer Corporation of America

Report Authors: (10) Christopher F. Herot, Jim Schmolze, Richard Carling, Mark Friedell, Jerry Farrell

~~Research Division~~
Computer Corporation of America

617-491-3670

The views and conclusions in this document are those of the authors and should not be interpreted as necessarily representing the official policies, express or implied, of the Advanced Research Projects Agency, or the United States Government.

Sponsor: Defense Advanced Research Projects Agency
Office of Cybernetics Technology

ARPA Order Number: 3487

ARPA Contract Number: (15) MDA903-78-C-0122
ARPA Order-3487

Contract Period: ~~15 February 1978~~ – 30 November 1979

DDC FILE COPY

This document has been approved for public release and sale; its distribution is unlimited.

387285

Table of Contents

1. INTRODUCTION 1

2. USER INTERFACE 2
2.1 Input/Output Devices 5
2.2 Command Structure 6
2.2.1 Modes of operation 7
2.2.2 Static menu 9
2.2.3 SQUEL Monitor 10
2.2.3.1 SQUEL transactions 11
2.2.3.2 Monitor commands 11
2.2.4 Editing ICDL - Icon Class Description Language 12
2.2.4.1 Menu commands 12
2.2.4.2 Using the ICDL editor 13
2.2.4.3 Testing ICDL 14
2.3 System Specification - Interactive Input 14

3. Text in SDMS 17
3.1 Static Text 17
3.2 Dynamic Text 18
3.3 Encapsulated Text 19
3.4 Generation of Dynamic Text 20

4. STRUCTURE OF SDMS 23
4.1 The SDMS Process Structure 23
4.1.1 INGRES interface 26
4.2 Concurrent process cooperation 27
4.2.1 Design 27
4.2.2 Implementation within UNIX 30
4.2.3 SDMS process interrupts 34
4.3 Log 35
4.4 SDMS Protection Mechanisms 37
4.4.1 User protection 37
4.4.2 Symbolic database protection 38
4.4.3 I-Space protection 38
4.4.4 Database Administrator 40

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
SPECIAL
A

1. INTRODUCTION

This report describes the third quarter of work on the design and implementation of a prototype Spatial Data Management System (SDMS). Spatial Data Management is a technique for organizing and retrieving information which enlists the user's sense of spatiality through the use of high bandwidth, color, interactive computer graphics.

The quarter was occupied by the detailed design phase of the SDMS, the progress of which is summarized herein. A full description of the detail design will be presented in the forthcoming design document [HEROT et al]. This report describes several key elements of that design.

Chapter 2 of this report gives an overview of the SDMS from the perspective of the users of the system, including the symbolic and graphical languages through which the users and administrators control the system.

Chapter 3 describes the techniques for dealing with text, which plays a crucial role, both as a data type and an element of the graphical data space.

Finally, Chapter 4 presents the framework in which SDMS will be implemented, with special attention given to the means by which the various UNIX processes are coordinated.

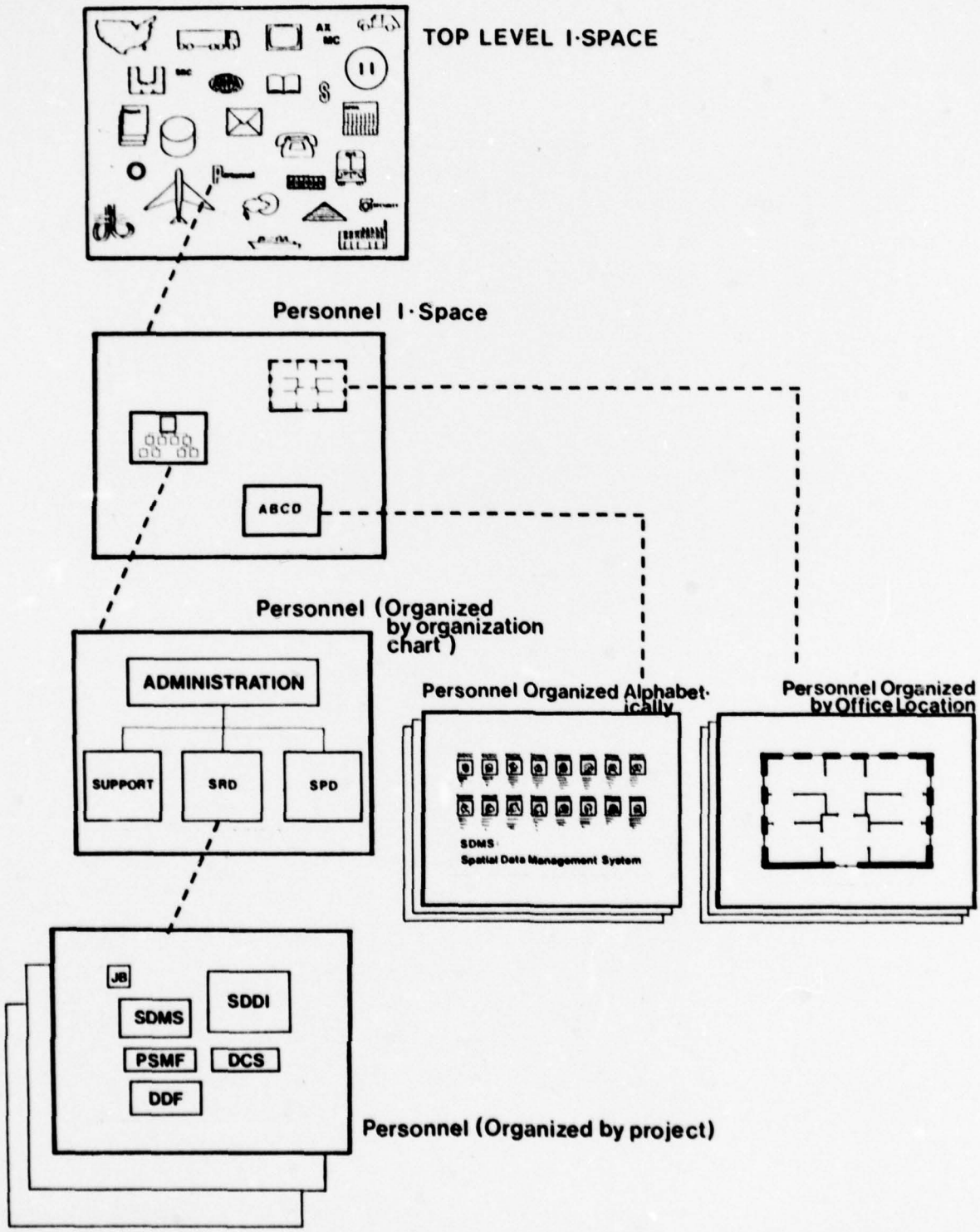
78 10 19 060

2. USER INTERFACE

The Spatial Data Management System provides each user with a graphical data space (GDS) consisting of nested surfaces of information, referred to as Information Spaces (I-Spaces). Each I-Space contains pictograms or icons which indicate to the user the location of particular items of information (see Figure 2.1). An I-Space is stored in the computer as one or more image planes (i-planes) which are the actual bit arrays used to generate the displayed image. An I-Space may be composed of more than one image plane in order to allow it to be viewed at several levels of detail. An I-Space is thus a two-dimensional world over which the user can "fly", changing his altitude in order to control how much information he sees at one time.

A user moves from one I-Space to another through ports, indicated by dashed lines in Figure 2.1. Ports can also be used to enter perusal spaces which provide techniques for examining data. A perusal space is a process which runs under UNIX and makes use of some combination of the graphical input/output hardware of the system. One such perusal space will allow a user to pass through a port in order to view selected frames from a video disk player. Another will provide for examining and editing documents by means of the Ned editor [BILOFSKY].

Example Graphical Data Space
Figure 2.1



In parallel to the graphical data space, the SDMS maintains a symbolic database management system (DBMS). The DBMS chosen for the prototype SDMS is INGRES [STONEBRAKER, HELD, WONG]. Tuples in the DBMS may have corresponding icons in the GDS, allowing the GDS to serve as a view of the database. The system provides tools for generating such views as a function of the contents of the database, so that the user may call up views tailored for specific purposes.

The following list summarizes the actions which the user and/or database administrator can perform:

1. Moving through the Graphical Data Space

    a. Scrolling (motion in the plane)
    b. Zooming (motion perpendicular to the plane)
    c. Passing through ports (to other I-Spaces or perusal spaces)
    d. Rapid transit directly to a point in an I-Space

2. Symbolic Queries

    a. Blink a specified icon
    b. Frame a specified icon
    c. Find (goto) a specified icon
    d. Associate an icon with a tuple
    e. Generate an I-Space of icons from a relation

3. Editing the Graphical Data Space

    a. Annotation

    b. Painting

    c. Defining Ports

        - Making I-Spaces
        - Establishing access controls
        - Making Perusal Spaces

    d. Making i-planes

4. Icon Class Description Language (ICDL)

    a. Creation/Editing

    b. Testing

2.1 Input/Output Devices

The SDMS user station contains a variety of interaction peripherals which provide for a wide range of data types and modes of interaction. They are:

1. three color CRT monitors which can be configured to display the contents of the graphical data space, maps of the data space, output from a video disk player, and menus;
2. two joysticks which control motion through the data

space;

3. a data tablet for input of icons, selection of menu items, and identification of positions in the graphical data space;
4. computer controlled sound playback equipment, to allow sound to be used as a data type; and
5. an alphanumeric keyboard to allow input of symbolic information.

2.2 Command Structure

SDMS interfaces to the user through two languages: SQUEL and ICDL. SQUEL is the primary command and query language, and is an extension of the QUEL query language of INGRES. It allows the user to move through the graphical data space, examine data, and create, modify, and destroy I-Spaces.

The Icon Class Description Language (ICDL) is used to define the correspondence between the symbolic data in the DBMS and the icons in the graphical data space. Through statements in ICDL, a user or database administrator can define icons whose appearance is a function of data in the DBMS.

Both languages make use of symbolic input (typing) and graphical input (menu selection, position input, and shape

description). The system is constructed to make typing unnecessary except in cases where the input of actual text strings (such as attribute names or values) is involved. In most cases, the user is given a choice of typing a command or selecting an item from a menu.

2.2.1 Modes of operation

The two languages of SDMS define two distinct modes of operation of the system. The system is normally in SQUEL mode, which allows manipulating the databases and the user's view of them. The system may also be used in ICDL mode in order to define new icon classes.

The mode of operation determines the kind of statements that a user can type at the keyboard and the information which is displayed on the various monitors. On the other hand, certain characteristics of SDMS are constant, in that they are a part of every mode of operation. These are:

1. The surface of the data tablet is marked with a static menu which is always available for global operations such as exiting from the system and entering various modes of operation. It is described in Section 2.2.2.

2. The main graphics screen always displays a view of

the graphical data space and the joysticks always allow the user to move around within it.

Other characteristics change depending on the mode of operation. The SQUEL mode of operation is characterized as follows:

1. The keyboard is connected to the SDMS monitor which accepts SQUEL statements and monitor commands (see Section 2.2.3).

2. The main graphics screen displays a view of the user's current location in the graphical data space.

3. The first auxiliary graphics monitor displays a navigational aid. This will be a "world-view map" of the top-level I-Space.

4. The second auxiliary graphics screen displays one of two possibilities, selected by the user. Either a second navigational aid is displayed, or the GDS Editor menu is displayed.

When a navigational aid is displayed on the second auxiliary screen, the entire screen is occupied. When the GDS Editor is displayed, only part of the screen is used. The remaining space will be used to post menus of commands. This feature is heavily used by the ICDL mode.

ICDL mode is used for editing icon class decriptions. ICDL mode is defined as follows:

1. The keyboard is connected to the ICDL editor of the ICDL monitor. The monitor allows manipulations of entire icon class descriptions while the editor allows manipulation of the statements in a description (see Section 2.2.4).

2. The main graphics screen displays a view of the GDS.

3. The first auxiliary graphics screen displays a navigational aid. This will be a "world-view map" of the top-level I-Space.

4. The second auxiliary graphics screen displays the GDS Editor plus one or more menus. The menus are described in Section 2.2.4.

2.2.2 Static menu

The tablet will have a permanent menu area, each having a special command associated with it. These commands are of general applicability and are always available to the user. The commands are:

1. QUIT - terminates the current session of SDMS.
2. ABORT - interrupts the current action and halts it. SDMS returns to a passive state. This can be used if

the user has done something incorrectly and he wants to abort before destructive action is taken.

3. HELP - offers information to the user concerning the use of SDMS.
4. STATUS - prints the current status of the system.
5. SQUEL MODE - puts the system into the SQUEL mode of operation.
6. ICDL MODE - puts system in the mode for editing ICDL.
7. ENABLE GDS EDITOR - makes available the full capabilities of the GDS editor. The second auxiliary screen then displays the GDS Editor.
8. DISABLE GDS EDITOR - disables the GDS editor, causing a navigational aid (if applicable) to appear on the second auxiliary screen.

2.2.3 SQUEL Monitor

The SQUEL monitor is the primary tool for communicating symbolically with SDMS. The SQUEL monitor is similar to the QUEL monitor supplied with INGRES, but it has been expanded to provide more control over the terminal session and to encompass operations on the graphical data space. Each transaction that is typed to the monitor must be terminated by a ";" or by a "\". If the termination character is the ";", the transaction is processed immediately. If the "\" is

used, a monitor command is expected to follow immediately after the "\". The monitor commands are listed and explained in Section 2.2.3.2.

2.2.3.1 SQUEL transactions

SQUEL transactions may be typed in directly to the SDMS monitor. After the transaction has been entered, it is sent to the SQUEL processor. Control does not return to the monitor until the transaction has been processed. This is the primary access method to the symbolic database.

2.2.3.2 Monitor commands

The monitor commands manipulate the input of previous transactions. Each transaction that the user enters is saved on a history list. The monitor commands allow the user to manipulate transactions on the history list. For example, he can edit a previous transaction and process the edited version. The monitor commands available are in a separate manual [SCHMOLZE, FRIEDELL] that will be included in the Final Design Document.

2.2.4 Editing ICDL - Icon Class Description Language

When the user wishes to define or edit Icon Class Descriptions, he enters ICDL mode. Since ICDL bridges the symbolic and graphical databases, it makes use of both symbolic and graphical input from the user. Although ICDL mode makes special use of some of the i/o devices, it still allows the user to navigate through the GDS as he wishes. The major departure from SQUEL mode is that SQUEL is not available from ICDL mode.

2.2.4.1 Menu commands

When SDMS is in ICDL mode, a menu devoted to ICDL will appear on the second auxiliary screen. These menu commands are useful for manipulating the user's collection of icon class descriptions (ICDs). When the user is not editing a particular ICD, keyboard input is directed to the ICDL monitor. The commands for this monitor are identical to the commands on the menu, allowing the user a choice of input device. The commands are:

1. directory - lists the ICDs in the user's (or other's) directory of ICDs.
2. copy - copies one ICD to another ICD.
3. delete - deletes an ICD.
4. edit - starts an editing session for a particular ICD. This is used for new or old ICDs.

5. test - enters the test mode for ICDL.
6. SQUEL mode - leaves ICDL mode and returns to SQUEL mode.

2.2.4.2 Using the ICDL editor

When a particular ICD is to be edited, a second menu appears and the terminal displays a special version of the Ned editor. The second menu contains the name of each statement in ICDL. To use this menu, the user positions the Ned cursor to the location where the statement should appear and touches a menu selection. If the statement requires information, such as an attribute name or a location, the user is prompted for it. Afterwards, the statement is generated automatically and inserted into the ICD at the cursor position. The user is also free to type in ICDL statements directly to the editor as well.

The OPEN and CLOSE features of Ned may be used to insert and delete statements in existing ICDs.

The pictures associated with an ICD will be stored in a special I-Space. The primary means of getting to this I-Space is via a menu selection from the second menu. This I-Space has only one i-plane which will hold the pictures for all ICDs. When in this special I-Space, the user may browse through other pictures, but he can only change those pictures associated with the ICD being edited.

2.2.4.3 Testing ICDL

The test mode allows the user to run an ICD and to look at the results. When test mode is entered, he is prompted for an I-Space to place the test icon and a qualification for retrieving a tuple for the test. The icon is generated and the SDMS flies to the location of the new test icon. At this point, the user can peruse the icon and the surrounding area. He leaves test mode with a "finished" command. The test icon is automatically deleted, leaving the I-Space unchanged, and the user is returned to his original position in the GDS.

2.3 System Specification - Interactive Input

This section describes the routines provided to enable the user to create and modify the Graphical Data Space (GDS) of the SDMS.

The usual mode of operation is to maneuver to an I-Space and issue the GDS edit command. This command causes a menu of interactive input commands to appear on the second auxiliary monitor.

The user may also invoke the edit command in such a way that the editor starts with a scratch i-plane. This i-

plane can later be inserted into the SDMS database or used as a source of graphical data to be copied piecemeal into various I-Spaces, or to be used with ICDL.

In order to perform editing of the graphical display space, the user is provided with a set of tools which can be used for generating and modifying the I-Space. These tools allow a user to work with general image areas when editing or with specific objects.

While editing, the user is at all times free to move about the GDS, allowing him to perform operations which involve more than one I-Space, such as copying information from one I-Space to another.

The graphics oriented functions are summarized as follows:

LINE - allows a user to draw a line of arbitrary width on the iplane.

FLOOD - allows a user to fill an area with a specified color.

PICK - defines an area or object to be copied.

PUT - defines an area or location for an object to be placed.

TEXT - allows the user to insert text onto the scratch pad.

GRID - places a grid on the screen for use in alignment of objects.

The remaining functions define an image area's characteristics.

MAKE ICON - defines a particular icon.

MAKE I-PLANE - defines a particular i-plane.

MAKE PORT - defines a location through which a user may pass to enter a new I-Space.

DELETE - allows a user to delete image areas, icons, i-planes and complete I-Spaces.

3. Text in SDMS

Text will appear in three guises in the SDMS system:

- static text is part of a picture
- dynamic text is generated in part of a picture in response to user actions (e.g. the result of a user query may be presented on a portion of the icon to which it refers.)
- encapsulated text is produced on a dedicated screen by some subsystem, such as a text editor or message program.

These three forms of text differ in the time and mode of their generation, in where and how they are stored, and in what contexts they may be seen by the user.

3.1 Static Text

Static text is generated as part of a picture, either directly by the user at the time an icon is defined, or when the ICDL defining the icon is executed (perhaps automatically at the time a new tuple is added to the symbolic database). It does not change; it is an integral part of the definition of the picture, stored in an image

form in the i-planes of its containing I-Space. From this, it follows that the text must be converted to pixel format at icon creation time. (Note that this approach allows a great deal of latitude in character format, from hand-drawn through carefully defined, shaded, and de-jaggied fonts which can be displayed in arbitrary attitudes.) Such text scrolls onto and off the screen along with the rest of the view; it grows as the user zooms in, and shrinks as he zooms out. In the usual case, parallel i-planes which show a text area at different scales will have different texts, or even have a text replaced by lines indicating its general shape, if it would be too small to be distinguishable.

3.2 Dynamic Text

Dynamic text is tied to a particular icon and position in I-Space, but the value of the text is not defined at the time the icon is generated, and in fact will normally change during the course of a user's interaction with SDMS. Only the area it occupies is predefined in ICDL. A typical use for dynamic text would be to allow symbolic response to a user query which applies to an indicated icon. A key issue is the efficient generation of dynamic text in acceptable time and format; this issue is

discussed in Section 3.4.

3.3 Encapsulated Text

Encapsulated text is not part of the standard SDMS world of I-Spaces. It belongs to subsystems which have been incorporated into SDMS, but which assert their own context once entered. When one of these subsystems, such as the text editor ned or the msg message system, is entered, it is allocated a display for its own use, on which its particular texts will appear. There is again a great deal of latitude in how the text will actually be presented on the screen (beginning on which screen holds it). At one end of the spectrum is a standard text terminal, 24 by 80 or 40 by 80 characters in the usual variety. At the opposite extreme, on the main SDMS display, text may be formatted with a high quality font, proportional spacing, justified margins, "turning" pages, and multiple colors. The choice within this range will be determined according to the needs of the subsystem; in an editor, quick response might prove more desirable than a very high quality image produced at the expense of additional processor overhead. A finished document merits more care in the production of its image, if it is going to be viewed critically.

3.4 Generation of Dynamic Text

Because of the ephemeral nature of dynamic text, it cannot be stored permanently in the i-plane in which it eventually appears; it must be generated anew in response to a particular user query. Nonetheless, once generated, it must be stored; at least for as long as it is possible for the user to scroll part of it off the screen and back on again. The location and format of this storage is one of the issues presented by dynamic text. The other concerns the time and location of the conversion from ASCII character codes to pixels in a 7 X 10 (or whatever) cell.

We have investigated three approaches to dynamic text storage:

1. Generate it into the i-plane which contains its window, erasing that window (that is, re-painting it with the background color) at some later time.
2. Maintain a parallel iplane (possibly with fewer bits depth) into which the image of the text can be written, which is combined with the real i-plane's image as it is moved to the display. This parallel plane can simply be discarded when the text is no longer needed.
3. Store the text in ASCII, and generate pixels only in the core buffer before an image is sent to the

display (or even have it generated in the display itself).

In the interests of minimizing disk i/o, we have settled on the third option. This has a beneficial side effect, in that dynamic text can be handled in a uniform fashion with modification functions (e.g. blinking, framing) specified for particular areas in an I-Space. For text, the modification function is simply TEXT, and the argument is a pointer to a string carried in the modification database. As an area is loaded into the core buffer on the way to the display, any applicable modifications are performed. In fact, if the display's character generator were acceptable, generation of the character images could be postponed until the image had actually been transferred to the display. This approach would require a fair amount of processing to ensure correct handling of texts which overlie the boundaries of the screen, however.

This raises the second issue, namely the actual conversion from ASCII characters to character images. The two alternatives are:

a. to do it in software in the PDP-11; or
b. to let the hard/firmware in the display take care of it.

The latter course appears faster, by a factor of at least 10, and will be used in most cases in SDMS. High-quality characters, with de-jaggied edges, such as may be required

by some perusal spaces, will require the software approach.

4. STRUCTURE OF SDMS

SDMS runs as a user job under the UNIX operating system. Invocation of SDMS results in the spawning of a hierarchical structure of concurrent cooperating processes (see Section 4.1) which communicate through the interprocess communication schemes described in Section 4.2.

Certain tasks performed by SDMS, especially those constructing graphical images, require rather lengthy computation and hence are incompatable with the highly interactive nature of the SDMS user environment. The multiprocess design of SDMS allows those processes to proceed in a backround mode as servers to interactive "front-end processes."

4.1 The SDMS Process Structure

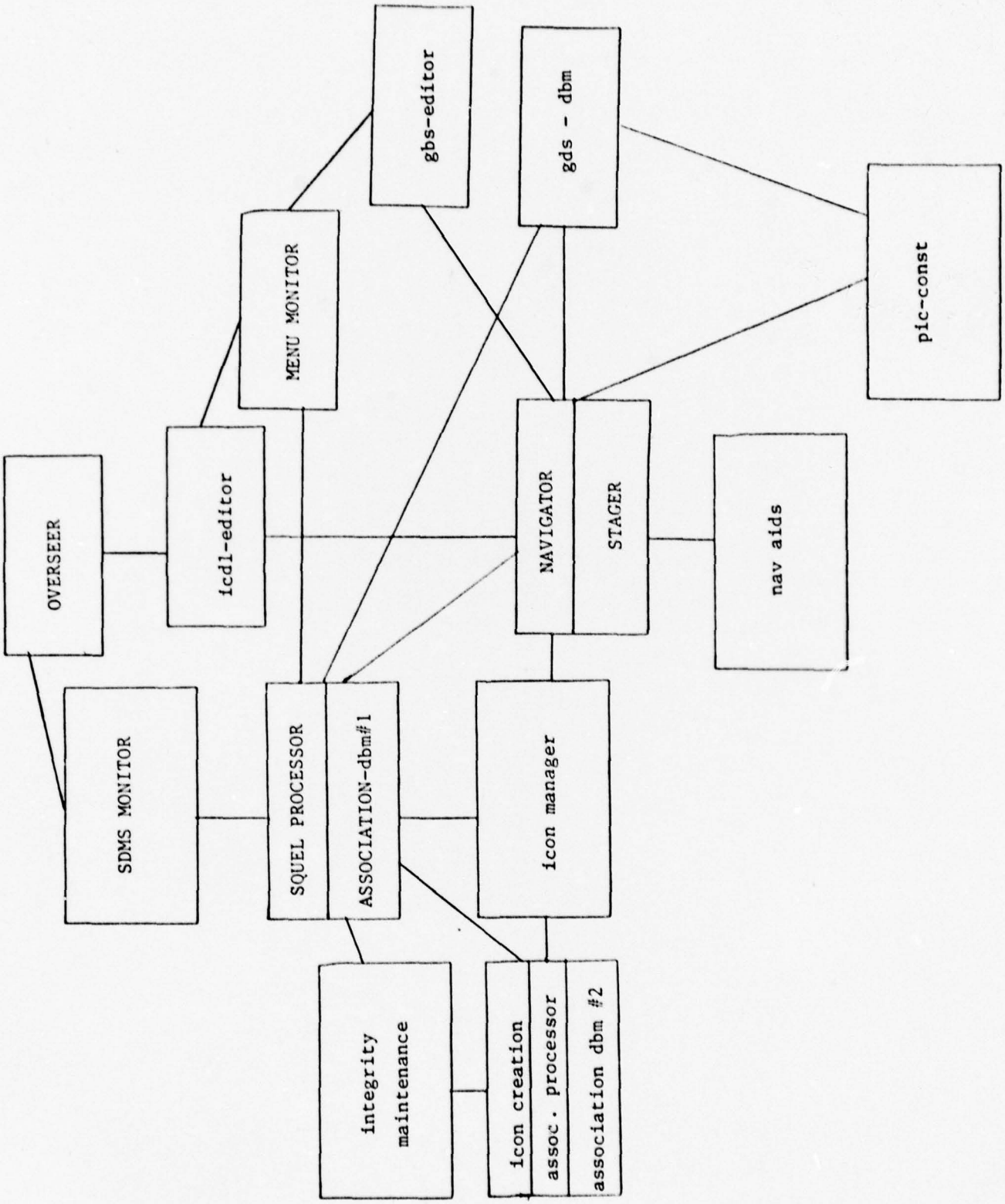
SDMS process consist of one or more functional modules. Modules are typically grouped into processes by related functionality. In exceptional cases, modules are grouped by common access to data structures or INGRES databases.

The SDMS process schema:

Modules of SDMS

Module Name	Description
overseer	Process overseer
sdms_monitor	SDMS Monitor
squel	SQUEL Parser
assoc_dbm	Association Database Manager
icon_creation	Icon Creation
assoc_proc	Association Processor
integrity	Integrity Maintenance
icdl_editor	ICDL Editor
icdl_dbm	ICDL Database Manager
icon_manager	Icon Manager
gds_editor	GDS Editor
navigator	Navigator
stager	Stager
disk_io	Disk IO
disp_io	Display IO
gds_dbm	GDS Database Manager
nav_aid	Navigational Aids
pic_const	Picture Construction
menu_monitor	Menu Monitor

SDMS Process Structure
Figure 4.1



4.1.1 INGRES interface

SDMS interfaces to INGRES through EQUEL. EQUEL is a mechanism which allows a programmer to access INGRES from within programs written in C, the standard language of UNIX. This mechanism includes a pre-processor which allows such C-programs to contain QUEL statements. Through EQUEL, the pre-processor converts the C-embedded QUEL statements into calls to subroutines which themselves are a part of INGRES. Currently, the EQUEL processor is insufficient to handle certain exceptional transactions with INGRES. Therefore, a few direct calls on the EQUEL subroutines appear in the SDMS implementation code. Further development of SDMS and/or EQUEL may make these direct calls unnecessary.

4.2 Concurrent process cooperation

As shown in the above discussion, the SDMS implementation involves a number of cooperating concurrent processes. For that reason, the various issues of concurrency, including those dealing with critical regions, shared resurces, and interprocess communication have been addressed. A discussion of these problems and their design solutions within the framework of SDMS is presented in Section 4.2.1. Section 4.2.2 describes in detail the actual implementation of this design through the facilities of the (UNIX) host operating system.

4.2.1 Design

The relationships between two cooperating processes within SDMS are of three types: the type associated with simple "co-routine" cooperating process environments, the "producer-consumer" relationship, and a more complex "requester-server" relationship often associated with physical resource management in operating system design.

The nature of the "co-routine" relationship eliminates the problem of critical regions when co-routine execution is explicitly interleaved: in such an environment, the active process can freely manipulate shared data as at

most one process is active at any time. The synchronization of such co-routines can be accomplished through the following message passing primitives:

SEND(message to cooperating process)

WAIT(message from cooperating process)

Optionally, processing may be included between the SEND and WAIT operations. This processing, however, must occur outside the critical region if it is to occur concurrently with the co-routine. The discipline of not accessing shared data in this non-critical region becomes a programmer's responsibility.

An alternative to the "co-routine" relationship which is also utilized in the SDMS implementation, is the "producer-consumer" relationship allowing essentially asynchronous open-loop operation of two concurrent processes connected by a single multiple slot message buffer. "Empty" and "full" signals denoting the availability of buffer slots and thereby constraining "producer-consumer" timing are provided by two additional buffers of very small (event) signals.

The somewhat more complex "requester-server" relationship involves a shared software resource, a "server process," which can used by one of two or more concurrent processes at a time. This relationship exists whenever any one process performs some function for at least two other

processes.

A resource allocation scheme similar to those used by operating systems to allocate physical computer system resources to a process has been applied to this situation. Simply stated,

1. a "requester" process issues a request to use the server and waits until it is granted
2. the "server" is allocated to a "requester" on a first-come first-served basis
3. when a "server" is allocated to a "requester," it is occupied until the "requester" releases it

A single message buffer per shared "server" process is sufficient to implement the described allocation scheme. The message buffer contains at most one message which is: "the 'server' is available." Any "requester" process wishing to use the "server" must issue a request of the form:

```
WAIT-IN-QUEUE(message in buffer)
READ(message).
```

The "server" freeing primitive becomes:

```
WRITE(message).
```

Having been allocated the "server" process, the "requester" may initiate the "co-routine" or "producer-consumer" relationship described above. As already discussed, this relationship continues until the "server" is explicitly

freed by the "requester."

4.2.2 Implementation within UNIX

In order to satisfy the inter-process communication needs of SDMS, the pipe mechanism which was the primary UNIX interprocess communication tool has been supplemented by a shared memory facility referred to as the "large core buffer area" or LCBA. This is a scheme for reserving a portion of primary memory which may be accessed by any number of concurrent processes. Selective use of both pipes and the LCBA provides the necessary process communication and timing facilities described below.

Utilization of the LCBA for storage of shared data provides an interprocess communication technique without the operating system overhead associated with pipes, and is therefore used whenever numerous and/or large messages are involved. Use of the LCBA entails a user process mapping of an area of the LCBA to an unused page of the process's virtual data address space. Multiple processes mapping the same area of the LCBA to their data address spaces may concurrently access the data stored there.

Process synchronization has been achieved through a feature inherent in pipe interprocess communication: a

read on an empty pipe causes the reading process to sleep in a queue until a message (at least one byte) is available in the pipe.

Implementation of the co-routine relationship between two processes is achieved by placing common data areas in the LCBA and utilizing the following co-routine calling convention:

```
WRITE(one byte to the co-routine's read pipe)
~ optional processing outside critical region ~
READ(one byte from co-routine's write pipe)
```

In addition, the single byte timing message may be interpreted to supply some further information. In the SDMS implementation, a timing byte's value is typically interpreted by the invoked co-routine to denote one of up to 265 unique messages. This is often a return code or a command to perform some operation on the shared data.

Implementation of the producer-consumer relationship is accomplished through a multiple slot message buffer (as described in Section 4.2.1) which is contained in the LCBA. Producer-consumer coordination is controlled as follows:

```
producer - (before writing message)
     IF(no message slots available)
          THEN WAIT(until slot available)
consumer - (before reading message)
```

IF(no messages slots filled)

THEN WAIT(until a slot is filled)

are implemented with pipes containing event messages which give the address of a message slot available for writing or reading.

As discussed in Section 4.2.1., the actual transactions of the requester-server are like that of any co-routine or producer-consumer. A scheme to allocate the server among competing requesters is what makes this relationship unique. The implementation of this allocation scheme involves one pipe per shared server process in addition to the two in use for interprocess timing and (sparse) communication. This pipe contains at most a single one byte message interpreted as meaning: "the server is available." In the requester-server environment, an allocation/ deallocation protocol must be observed by all requester processes as follows:

request server -

READ(one byte "server available" message)

free server -

write(one byte "server available" message)

Note that any process requesting the server (READ from pipe) while it is in use (pipe empty) will sleep until until it becomes available (one byte in pipe) through a free operation (WRITE to pipe) by the current user.

Finally, a mechanism suitable for exclusive locking of those data structures which may sometimes be available for shared access has been implemented using a semaphore technique. Initially the semaphore is equal to the maximum number of processes which may share the data. Shared use of the data is preceded by a decrement and test of the semaphore; a non-negative value indicates that shared use is allowed. If the semaphore's value is negative (shared access permission denied due to its use in exclusive access mode or by the maximum number of shared users) the requester is required to re-increment the semaphore and try again after putting itself at the bottom of the run-queue. Relinquishing shared access permission is associated with an increment of the semaphore.

A request for exclusive access entails a decrement of the semaphore by the maximum number of processes allowed shared access, and repetitive testing of the semaphore until it is non-negative, again with intervening reschedules. Freeing the data for shared access is simply a reinitialization of the semaphore to this maximum number of shared users. Note that this scheme forces shared access requesters to queue-up behind an exclusive access requester and thereby avoids a potential exclusive access requester lockout.

4.2.3 SDMS process interrupts

A mechanism to interrupt the normal execution of the various SDMS processes asynchronously has been implemented which provides the interrupted process with an ID and message from the interrupting process.

This mechanism can be described as a table-driven protocol for the use of the UNIX signal operations. A shared area of the LCBA contains a table with one entry per process. Each entry includes four fields:

1. UNIX_process_ID
2. interrupts_enabled
3. ID_of_interrupting_process
4. message_from_interrupting_process.

The UNIX signal operations are performed using signal number 14, a signal not currently in use for process termination by the operating system.

The specific interrupt primitives available and their realization in the SDMS implementation environment are:

1. INTERRUPTS ON

```
signal(14,handler);
var=1;
        /* here handler is the interrupt handler
        routine and var is the interrupts_enabled
        field of the process's own entry in the
        interrupt table */
```

2. INTERRUPTS OFF

```
while(--var)
```

```
        {
        ++var;
        reschedule();
        }
             /* var is the interrupts_enabled field of
             the process's own entry in the interrupt
             table */
```

3. INTERRUPT PROCESS

```
   while(--varA)
        {
        ++varA;
        reschedule();
        }
   varB="process's unique ID";
   varC="message to interrupted process";
   kill(varD,14);
             /* where varA is the interrupts_enabled
             field of the process to be interrupted; varB
             is the ID_of_interrupting_process field of
             the process to be interrupted; varC is the
             message_from_interrupting_process field of
             the process to be interrupted; varD is the
             UNIX_process_ID field of the process to be
             interrupted */
```

4.3 Log

Log(flags, format, arg, ...)

Log is provides a system-wide utility for noting interesting or exceptional conditions. In addition, it will be the standard means of invoking formatted reports of system status, and of suspending or halting system operation.

In the call to Log, flags is an integer consisting of a number of bit-flags as described below; format and the

remaining args are a string and other arguments used exactly as in a call to the system function printf. Log will format a message, with a timestamp and the name of the caller, as well as the text indicated by the second and following arguments, and write it to a log file. It may take further action, as requested by particular flags being set.

Currently defined flags include:

ERROR the invocation of Log indicates an error condition which requires recovery procedures.

DUMP a detailed, formatted report of common data areas and system status will be written to a file.

HALT sdms will exit, returning to unix

DEBUG sdms will enter a debugging mode in which normal operations are suspended and debugging tools are made available.

Combinations of these flags can be defined; for example, CRASH might be ERROR + DUMP + DEBUG. Other flags may be added as required, without impacting existing uses of the Log function.

4.4 SDMS Protection Mechanisms

Each I-Space in the GDS has exactly one supporting INGRES symbolic database. For the general user, protection is applied at three levels:

1. user
2. symbolic database
3. I-Space

each of which is described below.

4.4.1 User protection

User level protection is a privilege assigned to an individual user and controls whether he may create I-Spaces. This capability may be restricted in some systems to the Database Administrator. All other SDMS protection mechanisms are applied per symbolic database or per I-Space.

4.4.2 Symbolic database protection

Each Ingres symbolic database which supports SDMS I-Spaces has an associated list of SDMS users authorised to access it. Most users having access to the database will have private relations; however, one user may have shared relations. In effect, one user may make symbolic information available to all users with access privileges to that database through the use of publicly available relations.

4.4.3 I-Space protection

As in symbolic database protection, each I-Space has an associated list of SDMS users with access privileges. I-Space access privileges are of four types: read, write, control, and administer.

Read access to an I-Space provides the user with almost all the facilities of SDMS with the following provisions:

1. Graphical annotation of the GDS will be temporary.
2. Any new links or associations will be temporary.
3. i-planes may not be created.
4. Ports may not be created.
5. Access controls may not be changed.

Write access will imply all those privileges of read access. In addition, graphical annotation of the GDS will

be permanent.

Control access will imply all those privileges of write access. In addition, links and associations will be permanent, and i-planes and ports may be created.

Only users with administer privileges may destroy an I-space or change access controls on it. The creator of an I-Space is given administrator privileges; he may assign any combination of privileges to other users.

4.4.4 The database administrator

A very special user, the database administrator, or DBA, is immune to the SDMS protection mechanisms. The usual role of the DBA is simply that of administration, namely:

1. The database administrator maintains the list of valid SDMS users (who must also be valid UNIX users). The DBA specifies the create I-Space privilege for each user.
2. INGRES symbolic databases which support I-Spaces are created by the DBA. For each such symbolic database, those users authorized to access it are listed along with the shared relation user (if any).

References

[HEROT et al]
Herot, C.F.; Schmolze, J.; Carling, R; Farrell, J.; and Friedell, M. "SDMS Detail Design Document", Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139, October 1978.

[BILOFSKY]
Bilofsky, W. "The CRT Text Editor NED -- Introduction and Reference Manual" Technical Report No. R-2176-ARPA, The Rand Corporation, December 1977.

[STONEBRAKER, HELD, WONG]
Stonebraker, M.R.; Held, G.D.; Wong, E. "INGRES - A relational data base system", AFIPS Proceedings, Volume 44.

[SCHMOLZE, FRIEDELL]
Schmolze, J.; and Friedell, M. "SQUEL User Manual", Technical Report in progress, Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139.