PROGRESS REPORT ON THE INTERFACE BETWEEN ASTROPHYSICAL DATASETS AND DISTRIBUTED DATABASE MANAGEMENT SYSTEMS (DAVID)

Dr. S.S.Iyengar

Department of Computer Science

Louisiana State University

Baton Rouge, Louisiana 70803

January 1988

1. Introduction

This report gives a status report on the progress of the DAVID project being carried out at Louisiana State University, Baton Rouge, LA. The objective of this project is to implement an interface between Astrophysical datasets and DAVID. This report discusses the design details and implementation specifics of the generalized interface between the DAVID (Distributed Access View Integrated Database Management System) and Astrophysical datasets.

2. Project Details

Our thrust here is to interface the Astrophysical Datasets to the DAVID system. The motivation is to allow the DAVID users to use the DAVID primitives to operate on the astrophysical datasets. The main purpose of this interface is to simulate DAVID primitives and apply them on flat files as if they were applied to a database. Given the syntactical details, the application framework, interfacing specifications and a definition of the GSQL data manipulation language for the DAVID system, here is the design and implementation specs of a reliable software system which can synthesize, compile and execute queries on the astrophysical dataset system for data manipulation.

The details on the implementation are provided in the following sections.

2.1 An Overview of the Astrophysical Dataset Interface

As shown in the attached schematic (Figure 1) our software system includes code for interpreting and executing the following generic DAVID routines : ff_asgcluster, ff_trfirst, ff_trnext, ff_trdelete, ff_dasgcluster, ff_trupdate, ff_trprevious and ff_trlast. (working under 4.3 BSD UNIX on the VAX 11/780). The target framework for exploiting the interface capability is as follows. On receiving a query from the DAVID host, the query scheduler on determining that the query is on an external database would pass

control to the Resident GSQL primitive package(module 9) , the DAVID interface handler. Upon determining that the query is for the Astrophysical dataset, the package would invoke our system to execute the query on the flat file representing the Astrophysical dataset.

2.2 Files and data structures used in the Interface

Tapes containing the N30 data is read and transferred onto a disk file which has the following format.

Bytes	Description	Format
4	Record number	A4
1	Deleted flag	A1
88	Record read from tape as per specifications (see footnote*)	A88

This would be the flat file on which the Interface would operate on. The record number field as shown above indicates the record number relative to zero in the file. The deleted flag field indicates that this record has been deleted from the file. A '1' in this field indicates that this record has been marked deleted.

Structures like the CCA and TCA are used to store information to be passed to and from the interface and DAVID. To be more specific the CCA structure holds the flat file name and the flat file pointer. The TCA structure uses the buf_ptr field to get the contents of the tape and pass it on to DAVID and is also used to transfer information onto a specific record of the flat file. Details of the CCA and TCA structures are as shown in figures A1 and A2.

---

* Documentation for Machine-Readable version of the Catalog of 5,268 Standard Stars, 1950.0 Based on the Normal System N30

2.3 Process

The following is a brief write-up on the primitives used in the interface. These routines have been coded and tested.

a) FLAT FILE Install : This procedure is called to install a FLAT FILE onto the DAVID system. Here hand coding of the create-cluster is used.

b) FLAT FILE Drop Cluster : This procedure is used to delete a flat file containing DAVID cluster data. The name of the flat file is got from the gsql_row and the unix unlink command is used to delete the file.

c) FLAT FILE Define Cluster : This procedure creates a FLAT FILE capable of holding data from a DAVID cluster. Here the C creat command is used to create the file and the name is inserted in the arbi_file_name field of the gsql_row.

d) FLAT FILE Table Row Update : This procedure is used to simulate a DAVID table-row update on a flat file. The address of the data buffer of the TCA is obtained by making a call to a macro called DATA which has the following format :

d1 = DATA (t1) The d1 is a pointer returned by DATA which points to the buffer in TCA from where the data is taken and written over the current record in the flat file.

e) FLAT FILE Asgncluster : This procedure is used to open a flat file so that is can be read by DAVID. Here the file specified in the res_name of CCA is opened. The res_file_ptr field in the CCA is assigned to the file pointer obtained by opening the file.

f) FLAT FILE Table Row First : This procedure is used to simulate a DAVID table-row first . The address of the data buffer of the TCA where the data read is stored is obtained by making a call to a macro called DATA. The record read is the first record of the flat file specified by the res_file_ptr field of the CCA.

g) FLAT FILE Table Row Last : This procedure is used to simulate a DAVID table-

row last. The address of the data buffer of the TCA where the data read is stored is obtained by making a call to a macro called DATA. The record read is the last record of the flat file specified by the res_file_ptr field of the CCA.

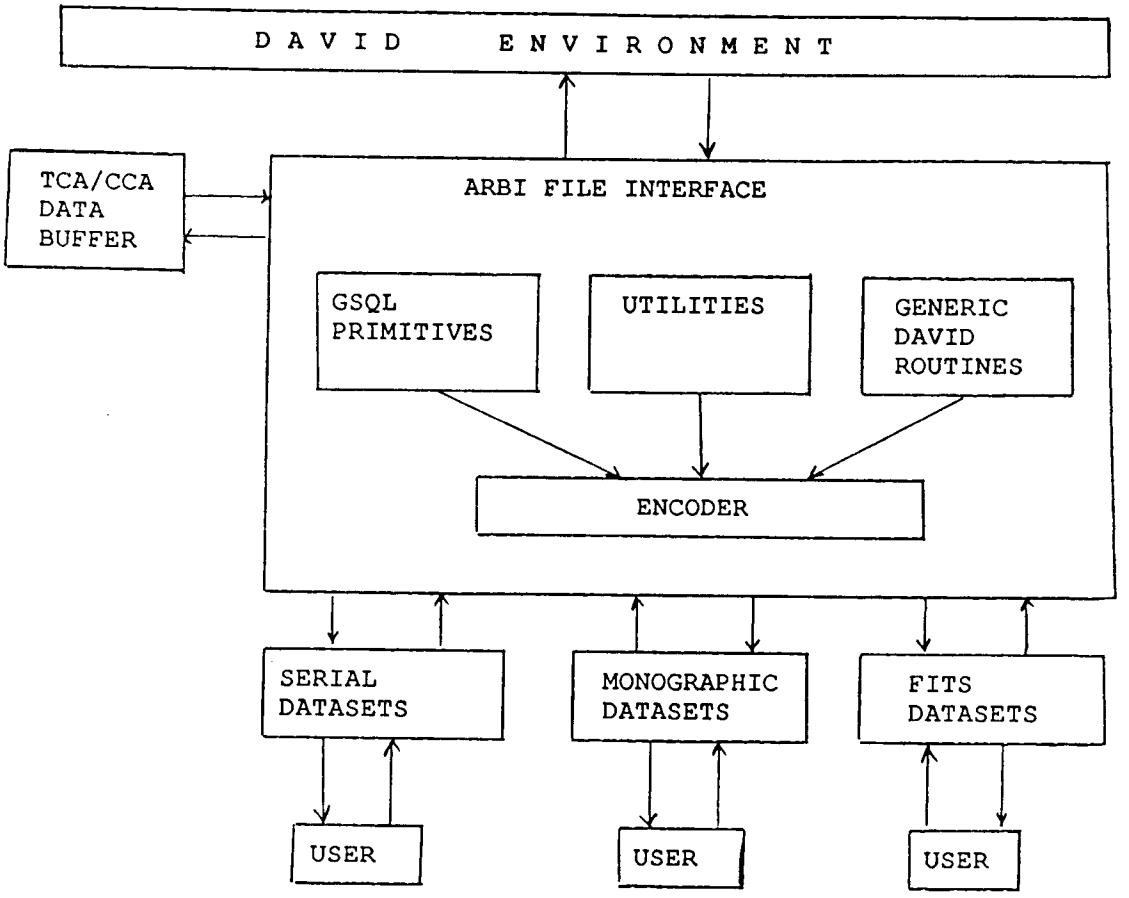
h) FLAT FILE Table Row Previous : This procedure is used to simulate a DAVID table-row previous. The address of the data buffer of the TCA where the data read is stored is obtained by making a call to a macro called DATA. The record read is the previous record of the flat file specified by the res_file_ptr field of the CCA.Here the record number is used to locate the previous record in the flat file.

h) FLAT FILE Table Row Delete : This procedure is used to simulate a DAVID table-row delete. The address of the data buffer of the TCA where the data read is stored is obtained by making a call to a macro called DATA. The record is marked deleted by setting '1' in the deleted flag field of the flat file.

i) FLAT FILE Dasgncluster : This procedure is used to close a flat file. The pointer to the file is obtained from the res_file_ptr field of the CCA and then closed.

3. Future Directions

Presently a driver has been written to read from a tape containing the Normal System N30 catalogue data and create the flat file in the format specified above. This driver would be enhanced to handle data from other tapes containing data in different formats viz., AGK3 Star Catalogue. Certain consideration would be given to specify tape format and content details to the driver.



GENERALIZED INTERFACE SCENARIO FOR HETEROGENOUS ASTROPHYSICAL DATASETS

FIGURE A1 and A2

```
/* type vca.h */
/* vca.h - vca */
#ifndef VCA_H
#define VCA_H
#define DTLENGTH 20
#define CLUSTER_HDG_LEN NODE_LEN + USER_LEN + FILE_NAME_LEN +  \
        NAME_LEN + 3
typedef char TRANID[15];
typedef struct daddr {
  unsigned            page_no;
  USHORT              record_no;
} DADDR;

typedef struct constraint
{ char          conid[12];      /* Unique id of the constraint KW*/
  char          contype[4];     /* GD, GK,... KW*/
  char          operations[3];  /* Insert/Update/Delete KW*/
  char          imm_proc[8];    /* Immediate Checking procedure name KW*/
  char          def_prof[8];    /* Deferred checking procedure KW*/
  STRING        sdef;           /* Constraint definition in string form KW*/
  struct constraint   *next;    /* pointer to next constraint */
} CONSTRAINT;

typedef struct privilege    /* R: read, W: write, E: execute, D: delete */
{ char          group[4];     /* group id or * */
  char          member[4];    /* member id or * */
  char          type[4];      /* R/W/E/D */
  struct privilege    *next;
} PRIVILEGE;

typedef struct  bind_info {
  POINTER destination;
  USHORT  length;
  USHORT  type;
  struct  bind_info  *next;
 } BIND_INFO; /*KW*/

typedef struct   field
{ char          name[NAME_LEN];   /* Name of FIELD */
  USHORT        id;         /* system assigned id of field */
  USHORT        type;       /* Type of FIELD int, real,..*/
  USHORT        length;     /* Length of FIELD */
  STRING        help;        /* Help text for the FIELD */
  BIND_INFO     *bind;       /* Pointer to bind table */
  struct field        *next;
} FIELD;

typedef struct   argument
{ char          name[NAME_LEN];   /* Name of argument */
  char          type[DTLENGTH];   /* Type of argument; int, real,..*/
  USHORT        length;     /* Length ofargument */
  STRING        help;       /* Help text for the argument */
  STRING        value;      /* Argument value */
  struct argument         *next;
} ARGUMENT;

typedef struct    anode   {
    USHORT   depth;          /* Recursive depth */
    USHORT   sib_id;         /* Child ID */
    char     *buf_ptr;       /* buffer to store table row */
```

```
    struct anode *sib_ptr;
    struct anode *successor;
} ANODE; /* KW */

typedef struct sort_field{
  FIELD          *field;     /* pointer to a field */
  USHORT         id;         /* sort field index: 1,2,3 etc major to minor */
  struct sort_field *next;   /* link */
} SORT_FIELD; /* KW */

typedef struct tca_link {
  struct tca     *tca_ptr;   /* pointer to next TCA
                                ON DISK VERSION THIS SHOULD BE name
                                char [20] the TABLE NAME KW */
  USHORT         id;        /* index of parent or child: 1,2,3 etc  KW*/
  struct tca_link *next;
} TCA_LINK;

typedef struct tca {
  char           name[NAME_LEN];   /* Name of table */
  unsigned       id;          /* system assigned unique id of table SS*/
  char           type[DTLENGTH];   /* Table type INDEX, CHAINED, BLOCKED,..*/
  char           structure;  /* i=index,$=$table  KW */
  USHORT         nfields;    /* # of fields */
  USHORT         size_of_row; /* size of row with pointers */
  USHORT         nchildren;  /* Number of children KW */
  USHORT         nparents;   /* Number of Ancestors KW */
  USHORT         nconstraints;   /* Number of constraints */
  USHORT         nlptrs;     /* number of logical (box 6) pointers */
  USHORT         npptrs;     /* number of physical (box 7) pointers */
  DADDR          special_row;  /*addr of spec.row containing
                                  q/tbase tables w/0 prnts KW*/
  DADDR          cur_tsmap;    /* disk pointer to current tsmap */
  DADDR          sav_tsmap;    /* disk pointer to committed tsmap */
  BOOL           rdrn;         /* reuse deleted record number flag */
  USHORT         nsort_fields; /* number of sort fields  KW*/
  char           sort_method; /*'A' (ascend) 'D' (desc) or blank (none) KW*/
  struct  curptr {
    unsigned     page;    /* current page index used in walking */
    USHORT       record;  /* through table pages  SS */
    USHORT       index;
    }curptr;
  STRING         help;       /* Help text for the table */
   BIND_INFO      *bind;
  SORT_FIELD     *sort_fields;  /* Pointer to sort fields KW*/
  struct tca     *ctca;       /* points to tca of chain parent */
  char           *buf_ptr;   /* pointer to current table buffer */
  ANODE          *aux_bufs;  /* Ptr to auxilliary buffers */
  FIELD          *field;     /* Field control areas */
  TCA_LINK       *children;  /* Child table control areas KW */
  TCA_LINK       *parents;   /* Ancestor table contrpol areas KW */
  CONSTRAINT     *constraints;  /* pointer to table of constraints KW*/
  struct tca     *next;      /* pointer to next tca */
} TCA;

typedef struct fca_link  {
  struct fca     *fca_ptr;   /* KW */
  struct fca_link *next;
} FCA_LINK;

typedef struct fca  {
  char           name[NAME_LEN];   /* Name of function */
  char           type[10];   /* Function type */
  USHORT         nargs;      /* # of arguments */
```

```
  unsigned        nconsts;    /* Number of constraints */
  DADDR           physfca;    /* Future use */
  STRING          help;       /* Help text for the function */
  ARGUMENT        *argument;  /* Argument control areas */
  CONSTRAINT      *constr;    /* Function constraints */
  FCA_LINK        *next;      /* Successor function control areas */
  FCA_LINK        *before;    /* Ancestor function control areas */

} FCA;

typedef struct path  {
  USHORT          altno;   /* 0,1,2,.. 0 is the primary path */
  struct path     *next;
  struct fnodes   *fnode_ptr;  /* KW */
} PATH;

typedef struct fnodes
{ char            id[20];     /* Unique id for the function execution */
  char            name[20];   /* Function name */
  char            type[6];    /* Procedure/command/function */
  char            parent[20]; /* Parent tree. Used only for root */
  char            from[100];  /* node that the function came from */
  char            to[100];    /* destination node */
  struct          variables
                              /* This table is used only if the type
                                 of the function is PROCEDURE.  This table
                                 basically contains the local variables
                                 used within the procedure's scope.
                              */
                  { char      name[30];
                    char      type[10];
                    unsigned  length[3];
                    STRING    value;
                  } variable;
  STRING          arglist;    /* Argument list separated by comma */
  struct          names
                  { STRING *name;
                    struct names *next;
                  } *clusters;  /* Name of the clusters assigned */
  USHORT          priority;    /* Priority assigned */
  struct          times       /* Performance measures */
                  { char  arrived[15];  /* Time queued */
                    char    start[15];  /* Time of execution start */
                    char     stop[15];  /* Time of execution stop */
                    struct  times *next;
                  } *time;
  char            status[2]; /* The status of execution */
  unsigned        size;       /* Operation size */
  PATH            *next;      /* Next function to be executed */
  PATH            *before;    /* Function to be executed before */
} FNODES;

typedef struct file_control  {
  char          file_name[FILE_NAME_LEN];  /* name of file */
  unsigned      sfile_ptr;  /*.'system file pointer */
  struct file_control  *next;
} FILE_CONTROL;

typedef struct ptr_alloc {
  char          *ptr;         /* ptr to allocated area for assigned stuff */
  struct ptr_alloc     *next;       /* pointer to next entry */
} PTR_ALLOC; /* KW */

typedef struct cca
```

```
{ char          name[NAME_LEN];  /* Name of the cluster only-no node,*/
  unsigned      id;          /* System assigned unique id of cluster SS*/
  char          type[20];    /* External, Actual, Tree,..*/
  char          created[20];  /* Time of the cluster installation */
  char          updated[20];  /* Last time of the cluster update */
  unsigned      structure;    /* int, corresponding to cluster
                              kind: QBASE, TBASE, DBASE, RBASE, etc */
  DADDR         sav_csmap;    /* disk address of commited cluster storage map */
  DADDR         cur_csmap;    /* disk address of current cluster storage map */
  USHORT        ntcas;          /* number of tables */
  STRING        sdef;           /* Text of defintion */
  STRING        help;        /* Help text */
  STRING        space_def;    /* Initial space definition used */
  STRING        res_name;       /* resident data base name */ /*MM*/
  STRING        res_pass1;      /* password for arbi cluster *//*MM*/
  STRING        res_pass2;      /* password for arbi cluster *//*MM*/
  STRING        res_pass3;      /* password for arbi cluster *//*MM*/
  STRING        arbi_file_name;  /* File name of arbi format *//*MM*/
  /* below are memory only fields */
  FILE_CONTROL  *file;          /* pointer to file control */
  USHORT        res_file_ptr;  /* resident file pointer */
  PRIVILEGE     *privilege;    /* Pointer to privilege table */
  char          access[3];     /* Access allowed to this cluster */
  TCA           *tca_ptr;      /* data table control area KW */
  struct cca    *next;         /* next cca in vca chain */
  PTR_ALLOC     *allocation;    /* table of allocated pointers for assign KW */
  PTR_ALLOC     *next_allocation;  /* ptr where next allocated ptr goes KW */
  TCA           *dollar;       /* pointer to $table- TEMP only KW */
  char          constraint_flag; /*I (immediate) or D (deferred-default) */
  char          verify[3];     /* contains "cca" to verify that this is
                                  an allocated cca */
  char          *wca_ptr;         /* pointer to window control */

} CCA;

typedef struct process_control
{ char          pname[30]; /* Name of the process file */
  char          name[20];  /* Name of the process */
  STRING        help;      /* Help text for the process */
  STRING        sdef;      /* String form of the process definition */
  char          created[20]; /* Time of the process installation */
  char          updated[20]; /* Last time of the process update */
  char          compile[20];  /* Name of the compile command file */
  char          link[20];    /* Name of the link command file */
  char          run[20];     /* Name of the run command file */
  USHORT        nfcas;       /* Number of fcas */
  FCA           *fcas;      /* Fcas table */
  PRIVILEGE     *privilege; /* Access privilege against groups
                             and their members of the user.
                             ex} {groupid, memberid, priv}
                                ={ 123jH, *, ED } would mean
                             that the members of the group
                             123 are allowed to execute or
                             delete the process.  If a wild
                             card is used as the group id, then
                             it would have indicated all the
                             group in entire DAVID network*/

  struct process_control *next;
} PROCESS_CONTROL;

typedef struct vca   {
  char          username[20];  /* Name of the user */
  char          groupid[20];  /* User's group id  */
```

```
  char          memberid[20];  /* Member id within the group */
  char          viewid[15];  /* Unique id for this view */
  char          autocommit_flag[3];   /* On/off */
  char          commit_point[5];    /* Last commit point */
  char          rollback_point[5];    /* Later than commit_point */
  char          sleep_flag[5];   /* awake/temporary sleep/permanent sleep */
  unsigned      status;      /* Completion status of operation */
  char          constraint_flag; /*I(immediate),D(deffered),O(off-def) KW */
  struct        synonym
                { STRING   left;
                  STRING   right;
                  struct synonym *next;
                } *synonyms;     /* Temporary synonyms. Effective for
                                    this login only */
  FILE_CONTROL     *files;
  CCA              *clusters;
  PROCESS_CONTROL  *processes;
  FNODES           *trees;
} VCA;
#endif
```