Case Study 1—Design Document for Structured Design

1 Overview

The goal of this project is to develop a system for scheduling the courses in a computer science department, based on the input about classrooms, lecture times, and time preferences of the different instructors. Different conditions have to be satisfied by the final schedule and which have been specified in the SRS.

This document specifies the final system design for the system. It also gives some explanations on how the design evolved and why some design decisions were taken.

1.1 DFD of the System

The DFD of the system which describes the various inputs and outputs and the flow of data in the system is shown in Figure 1. The DFD gives an overall idea of how this system processes the data and satisfies the various constraints.

1.2 Structure Chart

The structure chart of the proposed system describes the overall structure of the initial design that was created is given in Figure 2. After analysis, this design was enhanced. How this structure chart was evolved using the structured design methodology is discussed in the book in detail.

Note that this design is consistent with the architecture of the system given in its architecture document. Mapping the modules in this design to the components in the architecture is also quite straightforward.

- "Process File 1" component maps to "Validate File 1" module and its subtree.

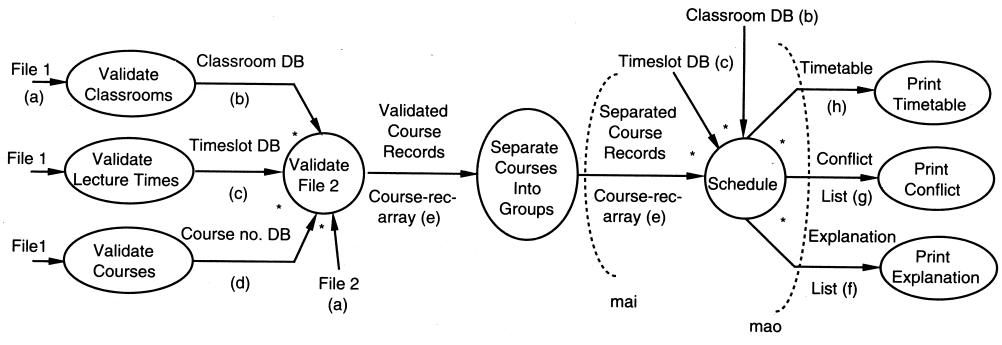

Figure 1: Data flow diagram for the case study.

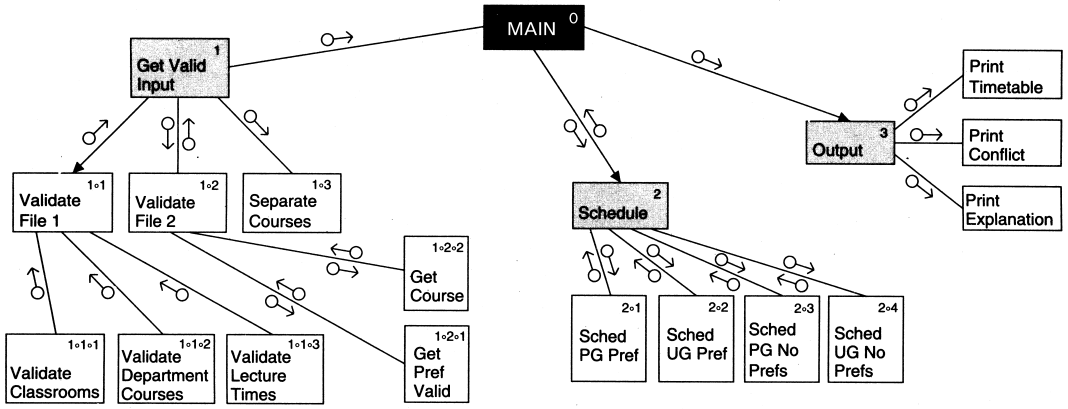

Figure 2: Structure chart for the system.

- "Process File 2" component maps to "Validate File 2" and "Separate Courses" modules (and their subtrees).
- "Schedule the courses" components maps to the "Schedule" and "Output" modules (and their subtrees)
- The pipes in the architecture are implemented in this design by the parameter passing mechanism.

2 Design Specification

The modules in the structure chart give the name and the main parameters passing between the modules. We first specified these modules precisely, and then we analyzed them using the information flow complexity metrics. Result of this analysis is given later in this document. Based on this analysis, the initial design was modified—the modifications made are also discussed later in the document. Here we specify the interface of each of the functions precisely in the final design of the system.

```
/*  Revised version. Last modification 24 Feb 1995   */

/* This software is for scheduling a set of courses offered by the Computer
 * Science Department - requirements of which are given in the SRS.
 * The design was done using structured design methodology, and the structure
 * chart is given elsewhere. This design document specifies the design formally.
 * It contains all descriptions of the major data structures and the modules
 * that are needed to implement a solution. This design was analyzed using
 * the dmetric tool for analyzing information flow complexity and was
 * found to be acceptable */
```

```
/*****************************************************************/
/* DATA DEFINITIONS         */
/*****************************************************************/


typedef char courseno[MAX_COURSENO_SIZE];
typedef char timeslot[MAX_TIMESLOT_SIZE];

typedef struct {
     int room_no;
     int capacity;
} room;

typedef struct LstNode {  /* A general linked list of integers */
   int index;     /* typically index in some DB will be kept */
   struct LstNode *next;
} lstnode;

typedef struct {     /* information about a course to be scheduled */
     int cnum;               /* index in CourseDB of this course */
     int enrol;              /* expected enrollment */
     lstnode *prefs;  /* list of prefs, kept as index in timeslotDB */
} course;

typedef struct {           /* to produce conflict or explanation info */
    int err_code;
    int cnum;              /* course no. for which error found */
    int roomno;            /* room no. may be needed in some error message */
    lstnode *prefs;        /* lst of prefs (time slots) */
} error;

/*-------------------------------------------*/

FILE *file1,*file2; /* input files */

courseno CourseDB[MAX_COURSES];     /* List of valid courses */
room ClassroomDB[MAX_ROOMS]; /* List of valid classrooms */
timeslot TimeslotDB[MAX_TIMES];     /* List of valid time slots */
course SchCourses[MAX_COURSES]; /* List of courses to be scheduled */

int TimeTable[MAX_ROOMS][MAX_TIMES]; /* rows are indices in ClassroomDB;
                              columns in TimeslotDB */

int PgAlloc[MAX_TIMES]; /* Time slots occupied by PG courses. An entry is -ve
                  if not allocated to a PG course, +ve otherwise. Used
                          for checking  conflicts while schduling PG courses*/

error ExplnLst[MAX_COURSES]; /* expln for each unschedulable pref */
error ConflLst[MAX_COURSES]; /* expln for unschedulable courses */



/*****************************************************************/
/*   MODULE SPECIFICATIONS    */
/*****************************************************************/
```

```
main(IN:file1,file2)
char *file1, *file2;
{
    SUBORDINATES : get_validated_input(),schedule(),print_output();
    SIZE: 10
}

/*****************************************************************/

/* This module validates the contents of file1 and file2 and produces
 * lists for valid courses, classrooms, time slots, and courses to be
 * scheduled. If errors are found in the input file format, it exits if
 * scheduling is not possible, else ignores the erroneous data */

get_validated_input(IN:fd1,fd2, OUT:TimeslotDB,ClassroomDB,CourseDB,SchCourses)
FILE *fd1,*fd2;
timeslot *TimeslotDB;
room *ClassroomDB;
courseno *CourseDB;
course *SchCourses;
{
    SUBORDINATES :validate_file1(),validate_file2(),separate_courses();
    SIZE: 20
}

/*-------------------------------------------*/

/* Top-level module to schedule courses. Besides timetable, it
 * gives explanation for courses/prefs that could not be satisfied */

schedule(IN:ClassroomDB,TimeslotDB,CourseDB,SchCourses,
         OUT:TimeTable,ConflLst,ExplnLst)
room *ClassroomDB;
timeslot *TimeslotDB;
courseno *CourseDB;
course *SchCourses;
int **TimeTable;
error *ConflLst;
error *ExplnLst;
{
    SUBORDINATES : sched_pg_pref(), sched_ug_pref(),
                        sched_ug_no_pref(), sched_pg_no_pref();
    SIZE:10
}

/*-------------------------------------------*/

/* Top-level Print module to  produce all outputs */

print_output(IN:TimeTable,ExplnLst,ConflLst)
int **TimeTable;
error *ConflLst;
error *ExplnLst;
{
```

```
    SUBORDINATES :print_TimeTable(), print_explanation(), print_conflict();
    SIZE:10
}

/*****************************************************************/

validate_file1(IN:fd,OUT:ClassroomDB,CourseDB,TimeslotDB)
FILE *fd;
room *ClassroomDB;
char *CourseDB;
char *TimeslotDB;
{
    SUBORDINATES :validate_classrooms(), validate_dept_courses(),
        validate_lec_times();
    SIZE: 10
}

/*-------------------------------------------*/

validate_file2(IN:fd, OUT:SchCourses)
FILE *fd;
course *SchCourses;
{
    SUBORDINATES :get_course_index(),chk_dup_sched_course(),
                        get_next_line(),form_pref_list();

    SIZE: 100
}


/*-------------------------------------------*/

/* Rearrange courses in this order:  PG courses with preferences, UG courses
 * with prefs, PG courses with no prefs, UG courses with no prefs */

separate_courses(IN:CourseDB,IN_OUT:SchCourses)
char *CourseDB;
course *SchCourses;
{
    SUBORDINATES :
    SIZE:60
}

/*-------------------------------------------*/

/* Schedules PG courses with prefs - allots highest possible preference
 * (records in PgAlloc also). For prefs it cannot satisfy, makes entry in ExplnLst.
 * If cannot schedule a course, makes entry in ConflLst */

sched_pg_pref(IN:SchCourses,OUT:TimeTable,ExplnLst,ConflLst,PgAlloc)
course *SchCourses;
int **TimeTable;
error *ExplnLst;
error *ConflLst;
int *PgAlloc;
```

```
{
    SUBORDINATES :get_room();
    SIZE:75
}

/*-------------------------------------------*/

/* Schedule UG courses with preferences. This is not straightforward.
 * Before alloting a <timeslot, classroom>, it has to check if this allotment
 * is "safe," i.e., the schedulability of PG courses with no prefs is not disturbed.
 * It does so by first "simulating" a scheduling of pg_no_pref courses - an allotment
 * is safe if the same no. can be still be scheduled after this allocation */

sched_ug_pref(IN:SchCourses,PgAlloc, OUT:ConflLst,ExplnLst, IN_OUT:TimeTable)
course *SchCourses;
int *PgAlloc;
error *ConflLst;
error *ExplnLst;
int **TimeTable;
{
    SUBORDINATES :get_room(),PgNoPrefSchlable(),is_safe_allotment();

    SIZE:100
}

/*-------------------------------------------*/

/* Schedule PG courses with no prefs. Returns the number of courses scheduled */
int sched_pg_no_pref(IN:SchCourses,PgAlloc, OUT:ConflLst,TimeTable)
course *SchCourses;
error *ConflLst;
int *PgAlloc;
int **TimeTable;
{

    SUBORDINATES :get_room();
    SIZE:50
}

/*-------------------------------------------*/

sched_ug_no_pref(IN:SchCourses, OUT:ConflLst, IN_OUT:TimeTable)
course *SchCourses;
int **TimeTable;
error *ConflLst;
{
    SUBORDINATES :get_room();
    SIZE:40
}

/*********************************************************************/

/* Checks if an allotment for a UG course with preference is safe, i.e.
 * the schedulability of PG courses with no preferences is not affected */
```

```
int is_safe_allotment(IN:time_slot,roomno,SchedPgNoPrefs,TimeTable,PgAlloc,SchCourses)
int time_slot,roomno;
int SchedPgNoPrefs; /* initially  schedulable pg_no_pref courses */
int **TimeTable;
int *PgAlloc;
course *SchCourses;
{
    SUBORDINATES : PgNoPrefSchlable();
    SIZE:50
}

/*-----------------------------------------*/

/* Returns the number of schedulable pg_no_pref courses without modifying
 * the timetable and PgAlloc */

int PgNoPrefSchlable(IN:TimeTable,PgAlloc,SchCourse)
int TimeTable[][MAX_TIMES];
int *PgAlloc;
course *SchCourse;

{
   SUBORDINATES : sched_pg_no_pref();
   SIZE : 15

 }


/*************************************************************************/

/* Get validated classroom info and build ClassroomDB.  If any error, continue
 * parsing, ignoring that room no. Room records sorted in increasing capacity */

validate_classrooms(IN:fd,OUT:ClassroomDB)
FILE *fd;
room *ClassroomDB;
{
    SUBORDINATES :get_next_line(),chk_fmt_room_no(),
  chk_range_cap(),chk_dup_room(),sort_rooms();
    SIZE:80
}

/*-----------------------------------------*/

/* Get validated list of courses from File 1*/
validate_dept_courses(IN:fd,OUT:CourseDB)
FILE *fd;
char *CourseDB;
{
    SUBORDINATES :get_next_line(),chk_dup_course(),chk_fmt_course_no();
    SIZE:75
}

/*-----------------------------------------*/
```

```
/* Get validated time slots from File 1*/
validate_lec_times(IN:fd,OUT:TimeslotDB)
FILE *fd;
timeslot *TimeslotDB;
{
    SUBORDINATES :get_next_line(),chk_fmt_time_slot(),
  chk_dup_slot();
    SIZE:70
}

/*-------------------------------------------*/

 /* Forms a list of validated preferences in the given order  */
form_pref_list(IN:line,TimeslotDB,OUT:pref_list)
char *line;
timeslot *TimeslotDB;
lstnode *pref_list;
{
    SUBORDINATES :
    SIZE: 45
}


/*****************************************************************/


print_TimeTable(IN:TimeslotDB,ClassroomDB,CourseDB,TimeTable)
timeslot *TimeslotDB;
room *ClassroomDB;
courseno *CourseDB;
int **TimeTable;
{
    SUBORDINATES :
    SIZE:40
}

/*-------------------------------------------*/

/* Print the reasons why higher prefs for a course could not be honored */
print_explanation(IN:ExplnLst,CourseDB,TimeslotDB,TimeTable,ClassroomDB)
error *ExplnLst;
courseno *CourseDB;
timeslot *TimeslotDB;
room *ClassroomDB;
{
    SUBORDINATES :
    SIZE:50
}

/*-------------------------------------------*/

/* Print the list of courses that are not scheduled in the final table
 * and specify the conflicts that prevented their scheduling */

print_conflict(IN:ConflLst,CourseDB)
```

```
error *ConflLst;
courseno *CourseDB;
{
    SUBORDINATES :
    SIZE:50
}


/*************************************************************************
 *                 Utility Routines                                      *
 *************************************************************************/


/* Check if capacity of a room is in the range [10 ... 300] */
int chk_range_cap(IN:cap)
int cap;
{ SUBORDINATES : SIZE:10 }

/* Sort the room records in increasing order of capacity */
sort_rooms(IN_OUT:ClassroomDB)
room ClassroomDB;
{ SUBORDINATES : SIZE:20 }

/* Checks if the room_no given already exists in the ClassroomDB */
int chk_dup_room(IN:ClassroomDB,room_no)
room *ClassroomDB;
char *room_no;
{ SUBORDINATES : SIZE:20 }

/* Check whether this course_no already exists in the CourseDB */
int chk_dup_course(IN:CourseDB,course_no)
courseno *CourseDB;
char *course_no;
{ SUBORDINATES : SIZE:20 }

/* check whether this time slot is a duplicate */
int chk_dup_slot(IN:TimeslotDB,time_slot)
timeslot *TimeslotDB;
char *time_slot;
{ SUBORDINATES : SIZE:20 }

/* Returns the index of the course number in CourseDB */
get_course_index(IN:course_no,CourseDB, OUT:course_index)
int course_index;
char *course_no;
courseno *CourseDB;
{ SUBORDINATES : SIZE:15 }

/* checks if a course number is a duplicate */
int duplicate_course(IN:CourseDB,cnum)
courseno *CourseDB;
char *cnum;
{ SUBORDINATES : SIZE:20 }

/* returns the index of the smallest room that can accommodate the course */
```

```
int get_room(IN:ClassroomDB,enrol)
room *ClassroomDB;
int enrol;
{ SUBORDINATES : SIZE:15 }

/* returns first nonempty line from the input file */
get_next_line(IN:fd,OUT:line)
FILE *fd;
char *line;
{ SUBORDINATES : SIZE : 25 }

/* checks the format of the room number */
int chk_fmt_room_no(IN:room_no)
char *room_no;
{ SUBORDINATES : SIZE: 25 }

/* checks the format of the given course number */
int chk_fmt_course_no(IN:course_no)
char *course_no;
{ SUBORDINATES : SIZE : 25 }

/* checks the format of the given time slot */
int chk_fmt_time_slot(IN:time_slot)
char *time_slot;
{ SUBORDINATES : SIZE : 25 }

/* checks whether given SchCourses_index is a duplicate occurrence */
int chk_dup_sched_course(IN:SchCourses,SchCourses_index)
course SchCourses;
int SchCourses_index;
{ SUBORDINATES : SIZE : 25 }
```

3 Requirements Tracing

Here we list each major requirement and then list the modules in the structure chart that implement that requirement. The tracing shows that each requirement has been handled in the design:

1. No more than one course should be scheduled at the same time in the same room: sched_pg_pref, sched_ug_pref, sched_pg_no_pref, sched_ug_no_pref
2. The class capacity should be more than the expected enrollment of the course: sched_pg_pref, sched_ug_pref, sched_pg_no_pref, sched_ug_no_pref
3. Preference is given to PG courses over UG courses for scheduling: sched_pg_pref, sched_ug_pref, sched_pg_no_pref, sched_ug_no_pref
4. The courses should be scheduled so that the highest possible priority of an instructor is given: sched_pg_pref, sched_ug_pref, sched_pg_no_pref, sched_ug_no_pref

   If no priority is specified, any room and time can be assigned: sched_pg_no_pref, sched_ug_no_pref

5. If any priority is incorrect, it is to be discarded: get_pre_valid

6. No two PG courses should be scheduled at the same time: sched_pg_pref, sched_pg_no_pref

7. If no preference is specified for a course, the course should be scheduled in any manner that does not violate the preceding constraints: sched_pg_no_pref, sched_ug_no_pref

8. The validity of the data in input_file1 should be checked where possible: validate_file1 and its subordinates.

9. The data in input file2 should be checked for validity against the data provided in input file1: validate_file2

4 Design Analysis Using Information Flow Metrics

Based on the structure chart, the design of the system was first specified completely. We then analyzed the design using information flow metrics. Results of this analysis are given here. Based on this analysis, the design was modified—the final design which has been specified earlier, is an outcome of this modification. We first give portions of the intial design, particularly the parts that we changed as a result of the analysis.

```
/*************************************************************************/
/* Following are for reserving slots for PgNoPrefs courses. A 3-D linked list
 * is maintained. For each PgNoPref course (in the list of all courses), list
 * of time slots is kept. For each time slot, list of possible rooms is kept.
 * Using this structure, before allotting a UgPref course, it can be checked
 * if by allotting it we are making a PgNoPref course unallottable. If that
 * is the case, that allotment for the UG course is not done */

typedef struct R_Node /* list of indices into class room_db; contains rooms
                  with capacity greater than enrol of PgNoPref course */
{
      int r_index;
      struct R_Node  *next;
}r_node;

typedef struct T_Slot_Node  /* list of possible time slots for pg_no_prefs */
{
      int time_slot;
      r_node *roomlist;
      struct T_Slot_Node *next;
}t_slot_node;

typedef struct Pg_Res_Node             /* list of pg_no_pref courses */
{
      int pgcourse;
      t_slot_node *time_slots;
      struct Pg_Res_Node *next;
}pg_res_node;

/*-----------------------------------------*/
```

```
/* Schedule UG courses with preferences. This is not straightforward. Before
 * alloting a <timeslot, classroom>, it has to check if this allotment is "safe,"
 * i.e., the schedulability of PG courses with no prefs is not disturbed.
 * Does so by "simulating" scheduling of pg_no_pref courses - an allotment
 * is safe if the same no. can be still be scheduled after this allotment */

sched_ug_pref(IN:SchCourses,PgAlloc, OUT:ConflLst,ExplnLst, IN_OUT:TimeTable)
course *SchCourses;
int *PgAlloc;
error *ConflLst;
error *ExplnLst;
int **TimeTable;
{
    SUBORDINATES :get_room(),is_safe_allotment(),
                      initialize_pg_reserve(), update_pg_reserve();

    SIZE:100
}


/* Initializes the 3-D linked data structure - pg_reserve */
initialize_pg_reserve(IN:PgAlloc, SchCourses, OUT:pg_reserve)
int PgAlloc[];
course SchCourses[];
pg_res_node pg_reserve;
{
    SUBORDINATES :get_room();
    SIZE:50
}

/* Updates the PG reserve after a UG course with preference is allotted */
update_pg_reserve(IN:time,roomno,IN_OUT:pg_reserve)
int time,roomno;
pg_res_node pg_reserve;
{
    SUBORDINATES :
    SIZE:50
}


/* Checks if an allotment for a UG course with preference is safe, i.e.
 * the schedulability of PG courses with no preferences is not affected */

int is_safe_allotment(IN:time_slot,roomno,pg_reserve)
int time_slot,roomno;
pg_res_node pg_reserve;
{
    SUBORDINATES :
    SIZE: 80
}

/*****************************************************************************/
```

For analysis, we followed the approach of comparing modules of the design among them-

selves and then highlight the "error-prone" and "complex modules", as described in the book. In the case study we used the metric where complexity of a module is defined as $D_c = fan\_in * fan\_out + inflow * outflow$. The definition of error-prone and complex is as given in the book, except that we also use size for classification; the size of the module must also be above average or above (average + standard deviation) for it to be classified as complex or error prone. A locally developed tool called dmetric was used to extract the information flow metrics. The overall metrics for the design and the metrics for the modules highlighted by the tool are given here.

```
------------------------------------------------------------------------
OVERALL METRICS
------------------------------------------------------------------------
#modules: 35  Total size: 1330      Avg. size: 38   Std.Deviation: 27
Total complexity: 595         Avg. complexity: 17   Std.Deviation: 33

Deviation of the structure chart from a tree  = 0
(without considering leaves)

------------------------------------------------------------------------
 ERROR-PRONE MODULES
------------------------------------------------------------------------
8) sched_ug_pref
    call_in: 1   call_out: 4   inflow: 5   outflow:13   size:100
    design complexity:   69

------------------------------------------------------------------------
 COMPLEX  MODULES
------------------------------------------------------------------------
5) validate_file2
    call_in: 1   call_out: 4   inflow: 4   outflow: 8   size:100
    design complexity:   36

7) sched_pg_pref
    call_in: 1   call_out: 1   inflow: 1   outflow: 6   size:75
    design complexity:    7

13) is_safe_allotment
    call_in: 1   call_out: 0   inflow: 3   outflow: 1   size:80
    design complexity:    3

15) validate_classrooms
    call_in: 1   call_out: 5   inflow: 3   outflow: 7   size:80
    design complexity:   26

16) validate_dept_courses
    call_in: 1   call_out: 3   inflow: 2   outflow: 5   size:75
    design complexity:   13

17) validate_lec_times
    call_in: 1   call_out: 3   inflow: 2   outflow: 5   size:70
    design complexity:   13
------------------------------------------------------------------------
```

This data flow analysis clearly points out that the module to schedule UG courses with preferences is the most complex, with a complexity considerably higher than the average. It also shows that the overall structure is a tree (with a 0 deviation). Hence, we considered the structure to be all right. Based on this analysis, parts of the design dealing with scheduling of UG courses was re-examined in an effort to reduce complexity.

During analysis we observed that much of the complexity was due to the 3-D linked data structure being used for determining safety. Through discussions, we then developed a different approach for determining safety. The idea was that instead of using a separate data structure, before allocating a UG course, we will "simulate" the scheduling of the PgNoPref courses, using the regular function for scheduling these courses. If the number of courses the function sched_pg_no_prefs() returns is the same before and after the planned UG course scheduling, then the current allocation is safe. For this approach, we just have to make sure that is_safe_allotment() invokes sched_pg_no_prefs() with temporary data structures such that the actual timetable is not affected during this "simulation." The design was then modified to incorporate this approach. On analyzing the complexity again, we found that this approach reduced the complexity of the sched_ug_pref() module significantly and the complexity of this module was now similar to complexity of other modules. Overall, we considered the modified design satisfactory. The final design was given earlier.