Scanning and Processing of Forms

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology

by
Nishit Verma (96167) and
Susanta Kumar Nanda (96294)

to the
Department of Computer Science & Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
April, 2000

Certificate

Certified that the work contained in the report entitled "*Scanning and Processing of Forms*", by *Nishit Verma (96167) and Susanta Kumar Nanda (96294)* , has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Pankaj Jalote and Dr. Rajat Moona

April, 2000

Preface

This report primarily describes the scanning and processing of forms devised specially for use by the scanner and the scanning software. These forms may be used for a variety of purposes like entrance examinations, surveys etc. Again we describe the Interface called Twain which connects the scanner to the application programme.

Acknowledgements

We would like to thank our supervisors Dr. Rajat Moona and Dr. Pankaj Jalote for helping us at every step.We would like to thank them for helping us plan our project in a better way and the help given to device the language to describe the forms. Under their able guidance and support we were able to implement our Btech project. We would also like to thank our batchmates for the help and support they have given us.

Contents

Chapter 1

Introduction

The traditional form processing methods are being carried out with the use of an **OMR machine**. The method that it generally uses is that, the forms that are to be processed, have special black marks in both of its sides, and the OMR machine goes through line by line processing and gives out the bubbles which are filled. This machine is a costlier one and some skill is required to process this.

In this project we have made a sincere effort to come out with an alternative for this. The problem that this project addresses is towards devising a software for **processing** forms using a scanner and an application software written on top of it. Since scanner is much widely available and that its fairly cheaper when compared to OMR so we hope this can work as a good alternative. It uses **TWAIN** as the interfacing software for the application and the scanner. It assumes only three kinds of data to be processed in the form viz. **bubbles, barcodes, and pictures**. Since the automatic detection of these kinds of regions in a form is not a possible task, the values that they take is not previously known, and that not every part may be of the users' interest, this brings out the need of a well defined manner to specify the details of the form.

The project is carefully designed so as to handle any generic kind of forms with the constraint that the processing can be carried out only on the three types of data described earlier. Unlike OMR which is partially manual, this method is completely automated. However, for every different **kind** of forms it requires an appropriate way of specifying the details. Hence the need of a formal way of defining and specifying the things in the forms and the output sought, i.e. a **specification language**. This takes care of how to make the software understand the exact format of the forms and the various values that various things can take.

After specifying the details in the specification language, the software scans the forms, and connects to the rest of the image processing part using the twain interface. Finally, using the data provided in the specification file, the outputs are written and saved into some file.

Chapter 2

Specification Language

Before scanning any kind of form this software needs to know the general format of the form, the kind of data involved in it, the exact location of the data, the values, the outputs required, and the filenames where the outputs are to be stored. All these are provided to the software in a prescribed format which we call as the *Specification Language*. The specification language should to be simple, designed so as to catch each and every aspect of the forms and should be generic enough to represent any such kind of forms that one usually comes across.

The specification file that is used as an input to the software, describes the following things

- **Global Allignment Mark**:
  - The coordinates of the *allignment mark* w.r.t. the bottom-left corner of the form must be specified.
  - The marker line lengths must be specified
  - The marker line thickness must be specified
- **Region Descriptions**: The size, location and types of regions alongwith some region specific attributes (such as the bubble size in case of the bubble-regions, barcode format in barcode regions etc).

  - Alignment with respect to G.A.M
  - Length and width of the region
  - If Bubble region then cell size
  - If Image region then Colortype , Resolution .
  - If Barcode region then nem of the output variable

- **Variables**: These are used to group certain chunks of regions (might be bubbles or anything else) which has some logical associations between them. The specification language also allows recursive use of variables (though not essential) which, in some cases help to group certain values better. The *language* also allows to use indices on the variables to help specifying a common set of bubbles. The things needed to be specified are –
  - The list of the cell coordinates of each variable
  - the corresponding values for those variables
  - It also needs to be specified what will happen if no bubble is filled or more than one bubble is filled .

- **Output Variables**: These are some special kind of variables whose values are required to be output and stored in some format in the files specified. So these are, in some sense, *variables* as above with some side effects.

- **Bubble-Value Map**: These are used to describe a variable and specifies the value of the varible depending on how the form is filled.

2.1 An Example File

```
# Global Alignment mark (7mm, 77mm ) , 181.5 mm , 77.5 mm , 1.4 mm , 1.4 mm
# wrt bottom left


   # Region definitions
   BUBBLE : R1 : P1
( 9 mm , 218 mm ) # Alignment wrt GAM

   ( 149 mm , 84 mm ) # Region height and length
( 4.257 mm , 3.231 mm ) ;# Cell width and Height
   BUBBLE : R2 : P1
( 58 mm , 115.4 mm )

   ( 35 mm , 39.5 mm )
( 4.257 mm , 3.231 mm ); # Cell width and Height
BARCODE : R3 : P1
(13.5 mm , 85 mm )
(12.5 mm , 31.5 mm )
BT
INTERLEAVE2OF5
=> F[1 : 8] ;
PICTURE : R4
( 5.8 mm , -0.9 mm )
( 35 mm , 45 mm )
DPI : 100
MONO8 GIF ;
Name = N[1:35]
N[i] = (i,1):(i,26) =>  'A' : 'Z'  @R1
DOB = Date , Month , Year
Date = D1 , D2
```

```
D1 = (1,2) : ( 1,5 ) => '0','1','2','3' @R2
D2 = (2,2) : ( 2,10 ) => '0' : '9' @R2
Month = (5,1) : (5,12 ) => "JAN" , "FEB" , "MAR" , "APR" , "MAY" , "JUN"
,
"JUL" , "AUG" , "SEP" , "OCT" , "NOV" , "DEC"
@R2
Year = Y1 , Y2
Y1 = (7,8):(7,11) => '6':'9' @R2
Y2 = (8,2):(8,11) => '0':'9'@R2
OUTPUT1 = Name , DOB => "data.txt"
```

2.2 Description

The specification file always begins with the global alignment mark . The global alignment mark is the intersection of the horizontal marker line and the vertical marker line . The first point desribes the x and y coordinates of the global alignment mark with respect to the bottom left corner . The rest of the terms describe the length and the thickness os the horizontal and the vertical marker lines .
After the description of the Global alignment mark the region desriptions follow . Before describing the region one needs to tell what type of region is it - BUBBLE , IMAGE OR BARCODE . The region R1 above is a bubble region . For a bubble region like R1 three things must be specified the alignmentof the region with respect to the G.A.M , the region width and the region height and the cell width and the cell height .
While specifying all the lengths the dimension is also specified with them . For eg it can be written mm,cm,inch etc . After the bubble region R1 comes the bubble region R2 .
After describing R2 the barcode region R3 is described . The barcode region description contains the coordinates of the origin of the barcode i.e bottom left corner of the barcode . After specifying the origin coordinates with respect to the G.A.M the length and width of the barcode is specified . Again we need to tell the scan

direction whether from Bottom to Top (BT) , Top to bottom (TB), Left to right (LR) , Right to left (RL) . After this the type of barcode is specified . Note that the type can be any of the types CODABAR , UPC , INTERLEAVE2OF5 etc . The arrow specifies the name of the output variable to which the barcode string should correspond to . Note that all the characters of the barcode string need not be used .

Finally the picture region R4 is specified . Again the first point is alignment with respect to the G.A.M. . After this the height and width of the image region is specified . After specifying these attributes the DPI at which the image should be scanned , the bits per pixel resolution of the image which can be MONO1,MONO8,COLOR8,COLOR16,COLOR24,COLOR32 is specified . After this the format in which the image is saved is specified . The format can be GIF , JPG, PPM ,BMP etc .

Now comes the turn of the exact specification of the bubble variables . Here the first variable Name is an array . The actual variable N varies from 1 to 35 . Note the coordinates specified are always in term of the cell coordinates of the region . The presence of a colon ":" indicates stepwise variation while indicating the coordinates . For example here N[i] = (i,1):(i,26) => 'A':'Z' @R1 indiactes that the i'th value of the string name can be any one of the cell coordinates (i,1) to (i,26) whichever is darkened . The right hand side of the arrow tells the values which the filled bubbles would take . The right hand side varies from A to Z . The value @R1 indiactes that the variable belongs to the region R1 . The second bubble variable DOB (Date of birth is composed of a variety of other variables , Date month and Year in this case . Note that the composition variables may or may not belong to the same region . Date is again composed of two variables D1 and D2 . Similarly year is composed of variables y1 and y2 .

Now come the output list . The output variable output 1 consisits of name and DOB . The output should go in the file specified after the arrow.

Chapter 3

TWAIN: The Scanner Interface

The scanner software uses the Twain interface to connect with the real scanner . For this a dyanamic link library Twain32.dll is used .

- The User Interface to Twain When an application uses Twain to acquire data , The acquisition process may be visible to the applications users in the following three ways.
  - The Application . The user needs to select the device from which they intend to acquire the data . They also need to signal when they are ready to have the data transferred .
  - The source manager . When the user chooses the select source option the application requests that the source manager display it's select source dialog box . This lists all available devices and allows the user to highlight and select one device . If desired the application can write it's own version of this interface .
  - The source . Every TWAIN compliant source provides a user interface specific to it's particular device . When the application user selects the acquire option , the source's user interface may be displayed . If desired the application can write it's own version of this interface , too .
- Communication between the elements of Twain . communication between the elements of twain is possible through two entry points . They are called

DSM-Entry and DS-Entry

- The application The goal of the application is to acquire data from a source . However the application cannot contact the source directly . All requests for data , capability information error information etc must be handled through the source manager . The application communicates to the source manager through the source managers only entry point , the DSM-Entry function . The parameter list of the DSM-Entry function contains
  - An identifier structure providing information about the application that originated the function call .
  - The destination of this request .
  - A triplet that describes the requested operation .
  - A pointer field to allow the transfer of data .
- The source manager The source manager provides the communication path between the application and the source , supports the users selection of the source and loads the source for access by the application . Communications from application to the source manager arrive in the DSM-Entry point.
  - If the destination is the source manager then the source manager processes the operation itself .
  - If the destination is the source , the source manager translates the parameter list of the information , removes the destination parameters and calls the appropriate source . To reach the source , the source manager calls the source's DS-Entry function . Twain requires each source to have this entry point .
- The source The source receieves operations either from the application via the source manager or the source manager itself . It processes the request and returns the appropriate return code . (The codes are prefixed with TWRC) indicating the results of the operation to the source manager . This return code is then passed back to the application as the return

value of its DSM-Entry function call . If the operation was unsuccesful , a condition code (the codes are prefixed with TWCC) containing more specific information is set by the source . Although the condition code is set , it is not automatically passed back . The application must invoke an operation to inquire abpout the contents of the condition code .

  - Communication flowing from the source to tyhe application . A majority of the operation requets are initiated by the application and flow to the source manager and the source . The source via the source manager is able to pass back data and Return Codes . However there are two times when the source needs to interrupt the application and request that an action occur . These situations are –
    - Notify the application that a data transfer is ready to occur . The time required for a source to prepare data for a transfer will vary . Rather than have the application wait for the preparation to be complete the Source just notifies it when everything is ready . The MSG-XFERREADY notice is used for this purpose .
    - Request that the source's user interface be disabled . This notification should be sent by the source to the application when the user clicks on the close button of the source's user interface . The MSG-CLOSEDREQ notice is used for this purpose .

- Twain States
  - twain state 1 Pre Session Source Manager not loaded Application asks to load the source manager .
  - twain state 2 Source manager is loaded . Get the entry point . At this point the source manager is ready to accept the operation triplets from the application .
  - twain state 3 Source manager is open . List all the sources and detect the source . The source manager will remain in states 5 to 3 until it is closed .

  - twain state 4 Source is open . Perform Capability Negotiations . The source should have verified that sufficient resources exist for it to run . The application can inquire about the source's capabilities (i.e. levels of resolution , support of color or black and white images , automatic document feeder available etc . The application can also set these capabilities to it's desired settings . For example it may restrict a source transferring color images to providing black and white only .
  - twain state 5 Source is enabled . Source will show it's user interface. When scan is over it will shut down .
  - twain state 6 Transfer ready . Application will inquire about the image information . It is possible for more than one image to transfer .
  - twain state 7 Transferring . Source is transferring data in one of the three modes - native mode , disk file transfer ,buffered memory . THe transfer will either complete succesfully or terminate prematurely . The source sends the appropriate return code indicating the outcome .

- twain capabilities
- some of the twain capabilities are
  - Some device shave automatic document feeders
  - Some devices are not limited to one image but can transfer multiple images .
  - Some devices support color images
  - Some devices offer a variety of halftone patterns .
  - Some devices offer a range of resolutions while others may offer other options .
- Capability Negotiation The following general process is followed
  - Determine if the selected source supports a particular capability.

  - Inquire about the current value for this capability . Also inquire about the capability's default value and the set of Available values that are supported by the source for that capability .
  - Request that the Source set the current value to the applications desired value . The current value will be displayed as the current selection in the ource's user interface .
  - Limit , if needed the source's available values to a subset of what would normally be offered . For instance , if the application wants only black and white data , it can restrict the source to transmit only that .
  - Verify that the new values have been accepted by the source .

- Modes available for file transfer
  - Native : Every source must support this transfer mode . On windows the format of the data is DIB (Device Independent Bitmap)
  - A source is not required to support this mode but it is recommended. The application creates the file to be used in the transfer and ensures that it is accesible by the source for reading and writing .
  - Every source must support this transfer mode . The transfer occurs through memory using one or more buffers . Memory for the buffers are allocated and deallocated by the application .

- Twain capabilities are divided into two groups .
  - CAP-xxxx Capabilities whose names begin with cap are capabilities that could apply to any general source . Such capabilities include use of automatic document feeders , identification of the creator of data etc .
  - ICAP-xxxx Capabilities whose names begin with ICAP are capabilities that apply to image devices . The I stands for image .

- capabilities exist in many varieties but all have a default value , current value and many other values that can be supported if selected . To help categorize the supported values into clear structures TWAIN defines four types of containers for the capabilities

- TW-ONEVALUE A single value whose current and default values are coincident . The range of available values for this type of capability is simply this single value . For example a capability that indicates the presence of a document feeder could be of this type .
- TW-ARRAY A rectangular array of values that describe a logical item . It is similar to the TW-ONEVALUE because the current and default values are the same and there are no other values to select from . For example a list of the names such as the supported capabilities list returned by the CAP-SUPPORTEDCAPS capability would use this type of container .
- TW-RANGE Many capabilities allow the user to select their current value from a range of regularly spaced values . The capability can specify the minimum and maximum acceptable values and the incremental step size between values . For example resolution might be supported from 100 to 600 in steps of 50 .
- TW-ENUMERATION This is the most general type because it defines a list of values from which the current values can be chosen . The values do not progress uniformly through a range and there is no consistent step size between the values . For example if a source's resolution options did not occur in even step sizes then an enumeration would be used .

Chapter 4

Form processing

The project assumes that the form has three kinds of data of interest, which are required to be processed. These three kinds of processable data are **bubbles, barcodes, and pictures**. Now, in order to process these data, they need to be scanned carefully considering every bit of details such as, their exact position in the form, size, values and so on. These data are initialized by retrieving them from the specified file. Once these are data are initialized, our work is much simpler.

After the form is scanned, then the TWAIN interface returns a handle of the image file of the form in **DIB** (device independent bitmap) format to the software. Once the file handle is transferred we carry out rest of the processing by carrying out some careful calculations and some image processing using the data in the specified file. We give some of the details for every part of the processing below this.

4.1 Bubble Processing:

This is an interesting part of our form processing. A traditional form contains a huge number of bubbles, e.g. names, biodata, examination details, other personal details, and so on. Here every bubble has a specified location as well as value. In addtion to that not every bubbles are independent. By this we mean, there may be some group of bubbles from which only one can be filled at a time, or some other constraints. However, these are specified in the specification file. So, what we do is that, go through a region of bubbles one at a time, find it exact location, and then try to process that bubble.

To find the exact location, and procees it we use the data from the specification file and go through the following procedures.

Calulate the offset of the bubble region by using its offset w.r.t. the global allignment mark (GAM) and the coordinates of the GAM. By using the cell coordinates of the particular cell in that region and the exact cell size, it is fairly easy to compute the x and y bounds of the cell. Now the important point to make here is that the cell size should also take care of the gaps between the cells, since this can cause a large amount of shift of the cell in both the directions. After finding the exact bounds of the cells the next thing we need to do is to find out the pixel values of all the pixels that lie inside that cell. For this we use several predefined variables and functions available for image files in DIB format. By using some thresholds for detecting the color and to detect whether the cell is filled we can decide the values of the cells.

4.2 Barcode:

A barcode is a kind of graphic representation of a number. It consists of a sequence of bars. The colors of bars are black or white, and they can be thin or thick. So, in this way, a bar code has a sequence of characters which are either of four types, i.e. thin white, thick white, thin black, and thick black. Typically the thick ones are two to three times wider than the thin lines. Now depending on these character sequence the number that the barcode represents, is calculated. Now, since there can be many interpretations to a particular sequence of bits or characters, hence are the variety of barcode formats. However, there are some special kinds of barcode formats that have international recognition. Some of them are, INTERLEAVED 2 OF 5, CODABAR 39, several versions of UPC (Universal Product Code) and many more.

Though we went through many of the recognized formats that are commercially available to decode in this project, but the forms that we came across had only

Interleaved 2 of 5 representations of the barcodes. In addition to that a barcode can have some additional check bits along with its start and end code as well.

- **Interleave 2 of 5:** As the name suggests, it interleaves bits that represent one character into another. Here four thin lines (two each of black and white) are used as the start code. Then one thick black followed by two thin lines (one black and one white) are used to represent the end code. The rest of the bits in the middle all represent numbers of the barcode. It uses five bits to represent a single digit. A thin line is considered to have the bit-value zero, whereas a thick line has the bit-value one. In this format, th numbers of digits that can be represented has to be even. If one has to represent a number having odd number of digits, then the simplest solution is to add a zero at the front or ignore the last digit. Since any barcode has to have the sequence in alternate white and black fashion, this brings the idea of interleaving. So what this format does is that, it uses first five black lines for the representation of the first digit. The interleaving five white lines represent the next digit. After a couple of digits, it starts again like this. The interpretation of the five digits to get the value of the digit are predefined and attached in the appendix.

- **Processing:** With this background given above, it is not much difficult to process a barcode of the above format and calculate the number it represents. The method that we are using is extremely simple. We just count the width (in number of pixels) of each of the bars (i.e. the count restarts when the color of the bar changes). After that we find out the minimum and maximum widths of both black as well as white lines. The average of these values work as a threshold for detecting the thin and thick lines. After detecting this, rest of our job is to extract the bit strings and find out the digit looking up the table. For better accuracy, we need to take care of the start and end white pixels which might occur due to wrong measurement.

4.3 Picture:

The picture regions are those which contain signatures or passport size photographs or say, any comments etc. So there is not much processing is needed about these except that they might be required to be saved into some user defined filename. So here what we do is, calculate the exact offset of the image (or picture) w.r.t. the GAM and then calculate the global offset of its left bottom point w.r.t. the form. Then using the size given in the specification file, and the calculation of the exact offset we get the pixels and then store it in the file, if required.

However, in each of these we did not care much about the output filename and just assumed that it is easily available from some source and used to store the data. But this might not be fixed or planned before. This filename might as well depend upon the data those are processed in the form itself, like say barcode. And this is really the case needed in many places with justified reasons because every form normally has a unique identity number which is printed by the help of a barcode. And since it is unique, so a user might be really interested in using this number or parts of it to name the files that are storing the data of the form. And that is what we describe in our next section.

4.4 Output:

The output are stored in some file. The filename are generally specified by the user while specifying the output details. In our specification format we allow it to be a concatenation of several strings which are not know prior to the processing of the form. They can include,

- **Constant Strings:** The file name can have a string constant that is known a priory, e.g. ”DATA.DAT” or whatever string.
- **User Input Strings:** The string size may be known before, but not the exact string. While naming the file, it can be asked to the user to input some string, which is then taken as the part of the name of the file. In our specification

model this is specified as a dollar variable, e.g. $[5] is a user input string of size 5.

- **Barcode dependent Strings:** In many a cases the part of the key barcode that appears in the form might be needed to use as a part of the file name. This can be useful because the key barcode is unique to a particular form. So the file name can uniquely identify its association with the particular form.

Chapter 5

Implementation

The project implements a hardware software interface. It interfaces a scanner to an application programme with the interfacing software as TWAIN. The overall software module can be shown by connecting some black boxes and defining their functionalities and their inputs and outputs. Once we connect all the boxes with their specified details, we can get a feel of the overall software with their detailed control flow paths. In the next few sections we describe all the modules in details hinting about their functionalities.

5.1 Overall Module

Figure 1 shows a rough diagram of what the overall software module looks like. The basic structure of the software can be thought of as having three parts, the *input module*, the *main module,* and the *output module.* The input module deals with the specification files and intializes the data structures by reading the values specified in the specification file. This is basically sets the input for the software to work. That is, it lets the software know every details of the form that it is going to process, what are the outputs required, and everything that it needs to know before going to scan the form and process it.

The main module is sort of central part of the software and it works as a bridge between the input and the output modules. It initializes the scanner and sets its

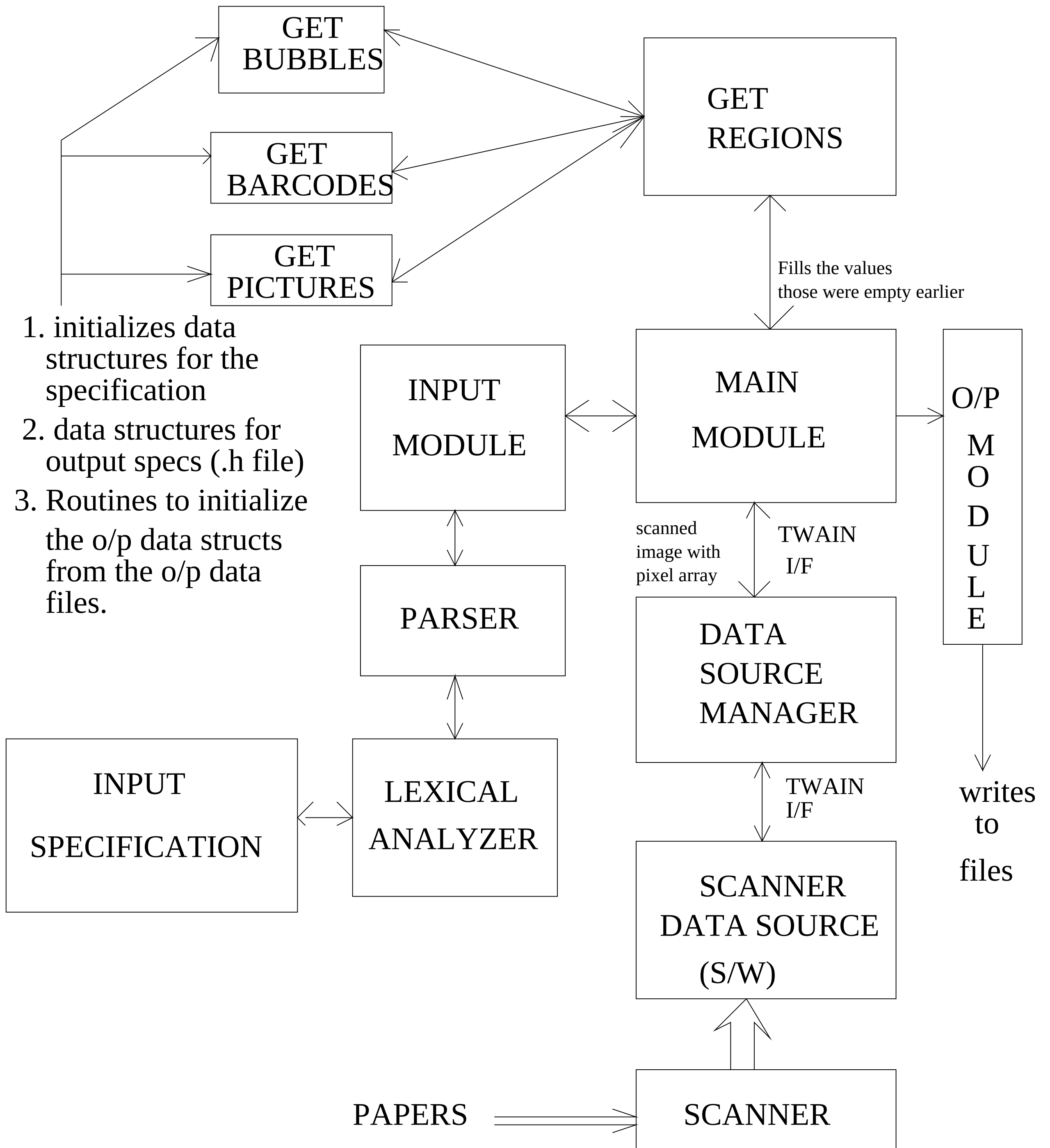
GET BUBBLES
GET BARCODES
GET PICTURES
GET REGIONS
Fills the values those were empty earlier
1. initializes data structures for the specification
2. data structures for output specs (.h file)
3. Routines to initialize the o/p data structs from the o/p data files.
INPUT MODULE
MAIN MODULE
O/P MODULE
PARSER
scanned image with pixel array
TWAIN I/F
DATA SOURCE MANAGER
INPUT SPECIFICATION
LEXICAL ANALYZER
TWAIN I/F
writes to files
SCANNER DATA SOURCE (S/W)
PAPERS
SCANNER

Figure 1: Overall Software Module

verious values, then scans the image and transfers the handle of the scanned image of the form in DIB format to the next step. The next step is where all the processing begins. This processes all the regions separately and gets their values depending on what are actually needed in the output (this can be known from the output format given in the specification file).

After the works done by the above modules, the job of the output module is pretty simple. It just does some processing using the values received after the processing in the main module and gets the filename where all the outputs are to be saved. And then just just saves the values into the specified file. With this kind of overview to all the modules we are now in a proper shape to go into the details of each of the modules. The next three sections provide the details of each of them.

5.2 Input Module

The *input module* functions as the initializer for the software with certain added routines. It takes input as the specification file that has been designed by the user to specify all the details of the form that is going to be processed. In a way, the importance of this module is that if anything goes wrong with this, the whole software fails, i.e. the processing does not come out with the accurate result. So while writing the specification file, every care should be taken to avoid any kind of errors whatsoever. There can be no error checking mechanisms incorporated here excepting some in this module, as the software has no prior information about the form that it is going to process, neither can it assume any.

After it gets the input specification file, it first runs a lexical analyzer on the specification file that recognises and then passes the tokens to the next phase. The next phase is a parser which parses these tokens according to the grammars of the specifications that are allowed. However, there are certain obvious constraints imposed on the specifications. Before using any variables corresponding to a region, user should make sure that the region is defined with all its details (refer chapter 2).

Every variable that are used in the specification file must be defined earlier or later. Our parser does not impose any condition on the definition of a variable prior to its use. As long as every variable that are used are also defined at some point of time all is well. However, an ouput variable is always necessary in a specification file as otherwise there will be no output and that means no processing needed. The global allignment mark is also compulsory for the file to have. So, with these constraints the parser parses the specification file. After this phase the initialization of all the data structures of the software begins. There are some global data structures maintained in this software that are required in every part of the project and which are initialized in this phase. We will go through a rough introduction on what kind of data structures used in this part.

- **Global Allignment Mark:** This stores the coordinates of the global allignment point, the approximate lengths of the horizontal and vertical lines intersecting at the GAM, and their approximate widths. These are used to allign the picture properly and find out the coordinates of all the other parts later.
- **Region Table:** This data structure is a generic one so as to store the data of each kinds of regions i.e. bubble, barcode and picture, along with all their details. So while processing a particular region, the details can be extracted from this data structure.
- **Symbol Table:** There is also a global symbol table maintained which keeps track of all the identifiers or symbols that are defined by that point of time in the specification file. This take care of the errors like redefinition of a variable, using a variable without ever defining it in the file etc. Along with storing the identifiers it also stores a pointer to the actual node which contains all the detailed description and defintion of that variable. The details of a variable are stored in its node in a member called components. The components are a list of one of the following kind of nodes,
  - **Variable:** This is a case of recursive definition of variables. That is, we define variable in terms of other variables. If these variables are not

defined earlier, they need to be defined later.

  - **Constant:** In this case again it can be a string constant or a character constant or any other types, if necessary.
  - **Coordinate to Value Map:** This is one of the termination criteria for the recursive variable definitions. The variable can have a value depending on the exact cell among a group of cells in a region which is filled. For this one needs to specify the list of cell coordinates in a region which together constitute an exclusive set and only one need to be filled out of them. After specifying the coordinates one also needs to specify the corresponding values that the coordinates will take if they are filled. So, this data structure stores these informations in it. This helps in retrieving the exact value tat the variable is going to take while that region is processed.
  - **Expression:** Though this type does not have of much use, but however it plays an important role if we need to store a value of a variable which is an expression, which is to be processed at the run time. Currently, we do not exploit much of its usage.

- **Output Table:** The output table is a data structure which stores the list of all the output variables. These output variables are like any other kind of variables with a side effect. The side effect is that the value of the output variable gets stored into a file. Besides this difference this table is more or less same as the symbol table. The node of this table contains a variable name and stores the description of the variable, and the description of the file name into which the value of the variable will be written. The file name may not be a straight forward one and may need some processing which is done afterwards.

After the initializations to these above defined data structures are over, we need to provide something else. That is, provision of some definitions and routines to access the output variables. For this we provide two more files. The first one is "spec.h", which contains all explicit definitions and declarations of the data structures related to output variables covering all the structures which might be used in

the structures of output variables.
The next one is a file called "init.c" which contains a set of routines to access and initialize these output data structures (those declared in the spec.h file) if the values are made available in a particular format in some file. So using these two files one can easily use the values received by the processing module in case any post-processing is desired. Now that we discussed some of the details of this module, we can move to our next part, where really the processing takes place.

5.3 Main Module

This is so to say the heart of the project, as the name suggests. It connects all the modules and organizes the data flow of the system. After the input module is done with its initialization part and the files are generated, then its the form is to be processed.

First of all the scanner is initialized. In the main function first the scanner is initialized. Once the scanner is initialized it performs all the operations necessary for image transfer. It negotiates for the capabilities, displays the source manager's user interface and then once the source is selected performs the actual image transfer. The handle of the image in Device Independent Bitmap format is returned.

After this a Scan-Card class is initialized. The scan-card class contains all the detailed information about the barcodes, images etc. After initializing the scan-card the function extract card is called which extracts the barcode, bubble and the picture values. If save scan-card is called after that the barcode, images are saved in the files specified.

- **Scanner:** First of all the constructor for the scanner is called. In this constructot there is the function which takes the scanner from twain state 1 to twain state 4. The following steps are taken. First of all the Data Source Manager is loaded then the operation open Data Source Manager is performed. After this the user selects a particular Source. After selecting a particular source, it is opened. After the source is opened capability negotiation is performed and default capabilities are set. Some of the default capabilities which are set are

DPI, XimageResolutuion, YImageResolution, Bits per pixel count etc. Once in state 4 the scanner waits for the data. After performing all the capability negotiations, the function transfer scan is called. It goes from Twain state 4 to 7 and comes back to state 4. In this function the actual image transfer takes place. The mode o image transfer here is the native mode, i.e. a handle of the Device Independent Bitmap is passed after the actual scanning has taken place . Note if there is some error while transferring, the function aborttransfer is called which results in the twain-state going back to 4. All the capability negotiations and the twain operations have a return-code. If this return code indicates error, then the scanner goes into twain state

- **Scancard:** The scancard is the data structure which contains the result of a single scan. It contains an array of images, an array of barcodes, their offsets with respect to the global alignment mark, their width and their height. While calling the constructor all these items are initialized. For the initializations of these items there respective constructors are called. Note that there are separate image and barcode classes. Once the scancard is initialized, it's handle is set to the handle which was passed in transferscan. The dpi of the scancard is also set to the dpi at which the image was scanned.

- **Locate and Extract ScanCard:** This is the place where we do all the image processing part. After retrieving the data from Global Allignment Mark, it calls a routine ExtractScanCard. This routine retrieves the offsets data from each of the regions and then processes each of these by calling several other routines. First, the barcodes images are extracted from the full scan card. By extraction we mean, a new image handle is created from the original one but the new handle contains only the image of the concerned portion. To do this, first the height and width of the image (to be extracted) are calculated in terms of number of pixels. After that starting from the offset of the region the particular image is extracted and the handle of the image is returned. Then the rest of the image processing for this region is carried out on this extracted image rather than doing it on the whole image. The details of the image processing of these regions are described in chapter 4.

5.4 Output Module

After the image processing part is over for these regions, the next step is to produce the output. For this we simply travel through the output list that had been created during the initialization process in the input module. Then for each output node, we go through the details of its components that need to be written. Then we go through the filename portion. The main processing involved in this module is to process and find out the exact filename in which the outputs are to written.

To find out the complete filename, we need to travel through the whole list of nodes those describe it, and process each one of them separately and finally concatenate the result. To process each of the nodes, we might need to retrieve data from the barcodes or might require to ask for user input etc. After getting the filename, the outputs are written and saved to the file.

Chapter 6

Conclusions

At the end of this project, we have learnt a lot about how to process forms using the scanner . We had also to implement the processing of course evaluation forms of the ESC101 , the forms have been designed by us but the final processing could not be implemented .This happened because the ESC101 survey could not be done on the special forms designed by us , as we could not get the specially printed copy of the forms due to lack of time . However we are able to scan the JEE and thei GATE forms . Another important fact to be kept in mind while running our software is that the specifications have to be very exact . Since the bubble size is 3 mm by 4 mm even a mistake of 1 or 2 mm in measuring can result in wrong processing of data . A remedy for giving the bubble sizes is that first we note the whole region width , region length and then divide the region area by the total number of bubbles in that area . There have been some inefficiencies somewhere but overall the project is working . One constraint is that different scanners allow different capabilities to be set . For example some scanners do not allow the DPI to be set less than a particular value .

Overall we spent a lot of effort in designing the language , parser , the scanner interface and the output module .Ou guides have also been quite helpful and generous to us .

Appendix A

Functional Specifications

- **main.cpp**
  This is the file which contains the main function . In the main function first the scanner is initialized . Once the scanner is initialized it performs all the operations necessary for image transfer . It negotiates for the capabilities , displays the source manager's user interface and then once the source is selected performs the actual image transfer . The handle of the image in Device Independent Bitmap format is returned . After getting the image information the parser is called . The parser takes as argument the specification file passed as the second argument of the main function . The parser initializes the data structures while parsing the specification file . After this a Scan-Card class is initialized . The scan-card class contains all the detailed information about the barcodes , images etc . After initializing the scan-card the function extract card is called which extracts the barcode , bubble and the picture values . If save scan-card is called after that the barcode , images are saved in the files specified .

- **Class-Barcode.cpp** The file contains the following things -
  - Constructor for the Class It initializes certain information about the barcode like offset w.r.t G.A.M , its length , its height etc.
  - Destructor for the Class . It performs the usual destruction operation.

  - - IT deallocates i.e. frees memory space reserved for that particular barcode but it is not a detructor i.e. it does not destroy the class itself .
  - Interpretbarcode This function first performs the check whether the barcode is valid or invalid . If the barcode is invalid , it calls the function reset-barcodeinterpretation which resets the information containe din the barcode . It initializes the variables bits (Which contains the barcode bits) and the variable Barcodestring which contains the whole barcodestring.
  - ExtractBarcode It rotates the subbitmap which contains the barcode , sets the handle in the barcode to the rotated bitmap and then calls the function which computes the barcode value .
  - GetBarcodeHandle Returns the image handle of the barcode
  - SaveBarCodeBMP Saves the barcode only in BMP format in the specified file and the directory .
  - SaveBarcodeGIF saves the barcode in GIF format in the specified directory and the file .
  - PrintBarcode It prints all the information related to the barcode , it's offsets , barcode value , bitstring , barcode bits etc .

- **Barcode.cpp** This is a file which contains function related to the barcode but accessible everywhere .
  - Invalidbarcodetype returns true if the barcodetype is invalid i.e. does not belong to one of i2of5 , codabar , upc .
  - GetNumber-I2OF5 returns the number corresponding to the i2of5 barcode given the start index and the interleave factor .
  - GetBarcodeString Given the barcodebits , it returns the barcode string depending on the type of barcode .
  - GetBarcodeBits This function returns the barcode bits . It reads the barcode image passed through the handle . The barcode bits are labelled as follows

        * thin white 0
        * thin black 1
        * thick white 2
        * thick black 3

- **bbl.cpp** This file contains all the functions necessary for bubble processing .
    - GetFileName Gives the file name where the output is to be stored given a node to the outputlist .
    - Save-bbl-op This function traverses the outputlist and for each member of the outputlist saves the bubble output .
    - gen-bbl-values This function generates and saves the bubble values for a particular variable .
    - Read-cell This function returns 1 if the cell is completely black .
- **Class-scancard.cpp** This file contains the following functions
    - Constructor Initializes the scancard . Initializes the variables like the number of barcodes , number of images , An array containing barcodes , an array containing images . This function calls the constructor for the barcode and the constructor for images .
    - Destructor Performs the normal destruction function for a class.
    - IsInvalid Returns whether the scancard is valid or not .
    - Reinitialize Initializes the scancard again .
    - SetId Sets the id of the scancard . The id depends mainly on the keybarcode . Here in this function id is passed as a string .
    - SetHandle Sets the handle of the scancard to the passed handle . Also sets the dpi of the scancard to the passed DPI .
    - GetId Returns the id of the scancard .
    - ExtractScanCard This function performs the actual extraction of the scancard given it's handle . The offsets of image , barcode and the bubble

region in the scancard are already known . Here the handle of the image region and the barcode region is also set . The handle of barcode and image points to the subregion which contains these barcodes and images respectively . After doing the computations necessary for images and barcodes , the bubble region computations are performed .

  – LocateandExtractCard This function is called from the main module itself . This function calls the function which calculates the rotation angle of the scancard . After calculating the rotation angle , it calls the function Extractscancard .
  – PrintScancrad Prints the current status of the scancard along with information about variables like barcodes , images in it .
  – SaveScancard Saves the parts which can be saved in the directory specified .
  – Traverseregionlist Traverse the region list and find out how many barcodes and images are there .

- **Class-scanner.cpp**
  – Constructor Normal constructor for the class . In this constructor Twain state is changed from 1 to 4 .
  – Destructor Normal destructor .Twain state is restored to 1 .
  – IsInValid Tells whether the scanner is valid or invalid .
  – LoadDataSrcMgr Loads the Data Source manager . The twain state changes from 1 to 2 .
  – OpenDataSrcMgr Opens the data source manager after loading it . The current twain state is 2 .
  – SelectSrc Select the source .
  – OPenSrc Open the source .
  – CreateCapstructure Create a capabilty structure .

- SetDefaultCap Set all the values for the default capabilities like DPI , image resolution in the X direction , image resolution in the y direction , bits per pixel count etc .
- SetCapCurrOneValue Set the value for the passed capability .
- Getcapvalues Get all the values of the default capabilities .
- GetCapCurrOneValue Get the current value of the capability passed .
- IfCapCanBeSet Returns true if the capability passed can be set .
- EnableSrc Enable the source .
- AlterEventLoop Start in state 5 and end in state 4 .
- ImageNativeXfer Start in state 6 and end in state 5 if succesful . Perform the actual image transfer in the native mode .
- EndTransfer Start in state6/7 and end in state 5 if successful .
- AbortTransfer Stop the transfer in between . Start in state 6 and end in state 5 if succesful .
- DisableSrc Disable the source .
- CloseSrc Close the source .
- CloseDataSrcMgr Close the data source manager .
- UnloadDataScMgr Unload the Data Source Manager .
- PrintFailureCause Every twain operation returns a status code . Reading this status code print the cause of failure if any of a particular twain operation .
- GoFromTwainstate1to4 Perform the requisite operations like loadDataSrcManager , openDataSrcManeger , selectsource , opensource , setdefaultcapbilities and change the twain state fro 1 to 4.
- GetDPI Get the current DPI .
- GetTwainState Return the current twain state .
- TransferScan Transfer the scan handle . The input twain state is 4 . Enable the surce and go into an event loop .

  - GoFromTwainstate4to1 Perform the requisite operations like closedatasrcmanager , closedatasrc and then go to twain state 1.
  - Gototwainstate1 Free the twain Dyanamic link library and go to twain state1 .

- **lass-Image.cpp**
  - Constructor Performs the usual operation for the constructor of a class. Initializes certain information like width , length , offsets etc .
  - Destructor Usual destructor for the class .
  - Reinitialize Reinitailizes the whole object .
  - ExtractImage Performs the image exttraction operation . This is only setting the handle to the area which contains this particular image .
  - PrintImage Print the information relating to the image .
  - GetImageHandle Return the handle of the image .
  - SaveImageGIF save the image in GIF format .
  - SaveImageBMP save the image in BMP format .

- **rotate-bmp.cpp**
  - GetRotatedBitMap Most of the complexity of this function is due to the fact that DIBs are organized differently depending on the number of colors it uses. There are two main attributes of a DIB that affect the flow of control . The first is the number of bits used to specify a pixel. This affects how the color information is read and written. The second attribute is the compression. This function does not support run length encoded DIBs. The actual rotation involves using a reverse transform. That is, for each destination pixel we determine the source pixel that should be copied there. The reason for this is that the straight transform will leave small blank spots. The main stuff here is getting and setting the color information. For bitmaps with 4 bits per pixel, we again have to deal with bitmasks. In this case the 4 most significant bits specify the

pixel on the left. When the bits per pixel is 8, 16, 24 or 32 we copy 1,2,3 and 4 bytes respectively. Also, when the bits per pixel is more than 8 and the destination pixel does not correspond to any of the source pixel, then we set it to the background color.

  - GetRotatedsubbitmap This function is similar to the above function . It differs only in the case that here the bitmap to be rotated is just a rectangle in the original bitmap .
  - ReducedDPIBitmap Create a new bitmap with reduced DPI image in the original bitmap .
  - ReducedDPIBitmap Create a new bitmap with reduced DPI image
  - ReducedBits-8-to-1-Bitmap - Create a new bitmap with reduced bpp image

- **write2file.cpp**
  - WriteDIB write a device independent bitmap to a file .
- **writegif.cpp**
  - DeleteGIFstream Delete the GIF stream
  - hbmp2GIFstream Convert the bitmap to GIF stream
  - GIFwrite write the GIF image to a file
- **parser.cpp**
  - generate(): Generates the output data structures and write to the file "spec.h"
  - gen(): Generates a set of routines to initialize the data structures created by the above routine, and writes them to a file "init.c"

Appendix B

Interleaved 2 of 5 Character Set

ASCII Character	Binary Word	Check Character Value
1	10001	1
2	01001	2
3	11000	3
4	00101	4
5	10100	5
6	01100	6
7	00011	7
8	10010	8
9	01010	9
0	00110	0

Start 0000 * Stop 100 *

Note : * Alternate bars and spaces, not interleaved