Chapter 5.

A case study.

Summary. The stamp dealer's business background, a requirements statement, developing the specification, generating the test sets, developing the implementation, looking at the final system.

In this chapter we will look at a real case study, a system that was built for a real, commercial, client using the ideas described above. The background was of a small retail enterprise that wished to computerise its entire operations. The challenge was to deliver the correct system at the correct time to a high level of quality. This was achieved, it took three person months, was delivered on time and has run continuously for 16 months with no faults reported. This chapter is based on the work of Mohammed Al-Qaed.

5.1. An example of a stream X-machine specification - basic background and the requirements capture process.

A critical part of this project, like any other project, was that of identifying, *correctly,* the required system and this is an area where the integrated X-machine approach taken was very helpful.

A number of discussions with the client took place and these could be summarised as follows:

- understanding the client's business and its environment;
- identifying the basic business processes and producing a business process model in simple English, which was approved by the client;
- translating the business process model into a sequence of specific activities defined around a number of user interface modes - essentially screens;
- formalising the top level interface structure using state diagrams;
- identifying the activity for each screen, the data to be input at appropriate points and the control buttons to move to other states of the system;
- checking with the client, through a number of simple non-functional prototype screen displays, that the correct information was available at the correct time and in the correct order during the proposed system operation.

Thus the outline structure of the system architecture were based on the identification of: the main "superstates" of the user interface; on the type of data to be input to or output from these screens and their relative relationship - which screen led to which screen and so on (that is the dynamics of the interface navigation). This is a very important and practical method since the client was able to see the relationship of the different part of the system in a meaningful and concrete way at an early stage and was able to relate it to the business process model that he understood. The approach was intuitive for all concerned and yet produced, in an hierarchical and structured way, a *complete* formal specification to the level of abstraction that was required for an implementation. It is unlikely that traditional formal methods - Z [16], VDM [17], OBJ [18], CCS [38], CSP [86] could be used in this way to the same effect. Yet the specification was entirely formal, was developed in a way that emphasised precision and conciseness and was used as a basis for the generation of a complete functional test set.

This case study is based on a real software development project undertaken by M. Al-Qaed and one of

the authors during June - September 1996. The client runs a postal philately business and wishes to computerise his operation. These notes describe the current business process and some initial outline requirements as captured by the software engineer during a number of discussions with the client. Later we will see a full formal development of this system using stream X-machines. The method is based around the identification of the structure of the user interface in consultation with the client and, eventually, the use of a visual programming language. The testing method was then applied be discussed in a later chapter.

The current business environment :

The dealer sells new (mint) and used postage stamps from any country to stamp collectors, who collect stamps out of interest or for investment, some of the stamps are extremely rare and valuable. These stamps are categorized by *countries* and every group of stamps from the same category is located on one or more *cards*. The cards are simply small albums of two sizes - the small ones are more expensive than the bigger ones. The *units* consist of one or more cards, and those are filed ready for sending to customers. Units are the smallest part in the system and they have an *identity number* with a *colour* to make the search easier. In the normal course of business a customer will indicate the type of stamps they are interested in and on a regular basis the dealer will send a collection of stamps by post to the customer for their approval, the customer retaining the ones he/she wants and returning the rest with an appropriate payment. Currently the dealer stores all the information required in manual files and when he wants to send some units he adds all the data manually and ensures that the units to be sent have not been sent before to the same customer.

Current manual procedures.

5.1.1. The dealer categorizes the stamps into groups and separates each group into units. Each unit contains a selection of stamps and all the stamps are from a specific country.

5.1.2. He then numbers each group using three digits and four colours to make the manual search easier.

5.1.3. All new stamps have to be categorized and numbered before they are added to the system.

5.1.4. The customer, initially, contacts the dealer and indicates that they wish to buy some stamps.

5.1.5. The dealer replies with a list of categories available.

5.1.6. The customer chooses some categories of stamps and selects an appropriate time interval for receiving the stamps (for example, immediately, two weeks, one month or any other date).

5.1.7. The dealer will check if these categories are available or not. He will send some units from the categories requested (if all the categories requested are not available then he writes to the customer asking him to choose others). He will write down the customer's name, address and the units to be sent.

5.1.8. The customer selects some stamps from the units delivered and keeps them. He/she returns the others to the dealer with the money for the stamps purchased and gives their next choice of categories and the date when they would like to receive another package of stamps.

5.1.9. The dealer writes down the cost of the stamps sold and puts the customer's name in the queue for his next order. This queue is subject to the choice of the customer for his next order date (if the customer does not give his/her next order then he/she will be deleted from the system automatically).

5.1.10. Repeat steps 5.1.1 to 5.1.3 for new stamps to be added to the system.

5.1.11. Repeat steps 5.1.7 to 5.1.9 for the process of selling the stamps (Steps 5.1.4 - 5.1.6 are for new

customers only).

This is a simple business process model which was confirmed by the dealer/client as representing the way he wished the proposed system to work. This activity was relatively straightforward, since there were no technical computing issues raised at that time.

Elementary data modelling.
We now start identifying some of the simple aspects of the system.

Units and customers.
Description of the sale units:

The system consists of 300-400 units (maximum).
There are two types of units (big/small).
Each unit has the following fields :

Number : Identity number (unique) to distinguish between cards.
Colour : To make the search easy.
Number : Number of cards in the units.
Category : Country or region.

Description of the Customers:

The system consists of 150 customers (maximum).

Each customer has the following fields :

Number : Identity number (unique).
Name : Customer's name.
Address : Customer's address.
Date of next order : immediately, one week, two weeks or one month.
Categories : Stamps requested from specific categories.
Unit numbers : which units were sent to him/her (could be more than one) ?
Cost : How much has he/she spent ?

Basic assumptions and the operating context.
The system will be installed in a "stand alone" computer and will be used by the dealer only, so no protections such as passwords are required by the dealer.

The dealer should have a basic knowledge of the Windows environment [87], and he must be able to run a suitable word processor application and be able to edit and print out texts.

The system must be easy enough to understand and use without a lot of training. The system will use the Windows environment and must provide appropriate help when needed at every stage of the system.

A search facility is needed for customer numbers and names but not for addresses.

Only the last six month's records of requests are needed to be stored in the system for each customer.

Specific constraints.
The user owns an Intel Pentium^{TM} PC with 8 MB of RAM, 800 MB of Hard Disk and 100 MHz processor.

The system was to be implemented using a Visual Basic [88] interface to an Access [89] database. This decision was made, primarily on the basis of what was available to the developer but also to see if the specification and testing methods were suitable for such a commonly used languages.
The system must be reliable and fully documented and must be developed rapidly (3 months at most).

Informal initial requirements.
Req. 1. To produce an application to automate the manual operations of the stamp dealer system. This application must do all the functions of the system that can currently be done manually quickly and provide the necessary data for the user at the appropriate time. The application should be easy to use and the process must be faster and easier than the manual method.

Req. 2. The manual method consists of *units* and each unit consists of *cards* that contains many *stamps* of the same *category*. These units have *identity numbers* and *colours* to distinguish them. The user should be able to **edit** units' data (*unit number, colour, category, number of cards* and *type*). The units' data should be stored in the computer to use them at the sale time (request and send orders' process).

Req. 3. The maximum number of units in the system is *400* units and there are four colours (*Red, Blue, Yellow, Green*). There should be no restrictions in permitted category names except the length, which is *30* characters in maximum. The card type is either small or large and number of cards is not more than *20* and not less than *1*.

Req. 4. *Customer names* and *addresses* need to be **stored** in the computer database, so *identity numbers* are important in order to identify each customer. There will be no more than 200 customers in total, so three digits could be enough for the identity numbers.

Req. 5. The dealer should be able to **add** new units or customers to the system. He should be able to **modify**, **browse**, or **delete** the existing ones.

Req. 6. The dealer could **send** existing units to existing customers at any time. This process is done in two steps: the first, the customer **requests** to see some stamps from a particular country (category) and gives a preferred date for receiving these stamps. The dealer **opens** a request transaction for the customer and **notes** the categories needed, with the date. If this is not the first request for the customer, then the dealer notes the cost of previous stamps sold to the customer to provide a check on the customer's usual purchasing habits.

Req. 7. When the dealer wants to **send** stamps to customers the system should suggest the first order to be done. The suggestion comes from the first due date of the requests, but the dealer should also be able to **examine** the order and **select** any request from the list.

Req. 8. The system should be able to **display** a customer's record (number, name, and address). Some useful information must be displayed with the customer record, this is the total number of orders the customer has made previously and the total value stamps purchased by the customer. This will give an indication about the customers previous purchasing record to allow the dealer to send him/her a suitable number of units.

Req. 9. The system should **suggest** the units to be sent to the customer, which should be different from those sent during the last six month's transactions with that customer. These units should, naturally, be available from stock and also from the same category requested by the customer. This suggestion will assist the dealer in sending suitable units that had not been sent before to the customer. This must be flexible and allow the dealer to select from the units sent to the customer before if he prefers.

Req. 10. The **replies** from a customer with his/her next request and the **send** to customer process should be repeated until the dealer **deletes** the customer from the system.

Additional requirements (*identified when version one of the product was installed*) :
NewReq. 1. *After* **selecting** some units in (step Req.9), the dealer should be able to **print** a *ready* letter (edited beforehand) to the customer including the unit numbers which will be send to him/her and the category names. A *special offer message* must be printed in the same letter if the customer has recently purchased more than 10 (or some other suitable number) collections of stamps over a given period.

NewReq. 2. If the customer **fails to return** the units sent to him/her on approval, then the dealer should be able to **send** him/her a letter requesting that he/she **returns** the unwanted items. This letter must include the units with the date of dispatching the stamps to the customer.

5.2. Developing the specification.

The first outline interface specification (identification of the top level control states) was drawn:

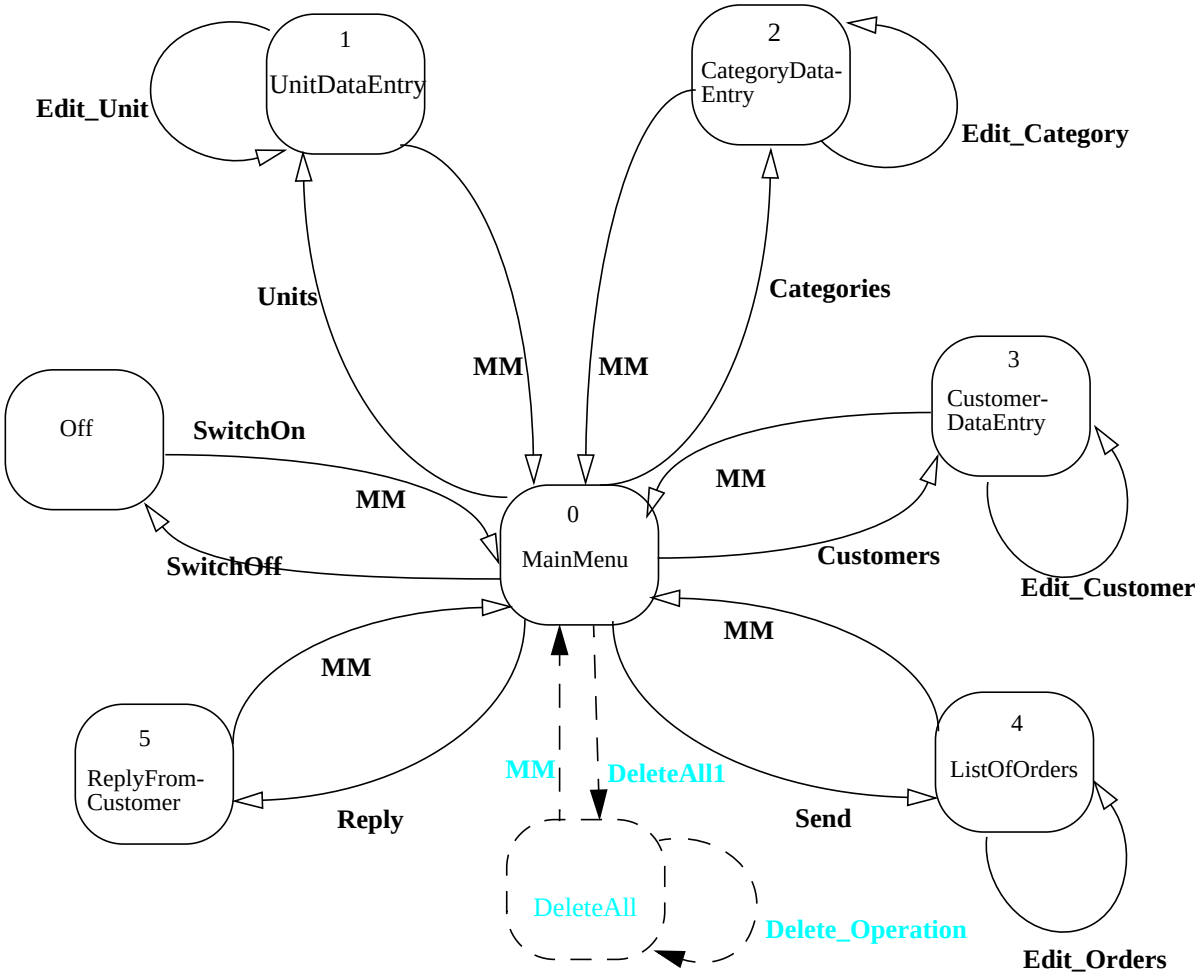

Figure 5.1 : An X-machine specification - first version (level 1)

This diagram presents the top level control structure of the proposed user interface. It presents to the client a series of forms that will implement, in outline, the processes identified in the requirements. The labels on the arcs represent functions which will be defined formally later, at this level these functions represent events such as clicking on a menu button to select a new state. The **MM** functions represent the return to the main menu - either at the end of an action or through an explicit “MainMenu” button click. In this particular example, prototype forms were implemented (using Visual Basic), to provide the client with a realistic simulation of a system.

Many of the states were then refined by introducing subsidiary states to reflect the "subfunctionality" of the state, thus we extend the CustomerDataEntry state to include a display and select operation, in other words the **Edit_Customer** function in the CustomerDataEntry state was refined by a suitably designed refining machine. Other states were introduced to cater for functions that were not in the original model, after discussion with the client. For example, the client wished to have a screen where various types of deletion operation could be carried out. This meant that the new top level diagram had an extra state, indicated by the dotted area of Figure 5.1. This is an example of changing requirements being managed in a simple and intuitive way. Having done this the **Delete_Operation** function could be refined which involved introducing a DeleteAll state and refining the diagram as shown in Figure 5.2.

.

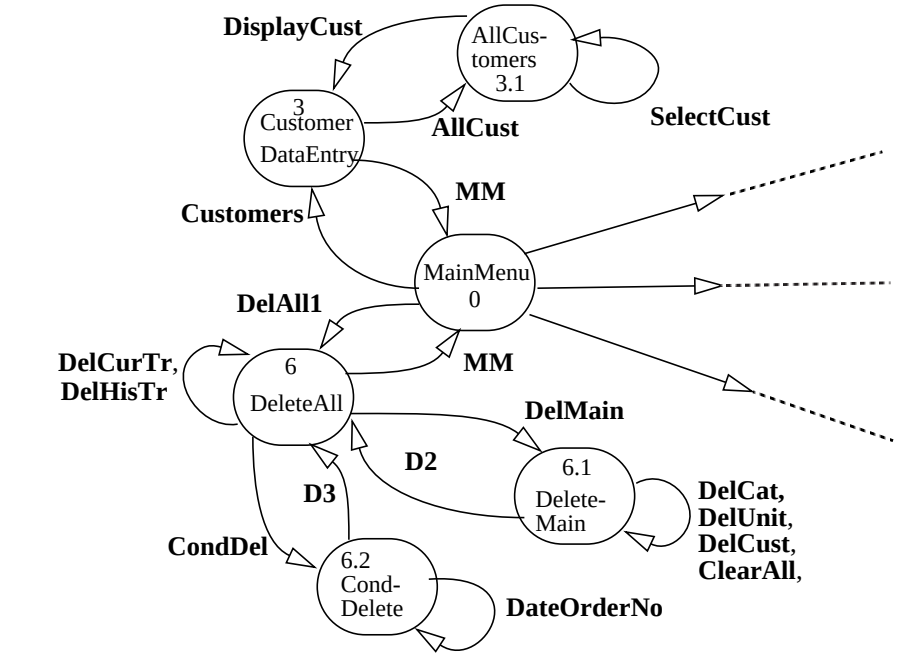

Figure 5.2 : A section of the refined machine from Figure 1.

Thus a number of refinements involving the refining of functions on specific states have been carried out. These can be refined further to allow for the full description of the functionality of the system down to the level of the basic Access functions described later in this section. It was thus possible to move to a full specification of the system in a few refinement stages. We present some of the details.

Developing the input-output.

The input for this system is of two types : constants (lower case characters) and variables (upper case characters). these two categories correspond to control inputs which will cause, in general, a change of the control state, and data entry inputs. The first simply are the mouse clicks over the objects on the displayed screens, and the second are the details of the form entry values. These second inputs must be of the correct type and lie within the range of valid values.

Y = {*on, off, data_maint, unit_data_entry, category_data_entry, customer_data_entry, all_units_list, all_customers_list, select_unit, select_customer, reply_from_customer,*

send_to_customer, list_of_orders, delete_all_menu, delete_main, cond_delete, delete_cats, delete_units, delete_custs, delete_cur_tr, delete_hist_tr, clear_all, main_menu, next, previous, delete, modify, add, save, cancel, ok, apply, change_order, yes, no, show_hide, select, de_select, select_all, selection, other, print} $\cup$ *CUST_NOS* $\cup$ *CUST_NAME* $\cup$ *ADDRESS* $\cup$ *UNIT_NOS* $\cup$ *CATEGORIES* $\cup$ *GROUP* $\cup$ *NO_OF_CARDS* $\cup$ *TYPE* $\cup$ *DATE* $\cup$ *COST* $\cup$ *ORDER_NO* $\cup$ *PERIOD*

Where :

- *CUST_NOS* : represents the set of all the valid customer numbers, (i.e. 100 ≤ *CUST_NOS* ≤ 999).
- *CUST_NAME* : represents a set string of characters, where the allowed length is between 1 and 50 characters.
- *CATEGORIES* : represents a set of string of characters, where the allowed length is between 1 and 30 characters.
- *GROUP* : represents a set of strings of characters, where the allowed length is between 1 and 30 characters.
- *NO_OF_CARDS* : represents the set of numbers starting from 1 to 20. (i.e. 1≤ *NO_OF_CARDS* ≤ 20).
- *TYPE* : is the type of cards. There are only two types of cards (i.e. *TYPE* = {*big, small*})
- *DATE* : any valid date of the format (*dd'-mm'-yy'*), where *d, m, y, d', m', y'* ∈ *DIGIT*.
- *COST* : is the cost of the purchased stamps. The *COST* could be any real number.
- *UNIT_NOS* : a character representing unit colour (*R, B, Y, G*) or mint (*M*) and a set of all the valid unit numbers, (i.e. 100 ≤ *UNIT_NOS* ≤ 999), so *UNIT_NOS* are the combination of one letter with 3 digits.
- *ADDRESS* : represents a set of strings of characters, where the allowed length is between 1 to 200 characters.
- *PERIOD* = {*Three months, Six months, One year, Two years*}
- *ORDER_NO* is any integer number.

The output will be divided into two streams: $Z = Z_1 \times Z_2$.

Z_1 = { *MAIN_MENU, DATA_MAINT, UNIT_DATA_ENTRY, CATEGORY_DATA_ENTRY, CUSTOMER_DATA_ENTRY, ALL_UNITS, ALL_CUSTOMERS, REPLY_FROM_CUST, SEND_TO_CUST, LIST_OF_ORDERS, DELETE_ALL, DELETE_MAIN, COND_DELETE* }

Z_2 = {*LETTER1, LETTER2*} x *MESSAGES*

Where :

- *MAIN_MENU* : is the main menu screen, which contains five options to choose [state 0];
- *UNIT_DATA_ENTRY = UDE_Screen* x *UNIT_NO* x *CATEGORIES* x *NO_OF_CARDS* x *TYPE* [state 1];
- *CATEGORY_DATA_ENTRY* = *CDE_Screen* x *CATEGORIES* x *GROUP* [state 2];
- *CUSTOMER_DATA_ENTRY = CuDE_Screen* x *CUST_NO* x *CUST_NAME* x *ADDRESS* [state 3];
- *ALL_UNITS = ALU_Screen* x *SeqOfUnit* (sequence of records), where each record contains (*UNIT_NOS* x *CATEGORY* x *NO_OF_CARDS* x *TYPE*) [state 1.1]
- *ALL_CUSTOMERS = ALC_Screen* x *SeqOfCust* (sequence of records), where each record contains (*CUST_NO* x *CUST_NAME*) [state 3.1]
- *REPLY_FROM_CUST = RFC_Screen* x *CUST_NO* x *CUST_NAME* x *RETURN_PRO* x *NEW_ORDER_PRO* , where :
  - fi *RETURN_PRO = LIST_UNITS* x *LIST_RET_UNITS* x *COST*
  - *fi NEW_ORDER_PRO = LIST_CAT* x *LIST_REQ_CAT* x *DATE*

[state 5]

- *SEND_TO_CUST = STC_Screen* x *CUST_NO* x *CUST_NAME* x *ADDRESS* x *TOT_ORDER* x *TOT_COST* x *DATE* x *LIST_BIG_UNITS* x *LIST_SMALL_UNITS* x *UNIT_NOS* x *SEL_BIG* x *SEL_SMALL* [state 4.1]
- *LIST_OF_ORDERS = LOO_Screen* x *SeqOfOrd* (sequence of records), where each record contains

(*CUST_NO* x *CUST_NAME* x *DATE*) [state 4]

- *DELETE_ALL* : is delete all screen, which contains five options to choose [state 6].
- *DELETE_MAIN* : is the delete main data menu, which contains five options to choose [state 6.1].
- *COND_DELETE = CD_Screen* x *PERIOD* x *ORDER_NO*, where *PERIOD* and *ORDER_NO* are introduced in the input alphabet [state 6.2].
- *LETTER1* : represents a sequence of characters saved as a text file named **return_a.txt**, which contains a letter to customers failed to return the approvals sent to them. This letter tells the customer to return the approvals as soon as possible.
- *LETTER2* : represents a sequence of characters saved as a text file named **send_apr.txt**, which contains a letter to customers including all the units sent to them.
  MESSAGES = {*msg1, msg2, msg3, ..., msg92, No_Mess* }, Where *msg1, ..., msg92* are messages or sequence of messages displayed by the machine. These are four examples of the messages :
  ⇒ *msg1* = "Duplicate category .. Please try again."
  ⇒ *msg2* = "You must fill the category field."
  ⇒ *msg3* = "You must fill in the group field."
  ⇒ *msg4* = "Are you sure you want to delete this category ?"
  ⇒ *No_Mess* represents the blank message.

The memory.

The memory does not contain any screen formats or messages, but it stores the information needed at the execution process of the X-machine model. We will be wishing to access the underlying database records and certain basic information is needed in the memory to do this conveniently. As described in the last section the memory consists of :

M = an array of global variables x Z_1

where :

an array of global variables : *FLAG* x *UNIT_NOS* x *CUST_NOS*

⇒ *FLAG* : an integer represents a flag and its valid values are *1 - 5* .

⇒ *UNIT_NOS* : a character representing unit colour (*R, B, Y, G*) or mint (*M*) and a set of all the valid unit numbers, (i.e. $100 \leq UNIT\_NOS \leq 999$), so *UNIT_NOS* are the combination of one letter with 3 digits.

⇒ *CUST_NOS* : represents the set of all the valid customer numbers, (i.e. $100 \leq CUST\_NOS \leq 999$).

- Z_1 :was defined above.

The initial memory value :

m0 = (*InitMem*, ^) where *InitMem* is the clear array of global variables (*0*, ^, *0*)

The processing functions.

In addition to the processing functions we require some other functions to manipulate the database functions or any other functions in the lower levels. The database used to develop the system is ACCESS2 [89]. We will require some functions to "write", "read", "find a record", "delete", "modify", etc. the database. Since these functions are available in the ACCESS2 database we will not be concerned about the way these functions are implemented within ACCESS2. We will also make an assumption that these functions are correct. There are also some other "system" functions available in Visual Basic [88], such as "select a cell" which we will assume to be available and correctly implemented.

These are some of the partial functions required in the level one of our X-machine, followed by the description of some of them as an example :

GetFirstUnit(UnitTable), GetFirstCat(CategoryTable), GetFirstCust(CustomerTable),

GetAllUnits(UnitTable), GetAllCust(CustomerTable), GetUnitNo(SeqOfUnit), GetUnit( ), SelUnit(SeqOfUnit), SelCust(SeqOfCust), GetRecords(TempTrans), SelRec(SeqOfOrd), GetCustNo(SeqOfOrd), GetCustData(CustomerTable, TempTransTable), GetRecords(TempTrans).

where :

GetFirstUnit(UnitTable) **:** retrieves the first record in the unit table and displays it on the screen if the table is not empty.

SelRec(SeqOfOrd) **:** this function selects the clicked record by highlighting it, once the record highlighted it means one of its attributes changed. Since all attributes are kept in the memory, so this function is going to change the memory.

GetCustData(CustomerTable, TempTransTable) **:** retrieves customer data from customer tables and the requested categories by the customer from TempTrans table.

The following is the complete set of functions, which have the effect of reading the input and current memory, moving to the next state, and altering the memory and the output.

Φ = { SwitchOn, SwitchOff, DataMaint1, MainMenu1, Units, DisplayUnit, AllUnit, SelectUnit, DataMaint2, Categories, DataMaint3, Customers, DisplayCust, AllCust, SelectCust, DataMaint4, Reply, MainMenu2, Send, MainMenu3, SelectRecord, Ok, List, MainMenu4, DelAll1, MainMenu5, DelCurTr, DelHisTr, DelMain, DelAll2, DelCat, DelUnit, DelCust, ClearAll, CondDel, DelAll3, Cancel, Date, OrderNo, Next1, Prev1, Delete1, Cancel1, Save1, NewUnit, IllegalUnitNo, UnitNo, Modify, IllegalCategory1, SelCategory1, IllegalNoCards, NoCards, IllegalType, Type, NewCategory, Cancel2, Save2, Next2, Prev2, Delete2, IllegalCategory2, Category2, IllegalGroup2, Group2, NewCustomer, IllegalCustNo3, CustNo3, IllegalCustName3, CustName3, IllegalAddress, Address, Ok3, Modify3, Cancel3, Save3, Next3, Prev3, Delete3, IllegalCustData4, ChangeOrder, DeleteTrans, CustData, Cancel41, Cancel42, Save41, Save42, Select41, DeSelect41, Select42, DeSelect42, Apply41, Apply42, IllegalCost, Cost, IllegalDate, Date, NewOrder, ReturnUnits, No41, No42, Yes41, Yes42, PrintLet4, YesPr41, NoPr41, NoPr42, YesPr42, All41, Selection41, All42, Selectio42, Ok5, Next5, Prev5, Cancel5, Save5, Select5, DeSelect5, ShowHide, OtherSelect, UnitNo5, TryAgain, IllegalUnitNo5, Ignore, Apply5, NoPr5, YesPr5 }

Some examples of the definitions of these functions are given next. We have introduced the memory and the output stream, so (for the sake of simplicity) we will introduce the processing functions in the following format, ϕ: $Y \times M \rightarrow M \times Z_2$. We will not have to mention Z_1, since it is part of the memory. These are some examples of the processing functions from level 1 with a brief description for a selection of them :

Some Level 1 processing functions :

- **SwitchOn**(*on*, (*InitMem*, ^)) = ((*InitMem*, *MAIN_MENU*), (^, *No_Mess*))
  The **SwitchOn** function starts at the initial memory (clear memory), removes the current input symbol (*on*), and displays the *MAIN_MENU* screen. This function does not give any messages or printings. The *MAIN_MENU* screen will be kept in the memory.

- **SwitchOff**(*off*, (*InitMem, MAIN_MENU*)) = ((*InitMem* , ^), (^, *No_Mess*))
  The **SwitchOff** function can be applied from the *MAIN_MENU* screen only. It removes the current input symbol (*off*), clears the memory, and clears the output.

- **DataMaint1**(*data_maint*, (*InitMem, MAIN_MENU*)) = ((*InitMem, DATA_MAINT*), (^, *No_Mess*))

- **MainMenu1**(*main_menu*, (*InitMem, DATA_MAINT*)) = ((*Init_Mem, MAIN_MENU*), (^,

No_Mess))

- **Units**(*unit_data_entry*, (*InitMemory*, *DATA_MAINT*)) = ((*Init_Mem*, (*UDE_Screen*, GetFirstUnit(UnitTable))), (^, *No_Mess*))

etc.

5.3. Data Design.

The first version of the specification of the entities contained four entities only (*Unit, Category, Customer,* and *Transaction*). Initially the customer entity consisted of three attributes but then the user requested two additional ones, the total number of orders that a customer made and the total price of stamps he/she sold, so these additional attributes were added after the first version of the product was installed. After the second version of the product was installed the user requested an additional requirement : the *special offer.* This new requirement can be provided by adding a counter attribute (*SaleAmount*). The four entities were normalized several times to resolve the one-to-one and many-to-many relationships as those are not accepted by the rules of the entity relationship model. The final entity relationship (the normalized one) is given below, in which all relations are one to many.

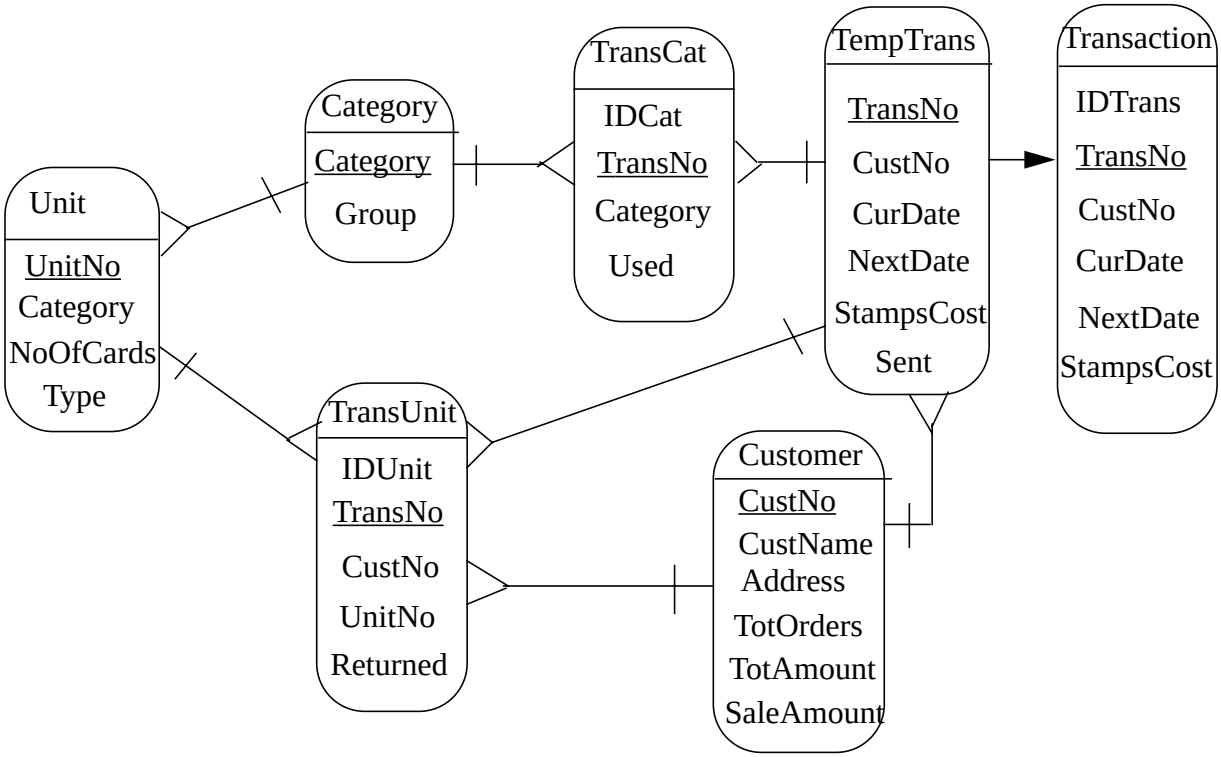

Figure 5.3. Refined entity relationship (all relations are one_to_many)

Some comments about this entity relationship.

The TempTrans entity was introduced to differentiate transactions under process from completed transactions. When any transaction was processed completely then it will be moved to the Transaction entity, this reduces the search time for current transaction and reduces the size of the TempTrans entity that is required to be used at every transaction.

The transaction entity is required only as a history at the conditional delete process, and the other attributes may be used in the future for some addition requirements from the user (such as detailed transaction information about every customer).

The TempTrans entity is separated into three entities (TempTrans, TransCat, and TransUnit) in order to resolve the many-to-many relationship. This normalization reduced the database size significantly, as in the first model for every transaction we needed to repeat fields of the TempTrans entity for every category the customer requested (it may be more than 20 for an order) and repeat them again for every unit that will be send to the customer (it may be more than 7 for every order). We can imagine the database size for only 100 transactions.

The CustNo attribute is added to the TransUnit entity (this is not required by the normalization rules). This is done because at every time the user requires to send some units to a customer the system must suggest the units under the specific categories he/she requested and those that were not sent to the customer before. This process needs to read from the transaction entity all the transactions made by the customer and then search the matched records in the TransUnit entity with every record in Transaction entity. This process reduces the search time of the system as it should be done for all transactions, so adding only one attribute is much better than affecting the performance.

Since all the entities required by the system have been introduced, we require to choose the most appropriate database to be created and linked to the system. Visual Basic supports many types of databases such as Access 1.1, Paradox [90], and FoxPro [91], thus we had to chose one of them. After a detailed investigation the final decision was to chose Access 2.0. The relational data analysis was not strictly necessary, however it was convenient to do this because of the chosen implementation package for the underlying database.

5.4. Testing.

One of the main benefits of the X-machine approach is the existence of a complete functional testing theory and algorithms for generating these test sets. This is the next task and tools would usually be used to construct the full test set. However it is possible to illustrate some of the constructions with this example. It may seem strange that we are discussing the test generation process before we have looked at the design and implementation process. This is deliberate and is intended to emphasise the importance of thinking about testing at an early stage. Issues of test refinement in conjunction with specification refinement will be discussed elsewhere and this relationship completes the essential link between specification and test that is one of the cornerstones of our approach.

The following is a small selection of the test set for the X-machine specification defined in 5.2 (The specification has to be augmented in the appropriate way so that the design for test conditions are satisfied, this is straightforward and we do not examine it in detail here.)

The first machine can be tested simply, because its input set is very small. All inputs of this machine are the clicks over different command buttons to move from one screen to another, so the complete test set can be constructed and applied in a reasonable time.

The test set for the top level machine is constructed from the specification involving the following transitions:

Φ = { reply, MM, SwitchOn, SwitchOff, Units, Edit_Unit, Categories, Edit_Category, Customers, Edit_Customer, Send, Edit_Orders }

The state cover is:

{ **1, Reply , SwitchOff , Units , Categories , Customers , Send** }

The W set is:

{ **Reply , MM , Edit_Unit , Edit_Category , Edit_Customer , Edit_Orders**}

The complete test set for the top level machine comprises 128 elements. As we discussed in Chapter 4 the process of testing a system implemented from a number of components involves transforming this test set and testing the implementations of the components separately. So, using the refinement testing method a series of further test sets can be generated, and the whole system tested incrementally.

These test sets were then applied to the implementation and were able to find all faults on the assumption that the basic underlying functions and the primitive Access functions were correct. In practice what happened was the implementations of the individual submachines, such as the CategoryDataEntry machine, was tested fully using the test sets generated. Then the complete integration of these machines was tested with the test set formed using the method described in the last Chapter. The details of the refining machines are too lengthy to describe here but the process of constructing the test set for the integration of the components was quite straightforward.

5.5. Implementation.

The role of the implementation is to translate the detailed design into a physical representation (code) using a programming language. The physical implementation should achieve the user's general and detailed requirements. In order to deliver a system, meeting all the agreed client's requirements, which is easy to test and maintain an appropriate method has to be applied during the development process.

General comments.

All objects and screens should carry meaningful names, satisfied by the naming rules, and use the graphical power provided by Visual Basic as much as possible. The created screens will not be thrown away (it is not a prototype), so special attention must be given at the implementation of these screen to make them friendly for the users and easy to use. Copying the components of a well-implemented screen to any similar screens and then modifying the copied components is a good idea and could save time in the event of re-designing the new screens (e.g. UnitDataEntry and CustomerDataEntry).

Level one of the X-machine state diagram.

We translate each state from the first level X-machine state diagram into a *form* (each form can be considered as a separate screen) and link them as specified in the diagram. The implemented screens will represent level one in the diagram and the arcs between any two states can be represented as an object (Command button) in the first screen (first state), which will unload the current screen and display (load) the second screen (second state) when clicked. This process will end with completed screens for those states that are not refined in the initial diagram (e.g. MainMenu).

Level two of X-machine state diagram.

The second level (refined from some states in level one) can be implemented by adding some objects (Command boxes, Text boxes, etc.) to the designed forms. Then the new objects should be linked together in a similar way to the linking between the states in the refined screens. This is not as simple as implementing level 1 by loading or unloading screens when applying the functions. The control movements this time are going to be done within the same screen between the objects, so we have to disable

all the objects on the screen except those that are allowed to be used at that point in the process.

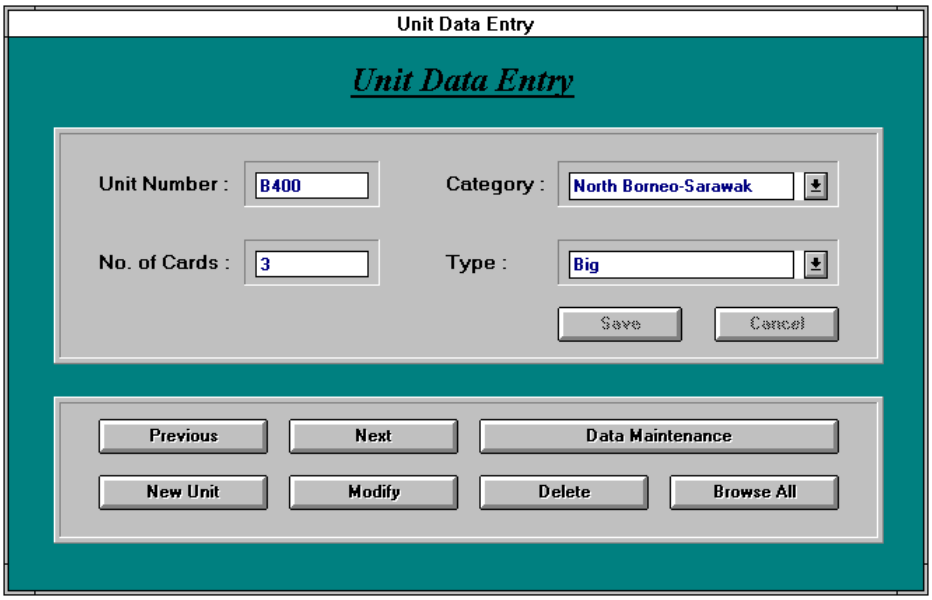

Figure 5.4. UnitDataEntry screen (screen dump)

Example. The UnitDataEntry screen (Figure 5.4.) implements the UnitDataEntry state (Figure 5.1), so all command buttons on the screen can be clicked (functions in the diagram) except **Save** and **Cancel**. The Text and Combo boxes cannot be accessed at this state. If the command button **NewUnit** is clicked then only the UnitNo Text box and Cancel button should be accessed and the others should be disabled. This is the way all the screens were implemented to make them similar to the states in the X-machine specification description. At this step the Text boxes will allow any type or length of edited strings as we are not concerned with the data structure and types. At the end of this step we will have many missing functions from the diagram (e.g. Prev1, Next1, IllegalUnitNo), since those require the data structure.

Note : The implemented screens can be used as a prototype version to give the user an idea about the final product layout and how it will perform. This can be done during the requirements capture stage and provides a mechanism for the client to explore the designer's understanding of the business processes. Since the method addresses the client's business processes through screen displays based on forms and the type of data that the client is used to dealing with it turned out to be very productive. The complete requirements must be confirmed at this step or as soon after as possible.

Add the conditions required to move between states.

At some states a string or number will be required to be edited. The state allows only a particular text, which meets a specific condition (e.g. WaitNoCards state is allowed only a numeric value within the range 1-20 to be inserted/edited in order to move to the next state, and if the input is not of this type the system should stay at the same state and give a message asking the user to edit a valid number within the range). At the end of this step all functions will be implemented except those that require a physical data storage (database, file, printer, etc.). The system will provide a convenient environment for coding the main procedures and functions. The programmer should add codes to the objects when a specific event happens (click, press-key, etc.) and code the procedures and functions required.

Create and link the database, tables, and files.
The database, tables, and files can be created, because the complete requirements were given, thus the system should be designed completely (the feedback may cause some modifications). After creating the database it can be linked to the system, the link should be started from the basic data (unit's, category's,

and customer's entry screens). The product constructed from this stage is ready for the installation as a first version to take the final comments or modifications required by the user (the user may find some errors when using the system in the real environment, so this is an additional means of testing - no formal specification can easily capture all of the environmental conditions that affect the system, in particular the client's machines may differ in subtle and undisclosed ways from the developer's).

Finally a complete set of report/user messages and facilities were added and the final modifications made. The complete set of messages in the system was 92 messages providing the user with a clear statement about the errors, invalid input, confirmation requests for risky functions, etc. Any additional facilities the programmer wishes to add to the system in order to improve the quality or to make the system easier can be added at this stage. The design of these was of great importance and should be discussed with the client.

5.6 A summary of the facilities provided by the completed system.

The system provides many facilities including:

5.6.1. It allows you to scroll through all the units' data in one screen and then to select the unit required for modification or deletion using a very simple and easy method (by clicking on the record only). This facility is available for the customers list as well.

5.6.2. It allows you to change the order of the units by unit number, category, or type, which could help the user and save time significantly.

5.6.3. It allows you to change the order of customers' data by customer number or name.

5.6.4. During the daily transactions it is very important to provide the user with a powerful searching facility, which is required many times every day. The system provides the user at the time of requesting new orders or at the time of returning units (ReplyFromCustomers screen) by four different ways to find customers. The customers' numbers and names are combined in an ordered list (ordered by number), so the user just has to scroll through them to find the appropriate customer. The second way is by changing the order of the same list to be by customers' name and then make the selection. The third is by typing the customer number and the system automatically will retrieve customer's information and the last is by typing customer's name.

5.6.5. The user can delete any untidy/unsatisfactory transaction for a specific customer from the ReplyFromCustomer screen.

5.6.6. To select the requested categories the user has only to click on the categories required from the list and they will be selected, or click over the **all** Option button to select all the categories. The selection of the returned units can be done in a similar way.

5.6.7. The searching facilities are not required at the time of units are sent, so a list of requests made by all customers is provided (ordered by the sending date) to allow the user to select from it (just click over the record). Then the system will display the complete information about the customer (total no. of orders, total stamps sold), which may help the user to decide what sort of customers is this (good, bad, etc.) and thus he can allow the appropriate units to be send at the same time.

5.6.8. The system will automatically suggest the units to be sent to a customer, which must be units that are available and which have not been sent to the same customer previously. It provides some flexibility by allowing the user to add one of those units sent to the customer, or any units not from the requested categories. He could display the categories requested on the same screen, which help in selecting the appropriate units.

5.6.9. To simplify a user's work the system displays the total number of big and small cards selected (each unit consists of one or more cards) as he will put them in an envelope and he is restricted with a

maximum limit of cards.

5.6.10. When the selection is completed by default the system will ask the user to confirm the details in order to print a letter for the customer informing him of the units included. If the letter is not printed for some reason (the printer not ready..), then the user could do so later from the ReplyFromCustomer screen. If any customer fails to return the units sent to him another letter could be printed (including the detailed description about the units previously sent) requesting him/her to return them back as soon as possible. Both letters could be edited or modified by the user, because they are saved as text files.

5.6.11. A complete set of delete facilities are available in the system (delete all units, delete all categories, delete all customers, clear the data base, delete the transactions under process, delete the history transactions, or delete history transaction by two conditions : date and number of orders for every customer). All deletion functions will ask for confirmation before applying the delete process.

Conclusions.
A very robust and successful system was developed over a short period of time. The way in which the method was able to capture the client's real requirements in a simple and effective way, together with the user-friendly nature of the formal language used in the specification were of considerable help in understanding the business processes involved. This precise information was then invaluable for constructing the code very rapidly and for generating the powerful test sets. These are all important strengths of the approach to building a *correct* system for the business. Using modern software development tools (a 4GL, Visual Basic and a proprietary database system, MS-Access) all add to the attractions and practical nature of the method.

So far the client has reported no faults and is entirely satisfied with the system. His company is now totally dependent on it.