2

UNLIMITED

RSRE
MEMORANDUM No. 4280

ROYAL SIGNALS & RADAR ESTABLISHMENT

AD-A212 402

THE APPLICATION OF Z TO THE SPECIFICATION
OF AIR TRAFFIC CONTROL SYSTEMS: 1

Author: L N Simcox

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

RSRE MEMORANDUM No. 4280

UNLIMITED

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4280

THE APPLICATION OF Z TO THE SPECIFICATION OF AIR TRAFFIC CONTROL SYSTEMS: I

April 1989

Author: L N Simcox

SUMMARY

This report describes an initial investigation into the formal specification language Z and its applicability to Air Traffic Control Systems. The software corresponding to the initial radar plot processing in the multi-radar automatic tracking system at the London Air Traffic Control Centre (LATCC) was used in the investigation. An informal pseudo code description of the radar plot processing function was taken as the 'requirements' and converted into a formal specification in the Z language. The specification was partly validated using an RSRE Z syntax and type checking tool. The experiences gained during the exercise are discussed and potential benefits for the Civil Aviation Authority are highlighted.

CARDPB Ref	2.4.1.1	
RSRE Task	900	
CAA Sponsor:	Director of Projects and Engineering DPE(R&DP)	
CAA Liaison:	S C Willmott	Hd DP1a
	C Metcalfe	RD1

Work done under contract to the UK Civil Aviation Authority

THE APPLICATION OF Z TO THE SPECIFICATION OF AIR TRAFFIC CONTROL SYSTEMS

I: An initial specification of the radar processing activity

CONTENTS

THIS PAGE IS LEFT BLANK INTENTIONALLY

1 INTRODUCTION

1.1 Background

The applicability of formal specification methods to Air Traffic Control (ATC) systems is one of the areas of the research being undertaken by the software engineering section of the ATC Systems Research division at RSRE for the Civil Aviation Authority. This report describes an investigation into the formal specification language Z and its applicability to radar data processing. The main thread of the investigation was the conversion of the algorithmic specification of a radar processing software module into the predicate-based language Z. To do this has involved learning about Z through the literature, attending a short course on formal methods, discussing problems with experienced RSRE Z users, and writing the formal specification which was partly validated using an RSRE Z type checking tool.

The particular software module used in the conversion exercise is called the RADAR PROCESSING ACTIVITY and corresponds to the initial radar plot processing in the multi-radar automatic tracking system of the London Air Traffic Control Centre (LATCC) radar data processing system. This software was chosen for the investigation because it represented a well defined piece of ATC software, and because the details were familiar to the investigator (the author). The informal algorithmic specification of the activity is taken from the pseudo code (subsequently referred to as PDL, Progam Design Language) developed in a MASCOT based design of the LATCC tracking system given in reference 1 and, for convenience, reproduced here in appendix 1.

The PDL looks much like the high level programming language Pascal, but it is not too precise on semantics. For example, it does not require a variable, say X, to be explicitly given a type; whether X is INTEGER or REAL has to be determined from supporting text or by context. The Z language, on the other hand is strongly typed, and variables must be given types, although it is not necessary to give details about a type if these details are not required - a name for the type is sufficient. A specification in Z is aimed at stating *what* is required, whereas the PDL defines *how* the requirement is to be achieved. The Z specification described in this report has been kept deliberately close to a one-to-one correspondence with the PDL specification since this eases the problem of ensuring the Z accurately captures the requirement of the radar processing activity as represented by the PDL; it is intended in future work to abstract away from the implementation-like specification to one more suitable for inclusion in a functional requirements document.

1.2 Structure of Report

The main body of the report is concerned with the development of the formal specification of the RADAR PROCESSING activity, as defined by the PDL in appendix 1, and is tutorial in form. Readers wishing to see the Z specification in a non-tutorial form that would be processed by a typical Z type checker should refer to appendix 3.

A brief description of the RADAR PROCESSING activity with the aid of a simple dataflow diagram is given in chapter 2. This is followed, in chapter 3, by a brief introduction to Z which should help in understanding the notation and concepts which are likely to be unfamiliar even to experienced software people. In this introduction a Z specification of the dataflow diagram of the RADAR PROCESSING ACTIVITY is developed. It should be emphasized that the main specification in chapter 4 was not derived from this example specification, indeed the example specification was produced while preparing this report, however it does provide an embryo of a method for top down development from a dataflow diagram. The chapter also describes some of the naming conventions used in Z specifications.

The derivation of the Z specification from the PDL is given in chapter 4. A two level specification approach is adopted where the first level of abstraction is chosen such that the main procedures of the PDL are revealed but not detailed. This level corresponds to a level slightly more detailed than that captured in the dataflow diagram of the RADAR PROCESSING activity.
The second level of the specification is a decomposition of the first level in a form that closely matches the structure of the PDL. This, together with keeping more or less the same names for variables, simplifies the correspondence checking between the PDL requirements and the Z specification, and eases the understanding for those already familiar with the LATCC-like radar processing programs.
This second level of specification presented here is derived from that which successfully passed through an RSRE Z type checker which ensured correct syntax and consistent use of types. The presentation is in tutorial form in a top down manner.

The lessons learnt in preparing the Z specification are described in chapter 5. Chapter 6 lists conclusions and discusses the possible benefits to NATS with proposals for follow up work.

2 RADAR DATA PROCESSING ACTIVITY

The general requirements of the activity can be understood with the aid of the data flow diagram of the activity shown in figure 1. The activity takes a plot (which will be either target or weather data) from the input buffer (RADAR DATA INPUT) and performs either target or weather processing. Target processing involves converting the plot position to system coordinates, and deciding if the plot is to be rejected, or to be used for display only purposes, or to be sent for possible correlation with tracks. The weather processing determines if the weather data is to be used for display purposes.

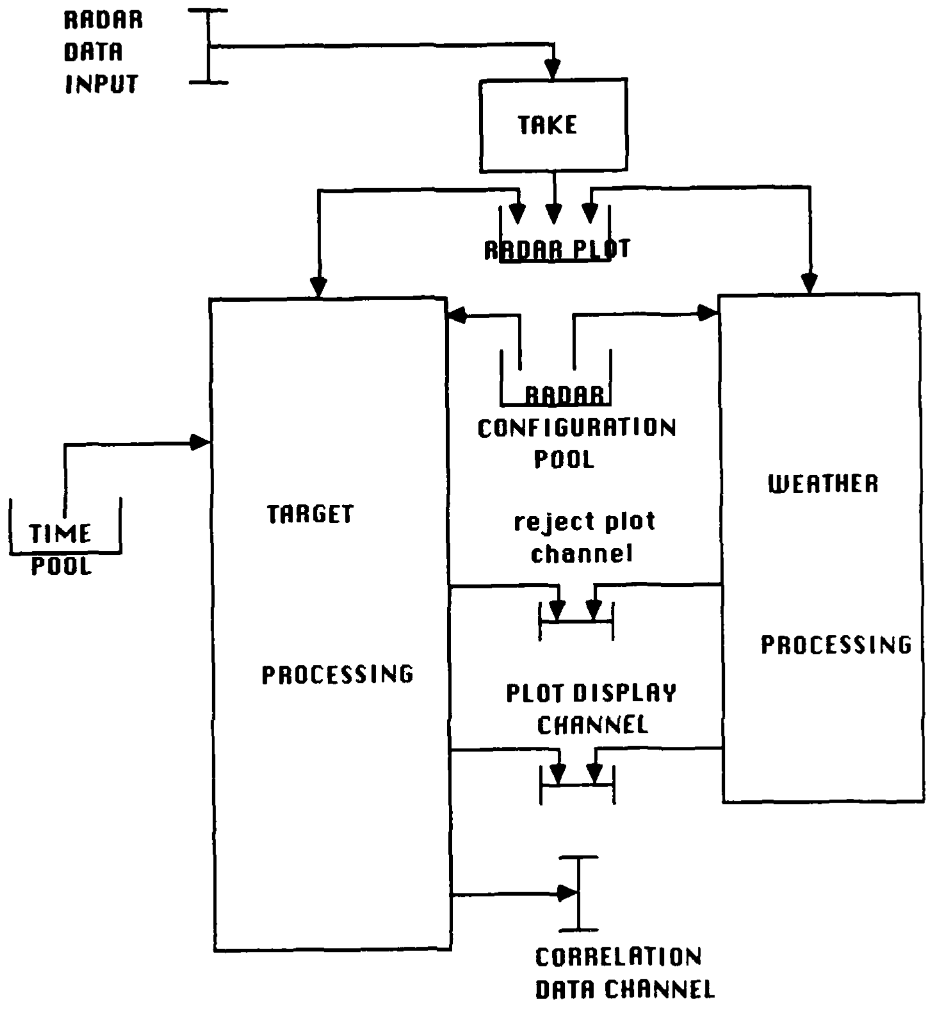
The RADAR DATA PROCESSING activity carries out the initial radar plot processing. It takes a digitized radar plot, which has come from one of a number of surveillance radars, from an input buffer and checks to see if the plot is from an aircraft target or if it corresponds to a weather strobe.
If it is a target plot then a time stamp is added and its position is converted into a common coordinate system. Subject to certain criteria which are defined later, a plot is then

a) thrown away as not required (which may occur during coordinate conversion), or
b) tentatively correlated with a track, or
c) sent for display.

If it is a weather plot, it is either

a) thrown away, or
b) sent for display.

The data flow diagram of the activity is shown in figure 1 which is used as a model for the Z specification. After processing a plot, the activity repeats the processing with the next plot. The informal specification of the activity is given by the PDL in Appendix 1.



<u>Data Flow Diagram: RADAR PROCESSING ACTIVITY</u>

Figure 1

3 THE Z LANGUAGE and NOTATION

A Z specification is composed of a mathematical text supported by a natural language description. The purpose of this chapter is to introduce sufficient Z notation to enable the Z specification of the radar processing activity given in the following chapter to be understood. Further details on Z are available in references 1, 2, 8 and 9.

A small Z specification

A specification of the data flow diagram (figure 1) provides a convenient way of introducing some of the terms and concepts. First the data areas are specified simply by listing them as follows

```
[ RADAR_DATA_INPUT, PLOT_DISPLAY_CHANNEL,
   CORRELATION_DATA_CHANNEL, reject_plot_channel,
   RADAR_CONFIGURATION_POOL, TIME_POOL, RADAR_PLOT ] .
```

This square bracket notation serves to introduces new types about which no more detail is known at this level of specification. Note: Integer types are assumed to be in the standard Z library; the character $\mathbb{Z}$ is used to denote the type integer (positive, zero, and negative) and $\mathbb{N}$ to denote natural numbers (non negative integers).
The Z specification is completed by defining the three radar processing operations in terms of these data types. One way of doing this is by stating the names of the operations with their input and output types, for example

```
TAKE: RADAR_DATA_INPUT ⇸ RADAR_PLOT
```

however a slightly more complex form is used here since it aids the tutorial:

```
TAKE──────────────────────────────────
│ take: RADAR_DATA_INPUT ⇸ RADAR_PLOT
└─────────────────────────────────────
```

this states that TAKE consist of a (partial) function, given the same name in lower case for convenience, which maps RADAR_DATA_INPUT to RADAR_PLOT. The other operations are similarly specified.

```
TARGET_PROCESSING─────────────────────────────────────
│ target_processing :
│     TIMEPOOL × RADAR_CONFIGURATION_POOL × RADAR_PLOT ⇸
│        RADAR_PLOT × CORRELATION_DATA_CHANNEL ×
│        PLOT_DISPLAY_CHANNEL × reject_plot_channel
└─────────────────────────────────────────────────────
```

this states that TARGET_PROCESSING contains a (partial) function which maps the pools and RADAR_PLOT to the channels and RADAR_PLOT, and

```
WEATHER_PROCESSING──────────────────────────
│ weather_processing :
│       RADAR_CONFIGURATION_POOL × RADAR_PLOT ⇸
│RADAR_PLOT × PLOT_DISPLAY_CHANNEL × reject_plot_channel
└───────────────────────────────────────────
```

this is similar in form to TARGET_PROCESSING.

The three boxed constructs are particular forms of a Z structure called a **schema**. Schemas can be manipulated with a set of rules defined in the Z schema calculus (reference 1). Thus in terms of the three schemas defined above the specification of the data flow diagram will be of the form

```
(TAKE and TARGET_PROCESSING)or(TAKE and WEATHER_PROCESSING)
```

which indicates the alternative paths through the flow diagram: it will be seen in the next chapter this does not capture exactly what is required.

The more general form of a schema contains two parts, the first part called the signature which groups together declarations, and a second part called the 'predicate' which enables constraints to be placed on the variables defined through the declarative signature part. The above schemas consist of only a signature, each with only one declaration; the predicate part is omitted as no constraints have been defined.

Example schemas
An example of a more general schema, not connected with the data flow specification, is the representation of a simple record of personal identity number with limits placed on the values of the identity number :-

```
P_IDENTITY──────────
│ Name     :TEXT
│ Ident_No :ℤ
├───────────────────
│ Ident_No >0
│ Ident_No <1000000
└───────────────────
```

In general, the upper part of a schema contains declarations separated by line breaks and/or semicolons: no meaning is to be attached to order of the declarations. The lower part contains predicates (constraints) which are also separated by line breaks and/or semicolons: the predicates so separated are considered to be conjoined (i.e. the 'and' is implicit).

There is an alternative horizontal form in which only semicolons are used as separators; in this form the above example is

```
P_IDENTITY ≙ [Name :TEXT; Ident_No :ℤ |Ident_No >0; Ident_No <1000000 ]
```

where the symbol ≙ can be read as "is defined as", and the vertical line symbol | as "such that".

Schema names can be used in various ways. One common use is in the declaration part of another schema, for example in the personal record

```
P_RECORD────────────
│ P_IDENTITY
│ Salary :ℤ
├─────────────────
│ Salary >0
└──────────────────
```

is equivalent to

```
P_RECORD────────────
│ Name  :TEXT
│ Ident_No  :ℤ
│ Salary  :ℤ
├─────────────────
│ Salary >0
│ Ident_No >0
│ Ident_No <1000000
└──────────────────
```

Some Naming Conventions

The definition used in specifying the data flow diagram will need to be refined as more details about the radar processing activity are obtained. For instance the `CORRELATION_DATA_CHANNEL` will be modelled as a variable called `Correlation_Data_Channel` which is a sequence of `RADAR_PLOT`:-

```
CORRELATION_DATA_CHANNEL─────────────────────
│ Correlation_Data_Channel: seq RADAR_PLOT
└────────────────────────────────────────────
```

or equivalently

```
CORRELATION_DATA_CHANNEL≙ [Correlation_Data_Channel: seq RADAR_PLOT]
```

Many operations read and/or change the state database. Certain conventions are used to indicate this. Δ as the first character of a schema name usually indicates an operation may read and change the indicated part of the state, while Ξ or ≡ indicates the operation may read but will not write to the state.

For instance, in defining an operation that may read and alter the CORRELATION_DATA_CHANNEL buffer, such as by adding a plot, it will be found convenient to define the schema ΔCORRELATION_DATA_CHANNEL by

```
ΔCORRELATION_DATA_CHANNEL──────────────────────────────
│ Correlation_Data_Channel, Correlation_Data_Channel'
│                            :seq RADAR_PLOT
└──────────────────────────────────────────────────────
```

here the dashed character ' is used to indicate the state after an operation. The more usual way to define this schema is

```
ΔCORRELATION_DATA_CHANNEL──────────────────────────────
│ CORRELATION_DATA_CHANNEL, CORRELATION_DATA_CHANNEL'
└──────────────────────────────────────────────────────
```

The characters ? and ! are also appended to a name to indicate an input variable and an output variable respectively. Names are case sensitive so for example TAKE is different from take.

4 THE Z SPECIFICATION (tutorial form)

4.1 Specification Structure

The formal Z specification is based on a model in which the activity is treated as a sequence of operations which access the state variables (the database as represented by the data Pools and Channels shown in figure 1). Note the RADAR_PLOT is a local data storage and is not part of the state. 'reject_plot_channel' is a 'waste bin' and is only used explicitly in the example specification of the previous chapter.

4.2 First Level Specification

The first level specification decomposes the operations and data objects of the data flow diagram; the degree of decomposition has been chosen to ensure that the procedures in the body of the Radar Processing activity given in the appendix are revealed. To do this it is necessary to describe the structure of the data areas in a little more detail.

The channels are first-in first-out buffers which are modelled here by sequences of plots. The RADAR_DATA_INPUT channel contains a sequence of plots of type SITE_PLOT from the different radar sites, which in Z can be denoted by seq SITE_PLOT. SITE_PLOT can be thought of as a record, the details of which are of no concern at this level of the specification. The types of plot contained in the other two channel buffers are DISPLAY_PLOT and RADAR_PLOT respectively. All this is captured in the Z notation by

```
[SITE_PLOT, RADAR_PLOT, DISPLAY_PLOT]
```

which introduces three new types, and

```
RADAR_DATA_INPUT          ≙ [Radar_Data_Input: seq SITE_PLOT]
CORRELATION_DATA_CHANNEL≙ [Correlation_Data_Channel:
                                          seq RADAR_PLOT]
PLOT_DISPLAY_CHANNEL      ≙ [Plot_Display Channel:
                                          seq DISPLAY_PLOT]
```

which defines the buffers as sequences. Note for convenience the names of the sequences are the capitalized lower case text ot the buffer names, although other names could have been used.

The RADAR_CONFIGURATION_POOL contains data about the radar sites such as site position, and information about which radar(s) are to be used for tracking in different geographical areas (called radar sort boxes - RSB for short). Details of the pool are given later. This pool is used only as a source of data by the RADAR_PROCESSING activity.

The TIME_POOL is provided to enable the current time to be read.

The Operations

The first part of the activity is to remove a SITE_PLOT from the input buffer and decide if it is a target or weather. The TAKE operation of the high level specification is replaced by the two operations TAKE and TARGET:-

TAKE	. remove site plot from input buffer to local radar plot storage area,
TARGET	. decide if local radar plot is target or not.

If it is a target, the following sequence of operations on the local radar plot replaces the high level TARGET_PROCESSING operation:-

TIME_STAMP	. add a time stamp to plot
REGISTRATION_COLLIMATION	. systematic position errors corrected
R_AZ_FILTER	. reject plot in preset r-az bands
COORD_CONVERT	. position to system coordinates
RSB_FILTER	. plot for correlation or display.

If it is weather then the high level WEATHER_PROCESSING operation is replaced by:-

WEATHER_FILTER	. convert for weather display or reject.

Each of the operations is defined in more detail later.

The RADAR_PROCESSING activity is defined in the Z notation by

```
RADAR_PROCESSING ≙
(TAKE and TARGET) ≫ TIME_STAMP ≫ REGISTRATION_COLLIMATION ≫
R_AZ_FILTER ≫ COORD_CONVERT ≫ RSB_FILTER
or
(TAKE and not TARGET) ≫ WEATHER_FILTER.
```

The symbol $\gg$ is called 'piping'[1] and is used to indicate the output components from one operation being used as input components to the subsequent operation. There is only one component concerned here, namely the local radar plot.

The logic symbols **and**, **or** and **not** are denoted in Z by the symbols $\wedge$, $\vee$ and $\neg$ respectively.

It is not considered profitable to provide any more details on the individual operations at this level because most of them require more information on the data structures before any useful meanings could be specified.

4.3 Second Level Specification

The degree of refinement to reach the second level of specification has been such that the resulting Z specification corresponds quite closely both in structure and detail to that of the defining PDL in the appendix. Some extraneous pieces of Z have had to be added in order that the the Z type checker could be used. The particular type checker used did not have REAL types or common functions such as SQRT so these had to be introduced into the "local library" by the following Z definitions:-

```
[ REAL ]

(_*_), (_-_), (_+_), (_/_) : (REAL×REAL)→REAL
_<_, _>_, _≤_, _≥_      :REAL ↔ REAL
COS, SIN, SQRT :REAL ⇸ REAL
TO_REAL    :ℤ → REAL
TO_INTEGER :REAL → ℤ
```

Note: The version of the type checker used did not allow overloading of operators, so for example, where * appears in this specification meaning multiplication of REALs, the symbol TIMES occurs in the text that was actually processed by the type checker. The Z language forbids overloading of variables (reference 3) but is less clear about overloading of operators.
To avoid having to define too many other operations, the types DISTANCE and TIME were redefined as REAL and integer respectively by

```
DISTANCE ≙ REAL
TIME ≙ ℤ .
```

[1] Piping is used here as a means of indicating sequential operation: it is assumed that if the specification of a component schema in a pipe is not satisfied then the pipe is terminated.

Another problem with Z and hence with the type checker is that names must be defined before they are used. Thus to the type checker, the specification is presented in a bottom-up fashion and the more primitive types defined first. Here the decision has been taken to present the specification in a top down manner so that detailed definition of subcomponents can be delayed.

Types for radar identities and secondary radar code are introduced by
`[ RADAR_ID, SSRCODE ]`
with no further details required.

4.3.1 Refinement of the CHANNELS

There is no refinement of the data channels, however for later convenience
Δ state change schemas are defined.

```
[SITE_PLOT, DISPLAY_PLOT]

RADAR_DATA_INPUT───────────────────
│ Radar_Data_Input: seq SITE_PLOT
└──────────────────────────────────

ΔRADAR_DATA_INPUT──────────────────
│ RADAR_DATA_INPUT, RADAR_DATA_INPUT'
└──────────────────────────────────

CORRELATION_DATA_CHANNEL───────────
│ Correlation_Data_Channel :seq RADAR_PLOT
└──────────────────────────────────

ΔCORRELATION_DATA_CHANNEL──────────
│ CORRELATION_DATA_CHANNEL, CORRELATION_DATA_CHANNEL'
└──────────────────────────────────

PLOT_DISPLAY_CHANNEL───────────────
│ Plot_Display_Channel :seq DISPLAY_PLOT
└──────────────────────────────────

ΔPLOT_DISPLAY_CHANNEL──────────────
│ PLOT_DISPLAY_CHANNEL, PLOT_DISPLAY_CHANNEL'
└──────────────────────────────────
```

4.3.2 Refinement of type RADAR_PLOT

RADAR_PLOT is the data structure type of the local radar plot. A schema for this structure can be obtained immediately from the corresponding PDL record,

```
RADAR_PLOT─────────────────────
  │ Message_Length: ℕ
  │ Site_Data: SITE_DATA
  │ Timestamp: TIME
  │ Corrected_R,X,Y: DISTANCE
  │ Rsb_id: RSB_ID
  │ Plot_Status: PLOT_STATUS
  │ Decay_Time: TIME
  └──────────────────────────────
```

SITE_DATA, RSB_ID, and PLOT_STATUS are the only undefined types. In the PD L SITE_DATA represents either target or weather information and in Z this can be expressed as a disjoint union of types TARGET_DATA and WEATHER_DATA

SITE_DATA::=Target_Data<<TARGET_DATA>>|Weather_Data<<WEATHER_DATA>>,

where
::= can be read as "is a new basic data type defined by",
| denotes an alternative and can be read as or,
<< >> bracket the domain of the preceding function; for example Target_Data is a function mapping TARGET_DATA into SITE_DATA. The inverse function Target_Data^{-1} is used later, for example to check if the Site_Data component of a radar plot is target rather than weather information.

In this definition TARGET_DATA and WEATHER_DATA are defined by the schema types

```
TARGET_DATA──────────────────
  │ Message_Length: ℕ
  │ Receiving_Radar_ID: RADAR_ID
  │ Plot_Type: PLOT_TYPE
  │ Range: ℕ
  │ Azimuth: ℕ
  │ SSR_Code: SSRCODE
  │ Mode_C_Height: ℕ
```

```
| Time_Delay: TIME
| Run_Length: RUN_LENGTH
| Squawk: SQUAWK
|-----------------------------
| Plot_Type ≠ WEATHER
|____________________________
```

and

```
WEATHER_DATA─────────────────
| Message_Length: ℕ
| Receiving_Radar_ID: RADAR_ID
| Plot_Type: PLOT_TYPE
| Azimuth: ℕ
| Start_Range: ℕ
| Stop_Range: ℕ
|-----------------------------
| Plot_Type = WEATHER
|____________________________
```

Note TARGET_DATA is constrained to be not of WEATHER Plot_Type, and WEATHER_DATA is constrained appropriately.

The schemas use the following enumerated types

```
PLOT_TYPE::= SECONDARY_REINFORCED|SECONDARY|PRIMARY|WEATHER
RUN_LENGTH::= SHORT | LONG
SQUAWK::= NONE | IDENT | EMERGENCY
```

This completes the specification of SITE_DATA, leaving RSB_ID and PLOT_STATUS as the remaining undefined types in RADAR_PLOT. These types are

```
RSB_ID───────────────────
| Rsb_X, Rsb_Y : ℕ
|________________________
```

```
PLOT_STATUS::= PREFERRED|SUPPLEMENTARY|NULL
```

Plot_Status can have the status PREFERRED, SUPPLEMENTARY or NULL.

As the specification gets more concrete, constraints, such as limiting Azimuth to lie in the range 0 to 4095, can be added into the schemas.

4.3.3 Refinement of RADAR_CONFIGURATION_POOL

This pool consists of three main components, Station_Data for each radar, and radar_sort_box radar data and status. The corresponding schema types are STATION_DATA, RSB_DATA and RSB_STATUS. The number of radar stations is given by TOTAL_STATIONS and is kept in a schema called SYSTEM_PARAMETERS1.

```
RADAR_CONFIGURATION_POOL────────
│ STATION_DATA
│ Station_Data : ℙSTATION_DATA
│ RSB_DATA
│ RSB_STATUS
│ SYSTEM_PARAMETERS1
├──────────────────────────────
│ #Station_Data =TOTAL_STATIONS
└──────────────────────────────
```

ℙ means the power set, and can be read as "a set of", and # gives the size (number of distinct elements) of the set.

```
SYSTEM_PARAMETERS1──────
│ TOTAL_STATIONS :ℕ
└──────────────────────
```

The components STATION_DATA, RSB_DATA and RSB_STATUS are specified in more detail below. STATION_DATA is

```
STATION_DATA──────────────────
│ Radar_ID: RADAR_ID
│ X_Position, Y_Position :DISTANCE
│ Sweep_Time : TIME
│ Range_Correction :ℤ
│ Azimuth_Correction :ℤ
│ Rho_Theta_Mask : seq RHO_THETA_MASK
│ Weather_Mask :seq LIMITING_RANGE
├─────────────────────────────────
│ # Rho_Theta_Mask = 64
│ # Weather_Mask = 32
└─────────────────────────────────
```

Rho_Theta_Mask is sequence of pairs of integers, one pair for each of 64 azimuth sectors, and each pair is of type

```
RHO_THETA_MASK________
| Minimum_Range  :ℕ
| Maximum_Range  :ℕ
|_____________________
```

Weather_Mask is a sequence of items, one item for each of 32 azimuth sectors, and each item of type

```
LIMITING_RANGE________
| Limiting_Range  :ℕ
|_____________________
```

This leaves RSB_DATA and RSB_STATUS to be considered, both of which are 64 by 64 arrays in the PDL. Here they are modelled by simple functional mappings:

```
RSB_DATA______________________________________
| RSB_Data: ((1..64) x (1..64)) → RSB_DAT
|_____________________________________________
```

where
RSB_DAT ≙ [RSB_Preferred:RADAR_ID;RSB_Supplementary:RADAR_ID].
The notation (1 . . 64) is a convenient way of representing the set {1, 2, . . , 64}.
Note that RSB_Data(i,j) selects the i,j th element of RSB_Data, and RSB_Data(i,j).RSB_Preferred selects the RSB_Preferred component of the i,j th element.

Similarly RSB_STATUS is

```
RSB_STATUS______________________________________
| RSB_Status : ((1..64) x (1..64)) → RSB_STAT
|_______________________________________________
```

where
RSB_STAT ≙ [Status_of_RSB:STAT;Supplementary_Status:SUP]
STAT::= DATA_REQUIRED | NO_DATA
SUP::= ENABLED | DISABLED.

Notes that RSB_Status(i,j) is the i,j th element of type (STAT,SUP). Thus the value of RSB_Status(i,j).Status_of_RSB is either DATA_REQUIRED or NO_DATA.

<u>TIMEPOOL</u>
This a general purpose operation which provides the current time.

```
TIME_POOL────────────
│ time: TIME
└────────────────────
```

4.3.4 <u>Refinement of Operations</u>

A number of preset parameters are required by the operations and these are gathered together in the schema below

```
SYSTEM_PARAMETERS2───
│ MEAN_TRANSMISSION_TIME: TIME
│ TWENTY_NM: DISTANCE
│ R_TO_NM_CONVERSION : REAL
│ MIN_REASONABLE_MODE_C :ℕ
│ MAX_REASONABLE_MODE_C :ℕ
│ DEFAULT_MODE_C :ℕ
│ MODE_C_NM_CONVERSION: REAL
│ ACP_TO_RADIANS :REAL
│ NO_CYCLE_TO_DISPLAY :ℕ
└──────────────────────
```

<u>TAKE</u>
This operation takes a site_plot off the RADAR_DATA_INPUT buffer, reformats it and puts in the working space called plot.

```
TAKE──────────────────────────────────────
│ ΔRADAR_DATA_INPUT
│ site_plot: SITE_PLOT
│ plot! : RADAR_PLOT
│ Site_to_radarplot:SITE_PLOT⇸RADAR_PLOT
├──────────────────────────────────────
│ Radar_Data_Input * <>
│ Radar_Data_Input' ⁀ <site_plot> = Radar_Data_Input
│ plot! = Site_to_radarplot( site_plot)
└──────────────────────────────────────────
```

Here ΔRADAR_DATA_INPUT indicates the input plot buffer may be read and altered. It will be altered by removing a site_plot if one is present, i.e. if the buffer Radar_Data_Input is not equal to the empty sequence ⟨⟩. The site_plot is taken from the end of the sequence; this is ensured by stating that the Radar_Data_Input sequence before the operation is the same as the sequence at the end of the operation with the site_plot joined ($\frown$) to it.
The function Site_to_radarplot converts the SITE_PLOT format into the RADAR_PLOT format. The details are not too important here, however the variables Timestamp, Corrected_R, X and Y are not given values at this stage.
The output variable plot! is the local radar plot referred to in earlier text.

TARGET

This operation checks that the variable plot! is of type TARGET_PLOT.

```
TARGET──────────────────────────
│ plot! : RADAR_PLOT
├───────────────────────────────
│ plot!.Site_Data ∈ ran Target_Data
└───────────────────────────────
```

ran is the range operator, and used here to specify all the possible vales of Target_Data. Note that plot! rather plot? is used because the TARGET schema is used to check the output variable of the TAKE schema (see section 4.2).

TIME_STAMP

```
TIME_STAMP──────────────────────────────
│ TIME_POOL
│ SYSTEM_PARAMETERS2
│ plot?, plot! :RADAR_PLOT
├───────────────────────────────────────
│ plot?.Site_Data ∈ ran Target_Data
│ plot!.Timestamp = time - inplot?.Time_Delay-
│           MEAN_TRANSMISSION_TIME
│ where inplot?≙Target_Data^-1(plot?.Site_Data)
└───────────────────────────────────────
```

The input plot is checked to ensure that it is of type TARGET_DATA and not WEATHER_DATA. Here time is of type TIME and is a component of TIME_POOL, and MEAN_TRANSMISSION_TIME is a preset system parameter. Note the dot notation could be used, as in Site_Data.Time_Delay(plot?). The predicate Site_Data(plot?) $\in$ ran TARGET_DATA
is redundant as far as the whole specification is concerned since it will be propagated in under the piping operation from the TAKE schema. However its inclusion here ensures the TIME_STAMP schema is less dependent on such knowledge. Because this situation occurs with other operations it is convenient to rewrite TIME_STAMP as

```
ΔTARGET─────────────────────────
│ plot?, plot! : RADAR_PLOT
├───────────────────────────────
│ plot?.Site_Data ∈ ran Target_Data
└───────────────────────────────
```

```
TIME_STAMP──────────────────────────────────
│ TIME_POOL
│ SYSTEM_PARAMETERS2
│ ΔTARGET
├───────────────────────────────────────────
│ plot!.Timestamp = time - inplot?.Time_Delay-
│           MEAN_TRANSMISSION_TIME
│ where inplot? ≙ Target_Data^-1(plot?.Site_Data)
└───────────────────────────────────────────
```

A slight liberty with conventional use of Δ is taken since plot is not actually part of the system state, but can be thought of as part of the local Radar_Processing_Activity state.

REGISTRATION_COLLIMATION

This operation removes the small systematic errors from range and azimuth measurements. The output azimuth is MODULO 4096 and is achieved using the Azi_modulo function which is not defined in detail .

```
REGISTRATION_COLLIMATION
 | RADAR_CONFIGRATION_POOL
 | ΔTARGET
 | sd: STATION_DATA
 | Azi_modulo:ℤ→ℕ
 |--------------------------------------------------
 | sd ∈ Station_Data
 | sd.Radar_ID = inplot?.Receiving_Radar_ID
 | outplot!.Range = inplot?.Range+sd.Range_Correction
 | outplot!.Azimuth = Azi_modulo(Az_corrected)
 | where
 | inplot? ≙ Target_Data⁻¹(plot?.Site_Data)
 | outplot! ≙ Target_Data⁻¹(plot!.Site_Data)
 | Az_corrected≙inplot?.Azimuth + sd.Azimuth_Correction
 |__________________________________________________
```

The requirement sd ∈ Station_Data ensures that the radar station that is being referred to, is actually in the set of active radar stations kept in Station_Data.

R_AZ_FILTER

This operation rejects a plot if its range does not lie between limits defined in the RADAR_CONFIGURATION_POOL. Rejection is achieved if the predicate of the schema evaluates to false.

```
R_AZ_FILTER
 | RADAR_CONFIGURATION_POOL
 | ΔTARGET
 | sd : STATION_DATA
 |--------------------------------------------------
 | sd ∈ Station_Data
 | sd.Radar_ID = inplot?.Receiving_Radar_ID
 | inplot?.Range > Filter.Minimum_Range
 | inplot?.Range < Filter.Maximum_Range
 | plot! = plot?
 | where
 |    inplot?≙Target_Data⁻¹(plot?.Site_Data)
 |    mask_region≙inplot?.Azimuth div 64 +1
 |    Filter≙sd.Rho_Theta_Mask(mask_region)
 |__________________________________________________
```

The underlying assumption is an azimuth in 12 bits (0-4095) and mask_region is in the range 1..64.

<u>COORD_CONVERT</u>

This operation makes a correction for slant range, and converts the plot position to system coordinates, i.e.

```
COORD_CONV ≙ slant_range_correction ⨟ coordinate_conversion
```

No slant range correction is made to plots with ranges of 20 nautical mile and greater. The type of slant range correction used below 20 nautical miles depends on whether or not a valid secondary radar height is available. The ModeC_OK schema is used as a predicate (i.e. as boolean procedure would be used in a programming language) to indicate the validity of the secondary radar height.

```
ModeC_OK──────────────────────────────────
│ SYSTEM_PARAMETERS2
│ ΔTARGET
├──────────────────────────────────────────
│ ( inplot?.Plot_Type = SECONDARY_REINFORCED
│    ∨  inplot?.Plot_Type = SECONDARY )
│ inplot?.Mode_C_Height ≥ MIN_REASONABLE_MODE_C
│ inplot?.Mode_C_Height ≤ MAX_REASONABLE_MODE_C
│ where  inplot?≙Target_Data⁻¹(plot?.Site_Data)
└──────────────────────────────────────────

slant_range_correction────────────────────
│ ΔTARGET
│ SYSTEM_PARAMETERS2
│ ModeC_OK
├──────────────────────────────────────────
│ r? ≥ TWENTY_NM and cr! = (r? * R_TO_NM_CONVERSION)
│ ∨
│ not(r? ≥ TWENTY_NM) ∧ ModeC_OK ∧ cr!=convert1
│ ∨
│ not(r? ≥ TWENTY_NM) ∧ not(ModeC_OK)∧cr!=convert2
│ where
│     inplot?≙Target_Data⁻¹(plot?.Site_Data)
│    r?≙TO_REAL(inplot?.Range)
│    cr!≙plot!.Corrected_R
```

```
|    a≙ (r?*R_TO_NM_CONVERSION)
|   b≙ (TO_REAL(inplot?.Mode_C_Height)*MODE_C_NM_CONVERSION)
|    c≙ (TO_REAL(DEFAULT_MODE_C)*MODE_C_NM_CONVERSION)
|    convert1≙ SQRT( (a*a) - (b*b) )
|    convert2≙ SQRT( (a*a) - (c*c) )
|_____________________________________________
```

```
coordinate_conversion──────────────────
| RADAR_CONFIGURATION_POOL
| ΔTARGET,
| SYSTEM_PARAMETERS2
| sd:STATION_DATA
|──────────────────────────────
| sd ∈ Station_Data
| sd.Radar_ID = inplot?.Receiving_Radar_ID
| plot!.X= (cr?*SIN(az?)) + sd.X_Postion
| plot!.Y= (cr?*COS(az?)) + sd.Y_Position
| where inplot?≙Target_Data⁻¹(plot?.Site_Data)
|       cr?≙plot?.Corrected_R
|       az?≙TO_REAL(inplot?.Azimuth)*ACP_TO_RADIANS
|________________________________
```

RSB_FILTER

This operation uses the x,y coordinates of the plot to find the corresponding radar sort box coordinates. If the plot's radar matches one of the radars (indicated as preferred or supplementary) associated with the sort box and the appropriate status values are set the plot will be put on the correlation buffer. If no match occurs the plot will be rejected. If a match occurs but the appropriate status values are not set, the plot is sent to the display buffer. The operation consists of two main parts as defined in

RSB_FILTER ≙ calculate_rsb ≫ rsb_status_filter.

The rsb_status_filter corresponds to a complex IF..ENDIF statement in the main body of the radar processing activity PDL.

```
calculate_rsb──────────────────
| RADAR_CONFIGURATION_POOL
| ΔTARGET
| TO_RSB_ID : REAL⇸ℕ
```

```
plot!.Rsb_id.Rsb_X = x?
plot!.Rsb_id.Rsb_Y = y?
(plot!.Plot_Status=PREFERRED ∧
     plotradar?=rsb_data.RSB_Preferred )
∨
(plot!.Plot_Status=SUPPLEMENTARY ∧
      plotradar?=rsb_data.RSB_Supplementary)
∨
(plot!.Plot_Status=NULL ∧
     not (plotradar?=rsb_data.RSB_Preferred) ∧
     not (plotradar?=rsb_data.RSB_Supplementary) )

where inplot?≙Target_Data⁻¹(plot?.Site_Data)
    x?≙TO_RSB_ID(plot?.X / TO_REAL(16))+1
    y?≙TO_RSB_ID(plot?.Y / TO_REAL(16))+1
    rsb_data≙RSB_Data(x?,y?)
    plotradar?≙inplot?.Receiving_Radar_ID
```

```
To_Correlation─────────────────────
 ΔCORRELATION_DATA_CHANNEL
 plot? :RADAR_PLOT
 ─────────────────────────
 Correlation_Data_Channel' =
  Correlation_Data_Channel ⁀ <plot?>
```

```
To-Display──────────────────────────
 ΔPLOT_DISPLAY_CHANNEL
 plot? :RADAR_PLOT
 Display_Form :RADAR_PLOT ⇸ DISPLAY_PLOT
 ──────────────────────────────
 Plot_Display_Channel' = Plot_Display_Channel⁀
                         <Display_Form(plot?)>
```

```
rsb_status_filter──────────────────────────────
 │ RADAR_CONFIGURATION_POOL
 │ To_Correlation
 │ To_Display
 ├──────────────────────────────
 │ (To_Correlation ∧ RSB=on∧ plot?.Plot_Status=PREFERRED)
 │ ∨
 │ (To_Correlation ∧ RSB=on∧ plot?.Plot_Status=SUPPLEMENTARY
 │               ∧ rsb_status?.Supplementary_Status=ENABLED)
 │ ∨
 │ (To_Display    ∧ not(RSB=on)∧ plot?.Plot_Status=PREFERRED)
 │ ∨
 │ (To_Display ∧ not(RSB=on)∧ plot?.Plot_Status=SUPPLEMENTARY
 │               ∧   rsb_status?.Supplementary_Status=ENABLED)
 │    where
 │    x? ≙ plot?.Rsb_id.Rsb_X
 │    y? ≙ plot?.Rsb_id.Rsb_Y
 │    rsb_status? ≙ RSB_Status(x?,y?)
 │    RSB ≙ rsb_status?.Status_of_RSB
 │    on ≙ DATA_REQUIRED
 └──────────────────────────────
```

Note: `Display_form` converts the plot to a form that can be put on the display buffer. In the original PDL, a plot is also put on the display buffer if an attempt to put it on the correlation buffer fails because the buffer is full: that possibility is not considered here.

WEATHER_FILTER

The first part decides if the weather data is needed and adjusts the values if necessary. The second part sends the data, after setting a display rate parameter, to the display buffer.

```
    WEATHER_FILTER≙ weather_filter ≫ weather_display
```

The main objective of `weather_filter` is to ensure the values of the `Start_Range` and `Stop_Range` components do not exceed a specified limit. If `Stop_Range` exceeds the limit it is reset to the limit. The preset value of limit depends both on the radar site concerned and on the azimuth sector in which the plot lies.

```
weather_filter──────────────────────────────────────────
│ RADAR_CONFIGURATION_POOL
│ plot?,plot! :RADAR_PLOT
│ sd:STATION_DATA
├──────────────────────────────────────────────────
│ plot?.Site_Data ∈ ran Weather_Data
│ sd ∈ Station_Data
│ sd.Radar_ID=weather?.Receiving_Radar_ID
│ weather?.Start_Range ≤ limit?
│ (weather?.Stop_Range ≤ limit?)∨(weather!.Stop_Range=limit?)
│ where weather?≙Weather_Data^{-1}(plot?.Site_Data)
│       weather!≙Weather_Data^{-1}(plot!.Site_Data)
│       mask?≙weather?.Azimuth div 128 + 1
│       limit?≙(sd.Weather_Mask(mask?)).Limiting_Range
└──────────────────────────────────────────────────────────
```

```
weather_display──────────────────────────────────────────────
│ RADAR_CONFIGURATION_POOL
│ To_Display
│ plot! :RADAR_PLOT
│ sd:STATION_DATA
│ SYSTEM_PARAMETERS2
├──────────────────────────────────────────────────────────
│ plot?.Site_Data ∈ ran Weather_Data
│ sd ∈ Station_Data
│ sd.Radar_ID=weather?.Receiving_Radar_ID
│ plot!.Decay_Time=NO_CYCLE_TO_DISPLAY*sd.Sweep_Time
│ To_Display
│ where weather?≙Weather_Data^{-1}(plot?.Site_Data)
└──────────────────────────────────────────────────────────
```

NO_CYCLE_TO_DISPLAY is a system parameter.

5 DISCUSSION

5.1 Learning Z

One of the main objectives of the study was to gain experience in using Z: the other being to provide an initial assessment of Z's applicability to Air Traffic Control systems. Familiarity with set theory and a programming language such as Pascal should enable some basic understanding of Z without difficulty. The author attended an introductory course on formal methods which included a small amount on Z and followed the Alvey funded self-learning course on discrete mathematics and specification (reference 5). British Telecom (reference 4) consider that to become proficient at using Z, an intensive introductory course, an understanding of the underlying discrete mathematics, and several months practical experience is required.

One problem in learning Z while trying to use it, is judging how much one needs to know in order both to make a reasonable start on a specification and finally to complete it. The training discussed proved to be sufficient to enable an initial Z specification to be produced. This initial attempt then underwent a number of corrective and restructuring iterations. The most significant of these came as a result of a partial "walk-through" with Z experts from CC1 division at RSRE and by using their Z-type checker. The type checker proved to be an invaluable assistant in the learning process.

One consequence of not being experienced in Z was that it was sometimes difficult to choose the appropriate Z constructs in which to express well understood requirements. A couple of examples illustrate the problem: How are the notions of sequential operations and an infinite flow control loop expressed. As described in chapter 4, the sequential operation was solved using "piping" operations; the infinite loop remains a problem. Fortunately only a few of the mathematical symbols peculiar to Z were needed in the specification work, but even so initial encounters are likely to be off-putting.

5.2 Applying Z

The approach to producing a Z specification of the radar processing activity, described in the appendix, was the simple one of converting each PDL data structure and each PDL procedure/function in turn into a Z schema. Thus the overall structure of the radar processing activity remained essentially unaltered, resulting in a state based specification with a set of operations transforming one state of the database into another. The actual methods for producing the Z specification were dictated, to some extent, by having to learn Z while producing the specification. Small chunks of the PDL were selected as exercises in Z, until it was believed that all the different structures used in the PDL could be handled with confidence. This meant that some of the level-two specification had been completed before the level-one specification. In practice, it was found easier to tackle the data structures first, since in many cases, it was easy to construct a Z schema with a simple and exact one-to-one correspondence with a PDL data structure; indeed many of the Z data schemas look like (allowing for small notational differences) Pascal records. The only difficulties encountered were in representing a data item which could have different pre-defined structures and in representing a square array. In the former case a Z construct for a disjoint union provided the solution. In the case of the array - neither Z or the 'standard' Z library has built in array types - ' a sequence of sequences' was adopted initially but this was changed to matrix type when the mechanism for defining generic types in Z was better understood: after comments from CC1 division a much simpler mapping (see the schema RSB_DATA in section 4.3.3) was adopted for this report that also maintains a closer correspondence with the PDL. Neither difficulty would be a problem for an experienced Z user.

In converting the PDL procedures to Z, one of the main problems was unraveling the nested conditional IF statements. The basic technique used to convert these statements into Z was to replace the nested statements by a sequence of simpler IF statements. For example

```
  IF A THEN B ELSEIF C THEN D
```

would be replaced by

```
  IF A THEN B
  IF not A and C THEN D.
```

In this form only one of the conditions (the term after the IF) evaluates to **true** so, in effect, the two statements are 'or-ed' together. The Z equivalent will be of the form

```
      (A and B) or (notA and C and D)
```

or using mathematical notation $(A \wedge B) \vee (\neg A \wedge C \wedge D)$.
Other minor problems which arose included that of finding out how to refer to items in certain complex data structures, and of having to introduce type conversion routines and standard mathematical functions because these were not in the Z library. One error (of omission) - was found in the PDL; the PDL to calculate radar sort boxes refers to a plot_status of NULL, a value which had not been defined.
Although not part of the main exercise, Z was also applied in a top down approach by specifying the data flow diagram of the radar processing activity.

5.3 Tool Support

The only tool support used during the development of the first version of the Z specification was a Macintosh computer - bit graphics, mouse, and windows - with a set of Z fonts accessible to all Macintosh word processors that obey the standard operating system guide-lines. This has been the only tool used for document preparation.
Later versions of the specification were subject to a syntax and type checker developed in the CC1 division at RSRE. The checker was written in Algol68 and ran under a research operating system on an ICL Perq computer. Again the computer had bit graphics, mouse and windows. The type checker could

handle a proper Z specification consisting of both English text and Z mathematics, although only mathematical parts of the radar processing specification were entered.
The reliability and functionality of the tool were good, and it proved invaluable during the specification process. Only one error was encountered in the type checker; it accepted a 'macro' clause of the form **where** $x \mathrel{\hat{=}} (A = B)$ which is not allowed. A few irritations were caused because overloading of operators was not allowed, for example since the symbol $>$ had already been defined as applying to INTEGER, another name had to be used to mean 'greater than' when applied to REAL. The associated Z text editor, while handling all the Z notation and symbology, was very awkward to use on some of its editing operations particularly when compared with text editors on the Macintosh.

6 CONCLUSIONS

This initial application of the formal specification language Z to an Air Traffic Problem has been extremely valuable and instructive. Z rather than VDM (the other leading model-based formal specification language in the UK) was chosen because there was experience and expertise at RSRE, Z appeared to have better structuring facilities for large specifications, and the available tool support seemed marginally better (some being at hand). The application involved converting a Radar Processing program based on the one used in the LATCC multi-radar processing system and defined by an algorithmic Program Design Language (PDL).

6.1 Using Z

Some important points observed while learning and applying Z are listed below.

Learning Z
1 Formal training in some form is required for writing Z specifications.
2 Having experts at hand is almost essential for a beginner.
3 Tool support is invaluable.

Converting PDL to Z
1 Some techniques and guide-lines were established for converting from PDL to Z.
2 Data structures were usually the easiest to map.
3 Z requires data to be typed, although details of the type need not be revealed unless needed in subsequent Z specification.
4 Declaration of standard mathematical and type conversions are explicitly required.
5 Control flow achieved by the Z 'piping' operation.
6 Walk-throughs essential - this implies at least more than one person is needed to complete Z specifications.
7 Tool support invaluable.
8 Since the PDL did not explicitly handle concurrency, there were no problems in this area with the Z.

Tool Support
1 No Z syntax/type checkers were commercially available at the time, only text editors.
2 A commercial text editor with Z fonts on a Macintosh computer was used for document preparation. The use of dot matrix printers was cumbersome and very slow with relatively poor quality, although a laser printer was used for the final printings.
3 A syntax and type checker developed and owned by another division in RSRE was used. This is a research tool, although a more portable version is under construction.
4 The research tool used did not support top down development.

5 A Z type checker has been produced by the Alvey FORSITE project but attempts to obtain copies during this work and subsequently have been unsuccessful.
6 Since completing this work, two Z type checkers originating from the Programming Research Group at Oxford have been mentioned in Z bulletin boards, one programmed in the functional language ML, and the other called FUZZ which processes Z specifications written in the type setting language Latex for SUN and IBM PC.

6.2 Benefits To NATS

The benefits of using formal methods are claimed to be

1 Clear thought with improved ability to specify with precision and conciseness.
2 Improved precision, consistency, unambiguity, and completeness.
3 A precise notation in which specifications can be reliably communicated to others.
4 The use of refinement techniques to produce implementations that accurately satisfy their specifications and to assist in validation of requirements, and hence enable manuals related to implementation to reflect the specifications accurately.
5 The use of animation techniques to assist in the validation of the semantic aspects of specifications.
6 A basis for informal and formal reasoning and analysis about correctness, resulting in detection and elimination of errors.

These benefits are, with varying degrees of importance, relevant to various NATS procurement activities from analysis and requirements, through contract monitoring and acceptance to maintenance. The application described in this report addresses the areas of detailed requirements and design, although many of the lessons can be extended to other phases of the life cycle. The exercise confirms the first three benefits listed above. The Z specification is clearer and more precise than the PDL definition on which it is based. It is a better base for a requirements specification because it describes 'What is required' rather than 'How it is be achieved'. Thus the PDL should be thought of as a refinement of Z, although in this instance the WHAT was obtained by reverse engineering from the HOW.
At least for the particular areas of requirements and design covered in the exercise, it is considered formal specification could be of value to NATS in producing a better rapport with contractors through increased clarity and precision of specifications and leaving less room for argument. The improved communications should also increase visibility and increase confidence in acceptance of deliverables. The resulting use of precise and unambiguous specification should also be of benefit in subsequent maintenance.

The benefits of validation through animation, implementation through refinement techniques, and the value of reasoning methods will be addressed in subsequent work.

Application of formal methods is in general hampered by the training needs and the availability of good tools. Training and tools should not be seen as a great barriers in the future, since good software engineering courses will include formal methods and tools are just about entering the market place. Although not key issues in the work reported, consideration needs to be given to where formal specification methods can be best used and how they are to be integrated into the other system development methods and tools being used. One possibility is to use Z, or similar specification language, to specify the functionality of the actions/processes of methods using data flow diagrams; this approach is being adopted by BAe for development of avionics software (reference 6).

Finally, it is worth noting the encouraging claims of 80% reduced maintenance effort, 250% productivity gains and a sixfold decrease in software integration times reported in the NCC video course on 'Towards Formal Methods'.

6.3 Future Work

Apart from the example of a Z specification of a data flow diagram, which can be considered as an embryonic method for top down development, the specific Z radar processing specification given here is more design than requirements because of the close mapping from the PDL. The next stage of the work will be directed more to the Requirements phase. A more abstract version of the Z specification given here will be produced. This will be followed with a versions that will make use of the knowledge of those Requirements not captured in the specifications derived from the PDL and will be closer to that which would developed in a top down approach.

ACKNOWLEDGEMENTS

The author would like thank Dr.C.T.Sennet and his section, particularly R.McDonald and A.Smith, for their help in clarifying some of the Z constructs and for their assistance in the use of their Z syntax and type checker.

REFERENCES

1. Detailed Design of the Demonstration System, P.A.Barrett, RMCS 1049/TD.4, December 1986.

2. Specification Case Studies, I.Hayes (editor), Prentice-Hall, 1987.

3. The Z-package, Oxford University Computing Laboratory, Programming Research Group, 1987. Consists of a number of documents updated regularly to reflect the state-of-the-art. The first document has been published, see reference 9.

4. Z (a formal specification method), A Debrief Report, DTI STARTS (Software Tools for Application to large Real Time Systems), M.Norris (British Telecom).

5. Essential Mathematics for Software Engineers, IEE 1987.

6. Embedding Formal Methods in SAFRA, A.Bradley, Agard conference on Software Engineering and its Applications to Avionics, proceedings no.439, 1988.

7 An approach to animating Z using PROLOG, R.D.Knott and P.J.Krause, Department of Mathematics, University of Surrey, Report A1.1, Alvey SE/065, July 1988.

8 Understanding Z: a specification language and its formal semantics, J.M.Spivey, CUP, 1988.

9 The Z Notation - A Reference Manual, J.M.Spivey, Prentice-Hall, 1989.

A1 Program Design Language definition of the RADAR PROCESSING

INTRODUCTION

This a slightly shortened extract from reference 1 (1049/TD.4 Dec86) of the design of the Radar Processing Activitity in PDL. Diagrams and the PDL of the module display plot have been excluded. The formats of relevant Pool and Channels have been added.

3.1. RADAR PROCESSING ACTIVITY

This activity carries out the functions of multi-radar processing on plot data received from the radar stations. A number of masking techniques are used to separate out data which is not required for further processing, data which is not required for correlation but which should be displayed as plots on the radar screens, and data which is required for attempted correlation. The three types of data are, respectively, discarded, placed into the Plot Display Channel for processing by the Console Handler subsystem, or placed into the Correlation Data Channel for processing by the Correlation activity.

Two main masking techniques are applied. Firstly, each radar station has associated with it a geographical mask defining the area of coverage of that station. Plots received from a station which fall outside its area of coverage are discarded.

Secondly, the airspace is divided into 16 nautical mile squares, known as Radar Sort Boxes (RSB's). Each RSB has two bits of information associated with it: whether data from that box is required for correlation, and whether the supplementary radar site for the RSB is enabled. The information is maintained by the Tracking activity based on expected track positions (if returns from an aircraft being tracked could appear in a particular RSB, that RSB is enabled) and continuity of radar returns.

Radar processing is based around 'radar plot's. These are instances of the following data structure:

```
Radar Plot
    -  Message Length            (RADAR PLOT LENGTH)
    -  Site Data
          Target Data | Weather Data
    -  Timestamp
    -  Corrected Range
    -  X Coordinate
    -  Y Coordinate
    -  RSB ID
          -  RSB X
          -  RSB Y
    -  Plot Status               (PREFERRED | SUPPLEMENTARY)
    -  Decay Time
```

Target Data

- Message Length	(TARGET DATA MSG LGTH)
- Receiving Radar ID	
- Plot Type	(SECONDARY REINFORCED | SECONDARY | PRIMARY)
- Range	(Rho)
- Azimuth	(Theta)
- SSR Code	
- Mode C Height	
- Time Delay	(between receipt and transmission by radar)
- Run Length	(primary only - SHORT | LONG)
- Squawk	(NONE | IDENT | EMERGENCY)

Weather Data

- Message Length	(WEATHER DATA MSG LGTH)
- Receiving Radar ID	
- Plot Type	(WEATHER)
- Azimuth	(Theta)
- Start Range	
- Stop Range	

'Site Data' is the information supplied by the radar stations, while the rest (much of which is unused in the processing of weather data) is calculated during radar processing.

Structured English Description of the Radar Processing Activity

```
RADAR PROCESSING ACTIVITY

acquire circular channel (RADAR DATA INPUT, OUTPUT)
acquire circular channel (CORRELATION DATA CHANNEL, INPUT)
loop while TRUE
        take (RADAR DATA INPUT CHANNEL, radar plot.site data)
        if radar plot.site data.plot type <> WEATHER
            timestamp (TIME POOL, radar plot)
            radar plot.decay time = RADAR CONFIGURATION POOL.station data[radar plot.site data.receiving radar id].sweep time
            apply registration and collimation correction (RADAR CONFIGURATION POOL, radar plot)
            apply rho-theta filtering (RADAR CONFIGURATION POOL, radar plot, result)
            if result <> REJECT PLOT
                apply slant range correction (radar plot)
                apply coordinate conversion (RADAR CONFIGURATION POOL, radar plot)
                calculate rsb (RADAR CONFIGURATION POOL, radar plot)
                if   radar plot.plot status = PREFERRED
                or (radar plot.plot status = SUPPLEMENTARY
                    and RADAR CONFIGURATION POOL.rsb status[radar plot.rsb id.rsb x,
                                radar plot.rsb id.rsb y].supplementary status = ENABLED)
                    if RADAR CONFIGURATION POOL.rsb status[radar plot.rsb id.rsb x,
                                radar plot.rsb id.rsb y ].status of rsb = DATA REQUIRED
                        place (CORRELATION DATA CHANNEL, radar plot, result)
                        if  result = FAIL
                            display plot (PLOT DISPLAY CHANNEL, radar plot)
                        end if
                    else
                        display plot (PLOT DISPLAY CHANNEL, radar plot)
                    end if
                end if
            end if
        else
            apply weather filter (RADAR CONFIGURATION POOL, radar plot, result)
            if result <> REJECT PLOT
radar plot.decay time = NO CYCLES TO DISPLAY * RADAR CONFIGURATION POOL.station
                                data[radar plot.site data.receiving radar id].sweep time
                display plot (PLOT DISPLAY CHANNEL, radar plot)
            end if
        end if
end loop

end RADAR PROCESSING ACTIVITY.
```

3.1.1 Timestamping

Each radar plot is timestamped as the first stage of its processing. This timestamp is calculated from the current time, the delay between receipt and transmission at the radar station (contained in the data from the radar station) and an estimate of the mean delay between transmission by the radar station and receipt by the Radar Processing activity.

```
module:    timestamp    (TIME POOL,  radar plot)

read current time  (TIME POOL,  time)
radar plot.timestamp = time - radar plot.site data.time delay - MEAN TRANSMISSION TIME

return
```

'read current time' is a standard access mechanism of the TIME POOL.

3.1.2 Registration and Collimation Correction

The range and azimuth (rho, theta) values of each radar plot are corrected for any alignment errors which may have been detected in the receiving radar station. Correction takes the form of the addition of correction values to range and azimuth.

Azimuth values are expressed in units of ACPs from North, where 4096 ACPs make up a circle. Range is expressed in units of 1/16 of a nautical mile.

```
module:  apply registration and collimation correction (RADAR CONFIGURATION POOL, radar plot)

          radar plot.site data.range  =  radar plot.site data.range  +  RADAR CONFIGURATION
POOL.station data[radar plot.site data.receiving radar id].error data.range correction
          radar plot.site data.azimuth  =  radar plot.site data.azimuth  +  RADAR CONFIGURATION
POOL.station data[radar plot.site data.receiving radar id].error data. azimuth correction
          if   radar plot.site data.azimuth  <  0
             radar plot.site data.azimuth  =  radar plot.site data.azimuth  +  4096
          else
             if  radar plot.site data.azimuth  >  4096
                radar plot.site data.azimuth  =  radar plot.site data.azimuth  -  4096
             end if
          end if

return.
```

3.1.3 Rho-Theta Filtering

Rho-Theta filtering is used to reduce the processing load on the system by rejecting as many as possible of the plots which, for a given radar, are from neither preferred nor supplementary sites of a Radar Sort Box (RSB).

In practice, each mask consists of 64 equal azimuth intervals (of 64 ACP's), with the minimum and maximum ranges to be considered for each. This is represented within the system as a series of tables, one per radar, stored in the RADAR CONFIGURATION POOL. The filtering operation consists of a simple check of plot range against the range limits of the appropriate region. Plots falling outside the range limits are discarded.

```
module:   apply rho-theta filtering    (RADAR CONFIGURATION POOL, radar plot, result)

          mask region = integer (radar plot.site data.azimuth / 64) + 1
          if radar plot.site data.range < RADAR CONFIGURATION POOL.station data[radar plot.site data.
                                                   receiving radar id].rho-theta mask[mask
                                                   region].minimum range
             result = REJECT PLOT
          else
             if radar plot.site data.range > RADAR CONFIGURATION POOL.station data[radar plot.site data.
                                                   receiving radar id].rho-theta mask[mask
                                                   region].maximum range
                result = REJECT PLOT
             else
                result = ACCEPT PLOT
             end if
          end if

return.
```

3.1.4 Slant Range Correction

Slant range correction involves adjusting plot range information to take into account the height of the aircraft involved. Slant range correction is not applied to plots with a range of greater than 20 nautical miles, as the correction then becomes insignificant. Two types of slant range correction are applied, according to the type of radar return concerned.

Conversion factors are incorporated into the algorithm to give the corrected range in units of Nautical Miles (NM). (N.B. Rho is in 1/16 of a NM, Mode C Height is in feet.) The constant DEFAULT MODE C HEIGHT IN NM is in units of NM.

```
module:   apply slant range correction    (radar plot)

          if radar plot.site data.range ≥ TWENTY NM
             radar plot.corrected range = radar plot.site data.range * RHO TO NM CONVERSION
FACTOR
          else
             if  (radar plot.site data.plot type = SECONDARY REINFORCED
             or   radar plot.site data.plot type = SECONDARY
                 and  (radar plot.site data.mode c height ≥ MIN REASONABLE MODE C
                 and   radar plot.site data.mode c height ≤ MAX REASONABLE MODE C)
radar plot.corrected range  =  square root of  (square  (radar plot.site data.range *
                                                   RHO TO NM CONVERSION FACTOR) /
                               square  (radar plot.site data.mode c height *
                                                   FEET TO NM CONVERSION FACTOR))
             else
radar plot.corrected range  =  square root of  (square  (radar plot.site data.range *
                                                   RHO TO NM CONVERSION FACTOR) /
                                                  square (DEFAULT MODE C HEIGHT  IN NM))
             end if
          end if

return.
```

3.1.5 Coordinate Conversion

The plot data received from the radar stations contains positional information in the form of range and azimuth values calculated in relation to the site of the station. These must be converted into the system of axes used by the system (x,y). Trigonometry, and a knowledge of the (x,y) coordinates of each radar station, are used to achieve this.

```
module:  apply coordinate conversion   (RADAR CONFIGURATION POOL,  radar plot)


degrees = radar plot.site data.azimuth * DEGREES IN AN ACP

radar plot.x coordinate  =   RADAR CONFIGURATION POOL.station data[radar plot.site
data.receiving radar id].x position + (radar plot.corrected range * sin (degrees))

radar plot.y coordinate  =   RADAR CONFIGURATION POOL.station data[radar plot.site
data.receivingradar id].y position + (radar plot.corrected range * cos (degrees))

return.
```

3.1.6 Calculate RSB

Radar Sort Boxes (RSBs) are fixed 16 NM squares covering the FIR in a grid of 64 by 64 boxes. The x and y values in the grid of the sort box in which a particular plot is located may be obtained by a simple translation of the plot (x,y) coordinates. The identity of preferred and supplementary radars for that RSB may then be obtained from a look-up table in the Radar Configuration Pool which associates RSBs and radar stations.

```
module:   calculate rsb   (RADAR CONFIGURATION POOL, radar plot)

          radar plot.rsb id.rsb x = integer (radar plot.x coordinate / 16) + 1
          radar plot.rsb id.rsb y = integer (radar plot.y coordinate / 16) + 1
          if radar plot.site data.receiving radar id = RADAR CONFIGURATION POOL.rsb data[radar plot.rsb id.rsb x, radar plot.rsb id.rsb y].rsb preferred
              radar plot.plot status = PREFERRED
          else
              if radar plot.site data.receiving radar id = RADAR CONFIGURATION POOL.rsb data[radar plot.rsb id.rsb x, radar plot.rsb id.rsb y].rsb supplementary
                  radar plot.plot status = SUPPLEMENTARY
              else
                  radar plot.plot status = NULL
              end if
          end if

return.
```

3.1.7 Weather Map Filtering

Weather map messages are subjected to a different type of rho-theta filtering from radar target messages. A weather rho-theta filter mask consists of 32 equal segments (of 128 ACP's), each of which has associated with it a limiting value. Weather map messages whose range start value is greater than the limiting value are discarded. Those whose range stop value is greater than the limiting value have their range stop values replaced by the limiting value. Those whose range stop value is less than the limiting value are accepted unaltered. Weather rho-theta filter masks are stored, like target rho-theta filters, in the RADAR CONFIGURATION POOL.

Weather map messages are not generated for every sweep of the radar, thus each weather map message is displayed for a time equivalent to MAP DISPLAY SWEEPS sweeps of the receiving radar.

```
module:   apply weather filter   (RADAR CONFIGURATION POOL, radar plot, result)

        mask region = integer (radar plot.site data.azimuth / 128) + 1
        if radar plot.site data.range start > RADAR CONFIGURATION POOL.station data[radar plot.site data.receiving radar id].weather mask[mask region].limiting range
            result = REJECT PLOT
        else
            if radar plot.site data.range stop > RADAR CONFIGURATION POOL.station data[radar plot.site data.receiving radar id].weather mask[mask region].limiting range
                radar plot.site data.range stop = RADAR CONFIGURATION POOL.station data[radar plot.site data.receiving radar id].weather mask[mask region].limiting range
            end if
            result = ACCEPT PLOT
        end if

return.
```

4.1.2 RADAR CONFIGURATION POOL

The Radar Configuration Pool contains the information on radar stations and Radar Sort Boxes (RSB's) which is used by the Radar Processing activity in the filtering and processing of radar plots. The data is in two parts:

a) Radar station data. For each radar station, data is stored on site identity, position, sweep-time, error correction values and the geographical rejection mask filters used in initial data selection processing.

b) Radar Sort Box data. For each RSB, the identity of the preferred and supplementary radar sites for that RSB, and the RSB status (whether data is required for correlation, whether the supplementary site is enabled) are stored. The RSB status information is maintained by the Tracking activity on the basis of expected track positions and continuity of radar *returns*.

The format of the Radar Configuration Pool is as follows:

Radar Configuration Pool

```
        -  Control Queue

        -  Station Data [1..TOTAL STATIONS]
              -  Radar ID
              -  X Position
              -  Y Position
              -  Sweep Time
              -  Error Data
                    -  Range Correction
                    -  Azimuth Correction
              -  Rho-Theta Mask [1..64]
                    -  Minimum Range
                    -  Maximum Range
              -  Weather Mask [1..32]
                    -  Limiting Range

        -  RSB Data [1..64,1..64]
              -  RSB Preferred
              -  RSB Supplementary

        -  RSB Status [1..64,1..64]
              -  Status of RSB              (DATA REQUIRED | NO DATA)
              -  Supplementary Status       (ENABLED | DISABLED)
```

A2 Z Glossary

Symbols used in chapters 3 and 4:

[ ]	Introduces new types
$\mathbb{Z}$	The integers; postitive, zero and negative
$\mathbb{N}$	The natural natural numbers, non negative integers
:	"of type"
$\rightarrowtail$	Function (partial)
$\rightarrow$	Function (total)
$\leftrightarrow$	Relation, i.e. a set of ordered pairs
$\times$	Cartesian product, denotes ordered pairs
$\vert$	"such that"
;	Conjunction, "and"
seq	Sequence
Δ	Indicates the associated state may change
Ξ	Indicates the associated state does not change
'	Denotes variable after an operation
$=$	Equality
$\neq$	Inequality
?	Input variable
!	Output variable
$\gg$	'Piping' operation
_	Anonymous generic parameter
$\widehat{=}$	"defined as"
::=	"is a new type defined as"
$\vert$	Exclusive or: disjoint union
$\langle\!\langle$ $\rangle\!\rangle$	Domain or type of preceding function
$^{-1}$	Inverse function
#	Number of distinct elements in set
$\langle\rangle$	Empty sequence
$\frown$	Sequence joining symbol
$\in$	"is a member of"
$\wedge$	Logical and
$\vee$	Logical or
$\neg$	Logical not
$\mathbb{P}$	Power set: read as "a set of"

A3 THE Z SPECIFICATION

1 Introduction

This appendix gives the specification of the RADAR PROCESSING activity in a non-tutorial form. The formal Z parts are presented in the order that a typical type checker would expect, i.e. definitions of named items such as variables and schemas must be given before they are used. The supporting English text has been kept to a minimum to help keep down the length of the report; normally a Z specification would be expected to contain a full description of the requirements. The specification is not exactly that which passed through RSRE type checker because overloading of the standard operators such as '+' has been allowed here, a simpler model for arrays than the one put through the type checker is used, and some errors may have been introduced by having to re-type on a different computer.

2 Specification Structure

The RADAR PROCESSING activity carries out the initial radar plot processing. It takes a digitized radar plot, which has come from one of a number of surveillance radars, from an input buffer and checks to see if the plot is from an aircraft target or if it corresponds to a weather strobe.
If it is a target plot then a time stamp is added and its position is converted into a common coordinate system. Subject to certain criteria which are defined later, a plot is then
a) thrown away as not required, or
b) sent for correlation with a track, or
c) sent for display.

If it is a weather plot, it is either
a) thrown away, or
b) sent for display.

The data flow diagram of the activity is shown in figure 1 (see chapter 3) which is used as a model for the Z specification. After processing a plot, the activity repeats the processing with the next plot. The informal specification of the activity is given by the PDL in Appendix 1.

In Z, the RADAR_PROCESSING activity structure will be defined by the schema following expression which corresponds to the data flow diagram in figure 1:

```
RADAR_PROCESSING ≙
(TAKE and TARGET) » TIME_STAMP » REGISTRATION_COLLIMATION »
R_AZ_FILTER » COORD_CONVERT » RSB_FILTER
or
(TAKE and not TARGET) » WEATHER_FILTER.
```

3 The Z Specification

3.1 Local Z library

Some extraneous pieces of Z have had to be added in order that the the Z type checker could be used. In particular REAL types, type conversions and standard mathematical functions form a "local library" :-

```
[ REAL ]
```

```
| (_*_), (_-_), (_+_), (_/_) : (REAL×REAL)→REAL
| _<_, _>_, _≤_, _≥_    :REAL ↔ REAL
| COS, SIN, SQRT :REAL ⇸ REAL
| TO_REAL   :ℤ → REAL
| TO_INTEGER :REAL → ℤ
```

The types `DISTANCE` and `REAL` are redefined for ease of type checking

```
DISTANCE ≙ REAL
TIME ≙ ℤ
```

3.2 Plot Formats

Types for the plots from the radar sites (from the RADAR DATA INPUT buffer to be precise) and for displaying are defined by

```
[SITE_PLOT, DISPLAY_PLOT]
```

and types for radar identities and secondary radar code are

```
[ RADAR_ID, SSRCODE ]
```

The following enumerated types occur in plot messages

```
PLOT_TYPE::= SECONDARY_REINFORCED|SECONDARY|PRIMARY|WEATHER
RUN_LENGTH::= SHORT | LONG
SQUAWK::= NONE | IDENT | EMERGENCY
```

Plots from the radar sites will be either target data or weather data defined by the following record schemas

```
TARGET_DATA────────────
| Message_Length: ℕ
| Receiving_Radar_ID: RADAR_ID
| Plot_Type: PLOT_TYPE
| Range: ℕ
| Azimuth: ℕ
| SSR_Code: SSRCODE
| Mode_C_Height: ℕ
| Time_Delay: TIME
| Run_Length: RUN_LENGTH
| Squawk: SQUAWK
|--------------------------------
| Plot_Type • WEATHER
|--------------------------------
```

and

```
WEATHER_DATA----------------------
| Message_Length: ℕ
| Receiving_Radar_ID: RADAR_ID
| Plot_Type: PLOT_TYPE
| Azimuth: ℕ
| Start_Range: ℕ
| Stop_Range: ℕ
|--------------------------------
| Plot_Type = WEATHER
|--------------------------------
```

Note `TARGET_DATA` is constrained to be not of `WEATHER` and `WEATHER_DATA` is constrained to be of type `WEATHER`.

Plots are taken from the input buffer and converted into a local plot format `RADAR_PLOT` which is defined below. Radar sort box identifiers and plot status variable are required and these are given by

```
RSB_ID-------------------
| Rsb_X, Rsb_Y : ℕ
|------------------------
```

```
PLOT_STATUS::= PREFERRED|SUPPLEMENTARY|NULL
```

The local plot format has to cater for the different structures of target and weather data. This is accomplished by defining a disjoint union type

```
SITE_DATA::=Target_Data<<TARGET_DATA>>|Weather_Data<<WEATHER_DATA>>,
```

where, for example, `Target_Data` is a function mapping `TARGET_DATA` into `SITE_DATA`. The inverse function Target_Data^{-1} is used later to check if the `Site_Data` component of a radar plot is target rather than weather information.

```
RADAR_PLOT──────────────────
│ Message_Length: ℕ
│ Site_Data: SITE_DATA
│ Timestamp: TIME
│ Corrected_R,X,Y: DISTANCE
│ Rsb_id: RSB_ID
│ Plot_Status: PLOT_STATUS
│ Decay_Time: TIME
└───────────────────────────
```

3.3 RADAR_CONFIGURATION_POOL

This pool consists of three main components, information about each radar, radar identities associated with radar sort boxes, and status information about the sort boxes. The corresponding schema types are STATION_DATA, RSB_DATA and RSB_STATUS. The number of radar stations is given by TOTAL_STATIONS and is kept in a schema called SYSTEM_PARAMETERS1.

```
SYSTEM_PARAMETERS1────────
│ TOTAL_STATIONS :ℕ
└─────────────────────────
```

STATION_DATA contains a range-theta mask for subsequent sector filtering of plots and a range based mask for subsequent weather filtering. Rho_Theta_Mask is sequence of pairs of integers, one pair for each of 64 azimuth sectors, and each pair is of type

```
RHO_THETA_MASK──────
│ Minimum_Range :ℕ
│ Maximum_Range :ℕ
└───────────────────
```

Weather_Mask is a sequence of items, one item for each of 32 azimuth sectors, and each item of type

```
LIMITING_RANGE──────
│ Limiting_Range :ℕ
└───────────────────
```

```
STATION_DATA────────────────────
│ Radar_ID: RADAR_ID
│ X_Position, Y_Position :DISTANCE
│ Sweep_Time : TIME
│ Range_Correction :ℤ
│ Azimuth_Correction :ℤ
```

```
| Rho_Theta_Mask : seq RHO_THETA_MASK
| Weather_Mask :seq LIMITING_RANGE
|-----------------------------------
| # Rho_Theta_Mask = 64
| # Weather_Mask = 32
|-----------------------------------
```

This leaves RSB_DATA and RSB_STATUS to be considered, both of which are 64 by 64 arrays in the PDL. Here they are modeled by simple functional mappings:

```
RSB_DAT ≙ [RSB_Preferred:RADAR_ID;RSB_Supplementary:RADAR_ID].

RSB_DATA-------------------------------------
| RSB_Data: ((1..64) x (1..64)) → RSB_DAT
|--------------------------------------------
```

and

```
STAT::= DATA_REQUIRED | NO_DATA
SUP::= ENABLED | DISABLED.
RSB_STAT ≙ [Status_of_RSB:STAT;Supplementary_Status:SUP]

RSB_STATUS-------------------------------------
| RSB_Status : ((1..64) x (1..64)) → RSB_STAT
|----------------------------------------------
```

```
RADAR_CONFIGURATION_POOL-----------
| STATION_DATA
| Station_Data : ℙSTATION_DATA
| RSB_DATA
| RSB_STATUS
| SYSTEM_PARAMETERS1
|----------------------------------
| #Station_Data =TOTAL_STATIONS
|----------------------------------
```

ℙ means the power set, and can be read as "a set of", and # gives the size (number of distinct elements) of the set.

3.4 The I/O Channels

The data channels are modeled as sequences:

```
RADAR_DATA_INPUT──────────────────
| Radar_Data_Input: seq SITE_PLOT
└─────────────────────────────────

ΔRADAR_DATA_INPUT─────────────────
| RADAR_DATA_INPUT, RADAR_DATA_INPUT'
└─────────────────────────────────

CORRELATION_DATA_CHANNEL──────────
| Correlation_Data_Channel :seq RADAR_PLOT
└─────────────────────────────────

ΔCORRELATION_DATA_CHANNEL─────────────────────────
| CORRELATION_DATA_CHANNEL, CORRELATION_DATA_CHANNEL'
└─────────────────────────────────────────────────

PLOT_DISPLAY_CHANNEL──────────────
| Plot_Display_Channel :seq DISPLAY_PLOT
└─────────────────────────────────

ΔPLOT_DISPLAY_CHANNEL─────────────────────
| PLOT_DISPLAY_CHANNEL, PLOT_DISPLAY_CHANNEL'
└────────────────────────────────────────
```

3.5 TIMEPOOL

This a general purpose operation which provides the current time.

```
TIME_POOL──────────
| time: TIME
└──────────────────
```

3.6 Operations

A number of preset parameters are required by the operations and these are gathered together in the schema below

```
SYSTEM_PARAMETERS2─────
| MEAN_TRANSMISSION_TIME: TIME
| TWENTY_NM: DISTANCE
| R_TO_NM_CONVERSION : REAL
| MIN_REASONABLE_MODE_C :ℕ
```

```
│ MAX_REASONABLE_MODE_C :ℕ
│ DEFAULT_MODE_C :ℕ
│ MODE_C_NM_CONVERSION: REAL
│ ACP_TO_RADIANS :REAL
│ NO_CYCLE_TO_DISPLAY :ℕ
└──────────────────────────
```

3.6.1 TAKE
This operation takes a site_plot off the RADAR_DATA_INPUT buffer, reformats it and puts in the working space called plot.

```
TAKE──────────────────────────────────
│ ΔRADAR_DATA_INPUT
│ site_plot: SITE_PLOT
│ plot! : RADAR_PLOT
│ Site_to_radarplot:SITE_PLOT⇸RADAR_PLOT
├──────────────────────────────────
│ Radar_Data_Input ≠ <>
│ Radar_Data_Input' ⁀ <site_plot> = Radar_Data_Input
│ plot! = Site_to_radarplot( site_plot)
└──────────────────────────────────
```

It will be altered by removing a site_plot if one is present, i.e. if the buffer Radar_Data_Input is not equal to the empty sequence <>. The site_plot is taken from the end of the sequence; this is ensured by stating that the Radar_Data_Input sequence before the operation is the same as the sequence at the end of the operation with the site_plot joined (⁀) to it. The function Site_to_radarplot converts the SITE_PLOT format into the RADAR_PLOT format. The details are not important here.
The output variable plot! is the local radar plot referred to in earlier text.

3.6.2 TARGET
This operation checks that the variable plot! is of type TARGET_PLOT.

```
TARGET──────────────────────────
│ plot! : RADAR_PLOT
├──────────────────────────
│ plot!.Site_Data ∈ ran Target_Data
└──────────────────────────
```

```
ΔTARGET─────────────────────────
│ plot?, plot! : RADAR_PLOT
├──────────────────────────
│ plot?.Site_Data ∈ ran Target_Data
└──────────────────────────
```

ran is the range operator, and used here to specify all the possible vales of Target_Data. Note that plot! rather than plot? is used because the TARGET schema is used to check the output variable of the TAKE schema (see section 4.2).

3.6.3 TIME_STAMP

The input plot is checked to ensure that it is of type TARGET_DATA and not WEATHER_DATA. Here time is of type TIME and is a component of TIME_POOL, and MEAN_TRANSMISSION_TIME is a preset system parameter.

```
TIME_STAMP──────────────────────────────
│ TIME_POOL
│ SYSTEM_PARAMETERS2
│ ΔTARGET
├───────────────────────────────────────
│ plot!.Timestamp = time - inplot?.Time_Delay-
│          MEAN_TRANSMISSION_TIME
│ where inplot?≙Target_Data^{-1}(plot?.Site_Data)
└───────────────────────────────────────
```

A slight liberty with conventional use of Δ is taken since plot is not actually part of the system state, but it can be thought of as part of the local Radar Processing state.

3.6.4 REGISTRATION_COLLIMATION

This operation removes the small systematic errors from range and azimuth measurements. The output azimuth is MODULO 4096.

```
REGISTRATION_COLLIMATION────────────────────
│ RADAR_CONFIGRATION_POOL
│ ΔTARGET
│ sd: STATION_DATA
│ Azi_modulo:ℤ→ℕ
├───────────────────────────────────────────
│ sd ∈ Station_Data
│ sd.Radar_ID = inplot?.Receiving_Radar_ID
│ outplot!.Range = inplot?.Range+sd.Range_Correction
│ outplot!.Azimuth = Azi_modulo(Az_corrected)
│ where
│ inplot? ≙ Target_Data^{-1}(plot?.Site_Data)
│ outplot! ≙ Target_Data^{-1}(plot!.Site_Data)
│ Az_corrected≙inplot?.Azimuth + sd.Azimuth_Correction
└───────────────────────────────────────────
```

The requirement sd ∈ Station_Data ensures that the radar station being referred to, is actually in the set of active radar stations kept in Station_Data.

3.6.5 R_AZ_FILTER

This operation rejects a plot if its range does not lie between limits defined in the RADAR_CONFIGURATION_POOL. Rejection is achieved if the predicate of the schema evaluates to false.

```
R_AZ_FILTER──────────────────────────────
│ RADAR_CONFIGURATION_POOL
│ ΔTARGET
│ sd : STATION_DATA
├─────────────────────────────────────────
│ sd ∈ Station_Data
│ sd.Radar_ID = inplot?.Receiving_Radar_ID
│ inplot?.Range > Filter.Minimum_Range
│ inplot?.Range < Filter.Maximum_Range
│ plot! = plot?
│ where
│    inplot?≙Target_Data⁻¹(plot?.Site_Data)
│    mask_region≙inplot?.Azimuth div 64 +1
│    Filter≙sd.Rho_Theta_Mask(mask_region)
└─────────────────────────────────────────
```

The underlying assumption is an azimuth in 12 bits (0-4095) and mask_region is in the range 1..64.

3.6.6 COORD_CONVERT

This operation makes a correction for slant range and converts the plot position to system coordinates.

```
ModeC_OK──────────────────────────────────
│ SYSTEM_PARAMETERS2
│ ΔTARGET
├──────────────────────────────────────────
│ ( inplot?.Plot_Type = SECONDARY_REINFORCED
│    ∨  inplot?.Plot_Type = SECONDARY )
│ inplot?.Mode_C_Height ≥ MIN_REASONABLE_MODE_C
│ inplot?.Mode_C_Height ≤ MAX_REASONABLE_MODE_C
│ where inplot?≙Target_Data⁻¹(plot?.Site_Data)
└──────────────────────────────────────────
```

```
slant_range_correction─────────────────────
 │ ΔTARGET
 │ SYSTEM_PARAMETERS2
 │ ModeC_OK
 ├─────────────────────────────────
 │ r? ≥ TWENTY_NM and cr! = (r? * R_TO_NM_CONVERSION)
 │ ∨
 │ not(r? ≥ TWENTY_NM) ∧ ModeC_OK ∧ cr!=convert1
 │ ∨
 │ not(r? ≥ TWENTY_NM) ∧ not(ModeC_OK)∧cr!=convert2
 │ where
 │     inplot?≙Target_Data⁻¹(plot?.Site_Data)
 │     r?≙TO_REAL(inplot?.Range)
 │     cr!≙plot!.Corrected_R
 │     a≙ (r?*R_TO_NM_CONVERSION)
 │     b≙ (TO_REAL(inplot?.Mode_C_Height)*MODE_C_NM_CONVERSION)
 │     c≙ (TO_REAL(DEFAULT_MODE_C)*MODE_C_NM_CONVERSION)
 │     convert1≙ SQRT( (a*a) - (b*b) )
 │     convert2≙ SQRT( (a*a) - (c*c) )
 └─────────────────────────────────────────
```

```
coordinate_conversion─────────────────────
 │ RADAR_CONFIGURATION_POOL
 │ ΔTARGET,
 │ SYSTEM_PARAMETERS2
 │ sd:STATION_DATA
 ├─────────────────────────────────
 │ sd ∈ Station_Data
 │ sd.Radar_ID = inplot?.Receiving_Radar_ID
 │ plot!.X= (cr?*SIN(az?)) + sd.X_Postion
 │ plot!.Y= (cr?*COS(az?)) + sd.Y_Position
 │ where inplot?≙Target_Data⁻¹(plot?.Site_Data)
 │      cr?≙plot?.Corrected_R
 │      az?≙TO_REAL(inplot?.Azimuth)*ACP_TO_RADIANS
 └─────────────────────────────────────────
COORD_CONV ≙ slant_range_correction ⨟ coordinate_conversion
```

3.6.7 RSB_FILTER

This operation uses the x,y coordinates of the plot to find the corresponding radar sort box coordinates. If the plot's radar matches one of the radars (indicated as preferred or supplementary) associated with the sort box and the appropriate status values are set the plot will be put on the correlation buffer. If no match occurs the plot will be rejected. If a match occurs but the

appropriate status values are not set, the plot is sent to the display buffer.
The operation consists of two main parts as defined in the schemas `calculate_rsb` and `rsb_status_filter`.

The `rsb_status_filter` corresponds to a complex IF..ENDIF statement in the main body of the radar processing activity PDL.

```
calculate_rsb──────────────────────────────
│ RADAR_CONFIGURATION_POOL
│ ΔTARGET
│ TO_RSB_ID : REAL⇸ℕ
├──────────────────────────────────────────
│ plot!.Rsb_id.Rsb_X = x?
│ plot!.Rsb_id.Rsb_Y = y?
│ (plot!.Plot_Status=PREFERRED ∧
│      plotradar?=rsb_data.RSB_Preferred )
│ ∨
│ (plot!.Plot_Status=SUPPLEMENTARY ∧
│       plotradar?=rsb_data.RSB_Supplementary)
│ ∨
│ (plot!.Plot_Status=NULL ∧
│      not (plotradar?=rsb_data.RSB_Preferred) ∧
│      not (plotradar?=rsb_data.RSB_Supplementary) )
│
│ where inplot?≙Target_Data⁻¹(plot?.Site_Data)
│     x?≙TO_RSB_ID(plot?.X / TO_REAL(16))+1
│     y?≙TO_RSB_ID(plot?.Y / TO_REAL(16))+1
│     rsb_data≙RSB_Data(x?,y?)
│     plotradar?≙inplot?.Receiving_Radar_ID
└──────────────────────────────────────────────────
```

Operation to put plots on the correlation and display buffers are needed and given by

```
To_Correlation─────────────────────
│ ΔCORRELATION_DATA_CHANNEL
│ plot? :RADAR_PLOT
├──────────────────────────────────
│ Correlation_Data_Channel' =
│  Correlation_Data_Channel ⁀ <plot?>
└──────────────────────────────────────
```

and

```
To-Display─────────────────────────────
│ ΔPLOT_DISPLAY_CHANNEL
│ plot? :RADAR_PLOT
│ Display_Form :RADAR_PLOT ⇸ DISPLAY_PLOT
├──────────────────────────────────────
│ Plot_Display_Channel' = Plot_Display_Channel⁀
│                        <Display_Form(plot?)>
└──────────────────────────────────────
```

```
rsb_status_filter──────────────────────────
│ RADAR_CONFIGURATION_POOL
│ To_Correlation
│ To_Display
├──────────────────────────────────────
│ (To_Correlation ∧ RSB=on∧ plot?.Plot_Status=PREFERRED)
│ ∨
│ (To_Correlation ∧ RSB=on∧ plot?.Plot_Status=SUPPLEMENTARY
│              ∧ rsb_status?.Supplementary_Status=ENABLED)
│ ∨
│ (To_Display    ∧ not(RSB=on)∧ plot?.Plot_Status=PREFERRED)
│ ∨
│ (To_Display ∧ not(RSB=on)∧ plot?.Plot_Status=SUPPLEMENTARY
│               ∧   rsb_status?.Supplementary_Status=ENABLED)
│    where
│    x? ≙ plot?.Rsb_id.Rsb_X
│    y? ≙ plot?.Rsb_id.Rsb_Y
│    rsb_status? ≙ RSB_Status(x?,y?)
│    RSB ≙ rsb_status?.Status_of_RSB
│    on ≙ DATA_REQUIRED
└──────────────────────────────────────
```

Note: Display_form converts the plot to a form that can be put on the display buffer. In the original PDL, a plot is also put on the display buffer if an attempt to put it on the correlation buffer fails because the buffer is full: that possibility is not considered here.

```
RSB_FILTER ≙ calculate_rsb ≫ rsb_status_filter.
```

3.6.8 WEATHER_FILTER

The first part decides if the weather data is needed and adjusts the values if necessary. The second part sends the data, after setting a display rate parameter, to the display buffer.

```
weather_filter─────────────────────────────
│ RADAR_CONFIGURATION_POOL
│ plot?,plot! :RADAR_PLOT
│ sd:STATION_DATA
├──────────────────────────────────────
```

```
| plot?.Site_Data ∈ ran Weather_Data
| sd ∈ Station_Data
| sd.Radar_ID=weather?.Receiving_Radar_ID
| weather?.Start_Range ≤ limit?
| (weather?.Stop_Range ≤ limit?)∨(weather!.Stop_Range=limit?)
| where  weather?≙Weather_Data⁻¹(plot?.Site_Data)
|        weather!≙Weather_Data⁻¹(plot!.Site_Data)
|        mask?≙weather?.Azimuth div 128 + 1
|        limit?≙(sd.Weather_Mask(mask?)).Limiting_Range
|______________________________________________________
```

```
weather_display_______________________________________
| RADAR_CONFIGURATION_POOL
| To_Display
| plot! :RADAR_PLOT
| sd:STATION_DATA
| SYSTEM_PARAMETERS2
|______________________________________________________
| plot?.Site_Data ∈ ran Weather_Data
| sd ∈ Station_Data
| sd.Radar_ID=weather?.Receiving_Radar_ID
| plot!.Decay_Time=NO_CYCLE_TO_DISPLAY*sd.Sweep_Time
| To_Display
| where  weather?≙Weather_Data⁻¹(plot?.Site_Data)
|______________________________________________________
```

```
WEATHER_FILTER≙ weather_filter ≫ weather_display
```

3.7 The Radar Processing

Finally the radar processing activity is specified in terms of the above operations by the schema expression:-

```
RADAR_PROCESSING ≙
(TAKE and TARGET) ≫ TIME_STAMP ≫ REGISTRATION_COLLIMATION ≫
R_AZ_FILTER ≫ COORD_CONVERT ≫ RSB_FILTER
or
(TAKE and not TARGET) ≫ WEATHER_FILTER.
```

DOCUMENT CONTROL SHEET

Overall security classification of sheet ....UNCLASSIFIED....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference MEMO 4280	3. Agency Reference	4. Report Security U/C Classification

5. Originator's Code (if known) 7784000	6. Originator (Corporate Author) Name and Location ROYAL SIGNALS & RADAR ESTABLISHMENT ST ANDREWS ROAD, GREAT MALVERN, WORCESTERSHIRE WR14 3PS
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title APPLICATION OF Z TO THE SPECIFICATION OF AIR TRAFFIC CONTROL SYSTEMS, 1

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials SIMCOX L N	9(a) Author 2	9(b) Authors 3,4...	10. Date 1989.4	pp. ref. VP

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract This report describes an initial investigation into the formal specification language Z and its applicability to Air Traffic Control Systems. The software corresponding to the initial radar plot processing in the multi-radar automatic tracking system at the London Air Traffic Control Centre (LATCC) was used in the investigation. An informal pseudo code description of the radar plot processing function was taken as the 'requirements' and converted into a formal specification in the Z language. The specification was partly validated using an RSRE Z syntax and type checking tool. The experiences gained during the exercise are discussed and potential benefits for the Civil Aviation Authority are highlighted.

S80/48