AD-A204 946

RSRE

MEMORANDUM No. 4114

ROYAL SIGNALS & RADAR ESTABLISHMENT

THE APPLICATION OF SOFTWARE FAULT TOLERANCE TO AIR TRAFFIC CONTROL: STUDY CONTRACT OVERVIEW

Author: L N Simcox

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

[illegible]

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4114

THE APPLICATION OF SOFTWARE FAULT TOLERANCE TO AIR TRAFFIC CONTROL: STUDY CONTRACT OVERVIEW

June 1988

Author: L N Simcox

SUMMARY

As part of its software engineering research for the Civil Aviation Authority (CAA), the ATC systems research division at RSRE initiated a study contract on the application of software fault tolerance to ATC systems. This memorandum gives an overview of a study contract in which a requirements analysis of the London Air Traffic Control Centre (LATCC) was developed using the CORE method with automated support. This was then taken as the basis for a MASCOT design of the LATCC Radar Data Processing subsystem. This design was converted into one with the software fault tolerant features of Recovery Blocks in a MASCOT-like environment.

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
	J Scott	DP1a
	B Bradford	RD1

This memorandum reflects the views of the author. It is not to be regarded as a final or official statement by the UK Civil Aviation Authority.

Work done under contract to the UK Civil Aviation Authority

THE APPLICATION OF SOFTWARE FAULT TOLERANCE TO AIR TRAFFIC CONTROL: STUDY CONTRACT OVERVIEW

CONTENTS

FIGURES

1 INTRODUCTION

Modern Air Traffic Control (ATC) systems are dependent on highly reliable computer based systems. This dependency is increasing with the growing volume of air traffic, making the control task after a system failure more difficult and prone to safety problems. Somewhat paradoxically, the addition of extra functionality to help air traffic Controllers with increased traffic implies even more reliance on the ATC computer system and, in turn, creates the need for even higher reliability and availability on new and evolving systems.

The Civil Aviation Authority (CAA) is aware of these problems and to help it keep both abreast of current techniques and methods, and assess their relevance to current and future projects and plans, it sponsors a research and development programme in various software engineering topics. Part of this programme is carried out in the ATC systems research division at RSRE Malvern and one item in the programme includes an investigation into software fault tolerance as a means of achieving high reliability.

Fault tolerance is about making use of component or information redundancy and is one of two approaches to achieving reliability in systems. The other approach is fault prevention which is concerned with using methods, techniques and technologies that aim to avoid introducing faults into the implementation, including removal of faults found during testing. Specifically software fault tolerance is about using software redundancy to minimise the effects of software design faults, recognizing that complete fault prevention is beyond the state of the art. In some instances, software redundancy may provide masking of hardware malfunction but that is not normally its prime purpose.

The RSRE research programme on software fault tolerance was mapped out to follow two lines of research:-

1 Study actual applications and existing research,

2 Study the applicability to Air Traffic Control (ATC) systems.

This report provides a summary of work on the second topic carried out under contract by RMCS-Cranfield. A more detailed account of the work is given in the contractors final report (reference 1f). The other aspects of the investigation into software fault tolerance will be covered in a subsequent RSRE memorandum (number 4237).

2 BACKGROUND

Initial investigations into software fault tolerance revealed that a significant amount of theoretical work had been carried out but that practical aspects had been confined largely to experiments of an academic nature on small scale software. There were some exceptions to this situation, two of which are worth mentioning here. The first is the use of functionally equivalent, but separately designed, software for the control of flaps and slats in the A310 aircraft (references 12,13): this is one of the few examples of software fault tolerance application in a real operational system. The second example involved a much more complex piece of software in an experimental Naval command and control system using the software fault tolerance recovery blocks techniques (reference 3 and appendix 1). This research was carried out at Newcastle University with the specific aim of evaluating the software fault tolerance techniques developed there. The success of the experiment in terms of increasing reliability, and the similarities between this command and control system and the London ATC Centre (LATCC) system suggested a similar experiment would be valuable in the ATC context.

With limited RSRE resources, discussions with Newcastle University on their possible participation were undertaken but were abandoned because of their resourcing problems at that time. However one of the leading designers of the Newcastle experiment, who was then at RMCS, was approached with the idea and responded with a proposal to study the application of the Newcastle techniques to LATCC. The essence of this proposal was to perform a limited requirements analysis of LATCC from which a subsystem would be selected for application of the fault tolerance design. The study would be conducted with an employee of the MARI company who had also been involved in the the Newcastle work, and was scheduled to last about a year and take about one man-year of effort. A contract based on the proposal was agreed between CAA and RMCS with the funding being shared between the NATS Directorates of Data Processing (now D PE(R&DP)) and the Chief Scientist and work began in October 1985.

An overview the work carried out during the study is given in Chapter 3, and the assessment and recommendations of the contractor are summarised in Chapter 4. For a more detailed account the reader is referred to the technical reports and final report listed in reference 1. The final report also provides a brief account of MASCOT, the CORE requirements method, and the software fault tolerant techniques used in the study. For convenience, a summary of the relevant software fault tolerance principles is provided in Appendix 1 of this memorandum.

3 STUDY ACTIVITIES

This chapter describes the main activities listed below:

1 A limited requirements Analysis of LATCC using the CORE method (reference 8).

2 A MASCOT (reference 7) design of the Radar Data and Flight Plan Processing subsystems.

3 A detailed design of the Radar Data Processing subsystem using a high level Structured English design language.

4 Application of Fault Tolerance,

4.1 Conversion of the Radar Data Processing MASCOT subsystem design into one with fault tolerant features based on Recovery blocks (reference 2 and appendix 1).

4.2 Detailed design of the Recovery Blocks using the same design language as step 3.

Note the contractor's final report breaks the work into eight tasks, but the partitioning chosen here is thought to be more appropriate for this summary document. Appendix 2 gives the nominal distribution of the contractor's effort between the tasks. Some assistance in devising the acceptance tests (para.3.4.2) and in the cost modelling exercise (para.4.1) was provided by the author.

3.1 LATCC CORE analysis

The objective here was to provide the contractor with an overall understanding of LATCC and with a set of sufficiently detailed requirement specifications for those parts of LATCC which would be represented in a trial application of software fault tolerance design techniques. The analysis was carried out following the procedures and guidelines of the CORE method (reference 8). Essentially the CORE method requires the system being analysed to be decomposed into a multi-level hierarchy of viewpoints called the viewpoint structure. Viewpoint is the name used by CORE for what is in effect a logical subsystem. In general, viewpoint structures are not unique. The viewpoint structure arrived at during the CORE analysis of LATCC is given in figure 1.

A viewpoint can then be analysed in terms of its processing actions and data flows to and from other viewpoints. Internal data flows, control of the execution of actions, and time ordering of actions are also determined. Only those viewpoints relevant to the problem in hand need be subjected to such a detailed analysis, and those that are are indicated using a solid outline in the viewpoint structure diagram. A given transaction will normally involve processing by actions from several viewpoints and an analysis of the processing thread can be made to validate the transaction. Because of the large number of transactions that are possible, analysis of such threads is usually limited to performance or reliability critical transactions.

Information gained in this analysis was captured using a graphical workstation with automated support for CORE through a rather immature software package called the Analyst running on an Apple Macintosh computer. The information was obtained by interviews with experienced LATCC staff, supported by observations of the LATCC operational system and by various documents.

A subsidiary objective of the study was to gain experience with the requirements analysis method CORE and an associated commercial computer workbench to support the method. A review of how the CORE method and it support tool were used was conducted under the guidance of experienced CORE users from ARE.

3.2 High level design - MASCOT

The objective of this task was to derive a MASCOT design for a demonstrator system based on the CORE analysis. Initially it appeared that the it might be possible to provide some reasonably mechanical translation from the activities defined in the viewpoints of the CORE analysis to the activities of a MASCOT ACP (activities, channels and pools) diagram. While such a derivation appears to have been achieved by BAe who took into account some implementation considerations in their particular CORE analysis, in this study it was found necessary to obtain a MASCOT design from the information gained during the LATCC analysis rather than directly from the resulting CORE documents. This approach was adopted partly because the LATCC CORE model largely ignored implementation issues which the MASCOT design needed to consider, and partly because of the complexity of the LATCC system. Validation of the MASCOT design was done by checking the correspondence back to the CORE requirements.

MASCOT ACP diagrams for both Flight Plan Processing (FPP) and Radar Data Processing (RDP) were produced. The one for RDP is shown in figure 2 where the interaction with other subsystems is accomplished using MASCOT channels. To minimise interactions between FPP and RDP, the Flight Plan data base informs the RDP (and FPP) subsystems of updates relevant to their processing.

There were only sufficient funds to enable one of the MASCOT subsystems to be designed in detail and the RDP subsystem was selected for its real-time characteristics.

3.3 Detailed design - PDL

The object here was to provide a detailed design of the RDP MASCOT activities and intercommunication data channels. This was done using a pseudo English Program Design Language (PDL) down to a level of detail which would enable an implementation in a high level programming language such as Pascal to be carried out with little more than a syntactic translation. Much of the information for this task had to be obtained from a significant informal analysis since the CORE analysis had not reached the required level of detail. The size of the PDL is indicated in the statistics given in the appendix 3.

3.4 Fault tolerant design

Here the objective was to modify the MASCOT design and the detailed design of the RDP subsystem to include the fault tolerant features of the scheme described in appendix 1. This scheme achieves redundancy by allowing module(s) of an 'alternate' design, but ostensibly to the same specification, to execute when an executing module fails its acceptance test. The primary module, its alternates, and acceptance test is called a recovery block. Before executing an alternate, the System State must be restored to that just before execution of the primary module. The need to restore to such recovery points entails additional coordination constraints on a MASCOT design otherwise, for example, two cooperating activities may never be able to recover to the same consistent state other than the initial system state. To overcome such problems, a common recovery point is enforced for pre-defined groupings of MASCOT activities and IDAs (intercommunication data areas), these groupings are called 'dialogues'. Dialogues may be nested.

3.4.1 High level design - dialogues

The outer-level dialogues for the RDP subsystem were determined from an examination of the interaction of the MASCOT activities, while the inner ones were deduced after deciding which modules within the activities should be made fault tolerant by converting them into recovery blocks. The outer dialogues are used to move subsystem-wide recovery points along in time rather than for recovery block purposes.

3.4.2 Detailed design of recovery blocks.

Two criteria were used in deciding where to place recovery blocks. The first was for complex functions, eg. position and velocity smoothing, and the second was for functions involving significant updating of the Track Table in order to ensure Track Table restoration should corruption occur.

The majority of the recovery blocks were designed on the principle that alternates should be degraded versions of the primary module, i.e. simpler algorithms were used. This is normally satisfactory where frequently refreshed data is involved as in the case of radar data, but for high-integrity data flows involving Flight Plan Data full function alternatives were preferred.

The acceptance tests were devised, as far as possible, to be independent of the algorithm used in the primary and alternate modules. They were simple in form, and many made use of comparisons between old and new track parameters, while others were based on reasonableness checks on values of data items.

The detailed designs of the recovery blocks were produced using PDL.

A review was conducted to validate both the application of fault tolerance techniques and ATC functionality.

4 STUDY ASSESSMENT AND RECOMMENDATIONS

4.1 Fault Tolerance Assessment

The production of both high level and low level designs based on dialogues demonstrated that software fault tolerance techniques could be applied to a LATCC-like system. Issues of software reliability improvements, run-time overheads and costs of building a dialogue machine with its recovery mechanisms, could not be addressed within the limited scope of the contract and a Demonstrator is proposed for this purpose.

It was found that some minor refinements to the original dialogue scheme (reference 4) were needed for this application.

Initial ideas on the interaction of dialogues across large-scale systems indicated further research was needed in this area.

The strategy (see section 3.4.2) for placing recovery blocks was in accord with previous experience (references 3 and 4), but the ability to create effective acceptance tests and alternate modules remains a difficult area for designers of recovery blocks. Analysis of the acceptance tests devised during the contract showed that many of the reasonableness comparisons could be effected using strong typing such as that provided by the Ada programming language. A difficulty encountered in the design of the alternates was that the functional specifications were contained within the PDL algorithms of the primary, and not as separate non-algorithmic specifications. Natural English abstraction was employed to overcome this, although it would have been preferable if the LATCC system functional specifications had been non-algorithmic in the first place.

A comparison of the size of the RDP application software with and without fault tolerance was obtained by comparing the lines of PDL in the two cases:

Lines of PDL without fault tolerance = 2555

Lines of PDL with fault tolerance = 3550,

representing a 39% increase to add fault tolerance features. Assuming an expansion factor of about 15-20% on conversion from PDL to Pascal, costs of the ATC application software were estimated using a cost model. It was emphasised that the proposed Demonstration system would also require a Flight Plan database, Flight Plan processing software, consoles, and a run time environment with a recoverable MASCOT machine.

4.2 CORE assessment

A subsidiary objective of the project was to assess the CORE method for defining ATC requirements, and how CORE specifications are mapped into MASCOT design. Particularly useful features of CORE were its consistency checking and its ability to handle manual functions. It was concluded that the systematic nature and graphical notations of CORE enabled high quality requirement specifications to be produced. However it was felt there was a need for some further refinement of the method, such as how to handle databases, together with better automated support. The System Designer's CORE Analyst package running on Macintosh XL used in the project was considered inadequate both in performance and functionality. Guidance in defining the lower levels of the viewpoint structure would have been helpful.

On the mapping from CORE to MASCOT the view taken was that requirements capture and design should be disjoint tasks. Design validation, therefore, should be based on correspondence checking back to the requirements, and this should be supported in any future integrated CORE/MASCOT environment.

4.3 Recommendations

The six recommendations of the final report are summarised below:-

i) A Demonstrator, based on the work done in the project, should be constructed to evaluate the cost-effectiveness of the technology for the ATC application. Ada should be used.

ii) The application of Formal specification techniques for recovery blocks should be investigated.

iii) The development of an integrated CORE/MASCOT support environment with user-defined consistency checking, preferably integrated with Ada, should be investigated.

iv) An education programme should be set up to educate relevant CAA staff on modern software engineering methods and tools, such as those encountered in this study.

v) Further research on extending the dialogue scheme to large scale systems should be carried out.

vi) CAA should develop their own in 'house standard' for using the CORE method.

5 CONCLUSIONS

The study has been successful in demonstrating, down to a detailed design, how software fault tolerance could be applied to an ATC system. An implementation of this design in the form of a demonstrator would enable a valuable cost benefit evaluation to be made. The study has also produced a requirements analysis of parts of LATCC using the CORE method and this analysis has been a catalyst in CAA adopting the CORE method for some of its study and experimental work. The method could become more widely acceptable both in CAA and MOD if an inexpensive and effective CORE workstation was available. CAA(DP1) has contracted System Designers to provide enhancements to the CORE/Analyst software which, together the new MacintoshII computer, would provide a solution.

The recommendation to integrate CORE and MASCOT has been taken up with a research study being placed by DPE(R&DP) with RMCS to use the AUTOG software (reference 11) as the basis for an integrated environment.

CAA have also agreed in principle to support a study on the use of formal methods as applied software fault tolerance with RMCS. This would complement the work on the application of formal methods to ATC systems already being undertaken by the ATC systems research division at RSRE.

6 ACKNOWLEDGEMENTS

Many individuals contributed to the success of this study. The design reviews were conducted by Mr.C.Young and Mr.A.McDowall of ARE (1st review), and Prof.T.Anderson of Newcastle University and Mr.D.Hatton of LATCC (2nd review). Mr.Hatton also made significant contributions during the detailed design work. After many tribulations, the CORE workstations were procured by the CAA liaison officers and the author. Credit is also due to the contractors, Professor M.R.Moulding (RMCS) and Mr.P.Barrett (MARI) for the additional effort they put into the study.

7 REFERENCES

1. List of Reports from the contract:
a) High-level Analysis of LATCC, RMCS Reference 1049/TD.1, August 1986.
b) Detailed Analysis of LATCC, RMCS Reference 1049/TD.2, October 1986.
c) High-level Design for the Demonstration System, RMCS Reference 1049/TD.3, August 1986.
d) Detailed Design of the Demonstration System, RMCS Reference 1049/TD.4, October 1986.
e) Application of Software Fault Tolerance to Air Traffic Control Systems, RMCS Reference 1049/TD.5, November 1986.
f) Project Final Report, RMCS Reference 1049/TD.6, September 1987.

2. B. Randell, System Structuring for Software Fault Tolerance, IEEE Trans. SE-1 (2), pp. 220-232, 1975.

3. T. Anderson, P.A.Barrett, D.N.Halliwell, and M.R.Moulding, Software Fault Tolerance: An Evaluation, IEEE Trans SE-11(12), Dec. 1985.

4. M. R. Moulding, An Architecture to Support Software Fault Tolerance and an Evaluation of its Performance in a Command and Control Application, Digest IEE Colloquium on Performance Measurement and Prediction, Feb. 1986.

5. B. W. Boehm, Software Engineering Economics, Englewood Cliffs, NJ: Prentice Hall, 1981.

6. P. M. Melliar Smith, Development of Software Fault Tolerance Techniques, NASA Contractor Report 172122, March 1983.

7. MASCOT Suppliers Association, The Official Handbook of MASCOT, RSRE, Malvern, UK, 1980.

8. G. P. Mullery, CORE: Method for Controlled Requirements Expression, Proc. IEEE Fourth International Conference on Software Engineering, New York, 1979.

9. T. Anderson and P. A. Lee, Fault Tolerance: Principles and Practice, Prentice Hall, 1981.

10 H.Hecht and M.Hecht, Software reliability in the system context, IEEE Trans. Software Engineering, SE-12, No.1, January 1986.

11. G.Hemdal and C.Coombs, Softchip Technology: a new architecture for telecommunications and other real time systems, 6th. International Conference on Software Engineering for Telecom. Switching Systems, Eindhoven, April 1986.

12. D.J.Martin, Dissimilar software in high integrity applications in flight controls, AGARD Conference proceedings No.330, 1982, pp. 36-1 to 36-13.

13. A.D.Hills, Digital Fly by Wire, AGARD Lecture Series No.143, 1985.

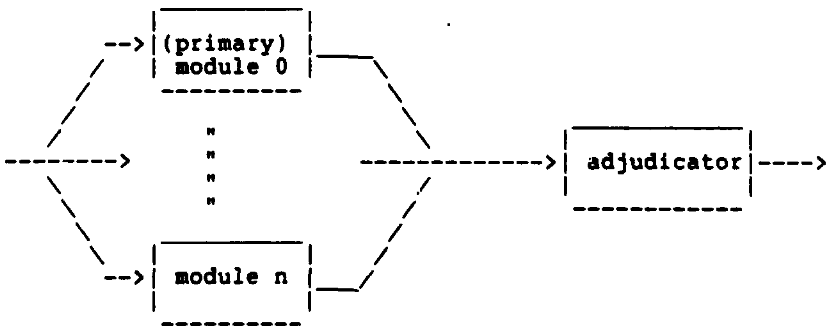
APPENDIX 1

A.1 OVERVIEW OF SOFTWARE FAULT TOLERANCE

Fault tolerance and fault prevention are two approaches to achieving reliability in systems. Fault prevention is concerned with using methods, techniques and technologies that aim to avoid faults existing in the operational system. It is the normal approach to reliability and involves the activities of fault avoidance and fault removal. Fault avoidance tries to exclude faults by use of the appropriate design and construction methods: examples of its use are the selection of reliable components and the adoption of good design methods. Fault removal attempts to locate and remove as many faults as possible by extensive testing, validation and verification.

Of course, fault prevention should not be expected to eradicate all faults, and it may be necessary to construct systems, usually involving redundant components, that tolerate faults and hence prevent system failure. Fault tolerance schemes require a combination of some or all of the activities of error detection, damage confinement, error recovery, and fault removal to reach the required reliability. Such schemes are well established in hardware systems where the physical nature and failure statistics of transients and component ageing are sufficiently understood that reliability predictions can usually be made with some confidence. In contrast, software faults are design faults and a software fault is either present or not present. Thus software does not have transients or wear out in the hardware sense. However, software faults may manifest themselves as errors in a computer system in such a way that the errors have characteristics similar to hardware failures (for example reference 10 describes an ageing characteristic arising through maintenance).

The various schemes for Software-fault tolerance can be described with the help of the diagram below in which the modules are different implementations from the same specification and the adjudicator makes decisions about the results from execution of the modules.



SOFTWARE FAULT TOLERANCE SCHEMATIC

The two main schemes are the recovery blocks scheme, which is relevant to this report and is described below, and the n-version scheme, in which the modules run concurrently and the adjudicator is typically a majority voter.

A.1.1 Recovery Blocks

In the recovery blocks scheme, the modules are executed serially until a module passes the acceptance test that is embodied in the adjudicator. A recovery block will have a syntax of the form

```
ENSURE   acceptance test
BY       primary module
ELSE_BY  1st alternate
..
..
ELSE_BY  nth alternate
ELSE_ERROR.
```

To ensure alternates can execute from the same consistent state as the primary module, the system state is stored on entry to a recovery block, and restored before an alternate is executed. These snapshots of the system state are referred to as recovery points. The concept of nested recovery blocks extends to sequential programs without difficulty.

However, in a system of concurrent interacting processes, recovery presents a problem because recovery of a process will normally include recovery of its interprocess communication data which in turn may force processes that have used or are using this data to recover. This backward recovery will need to continue until a consistent state is reached, and unless recovery is coordinated properly, a "domino" effect is likely and extensive restoration, even back to the initial system state, may occur. Particularly in a real time system such uncoordinated recovery is unacceptable.

A scheme for providing coordinated recovery in a MASCOT based system has been developed (references 3 and 4) in which a common recovery point is provided for predefined subsets of concurrent processes (Activities in MASCOT) and their intercommunicating data areas (IDAs of Pools and Channels in MASCOT). These predefined subsets of Activities and IDAs are called "dialogues" and are essentially a MASCOT specific version of the generalized "conversation" principle proposed in reference 2. Thus activities executing within a dialogue are constrained to access only those IDAs which are associated with that particular dialogue. The following paragraphs, taken from the final report (reference 1f) give more details about the dialogue scheme.

"Upon initial entry into a dialogue, an activity will become descheduled until all associated IDA's can be claimed by that dialogue. Thereafter, any other activity which wishes to enter the same dialogue may proceed without further delay. When an activity wishes to discard a recovery point (i.e. exit from a recovery block) it must wait for all other activities which have entered that dialogue to reach the same point in their processing. Then, and only then, can the dialogue be completed, its IDA resources released, and all participating activities allowed to discard recovery points and proceed with their processing. Conversely, if an activity wishes to recover, then all participating activities must also be recovered along with the IDA's of the dialogue. Thus the dialogue allows a number of activities to co-ordinate their recovery for a limited period of processing.

It follows from the description above that a dialogue effectively owns an IDA for the period of its activation. If an unrecoverable activity wishes to access an IDA, it must wait until that IDA is no longer owned by a dialogue (becomes quiescent). When it does access the IDA, that IDA becomes unrecoverable and can only be accessed by unrecoverable activities. When all unrecoverable activities have finished their accessing, the IDA will revert to its quiescent state and may then be claimed by a dialogue. In keeping with the requirement for multi-level recovery within a system, dialogues may be nested; an inner dialogue will consist of a subset of the IDAs and activities of the outer dialogue and when activated will automatically claim ownership of the appropriate IDA's, thus suspending the outer dialogue."

It is implicit in the preceding discussion that all IDAs of a dialogue are backward recovered when the dialogue is recovered. In some circumstances, however, it is more appropriate to provide forward recovery for an IDA. One example of this is where an IDA, holding the time-of-day, is forward recovered to ensure that it holds the current time following recovery. Forward recovery is also used to provide a means of interfacing between recoverable and non-recoverable subsystems.

In order to support the software fault tolerance scheme described above, the MASCOT (virtual) machine must be extended to provide recovery blocks and MASCOT construction and run-time facilities for dialogues. Further details of such extensions are given in reference 1e.

APPENDIX 2

DISTRIBUTION OF EFFORT

1 CORE Analysis	16 man weeks
2 High-level design	6 man-weeks
3 Detailed design	14 Man weeks
4 Application of Software fault tolerance	11 man-weeks
5 Final report	5 man-weeks

APPENDIX 3

SIZE OF STRUCTURED ENGLISH -PDL

	Lines of PDL		
	Non-FT	FT	% increase
Radar Processing Activity	163	163	-
Correlation Activity	478	566	18
Tracking Activity	1019	1644	61
Command Interpreter Activity	24	24	-
Track Table Updating Activity	348	355	2
Route Display Activity	22	22	-
Database Update Injection Activity	177	276	56
Flight Plan Maintenance Activity	20	36	80
Track/Plan Matching Activity	69	84	22
Flight Plan Activation Activity	12	28	33
IDA Access Mechanisms	223	352	58
Totals	2555	3550	39

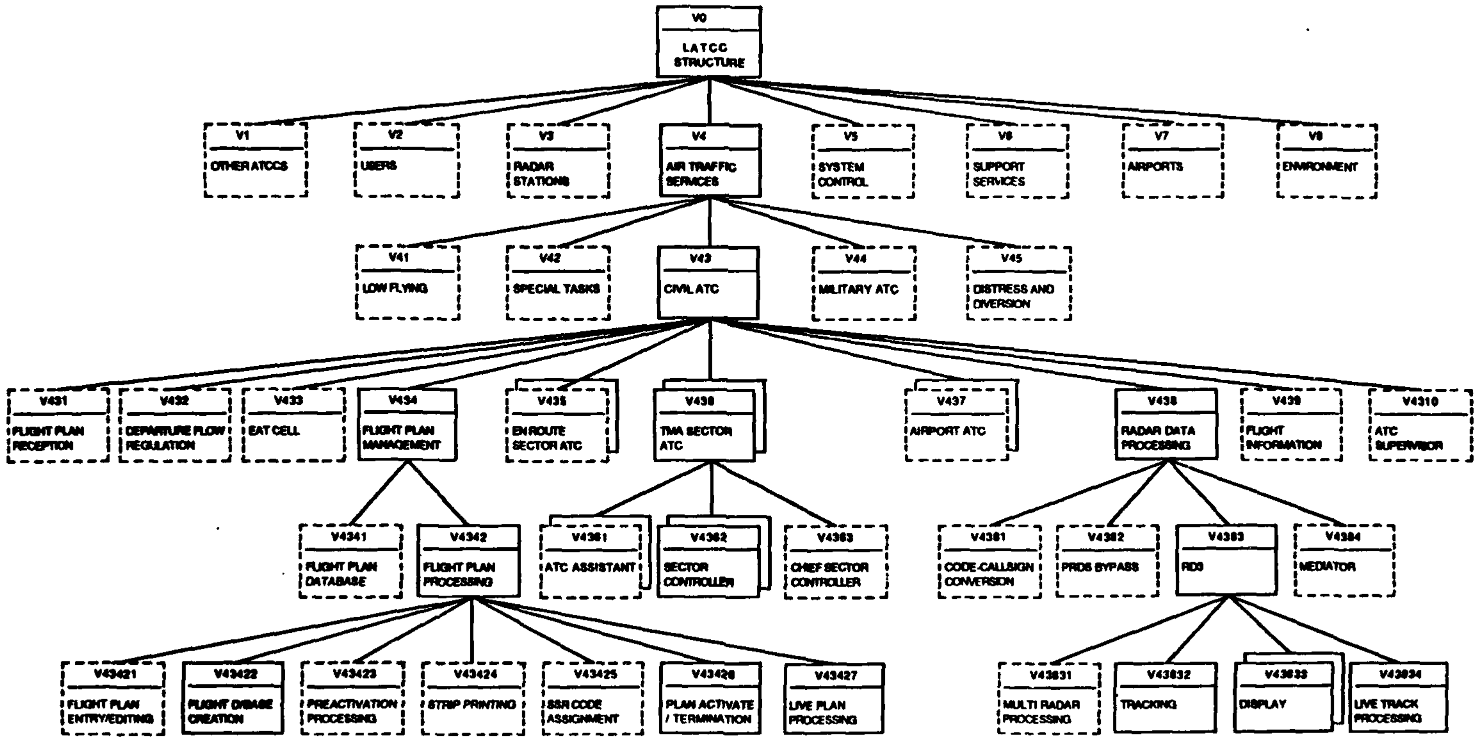
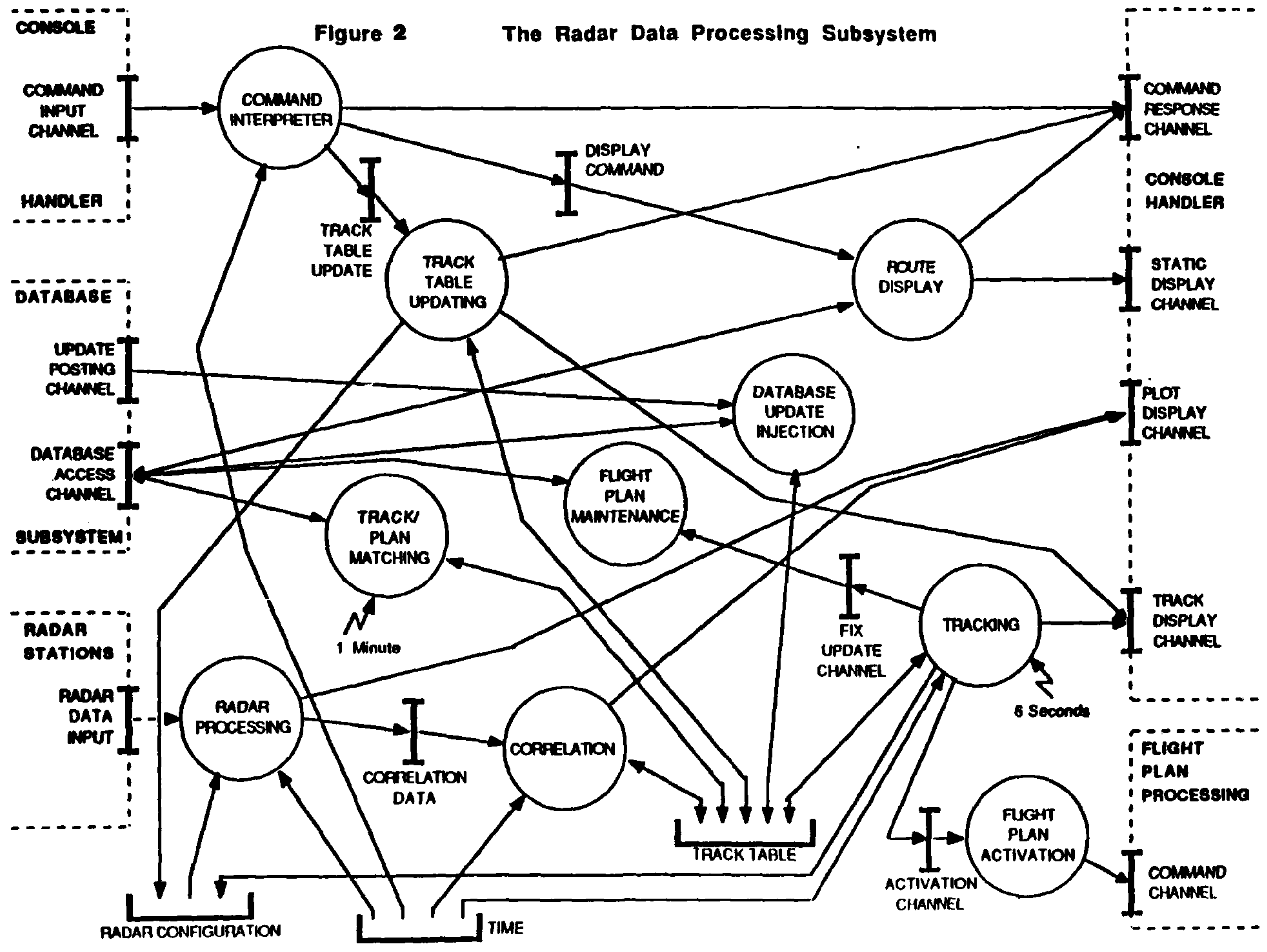


FIGURE 1. VIEWPOINT STRUCTURING DIAGRAM FOR LATCC

Figure 2 The Radar Data Processing Subsystem



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 4114	3. Agency Reference	4. Report Security Classification UNCLASSIFIED

5. Originator's Code (if known) 7784000	6. Originator (Corporate Author) Name and Location
5a. Sponsoring Agency's Code (if known) 2145000	6a. Sponsoring Agency (Contract Authority) Name and Location Civil Aviation Authority, London

7. Title
The application of software fault tolerance to Air Traffic Control: Study contract overview

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 06.1988	pp. ref. 16

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

15. Distribution statement
UNLIMITED

Descriptors (or keywords)

Fault tolerance	Recovery blocks
Software	MASCOT
Air Traffic Control	CORE

continue on separate piece of paper

Abstract

An overview of a study carried out by the Royal Military College of Science (RMCS) and its subcontractor, the Microelectronics Application Research Institute (MARI), is given. In the study a requirements analysis of the London Air Traffic Control Centre (LATCC) was developed, using the CORE method with automated support. This analysis was used as the basis for a detailed design of the LATCC Radar Data Processing subsystem. This design was converted into one with the software fault tolerant features of Recovery Blocks in a MASCOT-like environment. — ([illegible]) — ([illegible])

S80/48