SEVENTEENTH EUROPEAN ROTORCRAFT FORUM

Paper No. 91 - 74

# CREATION OF A LIVING SPECIFICATION FOR AN EXPERIMENTAL HELICOPTER ACTIVE FLIGHT CONTROL SYSTEM THROUGH INCREMENTAL SIMULATION.

G.D. PADFIELD* & R.BRADLEY**

*DEFENCE RESEARCH AGENCY (AEROSPACE DIVISION)
RAE BEDFORD, U.K.

&

**DEPARTMENT OF AEROSPACE ENGINEERING
UNIVERSITY OF GLASGOW
GLASGOW, U.K.

SEPTEMBER 24-26, 1991

Berlin, Germany.

Deutsche Gesellschaft fur Luft- und Raumfahrt e.V (DGLR)
Godesberger Allee 70, 5300 Bonn 2, Germany.

# CREATION OF A LIVING SPECIFICATION FOR AN EXPERIMENTAL HELICOPTER ACTIVE FLIGHT CONTROL SYSTEM THROUGH INCREMENTAL SIMULATION.

G.D. PADFIELD* & R.BRADLEY**

*DEFENCE RESEARCH AGENCY (AEROSPACE DIVISION)
RAE BEDFORD, U.K.

&

**DEPARTMENT OF AEROSPACE ENGINEERING
UNIVERSITY OF GLASGOW
GLASGOW, U.K.

## SUMMARY

In the field of helicopter flight control and handling qualities, the potential benefits offered by Active Control Technology are considerable. To support the development of appropriate handling criteria and carefree manoeuvring features, the UK Royal Aerospace Establishment has been engaged in the development of an ACT system for a research Lynx. As currently envisaged the system includes full authority fly by wire actuation and fail-operate / fail-safe hardware architecture. The impact of the required functionality on the system requirements dictated a need for a precise yet versatile specification of the system, and Jackson System Development (JSD) was selected as a design method since it provides a formal modelling of the pilot interface, and also operates at a sufficient level of detail necessary to ensure completeness and resolution of ambiguities. The tools which support JSD include automatic code generation, were further developed to accommodate changes to system architecture in an efficient manner. The code produced provides a direct simulation of the design and results in a living specification available for validation and behavioural investigations of the written specification .

## 1. INTRODUCTION

In-flight simulation provides the ultimate validation test of a new flight control concept. The realism of flight test overcomes the deficiences of ground based simulation associated with cue fidelity and modelling inaccuracies. On the other hand, cost and safety issues constrain what is achievable in experimental flight test. A balance between ground and flight test is required to mature a control concept fully. In the field of helicopter flight control and handling qualities, the potential benefits offered by Active Control Technology (ACT) are considerable [1] and results derived from ground and in-flight simulation in Europe and North America have demonstrated benefits at moderate performance levels. Future military rotorcraft will need to operate at considerably higher performance and in tougher environments than currently achievable. To support the development of appropriate handling criteria, carefree manoeuvring features, and the associated technologies in controls and displays , a number of research laboratories are exploring the options for enhanced in-flight facilities. In the UK, at the Royal Aerospace Establishment, attention has been focussed on studies into the development of an ACT system for a research Lynx [2,3]. Features of the system as currently envisaged include full authority fly by wire (FBW) actuation, safety pilot with back-driven controls, fail-operate / fail-safe (FOFS) hardware architecture coupled to a range of novel sensors and pilot inceptors providing inputs to the control laws. The FOFS architecture is proposed to enable safe experimental flight in the nap of earth and at the edges of the performance envelope. The impact of this functionality on the system redundancy requirements is considerable. RAE identified a need for a precise, yet versatile, specification of the system required to perform these functions - a specification developed through a formal design method and validated by simulation.

The specification needed to address functionality (for both normal and failed states), operation and performance of the integrated system, together with interfaces, constraints and testing requirements. The specification also needed to be fit for establishing realistic development costs and timescales. The approach taken has crystalised into two phases. Firstly, the development of a textual description of the system with accompanying illustrations. During this activity, a number of different methods were applied by different team members in an attempt to formalise the requirements, to tackle design issues and to provide a format compatible with the later stages of the system life cycle. The Jackson System Development (JSD) methodology was selected for several reasons:

(a) The JSD modelling produces a formal specification of all the pilot/system interactions and so forces the engineer to consider system behaviour from a constructional/design rather than hierarchical description viewpoint.

(b) The JSD network provides a complete description of all the external system interfaces required, plus a systematic partitioning of the system functionality.

(c) Ambiguities in the textual material are naturally identified.

(d) Tools were available to support the method including automatic code generation.

A most important feature of the specification is that it is an executable version of the functional behaviour of the system. Ada code is automatically generated from the specification and, when combined with a simulation of the flight model and various peripheral devices, becomes a 'living' specification of the system behaviour.

The second class of requirement involves the investigation of options for the exact nature of the final system implementation. This research is intimately connected with the number and types of processing element, and the form of fault monitoring and reporting. To this end a new language has been invented which allows description of hardware layouts and the provision of fault tolerance. The description language is supported by data entry and code generation tools that allow machine manipulation of the description and realistion of the specified system using Ada. This facility enables the investigation of various hardware architectures, providing the vital realism required to back up more conceptual research.

The RAE ACT Lynx project is currently 'on hold' due to UK funding limitations; the sophistication and complexity arising from the FOFS requirement has a significant cost penalty. The exercise of developing a living specification is continuing however and the key experiences are shared with the broader helicopter community in this paper, which draws on aspects of the system functionality to illustrate the JSD approach of modelling and network analysis. Section 2 covers the evolution of the ACT Lynx requirements leading to the need for a prototype simulation. Section 3 deals with the development and use of the Ada simulation, illustrating its investigative potential and Section 4 discusses the way forward for the specification and the project as a whole.

## 2. EVOLUTION OF THE SPECIFICATION

### 2.1 Background

In a series of technical memoranda and reports [2,3,5,6] RAE developed the rationale for a programme of research based on an ACT helicopter. Further studies have demonstrated the practical feasibility of modifying the RAE AH7 Lynx ZD 559 into a full authority ACT vehicle for such a purpose; encouraged by the feasibility of this approach, RAE embarked on the preparation of a specification for the airborne component of the ACT Lynx system. Figure 1 illustrates one possible design concept: The experimental pilot's conventional control system is replaced with an ACT system. The elementary modules of the new system are described more fully below, in section 2.3, but, in essence, a flight control computer connects a new set of inceptors and sensors to a group of parallel actuators driving the original actuation system. This approach, whereby the existing actuators are retained and the new parallel actuators backdrive the mechanical runs to the safety pilot's controls has been employed successfully in previous experimental ACT helicopters. An alternative configuration where the reversionary system is a core fly-by-wire system is also under investigation. The work reported in this paper is to a large extent generic in relation to architectural layout, although examples given will refer to the conventional situation.

From the outset RAE were determined that the specification should be the basis of a well managed procurement exercise, and as such, should solve all of the significant design issues of the system. Potential suppliers would then be able to assess accurately the costs of supplying the various components of the system, since the possibility of being involved in expensive open ended design work would be eliminated. Also, by solving the outstanding design problems *ab initio*, RAE would be sure that the system could actually be supplied in accordance with the specification.

## 2.2 Adoption of Jackson Techniques

With the objective of producing a complete, unambiguous specification it was decided to employ, as far as possible, the techniques of Software Engineering. The disciplines of these techniques would ensure a rigorous development of the design, and the associated CASE tools would assist in maintaining the precision and integrity of the specification. For the digital part of the total system, the methods could be applied directly but for other parts, which could include analogue, mechanical, hydraulic and even human components, it was not immediately clear how the software techniques could be adapted. Moreover, it was desirable that, at the specification stage, there should be some freedom as to the type of implementation.

Jackson System Development (JSD) , [7,8] was stipulated as the preferred methodology, but proved, at least initially, to be difficult to embrace in this novel, more general, context. While the difficulties relating to the use of JSD were being resolved, there was a partial application of De Marco [9] methods; consequently, when the first version (Issue 2.0 [10]) of the specification was circulated among suppliers, in addition to the conventional structure of paragraphs of text supplemented by a set of technical illustrations and diagrams, it contained a group of data flow diagrams and data compositions relating to a De Marco style enhancement to the text.

As a matter of deliberate policy the design team at RAE took Issue 2.0 and subjected it to careful scrutiny in order to identify those areas where it could be significantly improved. In particular, the possibility of using JSD was re-examined since, in the context of the ACT Lynx application, a methodology biased towards system development was considered to be more appropriate than a decompositional, hierarchical, descriptive technique. A strength of the Jackson method is that it spans the full range of activity from system definition to production of code [11], so that at one end it is concerned with modelling correctly, for example, the actions of the pilot when he uses the Pilots Control Panel (shown in Fig 2 and discussed fully in later sections) and, at the other end, contains the level of detailed specification necessary to generate code. Such a level of detail ensures that the design problems of the specification have been addressed even if the software is not actually produced, but in this application a further step has been taken and code produced to implement a simulation of the specified system of Issue 2.0 (section 4). These two areas had not been given sufficient emphasis in the earlier version, and the discipline of JSD would force attention to them.

## 2.3 Specification structure.

The specification structure describes the system in terms of its major functional elements. This decomposition was the only one that was imposed on the system *a priori* and reflected a separation which was unavoidably incurred by the nature of the project. Such a subdivision does not preclude further subdivisions should they evolve from the design process. The outcome is shown in Figure 3, where the square and rectangular components are those relevant to the specification exercise. The bold rectangles are referred to as processing elements embodied in a Flight Control Computers (FCC) although such terminology was not used in the specification.

The elements of the system are described in the order of the primary flow of the signal information illustrated by the arrows in Figure 3.

(i) Sensor Element (SE). This leading element contains the aircraft motion sensors - attitude, heading and rate gyros and accelerometers, and also the air data units for obtaining velocity components, pressure and temperature information.

(ii) Crew Station Element (CSE). The other leading element incorporates the conventional controls for the safety pilot and a versatile side arm controller facility for the experimental or evaluation pilot. For convenience these inceptor components were grouped together as an Inceptor Element (IE). The CSE also contains the various interfaces for the pilot to engage, operate and be cued by the ACT system (Figure 2) as follows:

(a) Pilots Control Panel (PCP) - used by the Evaluation Pilot for engagement and disengagement and also for conducting the system-test sequence. Engage and Disengage operations would normally be performed using switches on the pilot's controls.

(b) Repeater Panel (RP) - provides a copy of the displays for the Safety Pilot.

(c) Menu Panel (MP) - provides other ACT interactions, such as selecting one of the available control laws and sets of parameter values. The same panel provides the interface for injecting preprogrammed disturbances into the system, as part of a flight-test facility used, for example, in the validation of the helicopter mathematical models and in demonstrating compliance with handling qualities requirements of new control laws.

(d) Mode Select Panel (MSP) - available for in-flight selection of control modes, for example, height-hold and speed-hold.

Clearly the CSE would be expected to feature significantly in any JSD modelling exercise, with the pilot assuming a number of different roles as he interacts with different components of the system. Some of the related modelling issues are discussed in section 2.5, below.

(iii) Control Law Input Support Element (CLISE). This element has the main purpose of processing and managing the information from the Crew Station and Sensor Elements. It also contains the function for scheduling of a comprehensive system test.

(iv) Control Law Element (CLE). This element is supplied with inceptor, sensor, mode selection and related information by the CLISE. The CLE is the raison d'etre of the ACT Lynx since it hosts the experimental control laws which are to be evaluated. It is this element that the user of the ACT Lynx, the handing qualities engineer or flight dynamicist, will interact with. Carefully verified and validated control law software [4] will be plugged into and unplugged from this element. Typically six control laws will be selectable by the experimental pilot with an additional choice of up to six sets of parameters within each law. The demands produced by the CLE for each of the four axes may be separated into low and high frequency demands, if required, which are destined for the parallel and series actuators respectively (an option being currently evaluated). The separation algorithm is part of the user supplied CLE software. Alternatively this function could be achieved in software and a combined signal fed to full authority actuators.

(v) Control Law Output Support Element (CLOSE). The element following the CLE interfaces the demands produced to the remainder of the system. It also provides a selectable limiter on the demands produced by the control law as additional protection against immature software.

(vi) Actuator Drive and Monitoring Element (ADME). The final element to provide processing takes the demands from the CLOSE and produces drive signals for the parallel actuators resident in the Actuator Element, and the series actuators in the Primary Flight Control Units (PFCU). The ADME also manage the engagement of the ACT system through the energising of th parallel actuators, and supplies a normal autostabilisation function when the ACT system is not engaged.

(vii) Actuator Element. The parallel actuator system is last in the sequence. The parallel actuators are connected to the conventional control runs from the safety pilot; when the actuators are engaged (hydraulically powered), the controls are back driven to provide the safety pilot with essential control position cues and to aid in recoveries, and forward driven to the existing Lynx PFCUs.

(viii) External System Support Element (ESSE). In support of this network of elements is an element which essentially provides a catchment for all of the significant data in the system. It interfaces with the standard on-board data acquisition system MODAS [12] and also with the experimental displays such as helmet mounted or head down displays. A record of all system related events such as engagement, disengagement, and diagnostic messages is retained in a System Journal.

## 2.4 Element descriptions.

Issue 3.0 of the specification [14] contains a detailed description of each of the elements identified above. As far as possible, the recommendations of the STARTS [13] guide have been followed in the preparation of the specification. Each element is described in detail under the headings Type, Function, Operation, Performance, Inputs & outputs, Interfaces, Testing, and Failure reporting & recovery. Where a particular element is composed of replicated units, so that several **units** together comprise an **element**, the replication of units in the element is stated and the unit itself is described under the same headings. For example, the CLISE is a triplex element composed of three identical CLISUs (Control Law Input Support Units). In detail the descriptions are:

TYPE - Some indication is given here of whether implementation is anticipated as an analogue, digital, mechanical, hydraulic, electro-mechanical or human process. The suggested implementation is not intended to exclude alternatives if a supplier possesses a particular specialism or preferred approach. The view was taken, after some deliberation, that it was better to make specific recommendations rather than to leave the 'type' issue open. A general allowance could then be made for variations that nevertheless complied with the functional aspects of the specification.

FUNCTION - Under this heading is a complete statement of the tasks of the unit , that is,a statement of what job the unit has to do. For example, one of the tasks of the CLU (a unit of the CLISE) is inceptor management; the entry reads: "The inceptor displacements and inceptor switch positions shall be processed to provide consolidated signals for the associated Control Law Unit (CLU)"

OPERATION - This sub-section is concerned with how the unit will achieve its functions. This is done by detailed description, in text, of the processing required for each function. For the CLISE example above, the full details of the processing of the triplex signals would be supplied, including the consolidation algorithms for fault tolerance. The narrative under this heading is used to build the JSD Specification; the full JSD is not held within the text of Issue 3.0, but sufficient initial design work was undertaken to be confident that a JSD specification could be derived from the narrative.

PERFORMANCE - This deals with how much and how well issues, including a statement of the times within which the tasks must be completed and, where appropriate, the accuracy that must be achieved. For example, a certain part of the system test must be performed within a stipulated time. The sampling rates for the unit would be specified here. One important defined constraint is that the total system transport delay should be 25 ms.

INPUTS & OUTPUTS - This contains a list of all signals received by the unit and those transmitted by it. It includes the source of a received signal and the destination of a transmitted one. This information is also presented in diagrammatic form, Figure 4, for example, where the connections to neighbouring units are clearly visible (The network notation is discussed in section 3). There is, of course, a need to maintain consistency here, since for each input listed there must be a corresponding output on some other unit. Such consistency is easily maintained by a CASE tool such as Jackson Work Bench (JWB) [15].

INTERFACES - A list of the units and their types, both internal and external, to which the subject unit is connected. The purpose of this information is to identify the interfacing requirements between units - analogue to digital, for example.

TESTING - A statement of how the function, operation and performance of the unit is verified. In particular this may be done at a system test invoked prior to take off, or by the inbuilt monitoring.

FAILURE REPORTING AND RECOVERY- A statement of how errors, produced by a fault and having been detected, are reported within the system. Usually they are reported to the pilot via the Menu Panel, and sent to the system journal part of the ESSE. Cautions and Warnings may also be raised through the Central Warning System. In addition, a statement of the recovery of the system may be required; often the recovery is by returning to Standby via a controlled disengage - as would be the case when one of the monitoring tolerances within the system has been exceeded.

## 2.5 A living specification

Once Issue 3.0 of the airborne system specification was complete, it was decided to progress to a full JSD specification in order to identify and eliminate any residual ambiguity, vagueness or plain error. The full JSD would then be available to use as an adjunct to the written specification. It would give a precise description of the interfaces between the components of the system and between the system and any external devices, to the benefit of prospective suppliers.

A further decision was made to use the JSD to generate a simulation of the ACT system, to produce, in effect, a living specification which could be used to exercise and examine the specification dynamically. The novel features involved in this step are described in detail in section 3, but the six aims of the simulation in relation to authenticating and potentially enhancing the specification were:

(i) Control and human operation of the system. Pilot acceptance of the procedures for operating the system, for example, the arm/ engage/ disengage sequence can be evaluated through hands on experience. Also suppliers can directly examine the nature of the interface between their equipment and the rest of the system.

(ii) Synchronised control information. The techniques for managing and synchronising control information within an asynchronous or loosely synchronous system can be verified.

(iii) Establishing tolerances. An asynchronous system generally must allow some tolerance in the monitoring of the information from replicated units. Suitable tolerances can be verified or even derived.

(iv) Computational load. The processor power and memory requirements of the system can be more confidently deduced from a simulation than a paper specification. Alternative implementations may be evaluated for processing efficiency.

(v) Fault management. The mechanisms for reconfiguration, and the issuing of caution and warning signals may be verified directly.

(vi) Design Evolution. Alternative designs for the components of the system can be evaluated directly.

Before progressing from this discussion on the evolution of the specification to considering the development of the simulation in detail, there are two topics worthy of a special note.

## 2.6 The Supervisor as a modelling issue.

One area which, from the beginning, was subject to intense scrutiny was the control or overall supervision of the ACT system, including engagement and disengagement. Clearly this is a critical area where it is essential to get the specification and implementation correct. An example of an early model is shown in the flow chart in Figure 5, where the System Test, initiated by the pilot, if successful, is followed by a repetition of the arm, engage, disengage sequence of actions. While useful for conveying the general idea of the pilot's interaction in this area, it was not sufficiently precise to base software directly upon. For example, it is possible according to the specification, to return to Standby through a disengage action without an engagement of the system. This path is not shown in the flow chart. To express the requirements in a precise manner finite state machines (FSM) were mooted and proved a very useful approach. That shown in Figure 6 included the additional transitions to Standby omitted from the flowchart, but suffered from a shared disadvantage in not exposing that the system test, itself, included arm, engage, disengage sequences. FSMs have the advantage that they are readily transformed into software so they were seriously considered as a basis for a 'supervisor' process, which would have overall control and only permit defined transitions of the system to occur. The problems experienced with this approach were twofold. First, incorporating all of the possible states and transitions afforded by the pilot resulted in a very complex FSM, which was difficult to interpret and militated against a correct implementation. Secondly, the engage or disengage actions made within system test, gave different states from those occurring after a successful system test. Consequently, and very importantly, the system test did not exercise that part of the controlling software which would ultimately be used.

These problems were resolved in the final JSD modelling, part of which is shown in structure diagram form in Figure 7, where the system test and engage models are separately treated but have appropriate interlocks. In the structure diagram notation, which is described more fully in section 3, the leaves of the tree structure are actions (similar to transitions of the FSM) and in Figure 7 there is a repetition, denoted by the '*' symbol, of the alternatives, denoted by the 'o' symbol, of a normal engagement cycle or an early disengage. The Arm,Armed, Engage sequence can be quitted, denoted by the '!' symbol, at any stage to continue with an early disengage. The system test process is a simple cycle of alternatives of a successful or unsuccessful test.

The example above has been discussed, for clarity, in a simplified context, omitting such complications as control law selection and disturbance injection, but the same principles apply. The use of JSD in this area has helped to achieve a satisfactory modelling and, further, the model can be directly implemented as a process, upon which the whole of the software can begin to be constructed. It is also interesting to note that the separation away from a monolithic supervisor was also guided by the need for maintaining optimum integrity. The various roles of the pilot are modelled separately with appropriate interlocks preventing inappropriate actions, for example, a change of control law when the ACT system is engaged.

### 2.7 Fault tolerance and redundancy management

A second topic of fundamental concern within the function and operation sections of the unit descriptions consideration is the redundancy management and fault monitoring issues of the multiplex elements. The main criterion for tolerance is that the system should be first fail operative, and the identification of a fault should alert the evaluation or safety pilot, through the Pilot's Control Panel or the Repeater Panel respectively, to return control to the safety pilot and conventional inceptors, by a controlled disengagement of the ACT System. Faults in a unit are detected by downstream comparison of its outputs with those of its siblings (associated units or lanes within the element - its partners within the redundancy). This recognition is dealt with in three ways:

(i) The consolidation of the redundant signals must not be affected by one signal being in error. There are two types of information to consider here. The first type is 'analogue' or continuous type of data where the median select is used for triplex architectures, the second type is discrete data where a majority vote is employed; both of these are passive fault masking operations used to collect valid data for subsequent processing.

(ii) The error must be recognised and signalled to the system and the pilot via the appropriate panel lamp - this is the monitoring aspect. The event must also be logged in the system journal.

(iii)There must be a reconfiguration triggered by the signalling of the error in order to isolate the faulty unit. The isolation is done by ignoring all of the outputs of the faulty unit.

A dual-duplex arrangement operates in a different manner, where each pair of units carries a validity signal and outputs the validity status alongside the functional data. The downstream units can then mask the faulty unit by a tolerant behaviour or reconfiguration. The ACT System has in its initial form, a dual duplex ADME, originally to be compatible with the dual hydraulics of the existing Lynx actuation system, and the single fault tolerance arises from the disconnection of a faulty pair of units from the drive to the actuator; the performance of the drive being such that it can tolerate such a reduction of input. The processing elements, CLISE, CLE and CLOSE, are triplex, but have no cross connections at their mutual interfaces .(There is a modicum of sibling monitoring in the consolidation of discrete data.) Consquently they effectively form a single triplex module. The SE and the CSE are essentially triplex with a full number of cross connections to the CLISE.

### 2.8 The adequacy of the Issue 3 Specification.

The ACT System elements and the functions they perform were conceived and assembled from the combined engineering experience of the project team. This included first hand experience with design of conventional control systems and direct exposure to the helicopter digital flight control system programmes in foreign Industry and Government research laboratories. The FOFS requirement for ACT Lynx combined with the need for significant flexibility in operation created new problems however. The completeness and validity of the upgraded Issue 3 ACT Lynx specification had to be questioned. Were the performance figures achievable in practice? Would there be smooth operation throught the PCP? Would the redundancy managment logic work? In many projects it is apparent that answers to these kind of questions are deferred until deep into the detached design phase, often when the customer is no longer closely involved. RAE needed to increase confidence and reduce the risk associated with these questions; it was decided to embark on the development of a fully operational prototype simulation. An incremental approach naturally complemented the JSD methodology.

## 3. INCREMENTAL DEVELOPMENT OF AN ADA SIMULATION

### 3.1 Overview.

The development of a full JSD specification, and its implementation as a simulation of the ACT system required a number of adaptations and extensions to conventional JSD. There was, for example, the question of the treatment the non-digital aspects of the specification and, in addition, the need for a method of specifying triplex, and dual-duplex architectures in an efficient manner. Further, the CASE tools supporting JSD needed to be enhanced to support any extensions to the method. Finally, the simulation was to be implemented in defined stages so that lessons learned during the review of each stage can be included into subsequent stages in a disciplined manner. A consequence of the incremental approach was that the appropriate incremental stages had to be defined initially in outline but could, at a later stage, be altered in order to respond to revised, or corrected, requirements. All of these concerns are addressed in this remainder of this section after a brief introduction to the notation and concepts of conventional JSD for computer systems.

### 3.2 Jackson System Development

Jackson System Development is a method of analysing a written specification for a computer system to produce a formally executable specification. The method was jointly developed by Michael Jackson and John Cameron in the early 1980s [7,8]. It consists of three stages: model, network and implementation. There is considerable emphasis placed on the modelling stage in order to establish, unequivocally, the information available from the outside world.

3.2.1 Modelling. A JSD model is a description of the real world as it appears to the system. Entities are objects in the real world which have to be modelled by the system, and of particular interest in the modelling activity are those entities which perform discrete actions. For example, a press of the ARM button by the evaluation pilot is an action which is important to the system. The modelling stage requires that the actions be allocated to specific entities, and the main task of the modelling phase is to identify viable entities and allocate the relevant actions to them. For each entity the time ordering of the actions must be then be specified and, conventionally, a tree diagram is used for this purpose. As an example consider the truncated list of actions from the ACT Lynx system shown in Figure 8. Some of these are related to the pilot entity in his role of engaging the ACT System and to express the time-ordering a tree diagram using (Jackson Structured Programming) JSP notation is shown in Figure 9. The root is named after the entity which performs the actions, and the leaves (the lowest level boxes which are named rather than numbered) hold the names of the individual actions. The intermediate nodes or boxes describe the possible types of behaviour: sequence, selection and iteration, as denoted by the symbol in the top right hand corner of the box. An asterisk (*) represents iteration, that is, none or more occurrences; a circle (o) represents selection, that is, a mutually exclusive choice; an absence of a symbol represents a sequence, that is the branches of a node occur, as they are read, from left to right. The numbers in the lowest level boxes refer to changes in the state of the object (entity) as shown in the table in Figure 9. Thus Figure 9 expresses a model of the Pilot Engagement entity as a repetition of occurrences of Engagement Cycles. An Engagement Cycle can either be a Normal Cycle, composed of a sequence of Arm, Armed, Engage, Disengage, or alternatively an Early Disengage, composed of only part of the normal sequence followed by a Disengage. The appropriate changes of state are indicated by the numbered operations for each action, and it can be seen that prior to any action the engagement state is initialised to DISENGAGED by operation 13.

In a completed model, the total set of tree diagrams describes all of the time orderings of the actions plus the changes in system state. In real-time systems it is often only the controlling activities which require this type of modelling and much of the real world is modelled simply by polling sensor information. In the ACT Lynx application, for example, the Lynx helicopter is modelled kinematically by polling attitude and rate gyros and accelerometers. The tree diagrams in these cases are simply iterations of polling actions.

It is convenient here to look ahead to the implementation stage and observe that the model structure in Figure 9 can be used as a program structure for a process to control the engagement of the ACT system. Once operations have been added to read incoming action-messages then all that is required is for the operations to be expressed in the required language. The iterations can be expressed as loops and the selections as conditional statements with appropriate conditions. The result is that the tree diagram can be converted to code mechanically either by hand or, as in this work, automatically.

3.2.2 <u>Network</u>. Processes derived from the entities defined in the modelling stage are called model processes. Other processes are needed to make use of the data stored by the model processes in order to generate the outputs which are to provide the required functions of the system, and further processes, such as button-pollers, may be neeeded to provide inputs to the model processes. Communication between processes is either by a datastream, which is a queue of discrete messages, or by read-only access to another process's internal data, that is,by an inspection of its state vector. The specification of these additional processes and the communications between them is the developer's main task. All of the processes are described using the JSP tree-diagram notation and the result is a network of communicating sequential processes. A diagram of the network conventionally represents processes by boxes, datastreams by circles and state vector inspections by diamonds.

These features are illustrated in Figure 10 which shows part of the network diagram associated with the CLE of the ACT Lynx system. The input datastream to the control_law_algorithm process is the frame time-grain-marker (cla_tgm) which sends its outputs (actuator demands) as a datastream to the disturbance_imposer process. As can be seen, the control_law_algorithm process has access to the internal data of several other processes. For example, via the clise_amse_data state vector inspection, it has access to the Aircraft Motion Sensing Element data of the clise_amse_data process. Similarly the process has access to inceptor, mode-select and engagement status data etc. The engagement status data is obtained from the Pilot_engagement model process discussed earlier and described in Figure 9. The double bars across some of the connections merely denote enhanced multiplicity: for example, there are more than one inceptor providing positional data to the control_law_algorithm. An example of a JSP description of a network process is given in Figure 11, which describes the process for dealing with the interaction associated with the selection of control laws. The tree diagram, Figure 11(a), has had appropriate operations added as leaves to the structure, and the operations are defined in Figure 11(b). Some of the operations are concerned with modifying internal data, such as operations 50-52 and 21-24, with the latter group maintaining data that will be accessed by another process (The SV prefix denotes its state vector destination). The other operations are concerned with communications. The state vectors of some other processes of the network are inspected by operations 40 and 41. An input datastream is read by operation 30 and output datastreams are written to by operations 10-13. The tree diagram, together with its operations list, is to sufficient to generate program code from, once approriate conditions have been attached to the iterations and selections. All that is required is the addition of a declarative part and a decision about how to implement the datastream read and write operations for a particular implementation of the system. This latter aspect is discussed in the next section.

3.2.3 <u>Implementation</u>. The implementation stage is concerned with matching the network to the target environment. The flexibility which is available at this stage is a useful property of JSD in that it enables the specification to be verified prior to it being implemented on the target hardware. In the network each process executes concurrently with every other. This is unlikely to be the case in the target environment where a limited number of processors, often a single processor, will require that some of the processes are suspended while others execute. The @ symbol in the READ and WRITE operations of Figure 11(b) indicate the need for a Macro to deal with the suspension of the process. The code required is standard and can be generated automatically. Although the procedure may appear complex, it is its flexibility that defies a compact description not its difficulty. In practice there are standard strategies for implementation [7,8]. For example, by using a scheduler, all of the programs arising from the JSP description can be converted into subprograms, or procedures, and the datastreams implemented as procedure-calls. The final result of the implementation stage is code that will execute on the target hardware.

3.2.4 <u>Summary</u>. The principal aim of the JSD method is to create a specification which can be usefully viewed from both above and below. The modelling stage is an object oriented analysis of the real world which produces a description which users can readily grasp, because the result is described in terms of objects familiar to the user. The tree diagrams of the method also provide important detail about the model of the real world. The network stage uses two descriptions: (a) Data flows, which can be presented to the user to indicate the architecture of the system and (b) Tree digrams which the analyst can use to express the design of a particular function. The resulting specification can be viewed by the user from above because it is in terms of their real world and, simultaneously, the specification contains enough detail for the implementers below to perform their task. It is this general property that made JSD particularly attractive for the ACT Lynx specification, and encouraged a determined assault on the difficulties associated with the application of JSD to the ACT Lynx System.

### 3.3 Extensions to the JSD approach.

As a compositional method, JSD eschews a top-down approach to system development. The rationale is argued at length by its proponents and a convincing case can be made for it in software development; however for more general systems, the physical architecture imposes a decomposition by the nature of its elements. This is the approach described in section 2: each identifiable element can be viewed as an independent system communicating in a limited way with other elements. For elements which are composed of replicated units each unit is treated as independent. Figure 4 shows an example of such a top down view. The datastream into the CLU is a frame time-grain-marker, and the only inter-unit connections are state vector inspections. Each box represents a unit and JSD is applied in a conventional manner to that unit. For those aspects of the system which are not expected to be digital, such as the actuator element, the same approach is employed except that JSD is applied to a specified simulation of the element. Naturally, care has to taken to ensure that all of the relevant functional properties of the real element are included in the simulation specification with due authenticity. The integrity of replacing the real element in a specification by a simulation depends not only on authentically duplicating its relevant functions but also ensuring that the remainder of the system only has access to that data which the real system can provide. In the case of the actuator element, for example, the actuator positions are not directly available to the ADMUs; one of the four simulated position pick-off signals for each control lane which must be used. Another example is the engagement state of the actuator; signals corresponding to appropriate sensors mounted on the actuator must be used to determine whether the actuator is hydraulically energised or not. As a consequence the actuator entity must be modelled within the ADMU using JSD principles. The need for modelling one element within another is a natural consequence of the imposed decomposition into elements.

When system elements consist of replicated units, for example triplex or dual duplex, it is clearly undesirable to compose a JSD network diagram for each unit individually. At best it duplicates effort, and at worst introduces errors caused by accidental differences in the individual networks. What is needed is to reflect the written specification and describe a single unit in detail through a network diagram in the normal manner, supplemented by a formal description of the element in terms of its component units. Such a formal description is shown in Figure 12 (a) and (b) which depict descriptions of units of the Inceptor and Control Law Elements respectively as held on the CASE database. After some standard information (STD-INFO) consisting of its identifier and optional background detail, the MAIN-PART of the description includes a number of options such as:

(a) the type of unit - whether the unit is analogue or digital.

(b) the number of units - here both are simplex units replicated three times.

(c) Whether the units run synchronously or not.

There is also the possibility for connections between units of the element. To complete the description a list is required of all the JSD processes which belong to that unit, and thus need to be replicated; the final entry (UNIT-SID) being blank shows that the name of the list on the database defaults to the name of the unit.

A similar format is provided for the description of the connections between elements as shown by the example in Figure 12(c). The relevant fields are the source, destination and whether the connection is unit to unit individually (ONE-TO-ONE) or completely cross connected (BROADCAST). The connection description also holds some information relating to the fault tolerance implementation which is discussed in the following section.

### 3.4 Implementation of Fault Tolerance

Figures 14(a) and 14(b) describe the connections between the Inceptor Element and the CLISE, and the fault tolerant software sited in the CLISE to handle the data passing between the two units. The fault tolerant strategy was based on that described in Reference 19. The connections between the IE and the CLISE are BROADCAST as indicated in Figure 12(c), that is, every IE sends to every CLISE.Figure 12(c) also indicates that each of consolidation, downstream monitoring and sibling monitoring are enabled. The schematic diagram in Figure 14(b) shows the type of fault processing which takes place.Voting is always present where there are many sources for the same data. The voted value is obtained by either majority vote, or median select depending on the type of data. Downstream monitoring implies comparing the values coming from each of the data sources with the voted value. If any of the sources differs for more than a given number of frames (HISTORY-LENGTH in figure 12(c)), then an error is logged and the voter ignores all subsequent input from that source. Consolidation is performed by comparing the historical values from each sibling, gathered over previous frames. A consolidater will only output a new value if it perceives that all of its siblings agree with it. Sibling monitoring implies comparing the voted values coming from the siblings of the unit, rather from upstream sources. Otherwise the processing is identical, with an error being logged when a discrepancy occurs. The sibling which is diagnosed as being in error is then ignored by the consolidation process. The combination of unit and connection descriptions fully define the architecture of the fault tolerance and if a standard strategy is adopted for consolidation and voting the appropriate code can be generated automatically.

### 3.5 Incremental implementation.

The compositional, or "middle out", nature of the JSD method has the property that once a model has been built every new function added to it provides a potentially deliverable, working, system. In fact, at any stage of the development of the network it can be implemented. Incremental development takes advantage of this natural property of JSD and phases development of a system over a number of increments. The added functionality required from each increment is defined initially in outline, and as each increment is completed it is reviewed and the contents of future increments re-examined in the light of any modifications or additions that have been found to be necessary. The development of a system is thus responsive to an evolving specification but at the same time allows the project to be managed on the basis of milestones actually achieved.

The ACT Lynx simulation was developed over six increments distributed as follows:

Increment 1: A model of the pilot/ system interaction including engagement of the ACT system and inceptor movement. The Repeater Panel and a display of the control run position.

Increment 2: A model of the pilot/ system interaction as regards System Test, Control Law Selection, Disturbance Selection, Mode Selection, Parameter Set Selection. The Menu Panel, Mode Control Panel and Pilot's Control Panel.

Increment 3: A definition of a hardware description language for units and connections, and development of associated tools. The functionality of Increments 1 and 2 based on the specified hardware, including fault tolerance. Provision for injection of errors.

Increment 4: Completion of the Control Law Input Support Element including the development of a tool for building a System Test process from a non-procedural definition. The Aircraft Motion Sensor and the Air Data Elements

Increment 5: Completion of the Control Law Element and the Control Law Ouput Support Element.

Increment 6: Completion of the Actuator Drive and Monitoring Element and the Actuator Element. Further development of the System Test Builder.

The simulation also includes a simple model of a Lynx helicopter to provide sensor data from the actuator (PFC) displacements.

From the distribution of material in the six increments it can be seen that the primary concern was to establish an acceptable model of the pilot's interface. One of the early lessons was that different readers of a specification can place different meanings on the same words, and the sequencing of the lamps relating to engagement and system test on the Repeater Panel needed to be revised. The reference to system test in Increment 6 is indicative of the difficulties encountered in specifying a comprehensive test. The contribution from Increment 4 was not sufficient and more work had to be included in the final increment.

During the development of the simulation no fundamental flaw or omission has been discovered in the written specification. Nevertheless a wealth of additional detail has been accumulated mainly to reinforce inadequate descriptions or to compensate for minor omissions. The most significant inadequacy was the omission of a description about how to apply the consolidation algorithm of Reference 19 to replicated units in a fault tolerant manner.

### 3.6 Implementation of the simulation.

3.6.1 Ada as the implementation language. The selection of Ada as the implementation language was determined by the following considerations:

(i) DoD Language. Ada is a DoD mandated language, and is also "highly recommended" by the British MOD, which has provided a large, guaranteed market for Ada compilers ensuring a great deal of investment from compiler vendors. This fact coupled with the extensive validation tests required by the DoD has resulted in a number of very high quality compilers being available.

(ii) Language features. As has been discussed below, packages and tasks have been very important in implementing this system. In addition the comprehensive data typing provided through Ada has enabled a more precise specification to be constructed with a resulting increase in quality.

(iii) Tool Availability. The code generation tool Adacode, described below was already available in prototype form to serve as a basis for the project.

3.6.2 Target hardware and implementation strategy. The simulation has been implemented to execute on an IBM PC, or compatible. Therefore a strategy is needed, as discussed in section 3.2.2, to map the many processes of the specification onto the single processor of the target system. For automatic code generation, a set of CASE tools are also needed to implement the chosen strategy.

There are many possible mapping schemes between JSD and Ada; References 16 and 17 describe two. The mapping used for this project is based broadly on that described in Reference 17; it relies very heavily on packages, the aim being to produce a set of Ada packages where each correspond only to one specification object (e.g. process or data stream). This correspondence enhances the traceability from the JSD specification to the Ada. The extensions to this mapping scheme employed in the ACT application are:

(a) each unit is mapped onto a task type. Figure 13 shows the Ada for the unit described in Figure 12 (b). The replication of units within an element is achieved by declaring an array of the task type for the unit with a multiplicity equal to the REPLICATION factor specified in the UNIT object, which in the case of Figure 12(b) is 3.

(b) the infrastructure which provides the fault tolerance is described using a number of generic packages which are instantiated based on the information in the connection object.

An example of the implementation mapping between the specification network and a unit is depicted in Figure 12(d) for the Control Law Unit. The rectangle at the top corresponds to a task type, with the network processes converted via a standard transformation strategy into a procedure-calling hierarchy; processes nearer the top call those connected directly beneath them Data external to the CLU is shown via the disk symbols, with access shown.

3.6.3 Automatic code generationCode generation is provided by a prototype Ada code generation tool built by LBMS which has been significantly enhanced during the life of this project. Figure 15 illustrates the workings of the tool. It takes the description of the system, specified using the Jackson Work Bench (JWB) CASE product and generates the complete simulation from it.Data Extraction is done using built in facilities of the CASE tool. The code templates are combined with the specific parameters extracted from JWB by a proprietary tool called JSP-MACRO. The code generation approach provides a great deal of flexibility with respect to changes in the implementation of the system. Many simple changes can be achieved purely by amending the templates. Even large changes may only require changes to the data extraction, leaving specification of the system unchanged. As well as tools to build the whole system, there are others which rebuild the system regenerating a minimum of Ada based on changes and still others which create test harnesses for any sub network of the specification, providing a cost-effective way of ensuring quality.

The use of code generation was a significant contribution to the project. Several factors encourage its use in projects of this nature including:

(i) Productivity. The most obvious gain is productivity. The statistics concerning the number of lines of code and even number of functions (counted using function point analysis FPA)) were very high. The figures obtained from the second delivered increment were as follows:

| | |
|---|---|
| (a) Function Points per man day | 2.34 |
| (b) Source Lines of Code per man day | 204 |

(ii) Ease of Instrumentation. The requirement for dynamic analysis will doubtless change as the simulation is used. Because the system is generated using code generation, the instrumentation can be changed merely by altering the form of the Ada templates and regenerating.

(iii) Evolutionary Delivery. One of the important factors that supports evolutionary delivery is for user feedback at the specification level to be converted efficiently and accurately into implementation changes. With automatic code generation directly from the specification this is assured.

(iv) Living Specification. One of the major problems of maintaining computer systems is that the behaviour of the running system can diverge very quickly from the original specification of system behaviour, once maintenance and development begin. Code generation provides the ability to maintain a "living specification", i.e. one where changes to the specification are automatically represented in the implemented system.

This feature is especially important in the case of the ACT system because of the potential need to evaluate many hardware and error monitoring combinations and even new functions in the course of the planned ACT research.

(v) Re-Implementation. Another important benefit of code generation is that, without changing the JSD specification of the system, a completely different set of code can be generated, for example to fit the system onto real or alternative hardware such as transputers. This can sometimes be achieved merely by changing the code generation macros, but may require new implementation objects if the implementation is significantly different. Therefore the investment in a system specification is not compromised when evolving the system towards greater realism.

## 4. THE WAY FORWARD

At the time of writing the RAE ACT Lynx project is at a hiatus. Estimated procurement costs for the system and its certification are high and are likely to require a multi-partner team to be affordable. Both UK and International options are being explored but no clear way forward curently presents itself. Activities in support of the project are continuing at RAE including the study of performance / trade-off issues associated with trials in flight (safety) critical areas. The role of the safety pilot is crucial to this work and ground-based simulations [18] have been conducted - and are planned - to address critical functional questions such as optimum location of disconnects, backdriving frequencies, mismatch tolerances for failure managment, and PCP ergonomics. In parallel with these topics, the requirements specification will continue to be developed. The current operational form is essentially complete in its functionality. Future tasks include:

(a) Instrumentation of the simulation to evaluate end-to-end and internal performance and behaviour.

(b) Production of a comprehensive user guide to the simulation.

(c) Comprehensive exercise of the simulation to validate the specification across the operating spectrum.

(d) Upgrading the requirements specification in line with the results of (a) - (c).

(e) Upgrading the requirements specification to include a second level of JSD analysis, i.e. network and process diagrams together with text.

(f) Implementation of the Ada simulation in real time with representative pilots', engineer's and software development stattion.

Many of these tasks can be embarked upon concurrently and are not specific to the implementation in the final system. The results from these activities have generic value and can be used to guide and support similar projects for example. The current Ada simulation has been developed in seven increments and the approach has demonstrated the utility of this approach. The simulation has 'grown' in a controlled manner with each increment offering more functionality for review and revision if necessary. The approach has had the added advantage of enabling the software engineers to develop their understanding of the application incrementally. A top-down appraoch to the design would have required considerably reater investment in 'application learning' before any creative work could have been started. It is recognised that there are contentious issues in system development and that there are no right or wrong approaches. JSD has exposed functional anomolies and forced hidden issues into the open through its emphasis on design, however. The behaviour of the ACT Lynx system, as currently configured, is now well understood - the flight critical nature of the application makes this an attractive position to be in.

## 5. CONCLUDING REMARKS

The handling qualites opportunities offered by active control technology for helicopters require considerable research effort using both ground and in-flight simulation before the final and complete potential is realised. Much work has already been done but the peculiar problem areas, such as carefree handling, of high performance levels have yet to be explored in-flight. The safety-critical nature of such flight research demands that a fail-operate design concept be employed covering both system hardware and software. In the UK, the Royal Aerospace Establishment has proposed the procurement of an experimental ACT system for its research Lynx. This paper describes the development of the requirement specification for the airborne system including crew station, sensors, processing elements, actuation etc. In its current form the requirement is a textual and diagramatic description of the system behaviour covering functionality, operation, performance, testing and interface requirements. The specification is supported by design using the JSD methodology. An outcome of the design work is a prototype Ada simulation of the system. Examples of the JSD modelling and the mapping into Ada have been described. Initial results from exercising the simulation have been presented. Although the overall ACT Lynx project is on hold until an affordable package is defined, the requirement specification continues to be evolve, with an upgrading scheduled to folow from a comprehensive instrumentation and exercise of the simulation. A real time implementation is planned which could form the core element of a ground system to support software development.

## 6. REFERENCES

1 Padfield G D, (Editor), Helicopter handling qualities and control: Proceedings of the R.Ae.Soc Conference, London, 1988.

2. Winter J S, Padfield G D. A discussion paper on an ACT flight research programme using the RAE Bedford Lynx. RAE Tech Mem FS(B) 523, 1984.

3. Padfield G D., Winter J S, Proposed programme of ACT research on the RAE Bedford Lynx. RAE Tech Mem FS(B) 599, 1985.

4. Tomlinson B N, Padfield G D, and Smith P R. Computer - Aided control law research from concept to flight test. AGARD CP 473, 'Computer Aided System Design and Simulation', 1990.

5. Winter J S, Padfield G D, and Buckingham S L. The evolution of active control systems for helicopters; conceptual simulation to preliminary design. Proceedings of the AGARD FMP Symposium on ACS; Toronto, 1984.

6. Thomson K. The results of the WHL feasibility study in support of the RAE Bedford flight controls research programme: Systems Technology Note STN 19/84 Westland Helicopters, 1984.

7. Jackson M. System Development. Prentice Hall, 1983.

8. Cameron J R. JSP & JSD: The Jackson approach to system development. IEEE Computer Society Press, 1983.

9. De Marco T. Structured analysis and system specification. New York:Yourdon Press,1978.

10. Wright B P, RAE ACT Lynx -Airborne system requirement specification, Issue 2. WHL Flight Control Department Note FCDN 88/05, 1988.

11. Birrel N D, Ould M A, A practical handbook for software development. Cambridge University Press, 1985.

12. Jewel C. MODAS analysis system - system overview. Prosig Computer Consultants, 1986.

13. DTI/NCC. STARTS Purchasers' Handbook: "Procuring software-based systems" NCC Publications. Second Edition, 1989.

14. RAE ACT Lynx Airborne system requirements specification Issue 3.A, RAE 1989.

15. LBMS Plc, Jackson Work Bench User Guide (In preparation), 1991.

16. Cameron J R, Mapping JSD Specifications into Ada. Proceedings of the 6th Ada (UK) Conference. 1987.

17. Lawton J R & France N. The Transformations of JSD Specifications in Ada. Ada User, Jan 1988.

18. Kimberely A & Charlton M. ACT Lynx Safety Pilot Simulation - Trial Runaway.
RAE FM Working Paper (89) 031, June 1989.

19. Silva A. Mode synchronisation algorithm for asynchronous autopilot. Paper No. 38, Fourteenth European Rotorcraft Forum, Milan, 1988.

## 7. ACKNOWLEDGEMENTS.

The authors would like to acknowledge the contributions of Westland Helicopters Ltd and Theta Analysis and Systems Ltd to the work described in this paper.

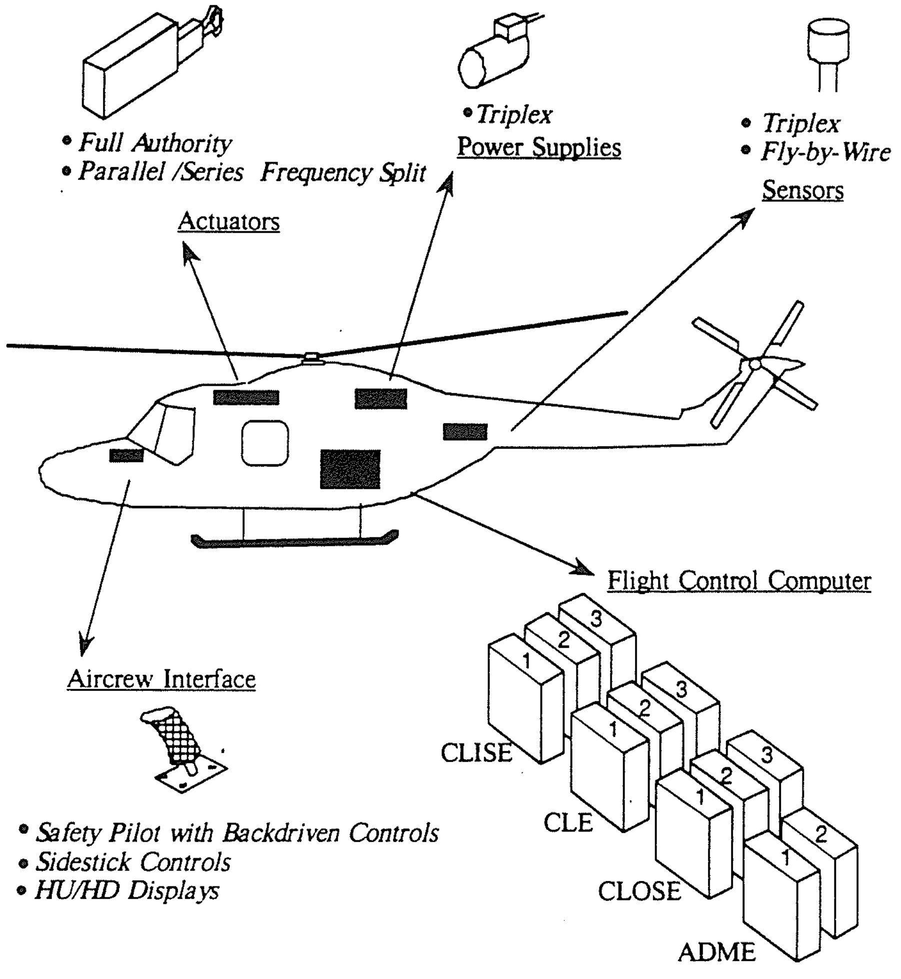


Figure 1. ACT Lynx System Elements

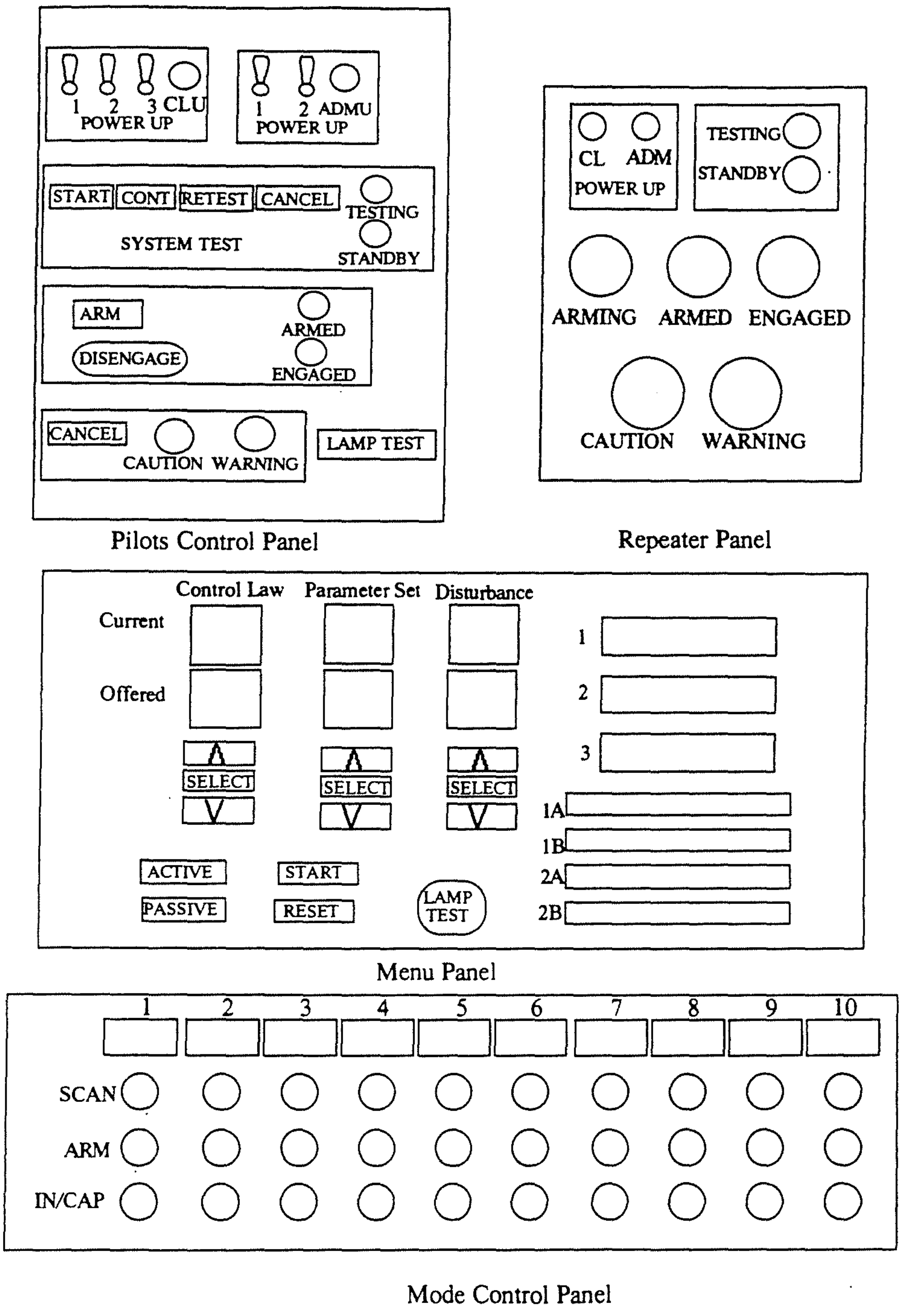


Figure 2. Act Lynx Control Panels

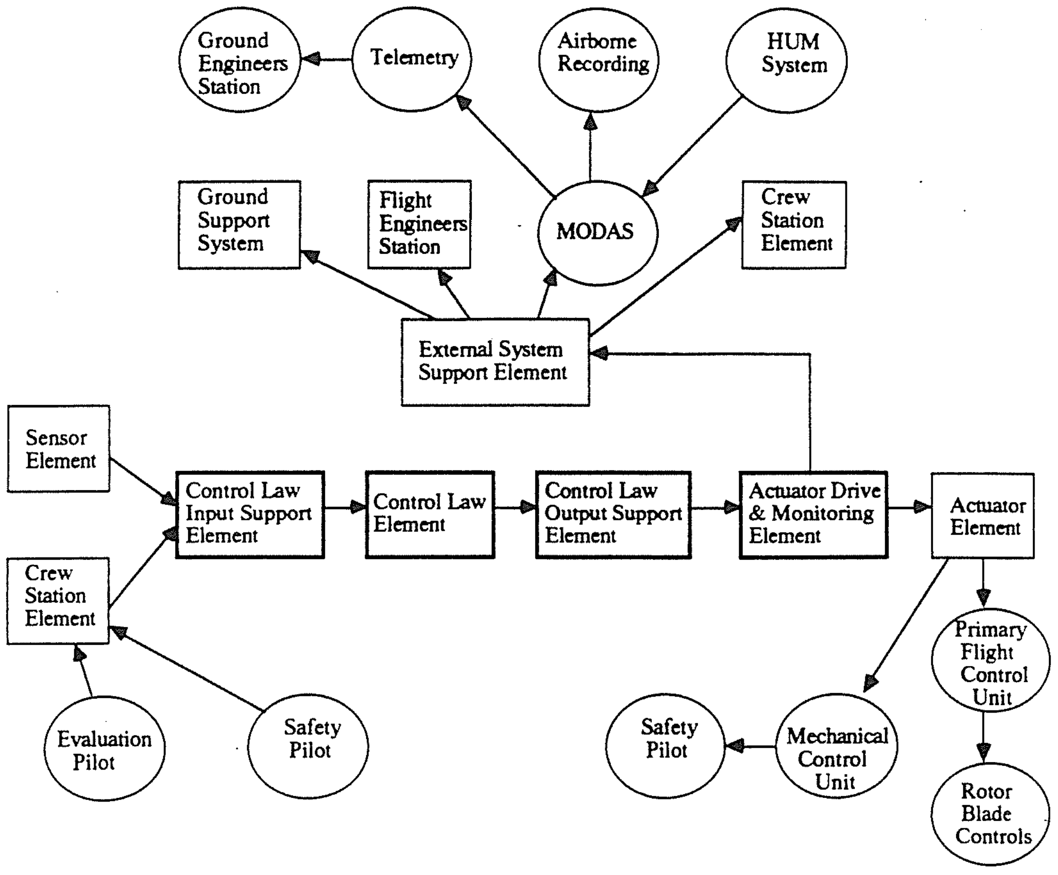


Figure 3: AACTS Logical Elements

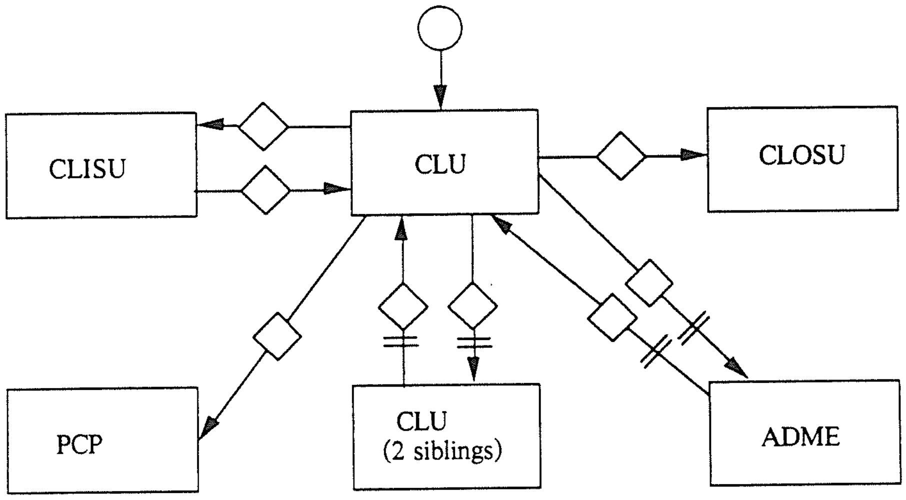


Figure 4. Connections to a CLU

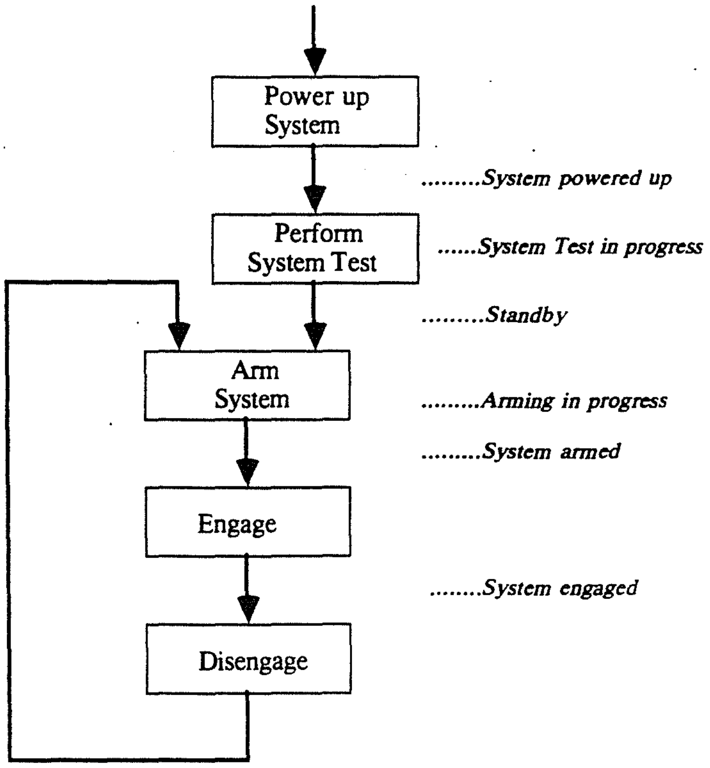


Figure 5. Possible System Control Flowchart

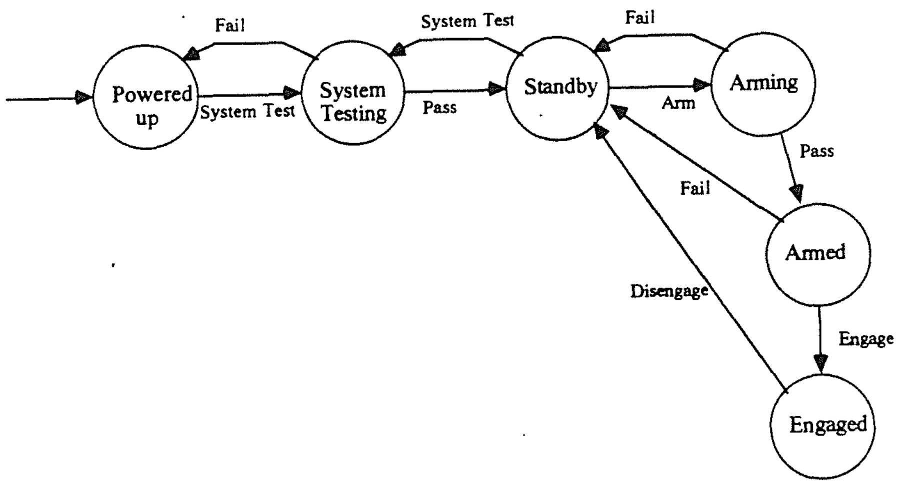
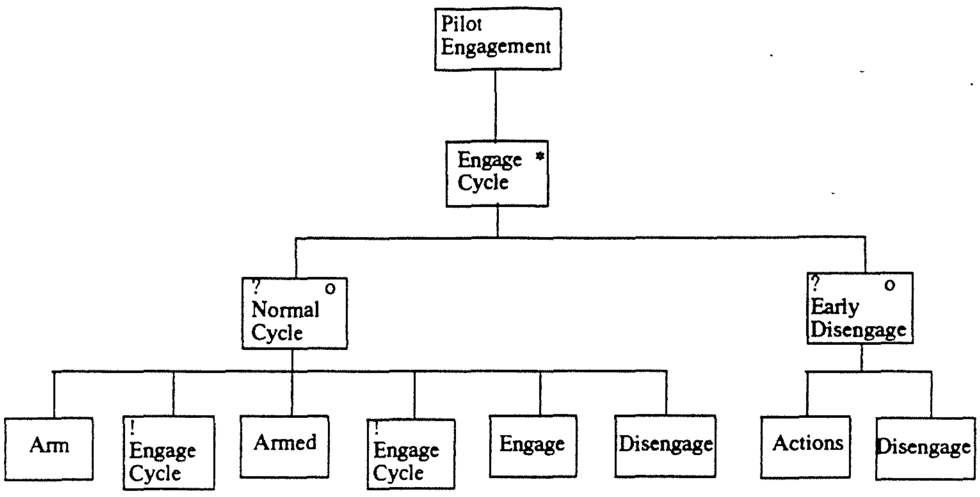


Figure 6. Possible FSM for System Control



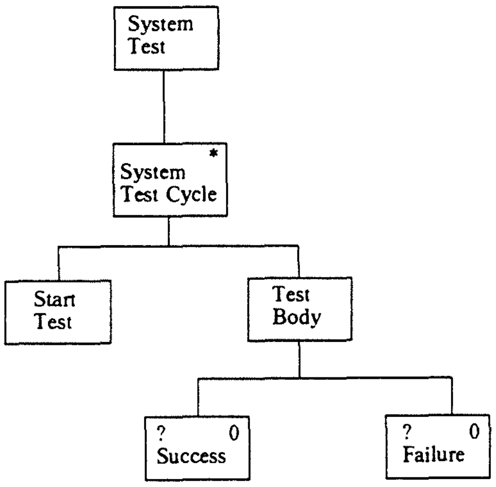
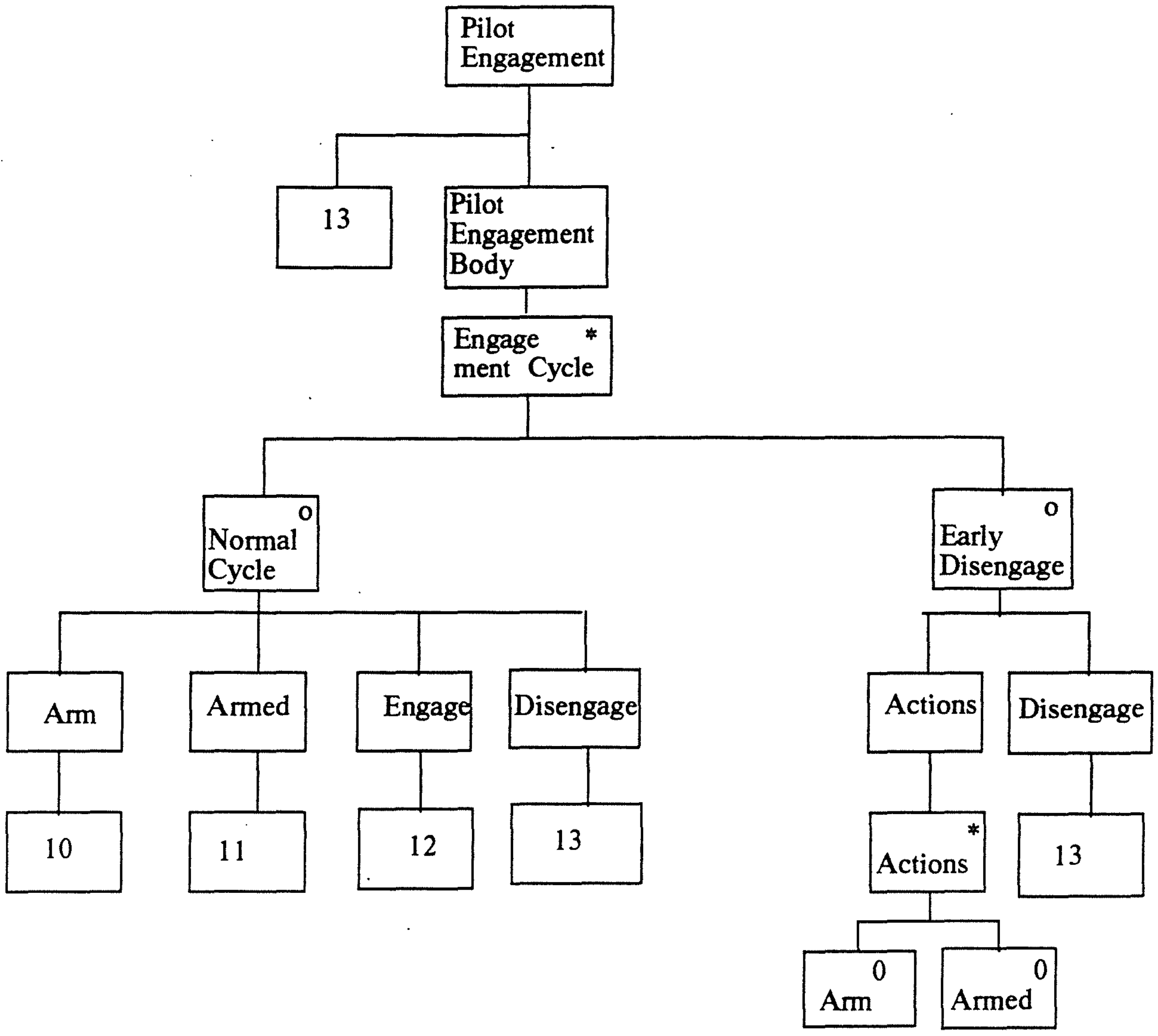


Figure 7: Pilot Engage and System Test Processes

| Action | Summary | Attributes |
|---|---|---|
| ARM | The pilot requests that the system be armed. | |
| ARMED | The actuator positions and the control law demands are in harmony | |
| ARM_DEFAULT_MODE | The initial arming of a default control mode. | ID: MODE_ID_TYPE |
| CANCEL_SYSTEM _TEST | A request to cancel the system test. | |
| CAPTURE | This is the signal to mode to go from ARM to ARM_AND_IN_CAP | ID: MODE_ID_TYPE |
| COMPLETED_SYSTEM _TEST | All tests of the system test have been successfuly completed | |
| CONTINUE_SYSTEM _TEST | Indication that the current test of the system test has been successfully completed. | |
| DISENGAGE | The system has been disengaged. This may happen before engagement (1) by the pilot pressing the disengage button or (2) by the system failing to get into the ARMED or ENGAGED state. It may happen whilst ENGAGED on receipt of a signal from an actuator relaying the fact that it has become disengaged | |
| DOWN_DISTURBANCE _REQUEST | The pilot wishes to be offered the previous valid disturbance, that is the first disturbance with a lower index number (ID) This is equivalent to the pilot presssing the DOWN button | |
| ENGAGE | The pilot requests (successfully) that the system be engaged. | |
| FAIL_TEST_STAGE | The current 'automatic' stage of the system test has not been successfully completed. | |

Figure 8. Typical list of actions.



Operations:
10. SV.SYSTEM_STATE:=ARMING;

11. SV.SYSTEM_STATE:=ARMED;
12. SV.SYSTEM_STATE:=ENGAGED;
13. SV.SYSTEM_STATE:=DISENGAGED;

Figure 9: Pilot Engagement

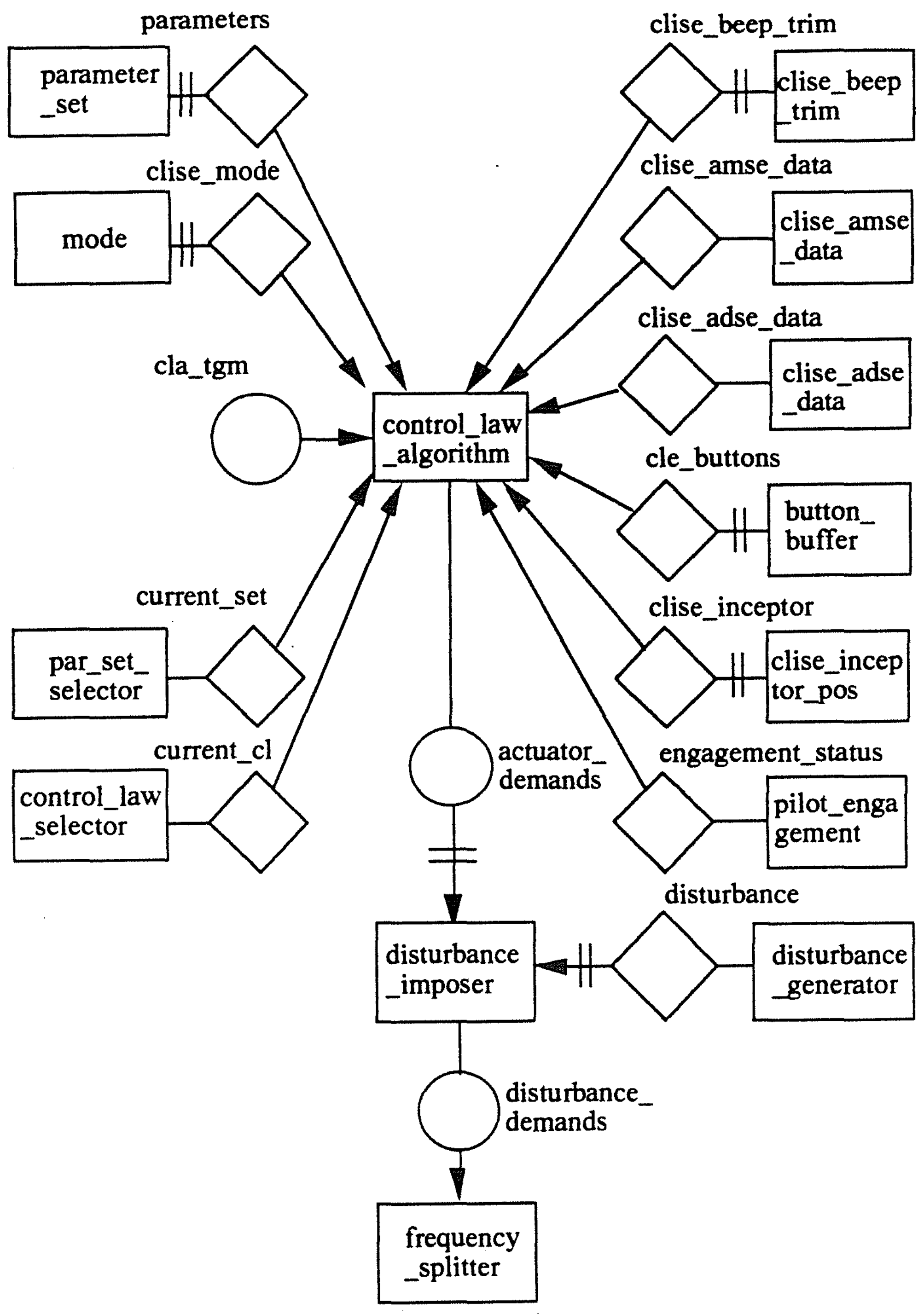
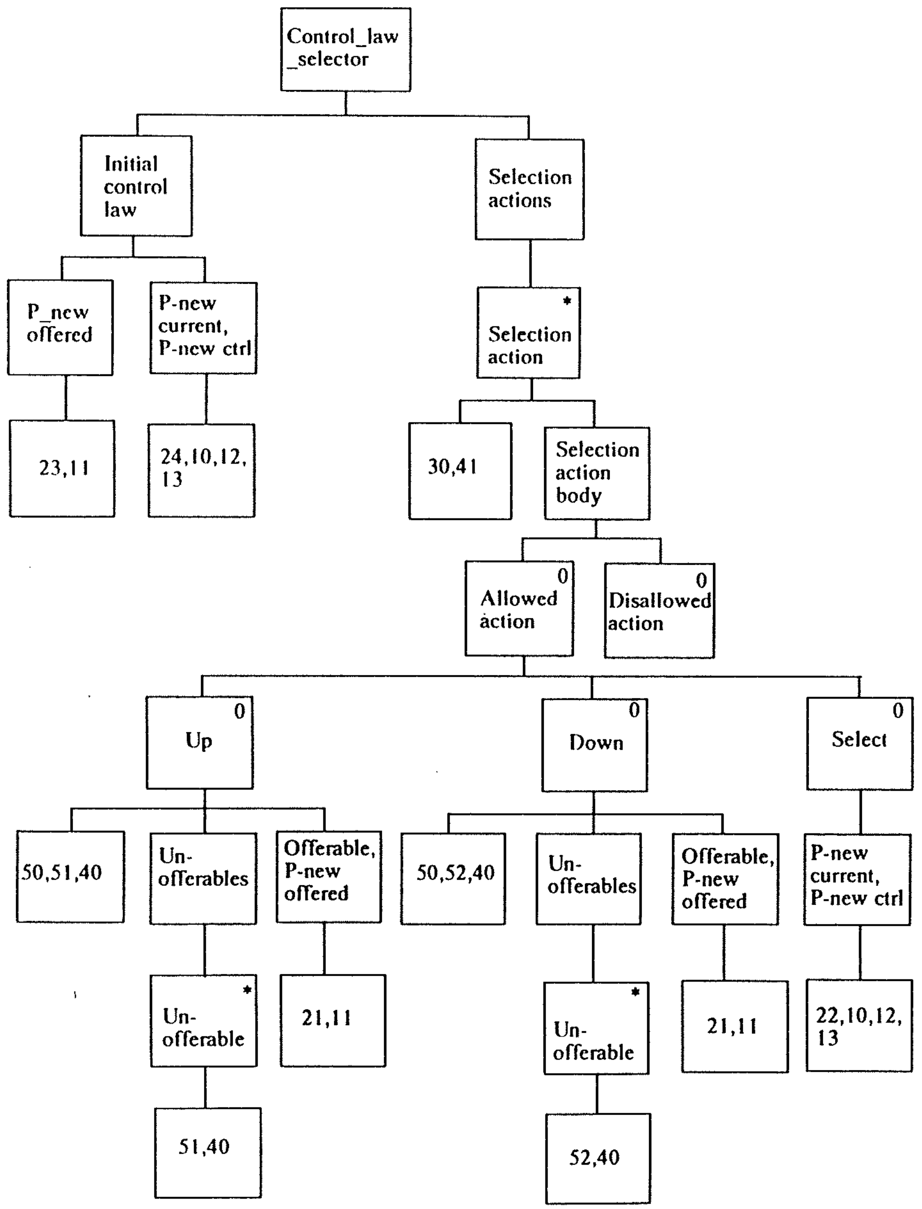


Figure 10 The CLE Control Law



(a) CLE Control Law Selector Process

```
10. @WRITE 1 CL_OUTPUTS NEW_CURRENT_CL
      ((ID =>SV.CURRENT_CONTROL_LAW))
11. @WRITE 1 CL_OUTPUTS NEW_OFFERED_CL
            ((ID => SV.OFFERED_CONTROL_LAW))
12. @WRITE 1 NEW_CONTROL_LAW NEW_CONTROL_LAW
          ((ID => SV.OFFERED_CONTROL_LAW))
13. @WRITE 1 NEW_LAW NEW_CONTROL_LAW
          ((ID => SV.OFFERED_CONTROL_LAW))
21. SV.OFFERED_CONTROL_LAW := CONTROL_LAW_ID;

22. SV.CURRENT_CONTROL_LAW :=
            SV.OFFERED_CONTROL_LAW;
23. SV.OFFERED_CONTROL_LAW := 1;
24. SV.CURRENT_CONTROL_LAW := 1;
30. @READ CL_SELECTION_CMDS

40. GET_SV(CONTROL_LAW_ID, PARAMETER_SET_SUBSET);

41. GET_SV(1,CL_PILOT_ENG_SUBSET);
50. CONTROL_LAW_ID := SV.OFFERED_CONTROL_LAW;

51. CONTROL_LAW_ID := CYCLIC_SUCC(CONTROL_LAW_ID);
52. CONTROL_LAW_ID := CYCLIC_PRED(CONTROL_LAW_ID);
```

(b) The CLE Control Law Selector Operations

Figure 11

91-74.17

```
UNIT IE
   STD-INFO
      LONGNAME
      REFERENCE IE
      [*]CLASSIFICATION-SET
      [*]SUMMARY
      This unit is connected to the
      inceptors of the evaluation
      pilot.
      [o]NARRATIVE
         NO
 MAIN-PART
    [o]TYPE
       ANALOGUE
    [o]BASE-REDUNDANCY
       SIMPLEX
    REPLICATION 3
    [o]UNIT-LVL-SYNCHRONISATION
       ASYNCHRONOUS
          FRAME-LAG
    [*]INTRA-UNIT-CONNECTIONS
  UNIT-SID
```

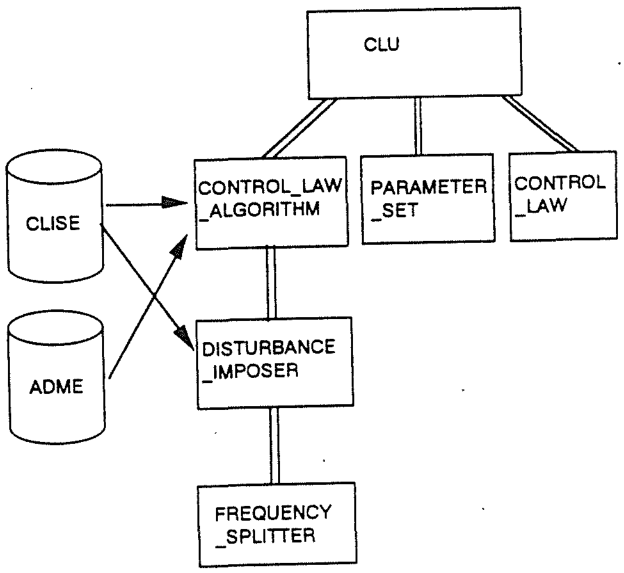
(a) Unit Description (analogue)

```
UNIT CLE
  STD-INFO
    LONGNAME
    REFERENCE CLE
    [*]CLASSIFICATION-SET
    [*]SUMMARY
    This unit houses the control
    law algorithm and associated
    processing. It is the middle processor
    in a three processor "lane".
    [o]NARRATIVE
       NO
 MAIN-PART
    [o]TYPE
       DIGITAL
    [o]BASE-REDUNDANCY
       SIMPLEX
    REPLICATION 3
    [o]UNIT-LVL-SYNCHRONISATION
       ASYNCHRONOUS
          FRAME-LAG 10
    [*]INTRA-UNIT-CONNECTIONS
    UNIT-SID
```

(b) Unit Description (digital)

```
CONNECTION IE_CLISE
   STD-INFO
      LONGNAME
      REFERENCE IECLIS
      [*]CLASSIFICATION-SET
      [*]SUMMARY
      [o]NARRATIVE
         NO
   MAIN-PART
      SOURCE IE
      DESTINATION CLISE
      [o]DATA-TRANSMISSION
         BROADCAST
      [o]SPEC-INTERFACE
         NO
      [o]CONSOLIDATION
         YES
            HISTORY_LENGTH 3
      [o]SIBLING_ERROR_MONITORING
         YES
            HISTORY_LENGTH 3
```

(c) Connection Description

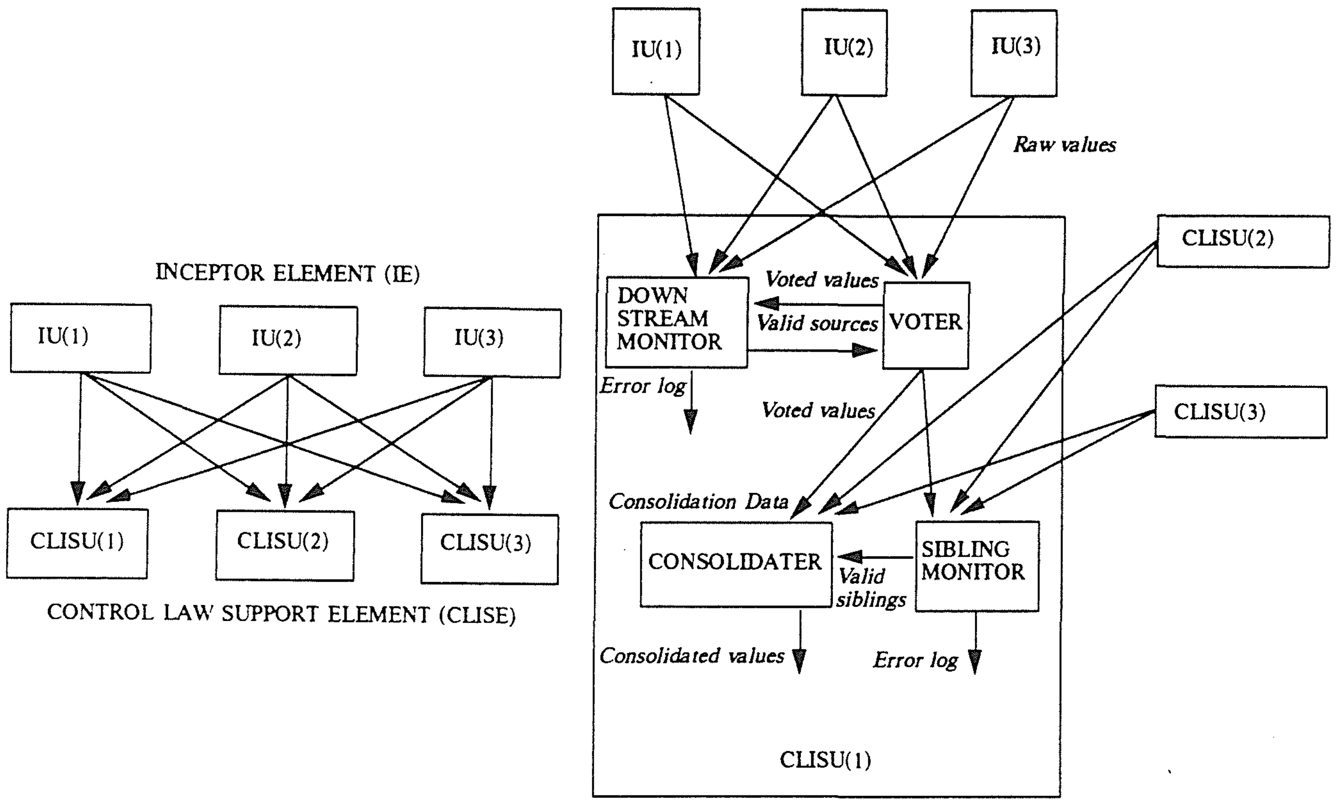


(d) CLU Implementation Diagram

Figure 12

```
with CLE_ID_TYPE_PACKAGE;
use CLE_ID_TYPE_PACKAGE; .
with SYSTEM;
package CLE_TASK_TYPE_PACK is
    function CURRENT_ID return CLE_ID_TYPE;
    task type CLE_TASK_TYPE is
        pragma PRIORITY (SYSTEM.PRIORITY'FIRST);
        entry INITIALISE(ID: in CLE_ID_TYPE);
        entry ENSURE_INITIALISATION;
        entry FRAME_START(FRAME_NUMBER: in NATURAL);
    end CLE_TASK_TYPE;
end CLE_TASK_TYPE_PACK;
```

Figure 13. CLE Package Specification



(a) IE to CLISE interconnection

(b) Schematic Diagram of Fault Processing

Figure 14

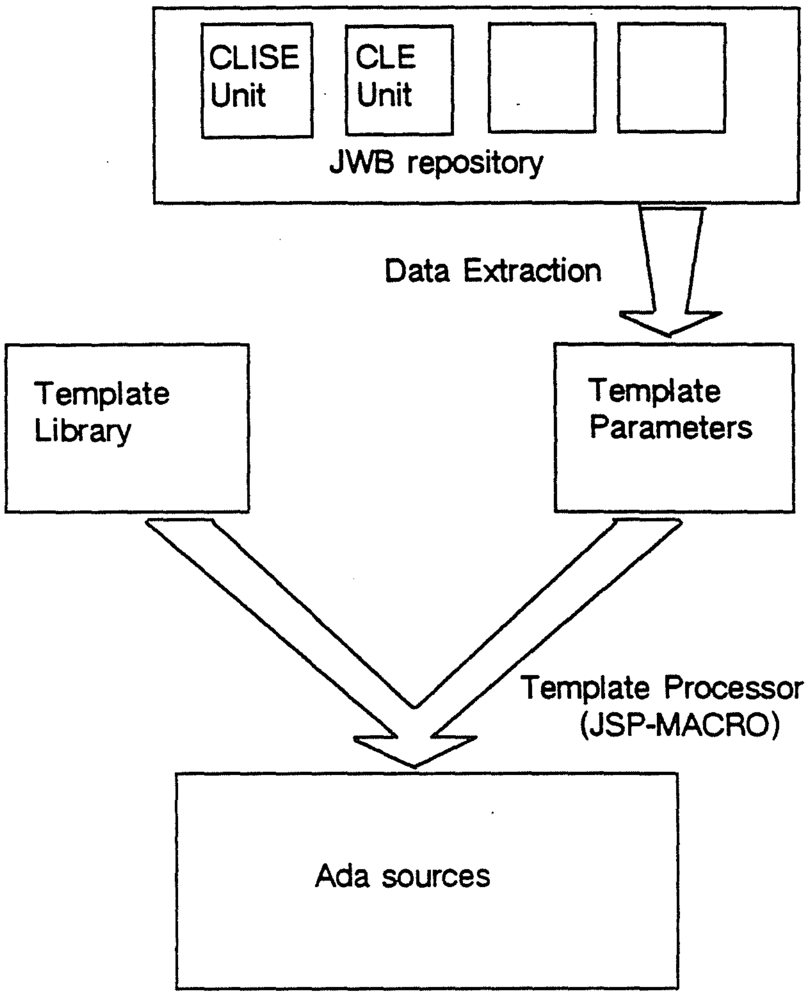


Figure 15 Operation of Code Generation Tool