Richard Botting,
Computing Science
University of Glasgow
Glasgow G12 8QQ
SCOTLAND
tel: +44 141 339 8855 ext. 2918
rmb@dcs.gla.ac.uk

A Formal and Structured Approach to the Use of Task Analysis in Accident Modelling.

R.M. Botting and C.W. Johnson
Department of Computing Science,
University of Glasgow,
United Kingdom,
G12 8QQ.
rmb@dcs.glasgow.ac.uk

11th September 1997.

Recent work(C.W.Johnson & A.J.Telford 1996, Johnson 1997) involving the application of formal notations to analyse accident reports has shown that the quality of these accident reports is poor, so much so that their conclusions can be misleading. The proposed solution has been to use formal notations in combination with traditional analysis to produce a report, the conclusions of which can be verified by formal reasoning. However, there are weaknesses with the formal notations used up until now. Firstly, they have not allowed the representation of all aspects of an accident or incident. In particular, human factors have either not been represented or not clearly delineated from system factors. In particular, there has not been an attempt to provide a task analysis of the incident. It will be shown how such a task analysis can be provided within a formal model. The advantages of this approach will be explained. Secondly, the notations used do not easily facilitate the system engineering concepts of modularity, encapsulation, or scalability. In consequence it is difficult to model different aspects of an accident, compose these different aspects to build up the model, or make changes to parts of the model without affecting the rest of the model. This paper explains how the most mature of the formal object-oriented specification languages, Object-Z, can remedy these problems. An air accident investigation report, issued by the Civil Aviation Authority, is used as a case-study.

1 Introduction

1.1 The problems with accident reports

Accident reports tend to be long documents containing a mass of data of varying degrees of relevance to the actual incident under consideration. Typically, an accident report will contain: a synopsis, a conclusion, and a body. The purpose of the conclusion is to explain the major causes of the incident and provide recommendations so as to avoid similar incidents in the future. The conclusions are claimed to be based on the body of the report. However, there are several problems with making such an assumption. The body consists of sections which tend to be written by different teams of experts. This often leads to inconsistencies. To ascertain all the available details of a particular event or state of the system it may be necessary to read through different sections of the report. Even if the report possesses adequate referencing it can still be an arduous task to glean the needed information.

Accident Reports are often incomplete. It has long been known that, in the field of Software Engineering, specifications of systems described using natural language often contain ambiguities and inconsistencies. They can also be incomplete. In an analogous way this is also true of accident reports. Given that the body of an accident report may contain such problems, the conclusions of the report can be erroneous or misleading.

Recent work(C.W.Johnson & A.J.Telford 1996, Johnson 1997) has shown that formal notations can be used in conjunction with traditional analysis to provide a better quality report. Building formal models of an accident can be used in helping to verify that:

- the main body of the report is unambiguous, consistent, and complete.
- the conclusions logically follow from the main body of the report.

The formal notations that have been employed(C.W.Johnson & A.J.Telford 1996, Johnson 1997) have been versions (temporal in some cases) of predicate calculus.

Predicate calculus only possesses constructs for reasoning but not for structuring a model. Notations without structuring capabilities do not easily facilitate: the modelling of different aspects in isolation, the composition of these different aspects to build up the model, the possibility of making changes to parts of the model without affecting the rest of the model. The fact that different aspects of an incident are not easily distinguished in such models would, in particular, imply that the modelling of users is not easily

distinguished from that of the system. In the analysis of an incident this would be an important distinction to make if a true understanding is to be gained.

Furthermore, in the analysis of human factors, it often provides insight to build a task analysis model. B.Kirwan & L.K.Ainsworth (1992) define task analysis as the study of what an operator (or team of operators) is required to do, in terms of actions and/or cognitive processes, to achieve a system goal. This analysis can be carried out separately from the analysis of the system and the two compared for incompatibilities. The information given by a task analysis will be structured. For example, a particular task may be described in terms of sub-tasks which may in turn be described in terms of sub-tasks. Building a model in a notation without the potential for such structuring would neither be easy, nor would the result be particularly legible. Thus a notation describing a task analysis should naturally provide structure. None of the previous work used such a notation, nor did they provide a task analysis model. The structuring facilities of the notation used in this paper will allow the building of a task analysis model.

1.2 The case-study.

The case-study concerns an aircraft incident which took place near Daventry. The aircraft was a Boeing 737-400, G-OBMM. Soon after take-off, from East Midlands airport, it was noted that guages indicating the oil pressures and oil quantities were low and decreasing. It was decided to bring the plane down. Once on the ground it was noticed that the cowls of both engines, and the underside of both wings and flaps immediately behind the engines were covered in engine oil.

The aircraft had been scheduled to undergo borescope inspections of the turbine sections of both engines during maintenance of the previous night. This was usually performed by the Line Maintenance shift. The Line Engineer, in charge of this shift, realised that they were understaffed that night. As he was the only Line Maintenance Engineer with the necessary authorisation he began to prepare one engine (engine no.1) for inspection. When he was ready he went over to the Base Maintenance hangar to ask for the borescope equipment as well as someone to assist him. The high pressure (HP) spool of the engine needed to be turned by a second person whilst he did the inspection. The Base Maintenance Controller was also qualified to do borescope inspections, but was in danger of allowing this authorisation to lapse. British Midland required staff to perform a minimum of two borescope inspections per year in order to keep their company authorisation. The

Base Maintenance Controller was also aware that Line Maintenance was understaffed, and therefore offered to take over supervision of the borescope inspections if the Line Maintenance engineer could supervise the moving of a Boeing 737-500. This offer was accepted. Although he had not carried out a borescope inspection for some time the Base Maintenance Controller did not make use of the task cards available.

There are three main stages to a borescope inspection of an engine: the preparation, the inspection itself, and the reassembly. Although the preparation and inspection were conducted as expected, the reassembly of the engines was not. Several tasks were omitted, the two high pressure(HP) rotor drive covers, one on each engine, as well as the corresponding O-rings, were not refitted, and an idle test run of the engines was not carried out. The entry relating to borescope inspections in the technical log was signed as being completed.

2 Existing Accident Analysis Techniques.

2.1 Fault Trees.

Fault tree analysis is widely used in the aerospace, electronics, and nuclear industries(Leveson 1986), for analysing the causes of hazards. It uses boolean logic to describe the combinations of individual faults that can constitute a hazardous event. For the purposes of accident analysis the semantics of fault trees have to be adjusted, however, the basic tree structure still remains the same. Tree nodes represent events or non-events such as 'fitter omits to put back O-ring'. It is a top-down method, the top event stating the result of the incident, whilst each level down describes the events in more and more detail. A simplified fault tree of the Daventry incident is given in Figure 1.

The most serious weakness with fault trees is that they do not explicitly show the event sequence. Simple AND and OR gates do not convey any notion of time ordering or time delay. It will often be the case that events will be interleaved time-wise, instead of in sequence, at any one level. Adding other types of gates to the syntax to help solve this problem only reduces the tractability of the technique, which is one of its major assets. Another weakness of fault tree analysis is that it does not provide for modularity, even the use of the extended technique of hierarchical fault tree analysis still provides only a very limited form of modularity. Finally, fault trees do not separate human from system factors.

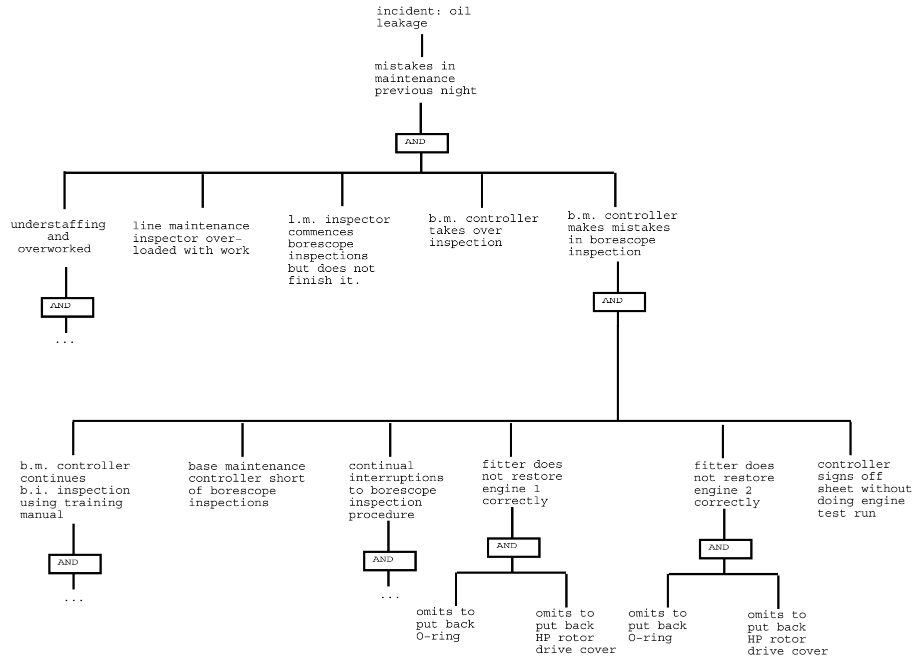


Figure 1. A fault tree representing the Daventry incident.

2.2 Cause Consequence Diagrams.

Cause-Consequence Analysis starts with a critical event and determines the causes of the event and the consequences that could result from it. The symbols available to the technique include the logic gates AND and OR, and in fact the causes can be given by fault trees. A cause-consequence diagram of the Daventry incident is given in Figure 2. Separate diagrams will be required for each critical event, and there may be several different causes of a critical event.

Unlike fault trees cause-consequence diagrams show the sequence of events explicitly. However, diagrams can become difficult to read. Again, there is no distinction made between human and system observations, and there is no way to integrate human factor techniques such as task analysis.

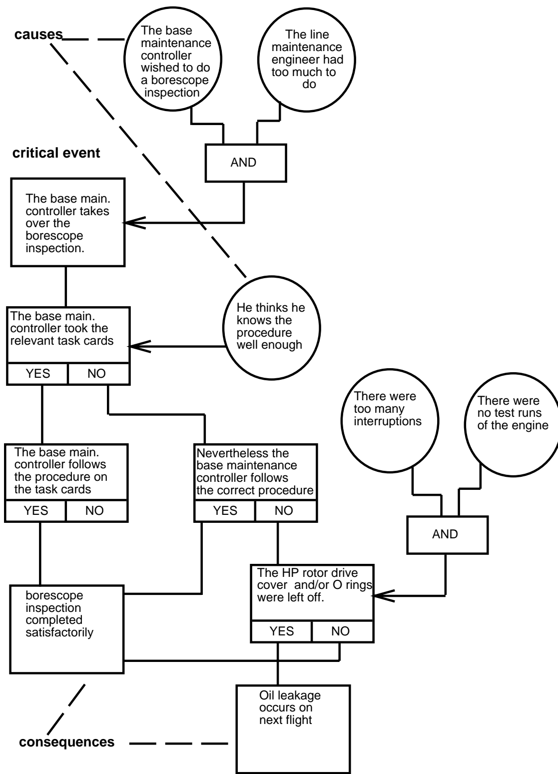


Figure 2. A cause–consequence diagram of the Daventry incident.

2.3 Predicate Logic.

The two techniques discussed above are graphical. Neither provides the possibility of reasoning about the particular incident. Predicate logic can be used both to represent and reason about the events that lead up to major accidents(C.W.Johnson & A.J.Telford 1996, Johnson 1997). In the Daventry incident the simple fact that oil leakage will occur if the O-ring or HP rotor drive cover is not replaced after the inspection of the turbine of engine no.1 can be expressed by:

$$\neg\ replace(engine1, O_ring) \vee \neg\ replace(engine1, HP_rd_cover) \Rightarrow oil_leakage(engine1)$$

Many such logic statements can be written to describe the incident, and combining these with the axioms of predicate logic, reasoning can be carried

out to prove or disprove assertions concerning the incident.

In conclusion, graphical notations such as fault trees and cause-consequence diagrams can quickly become intractable as more and more events are represented, and thus lose there main advantage over textual notations. For the same reason it would not be sensible to build a large model using the predicate calculus. What is needed is an engineering approach: a method which includes the concepts of encapsulation and composition so that different aspects can be built up in isolation and then composed in a structured way. None of the techniques discussed in this section distinguish easily between human and system factors. Neither would it be easy to integrate a task analysis of the incident.

3 Choosing a notation.

The well established formal specification languages Z(Dillar 95) and VDM-SL(Jones 1990) provide means of building more structured models which still possess all the reasoning power of the predicate calculus. However, these languages still do not offer some of the more powerful structuring capabilities associated with object oriented notations. Due to this weakness, formal specification languages have been developed, such as Object-Z(Duke, King, Rose & Smith 1991), MooZ and VDM++ which are object-oriented in nature but are based on either Z or VDM. Object-Z is the most mature of these notations(Lano & Haughton 1994) and it was therefore chosen to avoid the weaknesses inherent in previous notations used for accident analysis.

Object-Z is an extension of the formal specification language Z which accommodates object orientation. The main reason for this extension is to improve the clarity of large specifications through enhanced structuring(Duke et al. 1991). A typical Z specification consists of an unorganised collection of state and operation schemata. An example of a Z operation schema is given below. There is, however, no explicit statement of which state schema this operation schema applies. A large specification, containing many such operation as well as state schemata, is likely to become unmanageable.

Push

$\Delta(items)$
$item? : Item$

$\#items < max$
$items' = \langle item? \rangle \frown items$

The introduction of classes means that relevant operation schemata can be grouped with a particular state schema, allowing for a more structured and readable specification. Each class can be considered in isolation from the rest of the specification. Furthermore, complex classes can be specified in terms of simpler classes through the structuring techniques of inheritance and instantiation.

The philosophy encouraged by an object oriented approach is different from that of specification languages such as Z, as it leads to a stronger focus upon partitioning of a specification into well-designed and encapsulated subsystems(Lano & Haughton 1994). The syntax for Object-Z can be found in Appendix A. The remainder of this paper focuses on the application of this notation.

4 Introducing the model: Encapsulation and Composition.

In this section the staffing situation given in the case-study is represented in Object-Z. The Staffing situation can be considered in isolation from other aspects of the incident such as the borescope inspection or the state of the plane. In other words, the properties of the staffing situation can be encapsulated within an Object-Z class. For example the fact that the Line Maintenance shift is understaffed can be represented in one class while the procedural aspects of the borescope inspection can be encapsulated in another class. Furthermore, simple classes can be composed to form more complex classes, so a class representing the complete staffing situation can be built from smaller classes representing, for example, the Line Maintenance staffing and the Base Maintenance staffing.

4.1 Encapsulating properties of a supervisor.

The first thing which is usually done in writing an Object-Z specification, before any classes are defined, is to identify the basic sets which are needed. In the case of the staffing situation it will be assumed that there is a set $Personnel$ which consists of staffing roles such as $LM_manager$ and $BM_controller$.

$[Personnel]$

$$LM_manager, BM_controller : Personnel$$

The first class to be defined is a class *Supervisor*. Instances of such a class would provide details about a particular supervisor identified in the Daventry report, i.e. the Line Maintenance Engineer or the Base Maintenance Controller. The class *Supervisor* has two state variables (a third variable is added in the next section). The first variable *person* identifies a particular supervisor, whilst the second *BI_qualification_status* provides information concerning the status of the particular supervisor's borescope inspection authorisation status. This is justified by the following quotation from the accident report:

> *"British Midland Company rules required staff to perform a minimum of two 750 hour borescope inspections in a twelve month period in order to keep their Company Authorisation. The Controller (Base Maintenance) had recognised that, because of the scarcity of opportunities, he was in danger of allowing this authorisation to lapse ..." (D.F.King 1996)*

Supervisor

$BI_qual_status_values ::= completed_minimum \mid in_need_of_inspections$

$person : Personnel;$
$BI_qualification_status : BI_qual_status_values$

4.2 Including one class in another.

The night shift staffing is split into two sections, Line Maintenance staffing and Base Maintenance staffing. It is therefore natural to describe two classes, *LM_staffing* and *BM_staffing*, which encapsulate the properties of each particular section.

$LM_staffing$

$manpower_status_values ::= OK \mid understaffed$

$supervisor : Supervisor;$
$no_of_staff : \mathbb{Z}$
Δ
$manpower_status : manpower_status_values$

$(no_of_staff < 6 \land manpower_status = understaffed)$
$\quad \lor manpower_status = OK$

$\textit{INIT}$
$supervisor.person = LM_manager$
$no_of_staff = 4$
$manpower_status = understaffed$

For example, the class *LM_staffing* has three state variables. The first variable *supervisor* is of class type *Supervisor*. There are two ways one class can be included within another: either by instantiation as above, or by inheritance, examples of which are given later. The variables of the class *Supervisor* can be accessed by dot notation as given in *INIT*. The concepts of instantiation and inheritance allow for composition of classes so that classes can remain relatively small, and the specification manageable.

The variable *manpower_status* is a secondary variable. Its value depends on that of the value of the variable *no_of_staff*. Again, this part of the model can be justified by reference to the accident report:

> *"However, when he came back into work, at about 1930 hrs on the Wednesday evening, he found that the manpower of the shift had not been supplemented. Instead of the nominal shift complement of six, there were only four on duty that night, two of whom, including the shift leader, were doing extra nights to cover shortfalls."(D.F.King 1996)*

4.3 Composition of staffing classes.

The class *BM_staffing* is similar to the class *LM_staffing* and is therefore not given here. The two classes *LM_staffing*, and *BM_staffing* can be used to build a class which represents the complete night staffing situation.

Staffing

$lm_staffing : LM_staffing;$
$bm_staffing : BM_staffing$

$\text{INIT} \mathrel{\widehat=} [\, lm_staffing.\text{INIT} \wedge bm_staffing.\text{INIT} \,]$

4.4 Using the model to describe alternative scenarios.

The class *LM_staffing* has been defined in a general way. That is to say the state defines variables which allow the modelling of situations other than the one found in the Daventry scenario. In the above definition, these variables have been initialised to take their corresponding values found in the Daventry report. To allow for different scenarios, the initialisation values could be altered. For example, if the accident was to be avoided, the initialisation of the class *LM_staffing* should have been:

INIT

$supervisor.person = LM_manager$
$no_of_staff = 6$
$manpower_status = OK$

5 Introduction of tasks to the model.

The Daventry report often refers to tasks and procedures that the supervisors need to carry out:

> *"So, he (the Base Maintenance Controller) offered to take over the borescope inspections personally if the Line Maintenance Engineer could take over the moving of the (Boeing) 737-500 from the Ramp to Base. This offer was accepted and the transfer of the task was noted, ..." (D.F.King 1996)*

Thus it seems appropriate to represent the tasks (or actions) that a supervisor is expected or required to carry out, including how these tasks are related or structured. The field of task analysis(Diaper 1989, B.Kirwan & L.K.Ainsworth 1992) has many techniques for providing such representations. One general method is that of hierarchical task analysis (HTA). This technique includes the concepts of goals, sub-goals, tasks, and plans. This section will show how Object-Z can be used to represent HTA. However, the

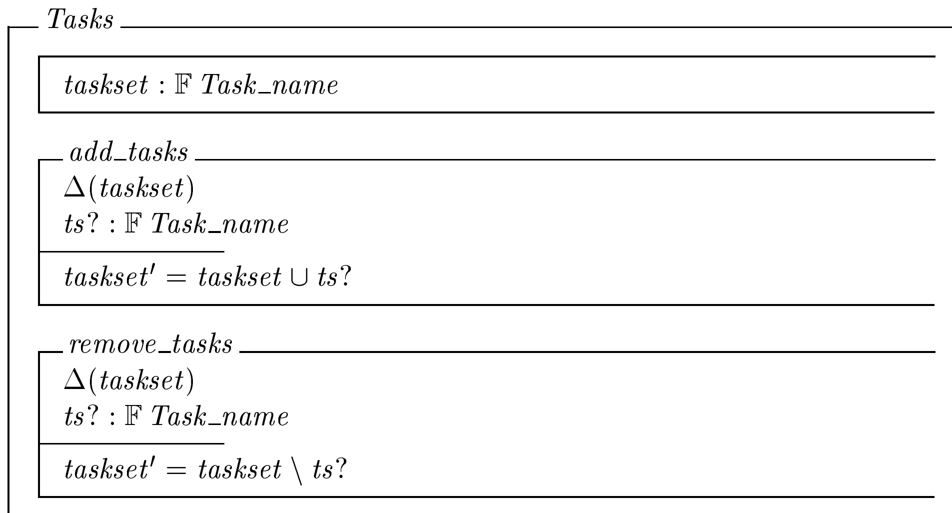
Object-Z model will not stick strictly to the approach of HTA. There are several reasons for this. Firstly, it is often difficult to differentiate between tasks and goals in the Daventry report. Secondly, task analysis notations, such as HTA, provide means of reasoning about expected user behaviour. They do not model erroneous behaviour. This clearly raises problems if we want to model the ways in which the omission of a task can lead to an accident.

5.1 Describing tasks by means of a class.

To introduce the concept of tasks, a new basic set *Task_name* is assumed to contain the names of those tasks which can be carried out.

[*Task_name*]

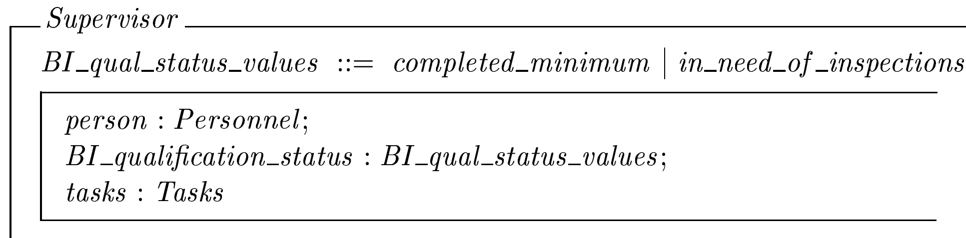
A new class *Tasks* can now be defined. One important event described in the Daventry scenario, and quoted above, was the transfer of tasks between the Line Maintenance Engineer and the Base Maintenance Controller. Given that a class *Tasks* possesses a set of tasks it would seem appropriate to have operations which can add or remove tasks from this set.



5.2 Including tasks in the model.

As it is the supervisors that are associated with carrying out tasks in the Daventry report an extra state variable *tasks* is now added to the class

Supervisor. Adding this variable means that all the information concerning supervisors is still grouped in the one class, *Supervisor*.



Supervisor

$BI_qual_status_values ::= completed_minimum \mid in_need_of_inspections$

$person : Personnel;$
$BI_qualification_status : BI_qual_status_values;$
$tasks : Tasks$

The class *BM_staffing* could be initialised so that the Base Maintenance Controller's initial tasks included *move*_737_500. Application of the appropriate operations of the sub-class *Tasks* would then describe the event given diagrammatically in Figure 3.

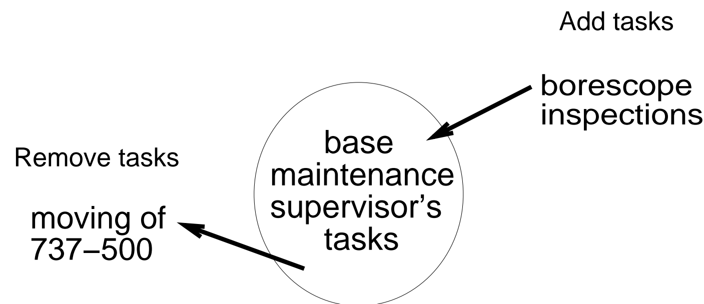


Figure 3. A change in the task set.

The last two sections have given an indication of how a model of the staffing situation can be built up using the concepts of encapsulation and composition, and has also introduced a simple model of the task situation. The next sections extend the idea of the task model by looking at the borescope inspection procedure. This can be considered in isolation from the staffing situation. Ultimately the top class of the staffing situation and the top class of the borescope inspection procedure can be combined into one main class. This again shows the power of Object-Z for building large scale models of complex human and system failures.

6 Developing the task analysis model: Modularity.

The last section considered how a simple class *Tasks* could be used to model the interchange of tasks between the 'users' involved in the Daventry incident. Consideration of the task situation with regards to the borescope inspection is also relevant to an understanding of the accident. Indeed some of the theory of task analysis is particularly appropriate to this aspect of the Daventry incident. The borescope procedure is described in Section 1.6.4 of the accident report. This is not shown here for reasons of brevity. A simplified version of the borescope procedure is given in Figure 4. The justifications for incorporating a task model within the Object-Z model rather than making use of HTA model will be discussed in this section.

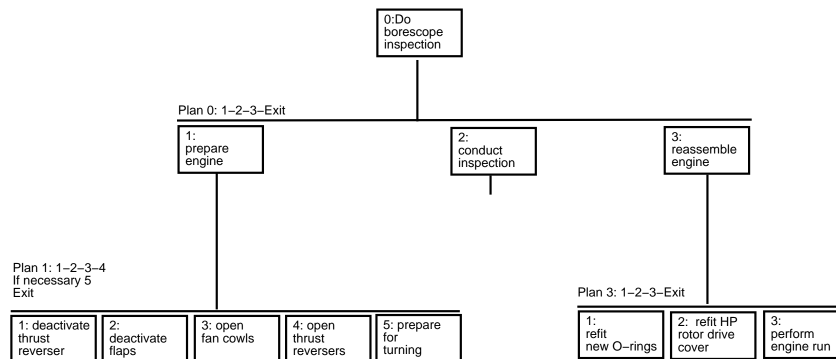


Figure 4. Hierarchical task analysis diagram of the borescope inspection.

6.1 The borescope inspection procedure.

The borescope inspection procedure is split into three sections: preparation of the engine, conducting of the inspection, and reassembly of the engine, which suggests there should be three corresponding classes. To show how task analysis can be developed in Object-Z, the reassembly section will first be considered in isolation. A class representing reassembly is given below. The operations of the class correspond to the tasks given in Section 1.6.4 C(1) of the Daventry report:

> *"...the fitting of new O-rings on the HP rotor drive cover, reinstallation of the cover onto the AGB and the requirement to perform an idle power ground run of the engine."(D.F.King 1996)*

$BI_reassembly$

$engine : Engine$

$Refit_new_Orings$ see Section 1.6.4, part of C.1

$$Refit_new_Orings \mathrel{\widehat=} [\, \Delta(engine) \mid engine.O_rings_status = unfitted \land engine.O_rings_status' = new_refitted \,]$$

$Refit_HP_rd_cover$ see Section 1.6.4, part of C.1

$$Refit_HP_rd_cover \mathrel{\widehat=} [\, \Delta(engine) \mid engine.HP_rd_status = uncovered \land engine.HP_rd_status' = covered \land engine.engine_run_performed' = not_performed \,]$$

$Perform_engine_run$ see Section 1.6.4, part of C.1

$$Perform_engine_run \mathrel{\widehat=} [\, \Delta(engine) \mid engine.engine_run_performed = not_performed \land ((engine.HP_rd_status = uncovered \lor engine.O_rings_status = unfitted) \Rightarrow engine.oil_status' = oil_leak) \land engine.engine_run_performed' = run_performed \,]$$

This may, at first, appear to be a complex and long-winded description. However, it is evident that this class contains a great deal more information than the corresponding section of the HTA diagram. This level of detail will be needed in any serious analysis of an accident.

The state variable *engine* is of class type *Engine*. *Engine* possesses such state variables as O_rings_status, HP_rd_status, etc. The class $BI_reassembly$ includes operations which describe what actions are taken corresponding to the tasks named above. Given the extra level of detail, analysis of task sequences can be carried out. In particular, the body of the operation $Perform_engine_run$ states that if the O-rings are not fitted or the HP rotor drive cover is not replaced then an oil leak will occur.

As it stands the operations can be carried out in any order, whereas the borescope inspection procedure implies an order in which the tasks are carried out. A class *Tasks* has been defined, in Section 5, which described a set of tasks which a supervisor needed to perform, whereas here there needs to be some order given to these tasks. For example, running the engines

before the rotor drive covers are refitted is clearly not the same as running the engine after they are refitted. To this end a class *Plan* will be defined, but first, so that this class can incorporate the notion of the omission of a task, a more general discussion on tasks and human errors is given.

6.2 Tasks and Errors.

Operators make mistakes. Such errors as omissions of necessary tasks and commissions of unwarranted tasks are included in Hollnagel's classification of human errors(Hollnagel 1993). In the Daventry incident the borescope inspection procedure is not followed. Some tasks are omitted. Moreover, Lucy Suchman says that the circumstances of our actions are never fully anticipated and are continuously changing around us(Suchman 1987). Task Analysis notations, typically, provide little means of modelling these adjustments to plans. Instead, it will be shown how Object-Z can be used to represent changes to a plan as well as the scenarios implicit in an HTA diagram. This work will build on the previous use of CSP to model human error scenarios(P.C.Wright, R.E.Fields & M.D.Harrison 1994, R.E.Fields, P.C.Wright & M.D.Harrison 1995).

6.3 Adjusting the class Tasks.

The class *Plan* given below is a modified version of the class *Tasks*. It contains a state variable *tasklist*, which is of type: a sequence of task names. The current task is given by the variable *current_task*. The operation *next_task* removes the head of the sequence *tasklist*. This is carried out when the current task has been performed. The next task in the sequence then becomes the new current task. There is also an operation *omit_a_task* which allows the explicit representation of a task being omitted, and an operation *add_a_task* which models commission. Operations which model other errors given in Hollnagel's classification(Hollnagel 1993) could be added if required.

Plan

$$tasklist : \mathrm{seq}\ Task_name$$
$$\Delta$$
$$current_task : Task_name$$
$$current_task = head\ tasklist$$

next_task

$$\Delta(tasklist)$$
$$tasklist' = tail\ tasklist$$

omit_a_task

$$\Delta(tasklist)$$
$$t? : Task_name$$
$$head\ tasklist = t?$$
$$tasklist' = tail\ tasklist$$

add_a_task

$$\Delta(tasklist)$$
$$t? : Task_name$$
$$tasklist' = \langle t? \rangle \frown tasklist$$

6.4 Ordering the operations.

The task *Plan* can be included in the class *BI_reassembly* so that the variable *task_list* now names the tasks to be carried out in some specified order. The operations of *BI_reassembly* should then be applied in a corresponding order. To be able to state this constraint a pre-condition is added to each operation so that it can only be applied if it matches up with the task name given by *current_task*.

$BI_reassembly$

$Plan$

$engine : Engine$

$\textsc{Init}$

$tasklist = BI_reassembly_tasks$

$Refit_new_Orings$ see section 1.6.4, part of C.1

$$Refit_new_Orings \mathrel{\widehat=} [\, current_task = refit_new_Orings \,] \land \\ [\, \Delta(engine) \mid engine.O_rings_status = unfitted \land \\ engine.O_rings_status' = new_refitted \,] \land \\ next_task$$

...

For example, the operation $Refit_new_Orings$ can only be applied if the pre-condition that $current_task = refit_new_Orings$ is satisfied. If the task is carried out, the sub-operation $next_task$ will update the task list by removing the task name $refit_new_Orings$. $current_task$ then takes on the value of the next task name in the list, which in this case would be $refit_HP_rd_cover$. An operation can now be considered as describing a task at two levels: at the human factors level there is the task name, and at the software engineering level this is made more precise by giving pre- and post-conditions to define the effects of carrying out the task.

The inclusion of the class $Plan$ means that operations such as $omit_a_task$ are available to the class $BI_reassembly$. As was said in Subsection 6.1 if either of the reassembly sub-tasks: refitting the O-rings or replacing the HP rotor drive cover, is omitted then carrying out the sub-task of performing an engine run will result in an oil leak. In the case of the Daventry incident none of the three reassembly sub-tasks are carried out. So the sequence of operations representing this scenario will include:

$omit_a_task(refit_new_Orings);$
$omit_a_task(refit_HP_rd_cover);$
$omit_a_task(perform_engine_run);$

This will leave the engine state with neither the O-rings fitted nor the HP rotor drive cover on. The fact that the engine run has not been performed

implies that an oil leak will occur when the plane is in flight.

In general, not all combinations of relevant class operations and the omission operation can occur. Whether a particular scenario can occur will depend on the pre-conditions, other than those stating the task name. A simple example is that of the operation *Refit_new_Orings* which can only be applied if $engine.O_rings_status = unfitted$. Thus, pre-conditions could be added to relevant operations of the model to stop certain unwanted scenarios occurring. Given these pre-conditions, the system - that the model represents - could then be considered to see if it could be re-designed to match the new model. In the case of the Daventry incident a required pre-condition of an operation *Fly_plane* should be that the HP rotor drive covers have been refitted. The procedures should be so designed so as to ensure that this pre-condition will be satisfied. For example, the carrying out of the operation *Perform_engine_run* would guarantee this. So a recommendation of the accident report should be that this task be made a priority.

7 Hierarchy of tasks: Scalability.

In the last section a class *BI_reassembly* was described. Similar classes, *BI_preparation* and *BI_conduct*, representing the other sub-tasks in the borescope inspection, can be described. Similarly to Section 4, the above classes could be combined to form a larger class *BI_procedure* . The tasks relevant to this larger class will be at the top level of the hierarchy. In *BI_procedure* the task list will contain the task names *bi_preparation*, *bi_conduct*, and *bi_reassembly*. Again there should be a relationship between the operations of the class and the task names given in the task list. In the class *BI_reassembly* this relationship was an isomorphism. However, it is not quite so simple if we want to model the omission of a task. One simple definition of, for example, the operation *Reassemble_BI* is:

$$
\begin{array}{l}
Reassemble_BI \mathrel{\widehat{=}} [\, current_task = bi_reassembly \,] \wedge \\
Refit_new_Orings \wedge Refit_HP_rd_cover \wedge Perform_engine_run \wedge \\
next_task
\end{array}
$$

This does not model the possible omission of sub-tasks. Instead, what is required are two operations corresponding to each named task. These operations are similar to pre- and post-conditions, but they can not be combined into one operation. The first operation names, by way of a list, what sub-tasks are to be carried out, whilst the second operation states that this sub-task list is now empty. For example, between the application

of the operations $Reassemble_BI_init$ and $Reassemble_BI_end$ given below, sub-operations from the class $BI_reassembly$ would be applied. If the correct procedure was followed these operations would be $Refit_new_Orings$, $Refit_HP_rd_cover$, and $Perform_engine_run$ in that order. However, it would also be possible, in this model, that one or more operations were omitted, or that, operations were committed.

$BI_procedure$

$Plan$
$BI_preparation[prep_tasklist / tasklist, omit_a_prep_task / omit_a_task]$
$BI_conduct[conduct_tasklist / tasklist, omit_a_conduct_task / omit_a_task]$
$BI_reassembly[reassembly_tasklist / tasklist, omit_a_reassemble_task / omit_a_task]$

INIT
$tasklist = BI_tasks$

$$Prepare_BI_init \mathrel{\widehat=} [\, current_task = bi_preparation \wedge prep_tasklist = BI_preparation_tasks \,]$$

$$Prepare_BI_end \mathrel{\widehat=} [\, current_task = bi_preparation \wedge prep_tasklist = \langle\, \rangle \,] \wedge next_task$$

$$Conduct_BI_init \mathrel{\widehat=} [\, current_task = bi_conduct \wedge conduct_tasklist = BI_conduct_tasks \,]$$

$$Conduct_BI_end \mathrel{\widehat=} [\, current_task = bi_conduct \wedge conduct_tasklist = \langle\, \rangle \,] \wedge next_task$$

$$Reassemble_BI_init \mathrel{\widehat=} [\, current_task = bi_reassembly \wedge reassembly_tasklist = BI_reassembly_tasks \,]$$

$$Reassemble_BI_end \mathrel{\widehat=} [\, current_task = bi_reassembly \wedge reassembly_tasklist = \langle\, \rangle \,]$$

The complete scenario of how the borescope inspections were carried out by the night maintenance staff can now be analysed by considering the relevant sequence of operations. The model also allows the analysis of other scenarios, the consequences of which may also be potentially disastrous. If changes to the system were considered desirable operations could be adjusted so as to analyse these changes.

The borescope inspection procedure modelled here has only two levels in its hierarchy of tasks. However, it can easily be seen that having more hierarchical task levels could be modelled using the technique described above.

8 Conclusion.

Previous work has shown that the use of formal notations in combination with traditional analysis can improve the quality of accident reports. However, there were several weaknesses in these notations. This paper has shown that Object-Z can remedy some of these weaknesses. Object-Z, through encapsulation and composition, provides the mechanisms for structuring a model. Encapsulation allows different aspects to be modelled separately, whilst composition implies that classes can be built-up from smaller classes. It is also suggested that the specifications are much more readable. Present work aims to provide empirical validation for this claim. A further advantage is that small changes to the specification will only affect the immediate class or classes involved. Lastly, it has been shown how a simple task analysis representation can be incorporated within a system model. This allows direct consideration of the consequences of operator actions.

The task model described here only considers the case when the task plans are simple sequences; this is not generally the case. A more typical plan may include repetition, choice, and concurrency. Ongoing work has shown that these constructs could be modelled by extending the Object-Z approach given here. However, application of the approach to a simple example produced a result which was complex and difficult to read. Even so it may still be worthwhile to use in safety-critical cases. An alternative notation, a combination of CSP and VDM-SL, is offered in R.E.Fields et al. (1995), but is not applied to a large example.

Another strength of the Object-Z model is that it allows consideration of scenarios other than that which occurred in the Daventry incident. This implies that other similar potential errors may be detected. Furthermore, the specification can be adjusted so as to consider the consequences of re-designing the system so as to avoid such errors.

References

B.Kirwan & L.K.Ainsworth, eds (1992), *A Guide to Task Analysis*, Taylor and Francis.

C.W.Johnson & A.J.Telford (1996), Using formal methods to analyse human error and system failure during accident investigations, *Software Engineering Journal* **11**(6), 355–356.

D.F.King (1996), Report on the incident to a Boeing 737-400, G-OBMM near Daventry on 25 February 1995, Technical report, Air Accidents Investigation Branch, Department of Transport.

Diaper, D., ed. (1989), *Task Analysis for Human-Computer Interaction*, Ellis Horwood.

Dillar, A. (95), *Z: An Introduction to Formal Methods*, second edn, Wiley.

Duke, R., King, P., Rose, G. & Smith, G. (1991), The Object-Z specification language: Version 1, Technical Report 91-1, Department of Computer Science, University of Queensland.

Hollnagel, E. (1993), The phenotype of erroneous actions, *International Journal of Man-Machine Studies* **39**, 1–32.

Johnson, C. (1997), Proving properties of accidents, *in* C. Holloway, ed., 4th NASA Langley Workshop on Formal Methods, NASA Langley Research Centre, Hampton, United States of America.

Jones, C. B. (1990), *Systematic Software Development Using VDM*, International Series in Computer Science, second edn, Prentice-Hall.

Lano, K. & Haughton, H., eds (1994), *Object-Oriented Specification Case Studies*, The object-oriented series, Prentice Hall.

Leveson, N. (1986), Software safety: Why, what and how, *ACM Computing Surveys* **18**(2), 25–69.

P.C.Wright, R.E.Fields & M.D.Harrison (1994), Deriving human-error tolerance requirements from tasks, *in* Proceedings,ICRE'94 - IEEE International Symposium on Requirements Engineering., Colorado.

R.E.Fields, P.C.Wright & M.D.Harrison (1995), A task centered approach to analysing human error tolerance requirements, *in* Proceedings,RE'95 - Second IEEE International Symposium on Requirements Engineering.

Suchman, L. (1987), *Plans and Situated Actions*, Cambridge University Press.

A The syntax of Object-Z.

Syntactically, a class of Object-Z is a named box with zero or more generic parameters. The constituents of the class may include inherited classes, type and constant definitions, at most one state schema, at most one initial state schema, zero or more operation schemata, and an optional history invariant.

ClassName[*generic parameters*]

inherited classes
type definitions
constant definitions
state schema
initial state schema
operation schemata

---

history invariant

These constituents can be seen in classes defined in the sections of the paper. For instance, the class *LM_staffing* of Section 4 contains, in the order given, a type definition, a state schema consisting of state and an initial state schema which is identified by the word *INIT*. The Δ-list of an operation, see class *Tasks* in Section 5, lists those variables of the state whose values may change when the operation is applied. Operations can also be given in terms of other operations using a non-boxed format. State variables and constants are known collectively as attributes. Instances of a class are called objects. A class can be used as a type. Thus, if C is a class, the declaration $c : C$ declares the object c to be of class C. If a is an attribute and Op an operation both defined in C then $c.a$ denotes the attribute a of object c and $c.Op$ corresponds to the application of operation Op on c. The use of the dot operator can be seen in the class *LM_staffing*. Other operators specific to Object-Z are introduced in the body of the paper as the case arises.