

SOFTWARE VERIFICATION RESEARCH CENTRE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072

Australia

TECHNICAL REPORT

No. 01-31

**A formal model of cognitive processes
for an Air Traffic Control task**

**Simon Connelly, Peter Lindsay,
Andrew Neal¹ and Mike Humphreys¹**

**¹Key Centre for Human Factors and Applied Cognitive Psychology
The University of Queensland**

August 2001

Phone: +61 7 3365 1003

Fax: +61 7 3365 1533

<http://svrc.it.uq.edu.au>

Note: Most SVRC technical reports are available via anonymous ftp, from [svrc.it.uq.edu.au](ftp://svrc.it.uq.edu.au) in the directory `/pub/techreports`. Abstracts and compressed postscript files are available via <http://svrc.it.uq.edu.au>

A formal model of cognitive processes for an Air Traffic Control task

Simon Connelly, Peter Lindsay,
Andrew Neal¹ and Mike Humphreys¹

¹Key Centre for Human Factors and Applied Cognitive Psychology
The University of Queensland

Abstract

This document describes a formal model of the cognitive processes involved in a simplified Air Traffic Control task. The model has been developed as part of the SafeHCI project, which is investigating detection and prevention of human error in safety-critical systems. The model will serve as the basis for development of new techniques for prediction of error sources and classification of error types. This document describes the cognitive model in detail.

1 Introduction

This document uses notation from computer science to model the cognitive processes underlying a simplified Air Traffic Control (ATC) task. The cognitive model has been developed as part of a collaboration between the SVRC and the Key Centre for Human Factors and Applied Cognitive Psychology, to investigate prediction of human (operator) errors associated with computer based systems. Experiments conducted on an ATC simulator are being used to validate the underlying cognitive psychology theories.

The purpose of the cognitive model is to identify the main cognitive processes involved in using the ATC simulator, and how “control” (roughly, the operator’s attention) moves from one process to another. The model is hierarchical, in that some processes are broken down into component subprocesses: in particular, both memory-based and rule-based (concurrent) processes are considered [4].

The model is reasonably generic but is primarily intended to represent the behaviour of experienced air-traffic controllers. Thus for example the model is broadly consistent with the Eurocontrol task analysis of en-route control [3], although of course our simulator and task are far simpler than in a real ATC system. Subjects in the simulator experiments were trained along the same lines as given here, so the model can reasonably be taken to represent the behaviour of novice (trained) users also.

This document is structured as follows: Section 2 describes the ATC simulator and the ATC task for which the cognitive model was developed, and introduces the main domain terminology. Section 3 describes the memory mechanisms used in the task, focusing on the nature of task-related data stored in short and long term memory. Section 4 describes the cognitive processes and the flow of control between processes.

2 Background and terminology

This section gives details of the simulator, its functionality and use in experiments, and introduces task-related terminology used in the cognitive model. Section 2.6 describes some of the ways this case study differs from “real” en-route ATC systems

2.1 Overview

In broad terms, the simulator presents a highly simplified en-route ATC system in which aircraft fly along straight-line segments – called *flight paths* – between *waypoints* within a fixed sector of airspace (see Figure 1). The primary task of the operator is to ensure that the aircraft moving through the sector remain separated by no less than the defined minimum separation distance (5000m): failure of this requirement is called *separation violation*. Within the simulator the only control that the operator can exert is to change the speed of individual aircraft: see Section 2.3 for details.

2.2 ATC simulator display

The ATC simulator has a display which depicts a simulated sector of airspace – consisting of airports, waypoints and flight paths – together with the location and details of aircraft currently flying within the sector: see Figure 1.¹ Flight paths are shown as dark lines, aircraft are represented by circles, airports are shown as squares and waypoints as triangles. Details of each aircraft (call sign, aircraft type, speed and flight route) are shown on labels attached to the aircraft symbols. *Flight routes* are represented as sequences of waypoint/airport codes. The display is updated at short intervals to give the impression that the aircraft are moving. Figure 1 also contains a simulation timer in the top right hand corner. This timer is used to aid in analysis of the operator’s behaviour (such as how long it took the operator to notice a conflict after it entered the sector); it is also provided to aid the operator in time judgements. Separation violations are indicated in two ways: the colour of the two aircraft changes to yellow (see Figure 2) and an audible alarm sounds. This alert continues until the minimum separation between the two aircraft is restored (either through operator action, or by passing each other).

2.3 User Interface functionality

The operator is responsible for changes to aircraft speeds within the sector, due to the limited amount of input required, the simulator is fairly simple. This is a benefit, as the operator should not be overwhelmed with options if they need to perform actions in a hurry. The simulator User Interface (UI) provides the operator with two main functions:

1. selecting a single aircraft, and
2. changing the speed of the selected aircraft.

An aircraft is selected by clicking the left button when the cursor is positioned over an aircraft. The selected aircraft is highlighted using a solid dot within the circle that represents the aircraft

¹The scale of the sector shown is 160x120km

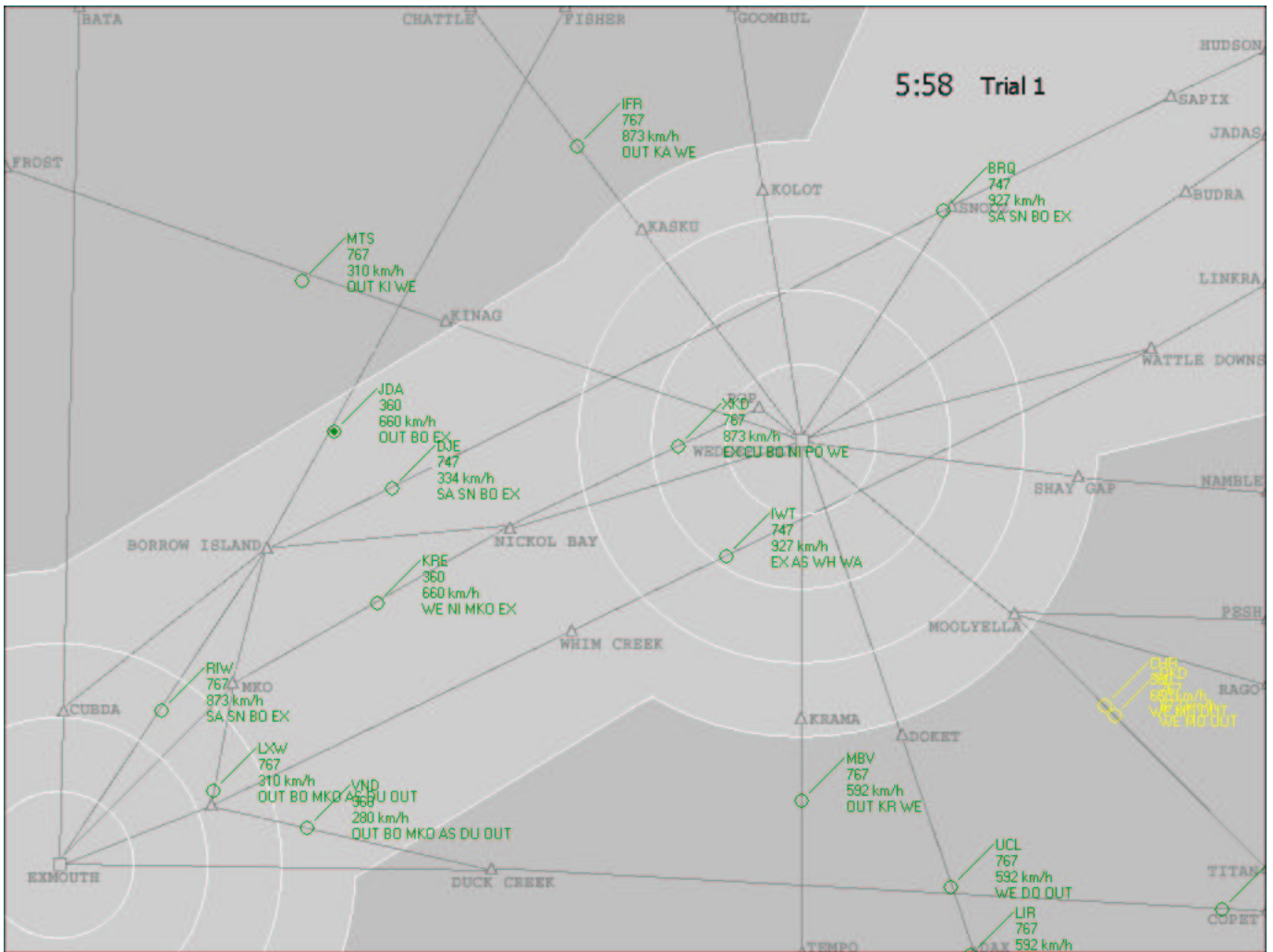


Figure 1: A (black and white) screenshot of the ATC simulator.

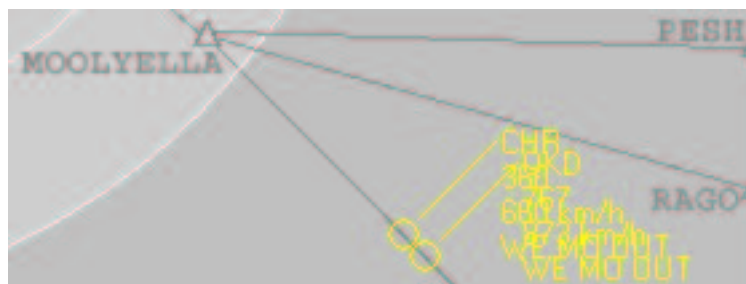


Figure 2: A partial screenshot showing two aircraft undergoing separation violation

(see Figure 3). Only one aircraft can be selected at a time: when a new aircraft is selected the previously selected aircraft loses its highlighting.

Changing the speed of the selected aircraft involves three steps:

1. Opening the *speed menu* by clicking the right button. The aircraft must be selected before the speed menu can be accessed. The menu appears at the position of the cursor.



Figure 3: A partial screenshot of the Speed Menu for the selected aircraft.

2. Navigating the speed menu by moving amongst the menu entries. The speed choices in the menu depend on the type of aircraft selected. A tick (✓) within the menu indicates the aircraft's current speed (see Figure 3).
3. Selecting a speed by left clicking on the desired menu entry.

The operator may abort this operation by clicking the left button when the cursor is positioned outside the speed menu.

2.4 Task terminology

A *conflict* is defined as two aircraft being on courses that, if their speeds are left unchanged, will lead them to violate the minimum separation standard at some time while they are within the sector.² Two distinct types of conflict occur within our case study:

- An *overtaking conflict* is one in which a faster aircraft approaches a slower aircraft flying ahead of it along the same flight route. In such a situation separation violation will result if the faster aircraft catches up to the slower aircraft while on the same flight path.
- A *convergence conflict* is one in which two aircraft on intersecting flight routes pass within 5000m of each other as they pass or approach the point of intersection.

For the purposes of this case study, a *problem* is defined to be a pair of aircraft to which the controller pays attention as possibly being in conflict. The term *episode* refers to a problem as it develops over time. An episode is said to be *active* if the problem is the focus of the operator's current attention. A single episode can be activated and deactivated many times as the controller switches attention between different problems. Only one episode can be active at a time.

When the operator has studied a given problem they will determine one or more *corrective actions* to be taken: namely, the aircraft whose speed they should change, the new speed, and when or where the change should take place (e.g. "after waypoint X"). Some of the details may not be determined precisely (e.g. simply "slow down aircraft A").

²This differs from real air traffic control, where the operator is also concerned with separation violations which would occur outside their sector.

A plan may have an associated *window of opportunity* during which it is safe to take corrective action. A window of opportunity can be discontinuous: for example, it may be safe to take action between times t_1 and t_2 , and between t_3 and t_4 , but not between t_2 and t_3 , because of the presence of other aircraft.

2.5 Scenarios used in the experiments

The air-traffic control simulation is run using scripted scenarios. A scenario script describes the starting positions, times, speeds, routes, etc of the aircraft involved in the scenario. The simulator animates (in real-time) the flight of the aircraft according to their scripted details, and according to any operator instructions from the UI.

Each script typically presents the air-traffic controller with a number of conflicts to be resolved at different times throughout the scenario and includes both the aircraft involved in those scripted conflicts and additional aircraft which are not involved in these conflicts.

2.6 Differences from real ATC

For the purposes of developing the methodology, the case study simplifies a number of aspects of Air Traffic Control. These simplifications include the following:

- Aircraft altitude is ignored. Consequently the sector is two dimensional and not three dimensional.
- The aircraft fly only on flight paths. The aircraft cannot diverge from flight paths.
- The only UI functions provided are selecting an aircraft and changing the aircraft speed. For example, aircraft flight routes cannot be modified and only a limited set of speeds is available.
- Aircraft respond to instructions instantly: there is no period of acceleration or deceleration involved when an aircraft changes speed.
- Collisions are not simulated. Two aircraft passing the same point, at the same time, will fly right through each other.
- Flight plans would be much more detailed and would contain, for example, estimates of waypoint passing times.

3 Underlying memory mechanisms

The cognitive model is memory-based, and involves a number of different hypothetical mechanisms, including episodic and short-term memories. These mechanisms are used to ensure that the model is able to simulate a number of well-known empirical phenomena within the memory literature (e.g., the use of cues, and the effect of the number of rehearsals, recency of occurrence and capacity of memory); they are not however intended as psychological or physiological hypotheses about the fundamental structure of human memory.

3.1 Episodic memory

The operator’s episodic memory is used for recording the episodes that the operator experiences. The details stored in memory for a particular problem, at a particular time, are represented by a cognitive *data relation*: see Section 3.3 below. The memory for a particular problem is cued by the information presented on the screen, such as the aircraft type, call sign, location, position in relation to other aircraft, etc. Depending on the way in which the information in memory is accessed and used, it is possible to retrieve a number of different types of knowledge from this memory system. These include so called “semantic knowledge”, which is accessed by abstracting information from a potentially large number of similar episodes which occurred previously, and so-called “episodic knowledge”, which involves the retrieval of information specific to individual episodes. These types of access and decision processes are assumed to be influenced by a range of factors, including the cues that are used for retrieval, and the frequency and recency with which specific data relations have been retrieved and stored.

3.2 Short-term memory

The operator’s short-term memory is used to temporarily record information regarding current problems, including the active problem (if any). We assume that the operator’s short term memory contains a collection of truncated versions of the data relations that describe the problem more fully in episodic memory, together with information about the priority of problems. There is no specific cue for these data relations so recall is determined by recency. Short-term memory has a very limited capacity.

3.3 Cognitive data relation

The data relation underlying the operator’s episodic memory, for a particular problem at a particular time, is modelled here as a tuple of the following form:

(aircraft attributes, context, classification, time of violation, priority, decision, decision stored?, windows of opportunity, behaving as expected?)

Not all of the data relation is necessarily known at one time, nor are all the known parts necessarily retrieved from memory at any given time. We do not model all the details that may be stored in memory, but simply those that are key to the choices made by the operator. The model uses abstract values and abstract predicates (classes of values) for simplicity and in order to highlight the main aspects of the cognitive processes. There is no implication that operators use these actual values or think in terms of these predicates; in practice operators’ thoughts are unlikely to be structured as suggested here – the model captures behaviour in a general rather than specific detail. In what follows, the symbol ? means the value is not known or not certain.

The various elements of the data relation are as follows:

aircraft attributes: The attributes of each of the aircraft involved in the problem including, for example, call sign, aircraft type, speed, flight route, etc. ³

³Typically exactly two aircraft will be involved in any one problem, but the model allows for more than two if necessary.

context: The context of the problem. This may be one or more of: the time of day, the position of the problem in the sector, etc.

classification: The classification of the problem. When known, the value is one of the following:

conflict: The problem is expected to end in a violation of separation if corrective action is not taken

non-conflict: The aircraft are not expected to violate separation

time of violation: The projected time of separation violation. This is represented as the time delay from the time at which they were classified as being in conflict until the violation will occur. Time is represented in the model as time since the beginning of the simulation. This time shown to the operators in a clock in the top right hand corner of the screen.

priority: This is a measure of the problem's relative priority. For the purposes of our model, we abstract away from values and simply use the following abstract predicate to indicate the operator's belief.

Is Highest: This conflict should have the highest priority. (The negation of this predicate is used when the problem is not the highest priority problem.)

decision: The corrective action(s) to resolve the problem. Individual corrective actions are represented below in the format (callsign, newSpeed, position, time) where position and time refer to the position or time at which the action will be taken. – indicates no action required (meaning 'do nothing' on the given aircraft).

decision stored?: Whether the operator has determined the corrective actions and deferred taking them until later. The following predicate is used:

decision Stored: The corrective actions have been decided and stored in memory

windows of opportunity: The time periods within which it is safe for the operator to take the planned corrective action(s). The following abstract predicates are used to indicate the operator's belief:

Inside Window: Currently safe to take corrective action

Outside Window: Currently unsafe to take corrective action

Must Act Now: The violation is about to occur and corrective action should be taken immediately

(Note that **Must Act Now** is a special case of **Inside Window**.)

behaving as expected?: Whether the aircraft are behaving as the operator expects. We do not model the cognitive mechanisms involved, but simply note here that expectations are based on things like aircraft position, problem classification, and whether the operator thinks that corrective actions have already been taken. The following predicate is used:

As Expected: The problem is behaving as the operator expects. (The negation of this predicate is used when the problem is not behaving as expected.)

At any one time the data relation may contain incomplete information: e.g., the operator may have classified the problem as a conflict but may not have projected forward to estimate the time of violation. Real air traffic controllers would have much more sector-specific knowledge from experience, such as hot spots, etc.

3.4 Example data relation

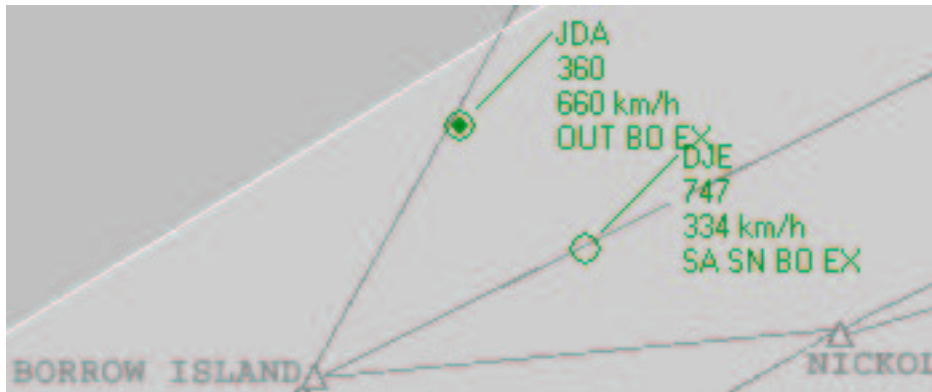


Figure 4: A partial screenshot showing two aircraft in conflict.

As an example consider the problem presented in the partial screenshot in Figure 4. The following describes a possible data relation for the problem just before the operator takes corrective action:

```
(({JDA, 360, 660km/h}, {DJE, 747, 334km/h}),  
  "Approaching Borrow Island en-route to Exmouth airport", conflict, 5:58+10, Is Highest,  
  ((JDA, 330km/h, ?, now), (DJE, 860km/h, ?, now)),  
  decisionStored, Inside Window, AsExpected)
```

The two aircraft involved in the problem are DJE (a Boeing 747 travelling at 334km/h) and JDA (a Airbus 360 travelling at 660km/h) as they approach the ‘Borrow Island’ waypoint en-route to ‘Exmouth airport’. The problem has been classified as a conflict because the controller thinks that they will pass within 5000m of each other as they approach Borrow Island. (In fact, this is probably not true.) The controller estimates that the conflict will occur at approximately 6:08 (10 seconds after 5:58 on the simulator clock). The controller intends to give this problem the highest priority, thus ignoring the active separation violation in the lower right-hand corner of Figure 1. They have decided to slow JDA down to 330km/h and speed DJE up to 860km/h immediately (“now”). They have taken mental note of the actions planned. They have noted that it is safe to act now, but there is still time left before it is absolutely necessary. The aircraft are behaving as expected from when the operator identified the problem and projected forward.

4 The cognitive processes and flow of control

The control-flow model is given in Figure 5. It identifies the main cognitive processes used by operators of the ATC simulator, and describes the “flow of control” through these processes: that is, the possible sequencing of processes, under different conditions.

The model uses the UML statechart notation [1, 5] which is based on Harel’s state chart notation [2]. A brief description of the notation used in the model is given in Table 1. In



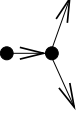

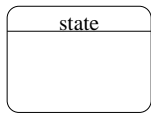
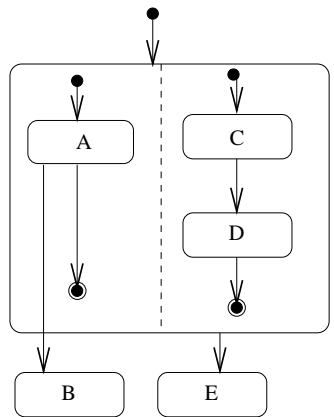
Notation	Meaning
	<p>A solid circle represents the start of a process or branch (choice) point. A solid circle inside a larger empty circle denotes the termination of a state.</p>
	<p>Used to denote a transition from one state to another</p>
	<p>This notation is used when there are two possible paths from the one state, the path to be taken is usually governed by preconditions. (Notation added to UML for convenience)</p>
<p>"Description of transition" event [Guard] /action</p>	<p>Descriptions can be added to transitions to clarify their purpose. An event is something that triggers the transition. A guard is a condition governing whether a transition can be taken. An action is an uninteruptable, brief process that takes place when the transition is taken. (In the preconditions the term != is equivalent to "is not equal to").</p>
	<p>A basic state that the system may be in. In this model, basic states are cognitive processes that are not analysed further.</p>
	<p>Some states may contain substates, which are shown by drawing them inside the 'superstate'</p>
	<p>An example of two processes being run in parallel. The system can be in the following states: (A,C) (A,D), B (in which case the 2nd process has been aborted) or E once both A and D are finished.</p>

Table 1: State-chart notation

the statechart model, each state represents an abstract cognitive process (such as monitoring a conflict), which in some cases is further subdivided into component cognitive tasks. The terms state and (cognitive) process will be used interchangeably within this section. Brief, uninteruptable processes will be written as "actions" as in UML notation (see Table 1); these should not be confused with the domain-specific term "corrective actions" also used below.

For the most part, the predicates from the data relation of Section 3.3 above are used to constrain flow of control. In some cases however "local" predicates are used, for data which is computed in the given state but not explicitly stored in memory. The User Interface also constrains some choices, as explained in Section 2.3 above.

In the remainder of this section each of the cognitive tasks in the control-flow model is briefly described, together with a brief indication of how the cognitive data relation is built up.

4.1 Scanning

The aim of the Scanning process is to identify problems to which the operator will need to attend. The operator systematically scans the display looking for specific patterns (“primitive geometries”) that characterise potential conflicts. Such patterns are the basic combinations of positions, speeds and flight paths which lead to conflicts. For example operators might attend to a pair of aircraft if:

- The aircraft are travelling at similar speeds and are approximately the same distance away from a point where their flight paths converge or intersect; or
- The aircraft are travelling at different speeds, and are at different distances from a point of convergence, and the slower aircraft is closer to the point than the faster aircraft; or
- The aircraft are flying on the same route, and the faster aircraft is behind the slower aircraft.

It is likely that there will be large differences between operators in their approach to scanning, which are at least partly attributable to their experience with the sector. Given the differences between operators, it is likely that a random scanning function will provide a relatively good fit to group data. Specifically the model randomly selects pairs of aircraft. If the pair does not match a known geometry that characterises potential conflicts, then the model will return to scanning and select another pair. In this manner the model cycles through pairs of aircraft, until it finds a pair that needs monitoring.

4.2 Monitoring

The `MonitorProblem` state is used to gather information about the problem being attended to and to classify whether it is a conflict or not. Monitoring a pair of aircraft has the following substates:

- Perceive problem details
- Lookup memory
- Project forward

The operator begins by encoding information from the display and checking episodic memory to see if this problem has been monitored previously (the `PerceiveProblemDetails` state): if so, certain parts of the data relation are retrieved from memory, as described below: Information encoded into the data relation from the display may include aircraft details and the context. This information is then used as a cue to retrieve any prior records of the current problem and records of previously seen problems that closely match. The records retrieved from memory (if any) may contain information about the problem type, relevant aircraft attributes (e.g. callsign, abridged flight path), context, any windows of opportunity and any corrective actions that the operator has decided upon (either taken or planned). The information encoded from the display and the information retrieved from memory is then used to create a new record.

The weighting of information encoded from the display and information retrieved from memory will vary across situations. Under some circumstances, prior records of the current problem

can be retrieved very quickly, with a high degree of confidence. The major factors influencing the speed and confidence of retrieval are recency and frequency⁴. If this is the case, then the operator does not need to encode very much information from the display. For example, retrieval might occur as soon as the operator encodes the position of the aircraft involved, if the operator has a high degree of confidence in the retrieved information. In this case, the new record would contain information that is predominantly derived from the prior records. If the operator does not have a high degree of confidence in the retrieved information, or if the information that has been encoded from the display is inconsistent with what they have retrieved from memory (i.e., the problem is not proceeding as expected), or if the retrieval does not occur, then the operator will continue to encode information from the display. Each additional piece of information from the display can act as a retrieval cue as above.⁵

If the operator has previously classified the pair, and has retrieved the classification from memory, then s\he will store the updated record in memory. If the classification is unknown then the operator will classify the pair (via the `Classifyproblem` state). Two processes are used to classify the pair `LookupMemory` and `ProjectForward`. These processes are assumed to run in parallel. Classification is terminated when either process returns a result. `LookupMemory` involves matching the current episode to all previous episodes that have been retrieved from memory and draw a classification from the results. At the same time, the operator will also engage in projection.

Projection is used to estimate the time and/or position of conflict (i.e. when or where the separation violation is likely to occur), and the windows of opportunity. We assume that projection is slow and difficult, as it imposes a high level of workload on controllers. When controllers are novices at the task, they have relatively few prior episodes stored in memory. Novices, therefore, should tend to rely on `ProjectForward` to classify problems, as it will return a result before `LookupMemory`. With practice however, controllers will have access to a larger number of episodes in memory, and the speed of retrieval will increase. As a result there is a shift away from the use of `ProjectForward` towards the use of `LookupMemory` with practice. Experts should only use `ProjectForward` when they are confronted with problems that do not match any prior episodes. Once the operator has matched the pair to existing patterns or projected forward, they will collate the information to make their final decision (`CollateExperience`).

Once the operator has identified whether the problem is a conflict or not, they are assumed to store the current record in episodic memory (the `StoreEpisode` action).

After the episode has been stored in episodic memory, the operator returns to `Scanning` or proceeds to `Prioritisation`. The operator may return to `Scanning` if the aircraft are not in conflict or they do not need to be acted upon immediately. The operator proceeds to `Prioritisation` if the aircraft are in conflict, or if the operator is still uncertain as to whether or not they are in conflict.

4.3 Prioritisation

The operator uses priorities to manage the order in which problems are dealt with (the `Prioritisation` state). In this process the active problem is first assigned a priority (the `AssignPriority`

⁴Records that have been encoded recently or frequently are likely to be retrieved faster and with greater confidence

⁵We assume that there is a monotonically decreasing probability of retrieval occurring as additional information is encoded, due to the limited capacity of Short-Term Memory.

state) which is associated with the data relation in short-term memory (the `StorePriority` action). Short-term memory is checked for priorities associated with any other current problems (the `Compare Priorities` state). If a higher priority problem is found, the operator will go directly to `MonitorProblem` for the highest priority problem (the “memory-cued problem monitoring” transition). If the current problem has the highest priority, the operator proceeds to determining corrective actions. If the problem time is anything other than `MustActNow`, the operator may defer deciding on the corrective action until a later time (the “defer” transition); if so, the operator returns to `Scanning`.

4.4 Decision

Next, the operator decides what corrective actions to take, if any (the `Decision` state). If the operator has previously made a decision (as indicated by the `decisionStored` predicate in the data relation) they are not required to make it again. They will merely retrieve the stored decision (`FetchDecision`), revalidate it against the current situation, and either defer or take action.

Two processes are used concurrently to generate corrective actions: one memory-based and the other rule-based [4]. The `Memory-basedDecision` process works by retrieving all previous episodes that closely match the current episode (`FetchEpisodes`) and selecting the most successful solution that the operator has previously used. A confidence level is associated with the selected solution: if it is high, the operator will use the solution to resolve the current problem. The Eurocontrol task analysis [3] showed that experienced controllers were likely to have developed a “conflict resolution library”, from which they draw solutions for the problem at hand.

If the operator cannot retrieve any solutions, or the confidence in the retrieved solution is low, the operator uses the rule-based decision process to deduce the actions to take (the `Rule-basedDecision` state). The `Decision Found` local predicate is used to show whether the operator has found a solution that they are confident in.

Once a decision has been made, the operator must then check that the actions would not lead to a conflict with other aircraft (the `ValidateDecision` state). Performing this validation is very similar to the `ClassifyProblem` process; the operator will project forward the action that they wish to take, and then perform a brief classification of each pair that the action might affect; if the actions would not lead to conflict then the operator is confident that the selected actions will solve the given problem and not produce any new conflicts. This process will probably not be as thorough as the classification process performed on single pairs. Through this validation it is also possible that the operator will formulate “windows of opportunity” for the chosen actions. The data relation is updated with the decided actions and stored in memory (the `StoreDecision` action). If validation reveals that the corrective actions may be undesirable, the operator formulates a new plan.

4.5 Perform Actions

Performing corrective actions is a multi-step process represented by the `PerformActions` state. First the actions are retrieved from the operator’s memory and the plan is reviewed (the `ReviewActions` state). The plan is reviewed after each attempted corrective action; this may result in return to the `Decision` state (via the “revise decision” transition).

It may be necessary to change which aircraft is currently selected (`SelectAircraft`) in order to

carry out the planned action. Once an aircraft is selected the speed menu is operated (the `ChangeAircraftSpeed` state). The speed menu can only be opened if an aircraft is selected (a UI constraint). To change an aircraft's speed the menu is opened and the cursor is moved within the menu, and finally a speed is selected (`OpenSpeedMenu`, `NavigateSpeedMenu` and `SelectSpeed` respectively). If the operator makes an error during the `ChangeAircraftSpeed` state (e.g. by selecting the wrong speed) they can abort the process and return to the `ReviewAction` state.

When all the corrective actions have been taken, the data relation in memory is updated with a record of the actions taken, together with the expected outcomes – in terms of how the conflict will proceed from here on – (`StoreActions`) and the operator returns to `Scanning`.

4.6 Other transitions

The alarm transition can occur anywhere within the control flow; it is triggered by the alarm event. This is shown in the model as returning directly to the `MonitorProblem` state. This means that the user, if interrupted by an alarm, may jump straight to monitoring the indicated problem. This does not mean that the operator *must* stop what they are doing immediately; they may continue through the normal flow until they are ready to deal with the alarm.

There is another transition that can occur anywhere in the cognitive flow; when the operator “forgets what they are doing”. This behaviour is possible during most of the cognitive tasks in the model, and relates to those error modes in which the input data relation is lost during the task (“forgotten”), and thus not output to the next task in the cognitive process. When this occurs, the operator returns to `Scanning` via the “Interrupted Awareness” transition.

Memory-cued transitions back to `Monitor Problem` are also possible from processes in addition to `Prioritisation` (e.g. from `ReviewActions` when considering the effect of actions on other pairs) – but these are not modelled here for simplicity.

5 Acknowledgements

The authors gratefully acknowledge the collaboration of the Key Centre's Shayne Loft for designing and supervising the experiments that are being used to validate the cognitive model and to calibrate the error model. The Key Centre developed the simulator used in the experiments. Finally, the ARC's support for the research reported here – by way of a Small Grant in 2000 and a Large Grant in 2001-3 – is gratefully acknowledged.

References

- [1] M. Fowler. *Applying the Standard Object Modeling Language*. Addison Wesley, 1997.
- [2] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
- [3] K. Kallus, D. van Damme, and A. Dittmann. Integrated task and job analysis of air traffic controllers – phase 2: task analysis of en-route controllers. Technical Report

HUM.ET1.ST01.1000-REP-04, European Organisation for the Safety of Air Navigation (Eurocontrol), October 1999.

- [4] G. Logan. Toward an instance theory of automization. *Psychological Review*, 95:492–527, 1988.
- [5] B. Oestereich. *Developing Software with UML*. Addison Wesley, 1999.