Integrating the Operator into Formal Models in the Air-Traffic Control Domain

David Leadbetter[1], Peter Lindsay[1],
Andrew Neal[2], and Mike Humphreys[2]

[1] Software Verification Research Centre
{davidl, pal}@svrc.uq.edu.au
[2] Key Centre for Human Factors and Applied Cognitive Psychology
{a.neal,mh}@humanfactors.uq.edu.au
The University of Queensland
Brisbane, Qld, 4072, Australia

Abstract. Growing use of computers in safety-critical systems increases the need for Human Computer Interfaces (HCIs) to be both smarter - to detect human errors - and better designed - to reduce likelihood of errors. We are developing methods for determining the likelihood of operator errors which combine current theory on the psychological causes of human errors with formal methods for modelling human-computer interaction. This paper outlines an approach to developing formal methods for evaluating safety of interactive systems, and illustrates the approach on a simplified problem from Air Traffic Control. We outline formal models for three components of an ATC simulator: the underlying computer system, the HCI and the operator.

1 Introduction

1.1 Motivation

Computers are increasingly being used in safety-critical systems. Examples include interactive control systems for transport (road, rail and air), medical equipment, power stations and process plants. As society increasingly relies on computerised systems for safety, this is an area that will continue to grow in importance. One of the key factors influencing the safety of computerised systems is the design of the Human-Computer Interface (HCI). A human-computer interface is safety-critical when the potential arises for injury or loss of life from defects in the design of the HCI. Safety has frequently been compromised and lives have been lost because of operator errors caused by HCI design deficiencies (e.g., see [11]). Current system safety standards - such as in defence [3], railways [5], and process industries [10] - mandate or highly recommend formal (mathematical) modelling for safety aspects of hardware and software functionality because such models promote methodical, reproducible and auditable software development, but the techniques used for evaluating and enhancing the safety of HCI designs are informal at best. This is largely attributable to:

– difficulty formally modelling interaction between operators and the system;
– lack of understanding of psychological processes leading to operator error;

– inability to formally specify the antecedent conditions that trigger those processes, and to estimate the resulting likelihood of errors; and
– lack of precise methods for mapping operator errors to design solutions.

We are conducting a project that will develop formal methods for evaluating the safety of interactive systems. Our ultimate goal is to develop probabilistic models of HCI use error suitable for use in risk estimation. In particular, the aim is to be able to compare the effectiveness of different HCI designs with respect to system risk and likelihood of use error.

Existing models of human error do not provide a precise specification of the conditions leading to error, or the mechanisms responsible for error. The complexity of modelling human behaviour in complex real world systems compounds this problem. We are developing a formal model of cognition in the simplified ATC simulation based on psychological theories of human error. The formal operator model is integrated with models of the ATC core system and HCI to enable tracing of operator errors to system hazards and to enable identification of HCI features that contribute to or diminish operator error. The ATC task being modelled is highly simplified, and is being proposed as a means for validating the approach. Experiments involving human subjects will be used to validate the cognitive models and, eventually, to calibrate the probabilistic models.

1.2 Approach

In this paper we model the ATC system in three parts: the core system functions, the human-computer interface, and the operator. Space limitations permit only parts of these models to be presented here.

The ATC core system: We model a simple sector of airspace such as that depicted in Figure 1. Our model only considers movement of aircraft in a two dimensional plane; vertical separation of aircraft is not considered.

The ATC Human-Computer Interface: The HCI visualises the state of the underlying simulation and provides a basic range of operations for acting on it - for simplicity operations are only provided to allow the aircraft speeds to be changed; more advanced operations, such as changing the flight plan are not provided. Figure 1 illustrates what the visualisation provided by the HCI may look like: aircraft are represented as dots over a graphical representation of the layout of the waypoints and flight routes in the sector; basic aircraft details (callsign, speed, etc) are displayed in boxes attached to the aircraft dots.

The ATC Operator: The primary task of the operator is to ensure that the aircraft moving through the sector remain separated by a defined “minimum separation” distance. The operator must use the operations provided by the HCI to avoid future violations of minimum separation that appear likely to occur. Real ATC controllers are also concerned with other objectives such as ensuring that aircraft move efficiently through the sector with minimal delay and disruption. Such concerns are not considered here.

The coupling between the three parts of the ATC system is illustrated in Figure 2. The models of the ATC core, HCI and operator presented in this paper represent the

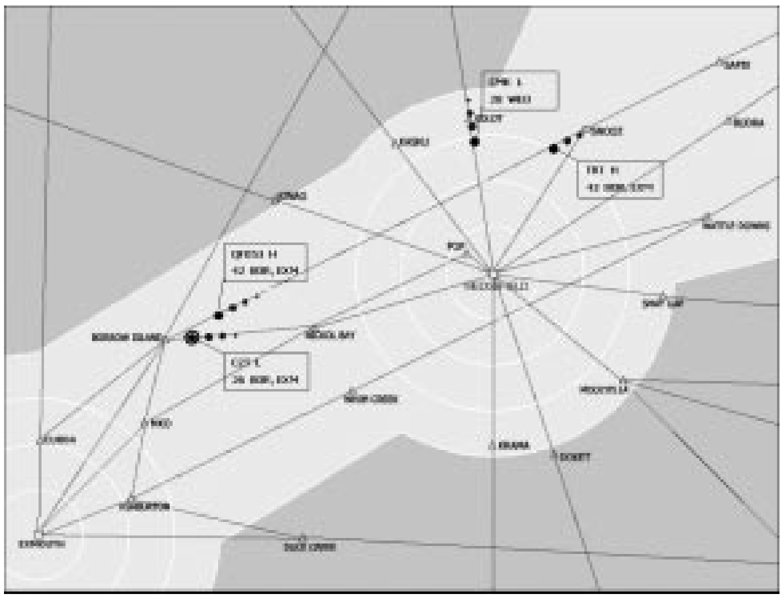

Fig. 1. A sector of controlled airspace

results of the first cycle in an iterative modelling process. As such, the models include many simplifications, primarily through the use of abstraction so that details of the system parts can be ignored at this stage. Such details will be added during later iterations of the modelling process.

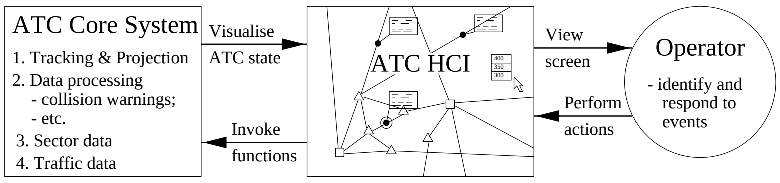

Fig. 2. Relationship between system, HCI, and operator models

1.3 Background and overview

Although considerable progress has been made in understanding the task conditions that are likely to lead to human error [16], these traditional approaches do not allow us to develop precise formulations of error. Our approach is based on a number of recent advances in cognitive psychology, including the development of connectionist models of memory [12], and mathematical specifications describing the input-output functions required to model different tasks [8]. These advances allow us to develop precise specifications of the conditions and processes that lead to a wide range of human memory errors.

A substantial body of related work deals with the integrated modelling of the operator with the computer aspects of a system. A range of integrated modelling approaches are summarised by Blandford and Duke [1]. As in Duke et al. [4], we advocate the integration of a cognitive model of the operator. The cognitive models presented in this paper differ from the work of Duke et al. because they describe the controller's abstract cognitive functions (rather than constructions of explicit cognitive subsystems) based on new psychological theories. Butterworth et al. [2] describe Programmable User Models (PUMs) using a state-based notation. PUMs focus on the users model of the system, with formalisation of the information needed and actually acquired by a user and the UI operations available to the user. PUMs are similar to the HCI model constructed in this paper but do not incorporate elements of the physical user actions and system presentation. Paternò et al. [15] have produced similar models of ATC operator tasks. However they do not explicitly model the ATC HCI, or the cognitive state and processes of the operator. Palanque et al. [13] have used Petri-nets to model tasks, systems and the user's view of the system. Their approach uses an object-oriented extension of the Petri-net notation and enables analysis of incompatibilities between the models. However their is no modelling of either the HCI presentation or the cognitive processes of the user. The method has been applied to analysis of an air-traffic control system [14].

In Section 2, we give a formal Z [17] model of the core state and operations of the ATC system simulation. Hussey and Carrington [9] have previously used an object-oriented extension of Z for modelling interactive systems. Section 3 provides a model of the HCI for the ATC system simulation which combines the UAN [7] and Z notations and Section 4 gives a model of the operator using Statecharts [6] and DFDs. The notations used enable a useful level of abstraction to be achieved for each of the models and are easily understood. In Section 5 we consider the significance of the work, and our future plans for the project.

2 The ATC System Core Model

The ATC system core model includes various subsystems that make up the functional core of the ATC system. This model does not define the HCI system through which the operator uses this functionality. The ATC system core models: Sector data; Aircraft specifications; Static aircraft details; Aircraft telemetry updates; and Warning systems for identifying separation violations.

2.1 Sector Data

A sector contains a variety of objects: waypoints, airports, routes, etc. All of these has an associated position in the sector (*Position*). The distance (*Distance*) between two positions is calculated using the function *distanceBetween*. For simplicity we consider airports to be waypoints (*Waypoints*).

$$\mid\ \mathit{distanceBetween} : \mathit{Position} \times \mathit{Position} \rightarrow \mathit{Distance}$$

A sector is modelled by defining the objects it contains: the *waypoints* and their positions in the sector; the *airports* in the sector; and the *routes* through the sector (where each pair of waypoints in the *routes* relation defines the single, straight route segment between the two waypoints).

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Sector}} \\
\hline
\mathit{waypoints} : \mathit{Waypoint} \nrightarrow \mathit{Position} \\
\mathit{airports} : \mathbb{P}\,\mathit{Waypoint} \\
\mathit{routes} : \mathit{Waypoint} \leftrightarrow \mathit{Waypoint} \\
\hline
\mathit{airports} \subseteq \operatorname{dom} \mathit{waypoints} \\
\operatorname{dom} \mathit{routes} \cup \operatorname{ran} \mathit{routes} = \operatorname{dom} \mathit{waypoints} \\
\hline
\end{array}$$

2.2 Aircraft and Air Traffic Data

We model aircraft callsigns (*Callsign*), types (*AircraftType*), and speed (*Speed*).

The air traffic in a sector consists of a set of *aircraft* (identified by callsigns). Each aircraft is of a specific *type*, has a current *telemetry* (position and speed) and a *flightPlan* (a sequence of waypoints). Lastly, the ATC operator's last instruction to each aircraft is recorded in *instructions*.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Traffic}} \\
\hline
\mathit{aircraft} : \mathbb{P}\,\mathit{Callsign} \\
\mathit{type} : \mathit{Callsign} \nrightarrow \mathit{AircraftType} \\
\mathit{telemetry} : \mathit{Callsign} \nrightarrow \mathit{Position} \times \mathit{Speed} \\
\mathit{flightPlan} : \mathit{Callsign} \nrightarrow \operatorname{seq}_1 \mathit{Waypoint} \\
\mathit{instructions} : \mathit{Callsign} \nrightarrow \mathit{Speed} \\
\hline
\mathit{aircraft} = \operatorname{dom} \mathit{type} = \operatorname{dom} \mathit{telemetry} = \operatorname{dom} \mathit{flightPlan} \\
\operatorname{dom} \mathit{instructions} \subseteq \mathit{aircraft} \\
\hline
\end{array}$$

The air traffic telemetry information is updated using the *updateTelemetry* operation. New telemetry for all of the aircraft in the sector is received and updated with this operation.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{updateTelemetry}} \\
\hline
\Delta \mathit{Traffic} \\
\mathit{newTelemetry}? : \mathit{Callsign} \nrightarrow \mathit{Position} \times \mathit{Speed} \\
\hline
\mathit{telemetry}' = \mathit{newTelemetry}? \\
\mathit{type}' = \mathit{type} \land \mathit{flightPlan}' = \mathit{flightPlan} \land \mathit{instructions}' = \mathit{instructions} \\
\hline
\end{array}$$

The operator's instructions to individual aircraft are recorded using the *changeSpeed* operation. This is the only instruction included in the simulation with only the latest instruction to each aircraft recorded.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{changeSpeed}} \\
\hline
\Delta \mathit{Traffic} \\
\mathit{aircraft}? : \mathit{Callsign} \\
\mathit{speed}? : \mathit{Speed} \\
\hline
\mathit{instructions}' = \mathit{instructions} \oplus \{\mathit{aircraft}? \mapsto \mathit{speed}?\} \\
\mathit{type}' = \mathit{type} \land \mathit{telemetry}' = \mathit{telemetry} \land \mathit{flightPlan}' = \mathit{flightPlan} \\
\hline
\end{array}$$

2.3 Separation Violation

The minimum separation between aircraft is defined by a regulatory authority and is modelled as a constant.

$$
\begin{array}{|l}
\mathit{minimumSeparation} : \mathit{Distance}
\end{array}
$$

The separation between aircraft is calculated using the telemetry data. Semantically meaningful functions are defined to extract information from the telemetry data.

$$
\begin{array}{|l}
\mathit{position} : \mathit{Position} \times \mathit{Speed} \rightarrow \mathit{Position} \\
\hline
\forall\, p : \mathit{Position};\ s : \mathit{Speed} \bullet \mathit{position}(p, s) = p
\end{array}
$$

The *detectSeparationViolation* function identifies (and outputs) the callsign of all aircraft that are currently within the minimum separation distance of another aircraft, and hence in violation of the separation regulations.

$$
\begin{array}{|l}
\hline
\textit{detectSeparationViolation} \\
\hline
\Xi\, \mathit{Traffic} \\
\mathit{violations}! : \mathbb{P}\, \mathit{Callsign} \\
\hline
\mathit{violations}! = \{ac1, ac2 : \mathit{aircraft} \mid ac1 \neq ac2 \land \\
\qquad \mathit{distanceBetween}(\mathit{position}(\mathit{telemetry}(ac1)), \\
\qquad\qquad \mathit{position}(\mathit{telemetry}(ac2))) \leq \mathit{minimumSeparation} \bullet ac1\} \\
\hline
\end{array}
$$

2.4 Aircraft specifications

Aircraft of the same aircraft type have some common characteristics. These ‘specifications’ detail the minimum and maximum speeds of the aircraft.

$$
\begin{array}{|l}
\hline
\textit{AircraftSpecification} \\
\hline
\mathit{minimumSpeed}, \mathit{maximumSpeed} : \mathit{AircraftType} \nrightarrow \mathit{Speed} \\
\hline
\operatorname{dom} \mathit{minimumSpeed} = \operatorname{dom} \mathit{maximumSpeed} \\
\forall\, \mathit{acType} : \operatorname{dom} \mathit{minimumSpeed} \bullet \\
\qquad \mathit{minimumSpeed}(\mathit{acType}) < \mathit{maximumSpeed}(\mathit{acType}) \\
\hline
\end{array}
$$

2.5 The ATC core system

The ATC core system consists of the sector data, the air traffic data, and the aircraft specification data.

$$
\begin{array}{|l}
\hline
\textit{ATCCore} \\
\hline
\mathit{Sector} \land \mathit{Traffic} \land \mathit{AircraftSpecification} \\
\hline
\operatorname{ran} \mathit{type} \subseteq \operatorname{dom} \mathit{minimumSpeed} \\
\hline
\end{array}
$$

3 The ATC HCI model

The ATC HCI is modelled using an integrated approach, blending a formal Z model of the interface state with the User-Action notation for describing the user actions. In these models low level details of the HCI are ignored so as not to obscure the high level functionality of the HCI and the actions of the user.

3.1 The Underlying Interface State

The underlying interface state defines the functional model of the ATC interface. The primary part of this is the mapping from the ATC system to the visual representation maintained on the interface devices.

The central aspect of this mapping is of the air traffic in the sector to the views that represent that traffic on the interface. The view of each aircraft is abstractly defined using the given type *AircraftView*. The function *makeView* creates the view from the telemetry and flight path information of an aircraft.

$$\begin{array}{|l} \mathit{makeView} : \mathit{Callsign} \times (\mathit{Position} \times \mathit{Speed} \times \mathit{Time}) \times \\ \qquad\qquad \mathrm{seq}_1(\mathit{Waypoint} \times \mathit{Time}) \rightarrow \mathit{AircraftView} \end{array}$$

The aircraft specifications also appear in the interface in the form of the speed selection menu used when instructing an aircraft to change speed. These menus are abstractly defined using the given type *SpeedMenu*. The list of speeds that may be selected from the menu is derived from the minimum and maximum speeds of the aircraft, and is created using the function *speedList*. The function *makeMenu* creates the menu object from this list of speeds.

$$\begin{array}{|l} \mathit{speedList} : \mathit{Speed} \times \mathit{Speed} \rightarrowtail\!\!\!\!\!\!+\; \mathrm{seq}\,\mathit{Speed} \\ \mathit{makeMenu} : \mathrm{seq}\,\mathit{Speed} \rightarrow \mathit{SpeedMenu} \end{array}$$

Whenever there is a separation violation in the ATC system, an audible alarm sounds in the interface. The status of this alarm is modelled using a free type.

$$\mathit{AlarmStatus} ::= \mathit{on} \mid \mathit{off}$$

The ATC interface consists of the aircraft views (modelled in the *views* function that maps each view to the callsign of the aircraft it represents, providing the coupling from the HCI to the core system), the *selected* aircraft view (modelled as a set allowing either no aircraft or a single aircraft to be selected), the *speedMenu* used by the operator to instruct the selected aircraft to change speed, the set of aircraft that are currently in a separation violation (*warnings*), and the status of the *alarm*.

Two basic actions on the HCI correspond to clicking the left mouse button (on an aircraft view) and right mouse button. These are selecting an aircraft (*selectAircraft*) and opening the speed menu (*openSpeedMenu*) respectively.

$$
\begin{array}{|l}
\hline
\textit{ATCInterface} \\
\hline
views : AircraftView \rightarrowtail\!\!\!\!\!\rightarrow Callsign \\
selected : \mathbb{P}\, AircraftView \\
speedMenu : SpeedMenu \\
warnings : \mathbb{P}\, AircraftView \\
alarm : AlarmStatus \\
\hline
selected \subseteq \operatorname{dom} views \\
\# selected \leq 1 \\
warnings \subseteq \operatorname{dom} views \\
alarm = \mathit{if}\ warnings = \varnothing \\
\qquad \mathit{then\ off\ else\ on} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{selectAircraft} \\
\hline
\Delta ATCInterface \\
aircraft? : AircraftView \\
\hline
aircraft? \in \operatorname{dom} views \\
selected' = \{aircraft?\} \\
views' = views \\
speedMenu' = speedMenu \\
warnings' = warnings \\
alarm' = alarm \\
\hline
\end{array}
$$

The *speedMenu* in the *ATCInterface* is only displayed when requested by the operator. Consequently a lazy approach to maintaining the consistency between *speedMenu* and *selected* is used: the menu is only updated when the speed menu is opened. Displaying of the menu is described using UAN in Section 3.4.

$$
\begin{array}{|l}
\hline
\textit{openSpeedMenu} \\
\hline
\Delta ATCInterface \\
\Xi Traffic \\
\Xi AircraftSpecification \\
\hline
selected \neq \varnothing \\
\exists_1\, acView : selected \bullet speedMenu' = \\
\quad makeMenu(speedList(minimumSpeed(type(views(acView))), \\
\qquad maximumSpeed(type(views(acView))))) \\
selected' = selected \land views' = views \\
warnings' = warnings \land alarm' = alarm \\
\hline
\end{array}
$$

Once the speed menu is opened, the operator may select one of the speeds in the menu. The following operation *selectSpeed*1 occurs when this happens. The index of the selected menu item is used to look up the corresponding speed.

$$
\begin{array}{|l}
\hline
\textit{selectSpeed1} \\
\hline
\Xi ATCInterface \\
\Xi Traffic \\
\Xi AircraftSpecification \\
menuIndex? : \mathbb{Z} \\
aircraft! : Callsign \\
speed! : Speed \\
\hline
selected \neq \varnothing \\
\exists_1\, acView : selected \bullet aircraft! = views(acView) \\
\mathbf{let}\ speeds == speedList(minimumSpeed(type(aircraft!)), \\
\quad maximumSpeed(type(aircraft!))) \bullet \\
\quad menuIndex? \in \operatorname{dom} speeds \land \\
\quad speed! = speeds(menuIndex?) \\
\hline
\end{array}
$$

3.2 Attaching the HCI to the Functional Core

The operation *selectSpeed*1 represents the interface portion of the speed selection operation. Associated with this, the selected speed must be recorded as the instructed speed of the aircraft in the ATC core system. The whole *selectSpeed* operation consists of the operation *selectSpeed*1 being piped to the operation *changeSpeed* (from Section 2.2). That is, the speed output from *selectSpeed*1 is input into *changeSpeed*.

$$\mathit{selectSpeed} \mathrel{\widehat=} \mathit{selectSpeed}1 \gg \mathit{changeSpeed}$$

Some changes to the ATC system occur independently of the operator, such as the *updateTelemetry* operation defined in the ATC system model. These changes in the ATC system must also be reflected in the HCI by updating the underlying interface state. The *refreshHCI* operation refreshes the interface state (using a brute force approach) – the aircraft views are re-created from the air traffic (and the selected view is updated accordingly), the warnings are updated, and the alarm status is set accordingly.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{refreshHCI}} \\
\hline
\Delta \mathit{ATCInterface} \\
\Xi \mathit{Traffic} \\
\mathit{violations}? : \mathbb{P}\, \mathit{Callsign} \\
\hline
\mathit{views}' = \{ \mathit{ac} : \mathit{aircraft} \bullet \\
\qquad \mathit{makeView}(\mathit{ac}, \mathit{telemetry}(\mathit{ac}), \mathit{flightPlan}(\mathit{ac})) \mapsto \mathit{ac} \} \\
\mathit{selected}' = (\mathit{views}')^{\sim} (\!|\ \mathit{views} (\!|\ \mathit{selected}\ |\!)\ |\!) \\
\mathit{speedMenu}' = \mathit{speedMenu} \\
\mathit{warnings}' = \mathit{views}^{\sim} (\!|\ \mathit{violations}?\ |\!) \\
\mathit{alarm}' = \mathit{if}\ \mathit{violations}? = \varnothing\ \mathit{then}\ \mathit{off}\ \mathit{else}\ \mathit{on} \\
\hline
\end{array}$$

When the interface is refreshed using *refreshHCI*, the violations input into the operation (used to update the warnings) need to be calculated. This calculation is performed and output by the operation *detectSeparationViolation* defined in the ATC core system. A complete update of the HCI thus involves piping the *detectSeparationViolation* operation to *refreshHCI*.

$$\mathit{updateHCI} \mathrel{\widehat=} \mathit{detectSeparationViolation} \gg \mathit{refreshHCI}$$

The entire ATC system includes both the *ATCInterface* defined above and the *ATCCore* system defined in Section 2.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ATC}} \\
\hline
\mathit{ATCInterface} \land \mathit{ATCCore} \\
\hline
\operatorname{ran} \mathit{views} = \mathit{aircraft} \\
\hline
\end{array}$$

3.3 The User Action Notation

The remainder of the HCI model integrates the Z notation with the User Action Notation (UAN) - a simple notation for describing “the behaviour of the user and the interface as they perform a task together” [7].

The symbols specific to UAN used in the HCI model are summarised in Figure 3 (taken from [7]). Some additional feedback symbols for the various forms of highlighting the aircraft views in the ATC HCI are defined in Figure 4. The two forms of highlight provided by these symbols are not exclusive: they can be applied simultaneously.

Action	Meaning
$\sim$[X]	Move the cursor into the context of object X
$\sim$[x,y]	Move the cursor to point x,y outside any object
$\sim$[X in Y]	Move the cursor to object X within object Y
[X]$\sim$	Move the cursor out of context of object X
$\mathrm{M}_L \vee \wedge$	Click (depress & release) the left mouse button
$\mathrm{M}_R \vee \wedge$	Click the right mouse button
	task is performed zero or more times
⨟	task interrupt symbol - indicates the user may interrupt current task at this point
:	separator between condition and action or feedback
Feedback	Meaning
!	highlight object
$-$!	dehighlight object
@x,y	at point x,y
display(X)	display object X
erase(X)	erase object X

Fig. 3. User Action Notation symbols used in the ATC HCI model

Feedback	Meaning
$!_{\mathsf{S}}$	highlight object using the selection highlight (a circle is drawn around the aircraft dot in the aircraft view)
$!_{\mathsf{W}}$	highlight object using the warning highlight (a different colour is used for the aircraft view)
$-!_{\mathsf{S}}$	turn off selection highlight on object
$-!_{\mathsf{W}}$	turn off warning highlight on object

Fig. 4. Additional feedback symbols for highlighting aircraft views

3.4 Displaying the System State

The underlying interface state (Section 3.1) defines those interface objects that are used to visualise the core system state, but does not define how those objects are composed on the HCI. Here we describe (using a blending of Z with UAN) this aspect of the HCI associated with the *refreshHCI* operation. Identical names indicate a coupling between these definitions and the definition of *refreshHCI*. In particular note that unprimed names (e.g. *selected*) refer to old interface objects before the update and primed names (e.g. *selected*$'$) refer to the new interface objects after the update.

Visualisation of the ATC system state consists of displaying the aircraft views at the appropriate positions on the screen. In UAN, screen positions are described as points, so the *convertPosition* function is provided to convert between sector positions and screen positions.

$$| \; convertPosition : Position \rightarrow \mathbb{Z} \times \mathbb{Z}$$

We describe the effects of *refreshHCI* using the feedback symbols of UAN.

Erase the old aircraft views: Because *refreshHCI* defines a brute force update (each aircraft view on the HCI is replaced with a new view), all of the old aircraft views must be erased.

$$\forall\, oldView : \mathrm{dom}\, views \bullet \mathsf{erase}(oldView)$$

Display the new aircraft views: The old aircraft views are replaced by the new aircraft views in the appropriate positions.

$$\begin{array}{l}\forall\, newView : \mathrm{dom}\, views' \bullet \\ \quad @convertPosition(position(telemetry(views(newView)))) \\ \qquad \mathsf{display}(newView)\end{array}$$

Apply the selection highlight: The selection highlight is applied to the new aircraft views.

$$\forall\, acView : selected' \bullet acView!_{\mathsf{S}}$$
$$\forall\, acView : \mathrm{dom}\, views' \setminus selected' \bullet acView{-}!_{\mathsf{S}}$$

Apply the warning highlight: The warning highlight is applied to the new aircraft views.

$$\forall\, acView : warnings' \bullet acView!_{\mathsf{W}}$$
$$\forall\, acView : \mathrm{dom}\, views' \setminus warnings' \bullet acView{-}!_{\mathsf{W}}$$

3.5 The User Actions

In the following definitions of the user actions the coupling between the user actions and the underlying interface state is implicit in the usage of common attribute and operation names. For example, the first user task defined below, *selectAircraft,* defines the behaviour of the user and the interface that accompanies the *selectAircraft* operation defined in the underlying interface state.

Task: *selectAircraft*

The operator selects an aircraft by moving the mouse over the appropriate aircraft view and clicking the left mouse button:

User Action	Interface Feedback	Operation input
~[aircraft_view]		
$M_L \vee \wedge$	$\forall\, acView : selected \bullet$ $acView{-}!_{\mathsf{S}}$ aircraft_view $!_{\mathsf{S}}$	*aircraft*? = aircraft_view

Task: *changeAircraftSpeed*

The operator instructs the selected aircraft to change speed by opening the speed menu, navigating the menu to the desired speed, then selecting it:

$openSpeedMenu \mathbin{;} navigateSpeedMenu \mathbin{;} selectSpeed$

Note that the '$\mathbin{;}$' symbol used above is the task interrupt symbol. If the user interrupts *changeAircraftSpeed* the effect is: erase(*speedMenu*)

Subtask: *openSpeedMenu*

If an aircraft view is selected, the operator can open the speed menu by clicking the right mouse button:

User Action	Interface Feedback	Interface State
$selected \neq \varnothing$:		
($\sim$[x,y] $M_R \vee \wedge$)	@ x,y display($speedMenu'$)	$speedMenu' = makeMenu(\ldots)$

Subtask: *navigateSpeedMenu*

The operator navigates within the speed menu by moving the mouse in and out of the lines in the menu:

User Action	Interface Feedback
$\sim$[line **m** in *speedMenu*]	line **m** !
($\mathbin{;}$ [line **m** in *speedMenu*]$\sim$ $\mathbin{;}$	line **m** $-$!
$\sim$[line **n** in *speedMenu*])*	line **n** !

If the user interrupts *navigateSpeedMenu* the effect is: erase(*speedMenu*).

In the above, 'line **m**' refers to the **m**th line in the menu.

Subtask: *selectSpeed*

The operator selects a speed from the speed menu when the mouse is over the appropriate speed line by clicking the left mouse button:

User Action	Interface Feedback	Operation input
$\sim$[line **m** in *speedMenu*] :		
$M_L \vee \wedge$	erase(*speedMenu*)	$menuIndex? = \mathbf{m}$

4 The ATC Operator Model

The operator model details the high level cognitive processes of the ATC operator, and avoids details of the low level mechanics of human memory. The model follows a basic Scanning-Decision-Action cycle depicted in Figure 5.

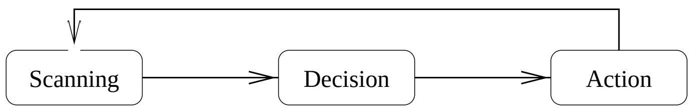

Fig. 5. The basic Scanning-Decision-Action cycle of the Operator model

The detailed model is presented in two parts. Section 4.2 describes the control flow through the cognitive processes of the ATC operator. Section 4.3 clarifies the flow of information created and used by these cognitive processes.

The cognitive model is a memory-based model, and involves a number of different memories. The most important of these memories are the operator's episodic and short-term memories. These memories are intended to capture the functionality of human memory (its dependence on cues, number of rehearsals, recency of occurrence and capacity) in a parsimonious manner and are not intended as psychological or physiological hypotheses about the fundamental structure of human memory.

Episodic memory The operator's episodic memory is used for recording the simulation episodes experienced by the operator. It contains a sequence of event relations (defined in more detail below) each describing a previous simulation experience. The memory for a particular event relation is cued by the information presented on the screen such as the aircraft, call sign, location, position in relation to other aircraft, etc. This memory has a long term or semantic component (it is affected by the memory for similar episodes which occurred on previous days), an intermediate term component (it is affected by the number of times the current event relation has been retrieved and stored during the scanning process), and a short-term component (it is affected by the recency of the last storage). The episodic memory has a large capacity.

Short-term memory The operator's short-term memory is used to temporarily record information of recent relevancy. This includes event relations for events recently experienced and either their associated priorities or a record that action has been taken. There is no specific cue for these event relations so recall is determined by recency. The short-term memory has a very limited capacity.

4.1 The Event Relation

The event relation is the main form of information stored in each of the above two memories in the cognitive model. An event relation is modelled as a tuple of the form:

$$\textit{event type}(\textit{aircraft attributes}, \textit{context}, \textit{classification}, \textit{time}, \textit{action})$$

The various elements of an event relation are as follows:

event type The relationship type of the event. This includes the following values:

nonevent	not an event at all;
converge	an event in which two aircraft are on converging flight paths;
overtake	an event in which two aircraft on the same flight path are in an overtaking situation.

aircraft attributes The attributes of the aircraft involved in the event including, for example, call signs, aircraft types, speeds, etc.

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

classification The classification of the event. The value is one of the following:

conflict	the event is expected to end in a conflict – corrective action is needed;
nonconflict	the event is expected to end without conflict;
?	it is unknown/uncertain whether the event will result in a conflict or not.

time The latest time at which corrective action can safely be taken to avoid a conflict. Time is described as the time delay from some time of day (usually the time of projection) and may involve a period of uncertainty. '?' indicates that the time is unknown.

When checking the event time the following three abstract times are used:

now	the current time of day;
later	any time of day after now (i.e. in the future);
passed	any time of day before now (i.e. in the past).

action The corrective action to resolve the event and avoid the conflict. The corrective action is something like: 'slow the trailing aircraft down and speed the leading aircraft up'. '?' indicates that the action is unknown.

Consider, for example, the following event relation that describes one of the events in the sector shown in Figure 1:

converge$((\{QF053, H, 42\}, \{CZT, L, 26\}),$
"Approaching Borrow Island en-route to Exmouth airport",
conflict, 11:23+10±1, ?$)$

This describes a convergence event involving the two aircraft – QF053 (a heavy aircraft travelling at 420knots) and CZT (a light aircraft travelling at 260knots) – as they approach the waypoint 'Borrow Island' en-route to 'Exmouth airport'. The event has been classified as a conflict, so a separation violation is anticipated as these aircraft converge on Borrow Island. In order to avoid this violation corrective action needs to be taken before sometime between 11:32am and 11:34am (10 minutes after 11:23am give or take 1 minute). The event relation does not provide any action from previous experiences.

4.2 Control Flow model

The control flow in the cognitive model is modelled using the statechart diagram presented in Figure 6. The basic Scanning-Decision-Action cycle is evident in this model, however substantial amounts of finer details are also included. The details of the complete model are now described.

Scanning: The ATC simulation operator begins in the Scanning state. This entails the physical monitoring of the ATC interface (the Monitoring state) until the representation on the interface matches some basic pattern. Such patterns are the basic geometric structures formed of planes and routes when involved in a potentially hazardous event. For example, an overtaking hazard involves the simple geometry in which the two involved aircraft are on the same route.

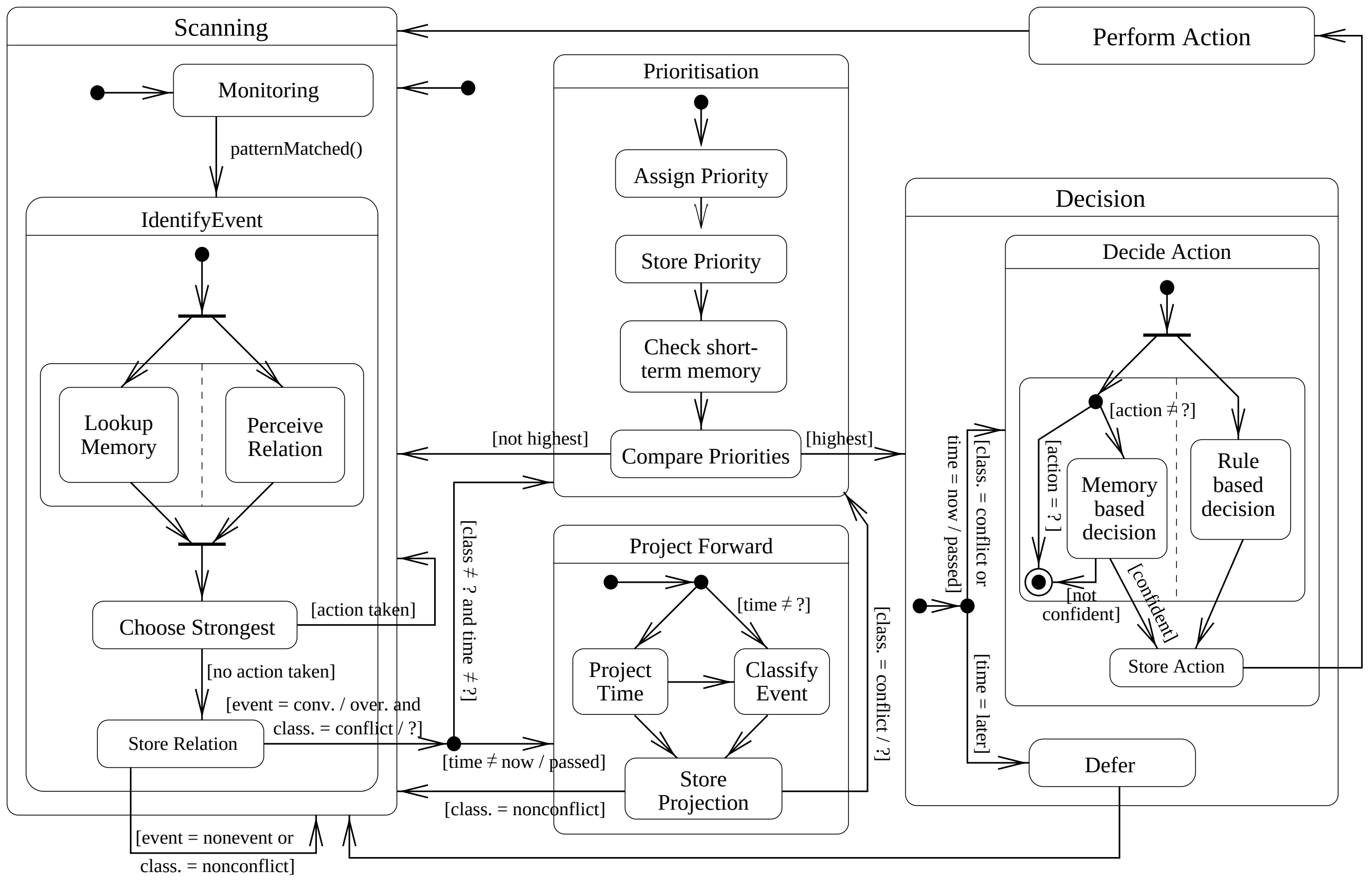

Fig. 6. The control flow model of the ATC simulation operator

Detailed identification of an event (the IdentifyEvent state) begins by simultaneously looking up the operator's memory (the Memory Lookup state) for the previous episode that best matches the current event (if there is any), and creating a new event relation (the PerceiveRelation state). The relation retrieved from memory will include the event type, aircraft attributes, and event context, but may also include additional information, such as the corrective action. If the matching event relation is drawn from short-term memory (rather than episodic memory), there may be an associated record that the corrective action has been taken. The created relation (whose type – overtake or converge – is derived from the geometry of the matched event) includes the aircraft attributes and event context for the event. This is typically the only information provided in the created relation. In specific situations, however, PerceiveRelation will provide specific information for the event classification and event time. This occurs when the event is immediately impending and action must be taken *now*!

Each of the two event relations has an associated strength. For the retrieved relation this is related to the strength of the match. The strength of the created relation is low, unless the event is immediately impending and the strength is high. Consequently the operator has a tendency to reuse past experience unless differences with the current event allow the created relation to take over. The weaker event relation is discarded (in the ChooseStrongest state). If there is a record that the stronger event relation has already been acted on, the operator returns to Scanning, allowing additional time for their actions to take effect. If there is no such record the event relation is stored in memory (the StoreRelation state) and the event is considered further.

The Decision-Action portions of the basic cycle are not relevant to events where the event type is nonevent, or where the event classification is nonconflict. For example: two aircraft matching the basic overtaking geometry where the leading aircraft is moving faster than the trailing aircraft would be a nonevent; two aircraft matching the basic convergence geometry on approach to the same airport and arriving a 'safe' time apart would be a converge event classified as a nonconflict. All other events must be considered further.

Project Forward: For those events where the event time is not now or passed (such events require immediate action!), the operator may project the event forward (the ProjectForward state) to estimate the event time (the ProjectTime state), and to classify the event (the ClassifyEvent state). These projections are added into the event relation, and the updated event relation is recorded in memory (the StoreProjection state). As the event classification may have changed, it is again checked. Events classified as nonconflict require no further consideration. Events with other classifications are considered further.

If both the event classification and event time are known, the operator can skip ProjectForward and proceed directly with prioritisation of the event. The operator may choose to re-project the event forward and will proceed to prioritisation afterward (provided the classification is appropriate, as noted above).

Prioritisation: The operator uses event priorities to manage the order in which events are dealt with (the Prioritisation state). The current event is assigned a priority (the

AssignPriority state) which is associated with that event relation in short-term memory (the StorePriority state). Short-term memory is checked for priorities associated with other event relations currently recorded (the CheckShort-TermMemory state). All priorities found (if any) are compared with the priority of the current event (the ComparePriorities state). If higher priority events are found these must be dealt with first, so the operator returns to Scanning. If the current event is the highest priority, it is dealt with immediately.

Decide Action: Next, the operator enters the decision making processes (the Decision state). If the event time is later, the operator may defer deciding on the corrective action until a later time (the Defer state). If so, the operator returns to Scanning. If the event is classified as a conflict the operator can immediately decide on the corrective action, and if the event time is now or passed the operator must immediately decide on the corrective action (the DecideAction state).

The corrective action is decided between two simultaneous processes. The first (the MemoryBasedDecision state) uses the event action recorded in the event relation. The action value is not unknown only if the event relation originally came from MemoryLookup – it is the action used to resolve the previous, matching event. A confidence level is associated with this action – if it is high, the operator will reuse this action to resolve the current event. When the event relation has an unknown action, or the confidence in the recorded action is low, the operator uses a rule-based decision process to deduce the action to take (the RuleBasedDecision state). The event relation is updated with the decided action (decided using either of the above two processes) and stored in memory (the StoreAction state).

Perform Action: Finally, the operator performs the decided action (the PerformAction state) using the ATC simulation interface and returns to Scanning.

4.3 Information Flow model

The cognitive process described above involves a continuous flow of information from one state in the process to the next as shown by the data flow diagram in Figure 7. The most significant component in this information flow is the event relation that is used and updated by most of the states. The updated event relation is frequently stored in the operator's memories, and associated information is frequently retrieved from the operator's memories.

4.4 Connection to the ATC HCI

The connection between the operator's cognitive model and the ATC HCI model is through the flow of data between the operator and the HCI as shown in Figure 7 (the data flows annotated 'raw screen data' and 'HCI actions').

Firstly, the visual (and audible) representation of the ATC system in the HCI provides the main external inputs to the cognitive model. These inputs are used in the Scanning states and while the operator performs any corrective actions.

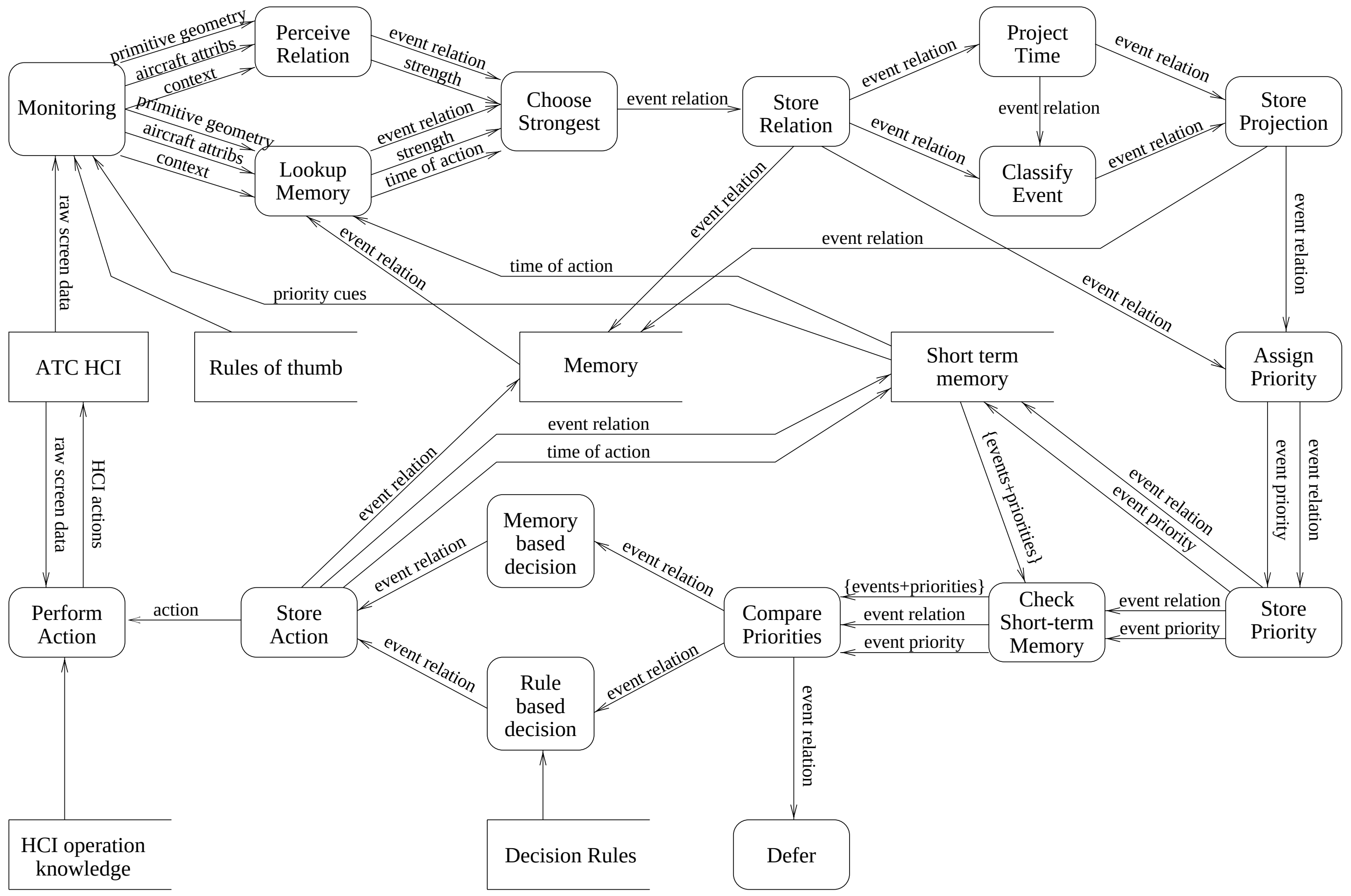

Fig. 7. The information flow model of the ATC simulation operator

Secondly, in executing any corrective actions the user must interact with the HCI. These are the outputs of the PerformAction state. The actions involved are those modelled in Section 3.5. Being modelled in UAN, the involvement of both the operator and the HCI in these actions is implicit.

Clearly the design of the HCI has a significant impact on the cognitive process – it determines the presentation of the information to the operator on which the cognitive process is dependent, and defines the interaction with the operator for performing actions. Differences in the HCI design can thus be expected to have potentially significant effects on the operator's cognitive process, including impacting on the likelihood of human error.

5 Conclusions

We have shown how appropriate formal models can be constructed of an air-traffic control simulation system, and the corresponding HCI and operator. Z, as a proven notation for modelling state-based system, has been used to model the core system state and functions, and the underlying state of the HCI. UAN, as a simple notation for describing the user and interface behaviour as a task is performed, has been used to abstractly model the user actions provided by the ATC system. Lastly, Statecharts and DFDs, as simple, diagrammatic notations, have been used to semi-formally model the operators cognitive process, such that the models can be easily validated by the psychologists.

The models are integrated in the sense that the communication between the components is modelled. In addition the HCI model integrates the UAN and Z notations to enable modelling of both the HCI actions engaged in by the operator, and the presentation of the underlying HCI state.

A unique feature of this work is that it incorporates a model of the operator's cognitive state and process. This model identifies the information focussed on, and the basic psychological process employed by the operator.

In later steps of the project, the cognitive model will be validated by observation of human subjects using the ATC simulator (on which our model of the core system and HCI is based). Analysis of the cognitive model is planned to identify the various sequences of errors in the cognitive process that can lead to hazards in the ATC system through erroneous operator decisions and actions. The conditions that cause these errors can then be used to quantify the likelihood of these errors occurring, and hence of hazards arising. These results are to be validated with those observed in empirical studies, and will be used to motivate improvements and changes to the notation and method. The contribution of HCI design decisions to the error producing conditions enables the likelihoods of these errors to be tied to the HCI design. Ultimately one of our aims is to utilise this to enable the effective comparison of different HCI designs with respect to system risk and likelihood of use error.

Acknowledgements

The authors wish to extend their thanks to Andrew Hussey for his contribution to this research during his time involved in the project. This work was supported by a grant from the Australian Research Council.

References

1. A.E. Blandford and D.J. Duke. Integrating user and computer system concerns in the design of interactive systems. *International Journal of Human-Computer Studies*, 46:653–679, 1997.
2. R. Butterworth, A. Blandford, and D. Duke. The Role of Formal Proof in Modelling Interactive Behaviour. In P. Markopoulos and P. Johnson, editors, *Design, Specification and Verification of Interactive Systems, DSV-IS '98*, pages 113–128. Springer-Verlag, 1998.
3. Commonwealth of Australia. Australian Defence Standard DEF(AUST) 5679: The Procurement of Computer-based Safety Critical Systems. Dept. of Defence, 1998.
4. D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–393, 1998.
5. European Committee for Electrotechnical Standardization. European Standard prEN 50128: Railway applications; Software for railway control and protection systems. CENELEC, 1995.
6. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, pages 231–274, 1987.
7. H. R. Hartson. Temporal Aspects of Tasks in the User Action Notation. *Human-Computer Interaction*, 7:1–45, 1992.
8. M. S. Humphreys, J. Wiles, and S. Dennis. Toward a theory of human memory: Data structures and access processes. *Behavioural and Brain Sciences*, 17(4):655–692, 1994.
9. A. Hussey and D. Carrington. Object-oriented user-interface development. *IEE Proceedings Software*, 146(5):221–231, October 1999.
10. International Electrotechnical Commission. 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. IEC, 1997.
11. N. G. Leveson. *Safeware: system safety and computers*. Addison-Wesley, 1995.
12. J. L. McClelland and D. E. Rumelhart. *Parallel Distributed Processing*. MIT Press, 1986.
13. P. Palanque and R. Bastide. Synergistic modelling of tasks, users and systems, using formal specification techniques. *Interacting with Computers*, 9(2), October 1997.
14. P. Palanque, F. Paternò, and R. Bastide. Formal specifications for designing user interfaces of air traffic control applications. In S. Gnesi and D. Latella, editors, *Second International ERCIM Workshop on Formal Methods for Industrial Critical Systems*. IEE, 1997.
15. F. Paternò, C. Santoro, and S. Tahmassebi. Formal models for cooperative tasks: concepts and an application for en-route air-traffic control. In P. Markopoulos and P. Johnson, editors, *Design, Specification and Verification of Interactive Systems, DSV-IS '98*, pages 71–86. Springer-Verlag, 1998.
16. J. W. Senders and N. P. Moray, editors. *Human Error: Cause, Prediction and Reduction*. Lawrence Erlbaum Associates, 1991.
17. J. M. Spivey. *The Z notation: a Reference Manual*. Prentice-Hall, 2nd edition, 1992.