DOI 10.1007/s00165-008-0102-7
BCS © 2009
Formal Aspects of Computing (2009) 21: 541–569

Formal Aspects of Computing

Verification-guided modelling of salience and cognitive load

Rimvydas Rukšėnas[1], Jonathan Back[2], Paul Curzon[1], and Ann Blandford[2]

[1] Queen Mary University of London, School of Electronic Engineering and Computer Science, Mile End Road, London E1 4NS, UK. E-mail: rimvydas@dcs.qmul.ac.uk
[2] University College London, UCL Interaction Centre, Gower Street, London WC1E 6BT, UK

Abstract. Well-designed interfaces use procedural and sensory cues to increase the cognitive salience of appropriate actions. However, empirical studies suggest that cognitive load can influence the strength of those cues. We formalise the relationship between salience and cognitive load revealed by empirical data. We add these rules to our abstract cognitive architecture, based on higher-order logic and developed for the formal verification of usability properties. The interface of a fire engine dispatch task from the empirical studies is then formally modelled and verified. The outcomes of this verification and their comparison with the empirical data provide a way of assessing our salience and load rules. They also guide further iterative refinements of these rules. Furthermore, the juxtaposition of the outcomes of formal analysis and empirical studies suggests new experimental hypotheses, thus providing input to researchers in cognitive science.

Keywords: Human error; Formal verification; Salience; Cognitive load; Model checking

1. Introduction

The correctness of interactive systems depends on the behaviour of both human and computer actors. Human behaviour cannot be fully captured by a formal model. However, it is a reasonable, and useful, approximation to assume that humans behave "rationally" in the sense of entering interactions with goals and domain knowledge likely to help them achieve their goals. If problems are discovered resulting from rational behaviour then such problems are liable to be systematic and deserve attention in the design. Whole classes of persistent, systematic user errors may occur due to modelable cognitive causes [Rea90, Gra00]. Often opportunities for making such errors can be reduced with good design [ByB97]. A method for highlighting those designs that allow users to make systematic errors, when behaving in a rational way, is important. It will allow such designs to be significantly improved.

1.1. Cueing mechanisms

After learning how to perform a task, some task steps are easily performed, while others have properties that make them difficult to remember, triggering slip errors [Gra00]. Remembering to collect your original document after making photocopies is a good example of a problematic task step. In this example when the primary goal of making photocopies has been achieved, this may act as a cue to the user that the task is complete and that

Correspondence and offprint requests to: R. Rukšėnas, E-mail: rimvydas@dcs.qmul.ac.uk

they are free to start a new task or resume a suspended task. Cognitive psychologists have found that individuals use a variety of different cueing mechanism when executing procedural knowledge. Two of these mechanisms (procedural and sensory) have been the primary focus of experimental manipulation in the laboratory [ChB04].

Procedural cues can be used to retrieve previously formulated knowledge enabling the next step to be performed. When a routine task is learned, task steps become associatively linked, i.e., action x (e.g. specifying the number of photocopies) becomes a procedural cue for action y (e.g. pressing the start button). Sensory cues (external to the cognitive system) can also be used to retrieve previously formulated knowledge. For example, if sensory cue p is attended to (e.g. paper jam warning) then it may indicate that q should be the next step (e.g. open up photocopier). In many domains, including safety-critical ones, it has been found (e.g. by Rasmussen [Ras80]) that these types of cueing mechanisms are not reliable. Rasmussen calculated that 34% of errors arose from the omission of functionally isolated steps, i.e., steps that are not cued by the external environment or internal user goals.

System designers recognise that some task steps are more error prone than others. Therefore, the sequence of actions required during human-computer interaction is sometimes imposed by the device to prevent errors from occurring. For example, some automated teller machines (ATMs) return a user's card before dispensing money so that a user cannot forget to collect their card. This ensures that remembering to perform the safety-critical step (taking your card) is not reliant on procedural cueing. However, there are many situations where changing the sequence of actions required is not straightforward. This is especially true in complex systems where goals are dynamic. Designers of simple systems such as an ATM, for example, rely on the assumption that getting cash is the primary goal. If a user's primary goal was to check an account balance then, with many designs, it is still possible for them to walk away from the ATM without their card.

Well-designed interfaces increase the sensory salience of signals that are used to cue actions that are frequently forgotten or are performed in the wrong sequence. Chung and Byrne [ChB04] found that a sensory signal has to be highly visually salient in order to ensure that a task-critical step, buried deep down in the task structure, is not forgotten. However, making a target action more salient by moving it to an area of the interface which is likely to be the focus of attention is often insufficient. The *specificity* of the cue is important. The cue has to be indicative of the type of action required, e.g., a red flashing arrow pointing at the button to be clicked. Low-level perceptual studies [CaL07] have shown that an individual is unable to process visually salient features if cognitive control functions are not available to maintain the active goal. This suggests that sensory cues are not always noticed under high workload scenarios.

1.2. Our earlier work

People make slip errors frequently, but do not make them every time. Our earlier laboratory work [BBC07], summarised in Sect. 4, showed that procedural and sensory cueing mechanisms are not resilient and can be influenced by cognitive workload in different ways. The focus of our laboratory studies was to identify and manipulate load variables that influenced the strength of procedural and sensory cues. We demonstrated that if a procedural or sensory cue was not strong enough to trigger a subsequent step then slip errors were more likely. To enable the generation of rules to represent how load variables influenced task performance the concept of *salience* was introduced into our formalism. The level of salience can determine the cue that is most likely to direct the interaction. Experimentation has revealed that the salience associated with a cue can be influenced by load.

The understanding gained allowed us to identify [BBC07] the abstract dependencies between salience and cognitive load. We then formalised [RBC+08] these dependencies within our abstract cognitive architecture developed earlier [CuB01, CRB07, RCB+07]. The architecture formalises abstract cognitive principles, such as a user entering an interaction with knowledge of the task and its subsidiary goals, and choosing non-deterministically between appropriate actions. Adding to our cognitive architecture salience and load concepts allowed us to refine the underlying principle of non-deterministic choice by introducing a hierarchy of goals, determined by the strength of cues and the level of cognitive workload imposed by the task performed.

To assess these developments, we undertook the formal verification of the fire engine dispatch task [BBC07] used in our empirical studies. Though overall this analysis yielded results [RBC+08] that are consistent with the human behaviour observed during the experimental studies, it also produced false positives and missed systematic errors in some cases. Such results suggest that our original salience and cognitive load rules were too abstract, missing important aspects of human cognition.

1.3. Contribution

This paper deals with the limitations of our earlier work while also demonstrating how formal tools can be of use to challenge and clarify results in cognitive science. We used our formal models as a tool for developing a *deeper* understanding of the experimental results with respect to cognitive salience and its dependency on workload. In particular, we experimented with different formal versions of salience and load rules using a model checker. Comparing, in each case, the formal verification results with our empirical data and guided by these comparisons, we stepwise refined our formal models. The focus of these refinements was to construct a formal user model that is more reliable in identifying systematic errors. This cyclic process of formal modelling and analysis revealed more detailed dependencies between salience and cognitive load, captured as formal, generic rules. As a result, the refined models can be used to verify interactive systems with respect to systematic human error in greater detail. Furthermore, our findings feed back to research in cognitive science suggesting new hypotheses to be tested in future experiments.

Developing this understanding raised some interesting issues that needed to be overcome and required further experiments (Sect. 7). These resulted in the conceptualisation of a new type of knowledge-based *cognitive cue*. Thus the refined version of our cognitive architecture encapsulates *procedural cueing*—where the performance of an action acts as an associative cue for the next action, *sensory cueing*—built-in to the device in an attempt to guide the interaction, and *cognitive cueing*—where an individual uses knowledge of the domain to select between possible actions. All of these cueing mechanisms can be assigned a level of salience within the execution of an interactive task. The overall salience level then determines the priority of a goal in our hierarchy of salience levels.

Summarising, this paper describes an investigation into the formal modelling of salience and cognitive load. Its main contribution is the following:

- A formalisation of salience and its dependence on cognitive load.
- A deeper understanding of the empirical data as a result of verification-guided iterative refinement of salience and load rules.
- A conceptualisation of a new type of cueing—the cognitive cue.
- An extension of our cognitive architecture, including a refinement of our hierarchy of salience levels, and our verification framework yielding more accurate results in the formal analysis of interactive systems.
- The generation of new hypotheses to be tested empirically.

Thus our work further advances interdisciplinary research by illustrating the mutual benefits of bridging cognitive science and formal methods in computer science.

1.4. Related work

People develop mental models of the systems and devices they interact with, and use those models to guide their interactions and predict how a system will behave and respond. Norman [Nor83] emphasized the need for a correspondence between a user's mental model and the conceptual model that a system is designed around, arguing that greater correspondence facilitates better learning of the system and leads to improved performance. This is typical of theories in human–computer interaction that suggest systems should be designed so that user intentions are easily executed; minimising cognitive complexity by ensuring that the system provides affordances. This type of theory is useful when developing walk-up-and-use systems since assumptions cannot be made about the expertise of the user. An operator in a safety-critical system, however, is an expert user who performs cognitively complex functions in demanding situations. An approach is needed so that the cognitive demands placed on individuals can be modelled.

There is little related work on salience and cognitive load. However, evidence from foundational studies, which have looked at how proceduralised knowledge is executed, suggest that individuals are reliant on cues internal and external to the cognitive system [Gra00], and that these cues can be influenced by oncoming task demands [BFH+08] and residual load [ByB97]. More generally, work on human error has shown that the provision of visual cues can strengthen procedural cueing providing they manage to capture attention [ChB04]. While developing causal accounts enables us to understand specific manifestations of error, thus far, no attempt has been made to consider causal accounts simultaneously. Being able to verify real-world systems requires this ability.

There are two basic approaches to formal modelling of specific user behaviours within the underlying system. In a simpler form, this involves writing both a formal specification of the device and task models for that device, to

support reasoning about the behaviour of the interactive system [PaM95, MoD95, Fie01, Cam03]. Task models, however, describe how users are intended to behave; as such they do not deal with human fallibility. An alternative approach is to specify users the way they are [BBD00]. This idea underlies approaches based on formal user modelling [DBD+98, BoF99, DuD99]. By representing the cognitive structures of the user in a generic way, formal user models specify how user behaviour is generated. To reason about the behaviour of an interactive system, a formal user model is combined with a formal specification of the device. Both models are then considered as central components of the system in such approaches known as *syndetic modelling* [DBM+95].

Within the paradigm of syndetic modelling, Duke et al. [DBD+98] and Bowman and Faconti [BoF99] use Interactive Cognitive Subsystems (ICS) [BaM95] as the underlying model of human information processing. Their models deal with information flow between the different cognitive subsystems and constraints on the associated transformation processes. As a result, their work focusses on reasoning about multi-modal interfaces and analyses whether interfaces based on several simultaneous modes of interaction are compatible with the capabilities of human cognition.

More specifically, Su et al. [SBB07] have developed an ICS-based formal cognitive model to explore the deployment of temporal attention in humans. They use this model to investigate how the salience of items is affected by their meaning. In particular, this model has been adopted [SBB+08] to simulate interactions in the presence of information overload. Compared to ours, this work is based on lower level cognitive models. Also, it focuses on simulation which would correspond to the laboratory work in our approach.

In the area of formal verification, Rushby et al. [Rus01] focus on mode errors and the ability of pilots to track mode changes. They formalise plausible mental models of systems and analyse them using the Murϕ verification tool. The mental models though are essentially abstracted system models; they do not rely upon structure provided by cognitive principles.

2. Cognitive architecture

Our cognitive architecture is a higher-order logic formalisation of abstract principles of cognition and specifies a form of cognitively plausible behaviour [BBD00]. The architecture specifies possible user behaviour (traces of actions) that can be justified in terms of specific results from the cognitive sciences. Real users can act outside this behaviour of course, about which the architecture says nothing. However, behaviour defined by the architecture can be regarded as potentially systematic, and so erroneous behaviour is similarly systematic in the design. The predictive power of the architecture is bounded by the situations where people act according to the principles specified. The architecture allows one to investigate what happens if a person acts in such plausible ways. The behaviour defined is neither "correct" nor "incorrect". It could be either depending on the environment and task in question. We do not attempt to model the underlying neural architecture nor the higher-level cognitive architecture such as information processing. Instead our model is an abstract specification, intended for ease of reasoning.

2.1. Cognitive principles

In the formal user model, we rely upon abstract cognitive principles that give a *knowledge level* description in the terms of Newell [New90]. Their focus is on the internal goals and knowledge of a user. These principles are briefly discussed below. Their formalisation within the SAL (Symbolic Analysis Laboratory) framework [MOR+04] is described in Sects. 2.2 and 3.

Non-determinism: In any situation, any one of several cognitively plausible behaviours might be taken. It cannot be assumed that any specific plausible behaviour will be the one that a person will follow where there are alternatives.

Relevance: Presented with several options, a person chooses one that seems relevant to the task goals. For example, if the user goal is to get cash from an ATM, it would be cognitively implausible to choose the option allowing one to change a PIN. A person could of course press the wrong button by accident, for example. Such classes of error are beyond the scope of our approach, focussing as it does on systematic slips.

Salience: Even though user choices are non-deterministic, they are affected by the salience of possible actions. For example, taking money released by a cash-point is a more salient, and thus much more likely, action to take than to terminate the interaction by walking away from the machine without cash. In general, salience could be affected by several factors such as the sensory (visual) prominence of an action, its procedural cueing as a part of a learned task, and the cognitive load imposed by the complexity of the task performed.

Mental versus physical actions: There is a delay between the moment a person mentally commits to taking an action (either due to the internal goals or as a response to the interface prompts) and the moment when the corresponding physical action is taken. To capture the consequences of this delay, each *physical* action modelled is associated with an internal *mental* action that commits to taking it. Once a signal has been sent from the brain to the motor system to take an action, it cannot be revoked after a certain point even if the person becomes aware that it is wrong before the action is taken. To reflect this, we assume that a physical action immediately follows the committing action.

Pre-determined goals: A user enters an interaction with knowledge of the task and, in particular, task dependent sub-goals that must be discharged. These sub-goals might concern information that must be communicated to the device or items (such as bank cards) that must be inserted into the device, for example. Given the opportunity, people may attempt to discharge such goals, even when the device is prompting for a different action. Such *pre-determined* goals represent a partial plan that has arisen from knowledge of the task in hand, independent of the environment in which that task is performed. No fixed order is assumed over how pre-determined goals will be discharged.

Reactive behaviour: Users may react to an external stimulus, doing the action suggested by the stimulus. For example, if a flashing light comes on a user might, if the light is noticed, react by inserting coins in an adjacent slot.

Voluntary task completion: A person may decide to terminate the interaction. As soon as the main task goal has been achieved, users intermittently, but persistently, terminate interactions [ByB97], even if subsidiary tasks generated in achieving the main goal have not been completed. A cash-point example is a person walking away with the cash but leaving the card. Users also may terminate interactions when the signals from the device or environment suggest that task continuation is impossible due to some fault. For example, if the cash-point signals that the inserted card is invalid (and therefore retained), a person is likely to walk away and try to contact their bank.

Forced task termination: If there is no apparent action that a person can take that will help to complete the task then the person is forced to terminate the interaction. For example, if, on a ticket machine, the user wishes to buy a weekly season ticket, but the options presented include nothing about season tickets, then the person will give up, assuming the goal is not achievable.

2.2. Cognitive architecture in SAL

We have formalised the above cognitive principles within the SAL environment [MOR+04]. It provides a higher-order language for specifying concurrent systems in a compositional way. State machines are specified in SAL as parametrised modules and can be composed either synchronously or asynchronously. Since our approach is based on a *generic* cognitive architecture, SAL support for higher-order specifications is an essential feature. The SAL notation we use here is given in Table 1. We also use the usual notation for the conjunction, disjunction and set membership operators. A simplified version of the SAL specification of a transition relation that defines our generic user model (SAL module `User`) is given in Fig. 1, where predicates in italic are shorthands explained later on. Below, whilst explaining this specification, we also discuss how it reflects our cognitive principles. The full formalisation of our cognitive architecture is presented in Appendix A.

Table 1. A fragment of the SAL language

Notation	Meaning
`x:T`	`x` has type `T`
`λ(x:T):e`	A function of `x` with the value `e`
`x′ = e`	An update: the new value of `x` is that of `e`
`{x:T | p(x)}`	A subset of `T` such that the predicate `p(x)` holds
`a[i]`	The `i`-th element of the array `a`
`r.x`	The field `x` of the record `r`
`r WITH .x := e`	The record `r` with its field `x` updated by `e`
`g` → upd	If `g` is true then update according to `upd`
`c [] d`	Non-deterministic choice between `c` and `d`
`[](i:T): c_i`	Non-deterministic choice between c_i with `i` in range `T`

```
TRANSITION
  [](g:GoalRange;sens,cog,someSens,someCog:Salience...):CommitAction:
      sens ≠ LowSLC ∧ cog ≠ LowSLC ∧
      someSens ≠ LowSLC ∧ someCog ≠ LowSLC ∧
      (HighestSalience(sens, cog, g, status, goals)(...)
       ∨
       HighSalience(sens, cog, g, status, goals)(...) ∧
       NOT(∃h : HighestSalience(someSens, someCog, h, status, goals)(...))
       ∨
       LowSalience(sens, cog, g, status, goals)(...) ∧
       NOT(∃h : HighSalience(someSens, someCog, h, status, goals)(...))) ∧
      (g ≠ ExitGoal ∨ MayExit) ∧
      NOT(acommitted) ∧ finished = notf ∧
      ...                                                          →
      commit′[act(Goals[g].subgoals)] =
        committed;
      status′ = status
        WITH .trace[g] := TRUE
        WITH .last := g
        WITH .length := status.length + 1
      ...
[]
  [](a:ActionRange):PerformAction:
      commit[a] = committed                                        →
      commit′[a] = ready;
      out′ ∈ {x : Out | Actions[a].tout(in, out, mem)(x)};
      mem′ ∈ {x : Memory | Actions[a].tmem(in, mem, out′)(x)};
      env′ ∈ {x : Env | Actions[a].tenv(in, mem, env)(x) ∧ possessions};
      ...
[]
  ExitTask:
      goals[TopGoal].achieved(in, mem) ∧
      NOT(acommitted) ∧                                            →
      finished = notf                                              finished′ = ok
[]
  [](someSens,someCog:Salience):Abort:
      someSens ≠ LowSLC ∧ someCog ≠ LowSLC ∧
      NOT(ExistsSalient(someSens, someCog, status, goals...)(...)) ∧
      NOT(goals[TopGoal].achieved(in, mem)) ∧                      →
      NOT(acommitted) ∧
      finished = notf                                              finished′ = IF Wait(in, mem)
                                                                               THEN notf
                                                                               ELSE abort
                                                                               ENDIF;
                                                                   ...
[]
  Idle:
      finished = notf                                              →
```

Fig. 1. Cognitive architecture in SAL (simplified). Its full specification is given in Appendix A.4

Guarded commands: SAL specifications are transition systems. Non-determinism is represented by the non-deterministic choice, `[]`, between the named guarded commands (i.e. transitions). For example, *CommitAction* in Fig. 1 is the name of a family of transitions indexed by goal g. Each guarded command in the specification describes an action that a user *could* plausibly take. The pairs *CommitAction*—*PerformAction* of the corresponding transitions reflect the connection between the physical and mental actions. The first of the pair models committing to a goal, the second actually taking the corresponding action (see below).

Goals structure: The main concept in our cognitive architecture is that of user *goals*. User goals are organised as a hierarchical (tree like) goal–subgoals structure. The nodes of this tree are either compound or atomic:

atomic goals: Goals at the bottom of the structure (tree leaves) are atomic: they consist of (map to) an action, for example, a device action.

compound goals: All other goals are compound: they are modelled as a set of task subgoals.

In this paper, we consider an essentially flat goal structure with the top goal consisting of atomic subgoals only. We will explore the potential for using hierarchical goal structures in subsequent work.

In SAL, user goals are modelled as an array, `Goals`, which is a parameter of the `User` module. Each element `g` in `Goals` is a record with the following fields:[1]

guard: This field is a predicate, denoted `grd`, that specifies when the goal `g` is enabled, for example, due to the relevant device prompts.

choice: This field is a predicate (choice strategy), denoted `choice`, that models a high-level ordering of goals by specifying when the goal `g` can be chosen. An example of the choice strategy is: "choose only if `g` has not been chosen before."

achieved: This field is a predicate, denoted `achieved`, that specifies the main task goal when `g` is the top goal. It is not used for atomic goals.

salience: This field is a value, denoted `slc`, that specifies the sensory salience of `g`.

cueing: This field is a function, denoted `cue`, that for each goal, `h`, returns the strength of atomic goal `g` as a procedural cue for `h`. In refined versions of the architecture, this field is also used for compound goals. In this case, it is a function that for each goal `h` returns the strength of compound goal `g` (parent of `h`) as a cognitive cue for `h`.

load: This field is a value, denoted `load`, that specifies the intrinsic load associated with the execution of `g`.

subgoals: This field is a data structure, denoted `subgoals`, that specifies the subgoals of the goal. It takes the form `comp(gls)` when the goal consists of a set of subgoals `gls`. If the goal is atomic, its subgoals are represented by a reference, denoted `atom(act)` to an action in the array `Actions` (see below).

Goal execution: To see how the execution of an atomic goal is modelled in SAL consider the guarded command *PerformAction* for doing a user action that has been previously committed to:

$$\texttt{commit[a]} = \texttt{committed} \rightarrow \begin{array}{l} \texttt{commit}'\texttt{[a]} = \texttt{ready}; \\ \texttt{out}' \in \{\texttt{x} : \texttt{Out} \mid \texttt{Actions[a].tout(in, out, mem)(x)}\}; \\ \texttt{mem}' \in \{\texttt{x} : \texttt{Memory} \mid \texttt{Actions[a].tmem(in, mem, out}')\texttt{(x)}\}; \\ \texttt{env}' \in \{\texttt{x} : \texttt{Env} \mid \texttt{Actions[a].tenv(in, mem, env)(x)} \wedge \mathit{possessions}\} \end{array}$$

The left-hand side of $\rightarrow$ is the guard of this command. It says that the rule will only activate if the associated action has already been committed to, as indicated by the element `a` of the local variable array `commit` holding value `committed`. If the rule is then non-deterministically chosen to fire, this value is changed to `ready` to indicate there are now no commitments to physical actions outstanding and the user model can select another goal. Finally, the three relations (set membership) represent the state updates associated with this particular action `a`. They are explained below.

The state space of the user model consists of three parts: input variable `in`, output variable `out`, and global variable (memory) `mem`; the environment is modelled by a global variable, `env`. All of these are specified using type variables and are instantiated for each concrete interactive system. The state updates associated with an atomic goal are specified as an action. The latter is modelled as a record with the fields `tout`, `tmem` and `tenv`. The array `Actions` is a collection of all user actions. The three fields are relations from old to new states. They are provided when the generic user model is instantiated and specify how two components of the user model state (outputs `out` and memory `mem`) and environment `env` are (non-deterministically) updated by executing the corresponding action.

Since we are modelling the cognitive aspects of user actions, all three state updates depend on the initial values of inputs (perceptions) and memory. In addition, each update depends on the old value of the component updated. The memory update also depends on the new value (`out`′) of the outputs. This provides a simple way to specify that the user remembers the action just taken. The update of `env` must also satisfy a generic relation, *possessions*. It specifies universal physical constraints on possessions and their value, linking the events of taking and giving up a possession item with the corresponding increase or decrease in the number (counter) of items possessed. For example, it specifies that if an item is not given up then the user still has it. The counters of possession items are modelled as environment components.

PerformAction is enabled by executing the guarded command for selecting an atomic goal, *CommitAction*, which switches the commit flag for some action `a` to `committed` thus *committing* to this action (enabling *PerformAction*). The four index variables (having type `Salience`) are used to model non-deterministic values

[1] Note that we are omitting from the description of the goal structure some aspects related to the relevance and timing of goals. They are not used in the work described here; for the omitted detail see [RCB+08].

of the strength of sensory and cognitive salience when these are affected by high cognitive load (see Sect. 3). A goal `g` *may* be selected only when one of the disjuncts specifying its salience level is true. We also distinguish the cases when the selected goal is `ExitGoal` or not (specified in *MayExit*). `ExitGoal` (given as a parameter of the `User` module) is intended to represent such options as "cancel" or "exit", available in some form in most interactive systems. We omit the definition of *MayExit* from Fig. 1 here, since it is irrelevant for this paper.

When an atomic goal, `g`, is selected, the user model commits to the corresponding action `act(Goals[g].subgoals)`. The record `status` keeps track of a history of selected goals. Thus, the element `g` of the array `status.trace` is set to true to indicate that the goal `g` has been selected, `status.last` records `g` as the last goal selected, and the counter of selected goals, `status.length`, is increased.

Task completion: In the user model, we consider two ways of terminating an interaction. Voluntary completion (`finished` is set to `ok`) can occur when the main task goal, as the user perceives it, has been achieved (see the *ExitTask* command). Forced termination (`finished` is set to `abort`) models random user behaviour due to there being no apparent way forward (see the *Abort* command). Since the choice between enabled guarded commands is non-deterministic, the *ExitTask* action may still not be taken. Also, it is only possible when there are no earlier commitments to other actions.

In the guarded command *Abort*, the condition of forced termination (no enabled salient actions) is expressed as the negation of the predicate `ExistsSalient` (it states that there exists goal `g` for which one of the predicates `HighestSalience`, `HighSalience` or `LowSalience` is true). Note that, in such a case, a possible action that a person could take is to wait. The user model will only do so given some cognitively plausible reason such as a displayed "please wait" message. The waiting conditions are represented in the model by predicate parameter `Wait`. If `Wait` is false, `finished` is set to `abort` to model a user giving up and terminating the task.

3. Salience and load rules

Here we discuss the connections between the salience of cues and cognitive load observed in our empirical studies [BBC+08]. We relied on the empirical results in our initial attempt [RBC+08] to formalise these connections, as presented in this section. Since the empirical studies were based on a 'Fire Engine Dispatch Centre' simulation (summarised in Sect. 4), we also developed a formal user model (Sect. 5) for that specific interactive system. The formal model is used to guide our refinements to the salience and load rules, as described in Sects. 6–8.

3.1. Cognitive salience and load

Slip errors are made frequently, but they are not made every time.[2] The frequency of these errors are determined by causal factors internal and external to the cognitive system. Procedural cues are useful when the sequence of performing a task does not vary since performing one action associatively cues another. Sensory cues are required to ensure that a user knows what action to perform next when this information cannot be known in advance.

Our experimental paradigm [BBC+08] explicitly manipulated load variables in an attempt to influence the strength of cueing mechanisms. Our hypothesis was that slip errors were more likely when the salience of cues was not sufficient to actively influence attentional control and drive the interaction. By manipulating cognitive load we found that the difficulty associated with performing a proceduralised task significantly influences the likelihood of making a slip error.

The inherent difficulty of the task at hand can be referred to as *intrinsic* cognitive load. Our experiments have shown that this load can influence the strength of procedural cues used to perform future task critical actions (background intrinsic cognitive load). Another load type, known as *extraneous* load, has been shown to influence the awareness individuals have of sensory cues when intrinsic load is high. Extraneous load is imposed by information that does not contribute directly to the performance of a specific goal. Activities such as attempting to find relevant information on the device display (visual search) or manipulating the user interface in an attempt to find relevant information (interactive search), that do not foster the process of performing a goal can be classified as extraneous. Our findings suggest that sensory cues will only be low in overall salience when both the intrinsic

[2] Note that our framework is binary in the sense that verification either proves a property or provides a counter example to it. We deal with the frequency of errors by using the concept of a threshold of error rates (Sect. 6).

and extraneous load imposed on the individual is high. These findings are compatible with Cartwright–Finch and Lavie's [CaL07] theory that a high extraneous load only reduces perception of a sensory cue (or distractor) when cognitive control functions are not available to maintain the active goal.

The informal analysis [BBC+08] of our empirical data suggested the following connections between salience and cognitive load:

sensory cueing: When both the intrinsic and extraneous load are high, the salience of sensory cues *may* be reduced.
procedural cueing: High intrinsic load reduces the salience of procedural cues.

Next we describe how these connections are formalised within our verification framework.

3.2. Formalisation

For practical reasons, our intention was to develop a formalisation as abstract and simple as possible. Since having a number of salience values would complicate choosing one as the most appropriate for a cue in a concrete scenario, we considered two basic values: `HighSLC` and `LowSLC`. Intuitively, the idea is that these are sufficient for most of the concrete user specifications derived from the experimental work by instantiating our cognitive architecture. The third value, `ReducedSLC`, is intended mostly for our generic architecture. It models the dynamic aspect of cue salience—its reduced level due to the effect of cognitive load at some stages of task execution. If needed, this dynamic aspect can be refined in the future by adding more intermediate salience values to our generic architecture. However, this will only be done when empirical findings show that more fine-grained differentiation is needed.

Further, we assume that both the intrinsic and extraneous load can be either `HighLD` or `LowLD`. Next, we chose to interpret the meaning of "reduced salience" in the above rules as salience possibly going from high to reduced. Then the sensory salience rule can be written as follows:

$$\begin{array}{l} \textit{if}\ \texttt{default} = \texttt{HighSLC} \wedge \texttt{intr} = \texttt{HighLD} \wedge \texttt{extr} = \texttt{HighLD} \\ \textit{then}\ \texttt{sensory} = \texttt{HighSLC} \vee \texttt{sensory} = \texttt{ReducedSLC} \\ \textit{else}\ \texttt{sensory} = \texttt{default} \end{array} \tag{1}$$

Here, `intr` and `extr` represent the intrinsic and extraneous load, respectively. The variable `default` denotes the salience of a sensory cue without taking into account the cognitive load experienced, whereas `sensory` denotes the actual sensory salience of that cue. Note that our formalisation is non-deterministic, i.e., we assume that a sensory cue can be salient (and thus be noticed by people) even under the high cognitive load condition. This reflects the modality *may* in the corresponding informal rule.

In the cognitive architecture, we use predicates, `SensHigh` and `SensLow`, that specify when the sensory salience of a goal is high and low, respectively. Thus, rule (1) is translated into the following SAL definitions:

```
SensHigh(sens,g,status,goals)(inp,mem,env) =
   IF status.intrinsic = HighLD ∧ extraneous = HighLD ∧
      goals[g].slc(inp,mem,env) = HighSLC
   THEN sens = HighSLC
   ELSE goals[g].slc(inp,mem,env) =HighSLC
   ENDIF

SensLow(cog:Salience, g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal) : Pred0 =
   λ(inp:Inp,mem:Memory,env:Reality): goals[g].slc(inp,mem,env) = LowSLC
```

Here, `goals[g].slc(inp,mem,env)` is the default salience (as determined in any specific case by HCI experts) of the goal `g`. The parameter `sens` represents a possible value (high or reduced) of the actual sensory salience. This value is chosen non-deterministically as an index of the guarded command *CommitAction* (Fig. 1).

The procedural salience rule can be written as follows:

$$\begin{array}{l} \textit{if}\ \texttt{default} = \texttt{HighSLC} \wedge \texttt{intr} = \texttt{HighLD} \\ \textit{then}\ \texttt{procedural} = \texttt{ReducedSLC} \\ \textit{else}\ \texttt{procedural} = \texttt{default} \end{array} \tag{2}$$

```
HighestSalience(sens,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (ProcHigh(g,status,goals)(inp,mem,env) ∨
    NOT(ProcLow(g,status,goals)(inp,mem,env)) ∧
        SensHigh(sens,g,status,goals)(inp,mem,env))

HighSalience(sens,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (ProcHigh(g,status,goals)(inp,mem,env) ∨ SensHigh(sens,g,status,goals)(inp,mem,env))

LowSalience(arb,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s)
```

Fig. 2. Levels of salience

In the cognitive architecture, this is translated into two SAL predicates, `ProcHigh` and `ProcLow`, that specify when the procedural salience is high and low, respectively:

```
ProcHigh(g,status,goals)(inp,mem,env) =
   goals[status.last].cue(g)(inp,mem,env) = HighSLC ∧ status.intrinsic = LowLD

ProcLow(g,status,goals)(inp,mem,env)
   goals[status.last].cue(g)(inp,mem,env) = LowSLC
```

3.3. Hierarchy of choices

Next we discuss how sensory and procedural salience influences the choice of goals in our cognitive architecture. Recall that the underlying principle is non-deterministic choice—any "enabled" goal can be chosen for execution. The addition of salience refines the notion of enabledness by introducing a hierarchy of choices into our cognitive architecture. We started with a version that included a two-level hierarchy: high salience and low salience. A goal was defined to have the high salience if either of the predicates `SensHigh` or `ProcHigh` was true. Otherwise, its salience was defined as low. We also assumed that high salience goals have priority over low salience ones. However, with this version of the architecture, our verification efforts identified [RBC+08] errors in the behaviours of the model that were not observed during our empirical studies. The analysis of these counter examples led to the discussions between the formal modelling and cognitive science members of our team. As a result, a refinement of the two-level hierarchy was developed. The new hierarchy is specified in Fig. 2.

The refined version consists of three levels of salience (Fig. 2). Assuming the choice strategy and the guard for an atomic goal is true, its salience belongs to the highest priority level, if:

- its procedural salience is high, or
- its procedural salience is reduced, and sensory salience is high.

It belongs to the middle level (high salience), if:

- its procedural salience is high, or
- its sensory salience is high.

Such a goal is only chosen, if there are no enabled goals in the highest level (Fig. 1). Finally, the lowest level includes all atomic goals whose choice strategy and guard are true. These goals can only be selected if there are no enabled goals in the higher priority levels.

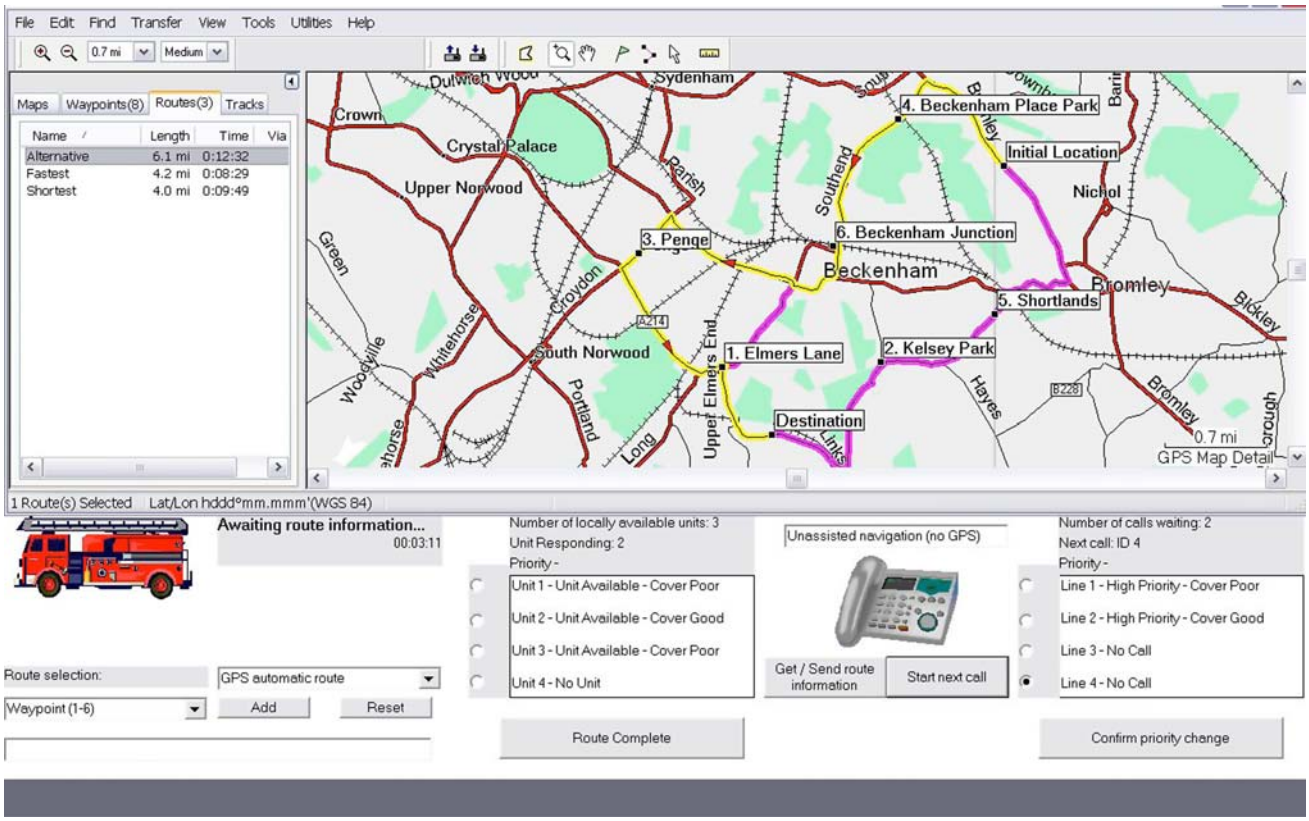

Fig. 3. 'Fire Engine Dispatch' interface

4. Fire engine dispatch task

Laboratory experiments provide well-established approaches within the cognitive sciences to further understanding of human error (e.g., [ByB97]). Although these approaches are not representative of real-world tasks, they provide valuable insight into the underlying cognitive mechanisms that individuals are reliant on. Manipulating intrinsic and extraneous load under controlled conditions allowed us to identify some important generic factors that shape human performance during procedural routine. In this section, we describe the task that was used in our laboratory work. The task is based on a simulation of a 'Fire Engine Dispatch Centre'.

The overall objective of the task was to send navigational information to fire engines enabling the fastest possible incident response times using the interface shown in Fig. 3. Training trials were used to ensure that participants became familiar with the sequence of actions. After performing two 'error free' training trials consecutively, a participant was allowed to move on to twelve experimental trials.

Two experiments using 24 participants each were run. These investigated the frequency of slip errors under different cognitive load scenarios. In total there were twelve trials: half imposed a low intrinsic cognitive load (less complex routes, fewer navigational waypoints to be considered) and the remaining half imposed a high intrinsic cognitive load (more complex routes, more navigational waypoints). Trial order was randomized. Extraneous cognitive load was manipulated between-subjects: participants in the high extraneous load condition were presented with a mixture of relevant and irrelevant information on the traffic information ticker; participants in the low extraneous load condition were only presented with relevant information.

4.1. Task scenario

When commencing the fire engine dispatch task an individual has to decide which call to prioritize before clicking on the 'Start next call' button. Choosing call priority involves clicking on the radio button that is located

alongside the required call ID (see the bottom right part of Fig. 3). However, call priority is actually set only when the 'Confirm priority change' button is clicked. Clicking on this button updates the visual confirmation of the selected call, located at the top of the priority selection window (ID 4 in Fig. 3). The selected call is then processed by clicking on the 'Start next call' button.

The second part of this task is to construct the optimal route and send the necessary information to fire engines. This is done using the bottom left part of the interface from Fig. 3 which is displayed only when a call has been processed. At this point, the location of the nearest fire engine and the location of the incident are displayed as waypoints on the map in the upper part of the interface. Depending on the availability of GPS signals, there are two options for constructing route information: automatic and manual. The most appropriate automatically generated route can only be used when GPS signals are being received by the fire engine attending the incident. When GPS signals cannot be relied upon, a route must be constructed based on waypoint information in the local area. The indicator located above the telephone image tells the operator which option must be used. The leftmost drop-down menu supports manual route construction by allowing the user to select waypoints and add them to the route by clicking on the 'Add' button. The selected waypoints are then displayed in the text box below. One of the automatically generated routes can be selected by clicking on the menu just above the 'Add' button. Selecting the wrong route construction method is regarded as a mode error.

The constructed route is sent by clicking the 'Get/Send route information' button, thus finishing the task. However, before this step is taken, a fire engine designated as the backup unit must be selected. This selection involves clicking the radio button alongside one of the units in the centrally located menu. Mirroring the call selection stage, the backup unit is only set once the 'Route complete' button has been clicked.

4.2. Errors under investigation

Our laboratory experiments focused on five potential errors during the execution of the dispatch task. Based on a pilot study, the interface of our 'Fire Engine Dispatch Centre' was intentionally designed to frequently (10% threshold) provoke three errors: initialisation, mode and termination (see below). It also presented opportunities for making two confirmation errors. In principle, other errors could have been made when completing the task. However, they did not occur during our experiments, and no cognitive mechanism suggests they should. Therefore, the five potential errors were:

initialisation: The first opportunity to make a device-specific error was classified as an initialisation error. If a participant clicked on the 'Start next call' button after starting a new trial without prioritising calls then this was recorded as an initialisation error.

confirmation 1: The next opportunity to make an error was labelled 'Confirmation Error 1'. Participants were instructed that the 'Start next call' button should only be clicked when both the new call ID has been selected and the 'Confirm priority change' button has been clicked. Forgetting to click the 'Confirm priority change' button after selecting the new call ID was recorded as 'Confirmation Error 1'.

mode: An opportunity to make a mode error occurred during the route construction stage after a new call was started. Participants were required to attend to a signal so that they could determine what type of route information was needed (the signal appeared 30–45 seconds after a new call was started). The mode error was to construct the route using the wrong mechanism. This is referred to as a mode error because it indicates that the participant is behaving as if the system were in the other mode (requiring input of the other type).

termination: A further opportunity to make a device-specific error occurred before the 'Get/Send route information' button was clicked. Procedural steps were required before clicking on this button, which terminated the trial. Clicking 'Get/Send route information' without selecting a backup unit was considered to be erroneous. This error is known as a termination error because the action omitted occurs at the end of the task, after the main task (route construction) has been completed.

confirmation 2: After selecting a backup unit participants were then required to click on the 'Route complete' button. Clicking 'Get/Send route information' without clicking this button was considered to be erroneous and was labelled 'Confirmation Error 2'.

The rates of these errors during our laboratory experiments are shown in Fig. 4. Note that both confirmation errors are not represented in the graph since their rates were negligible (less than 1%).

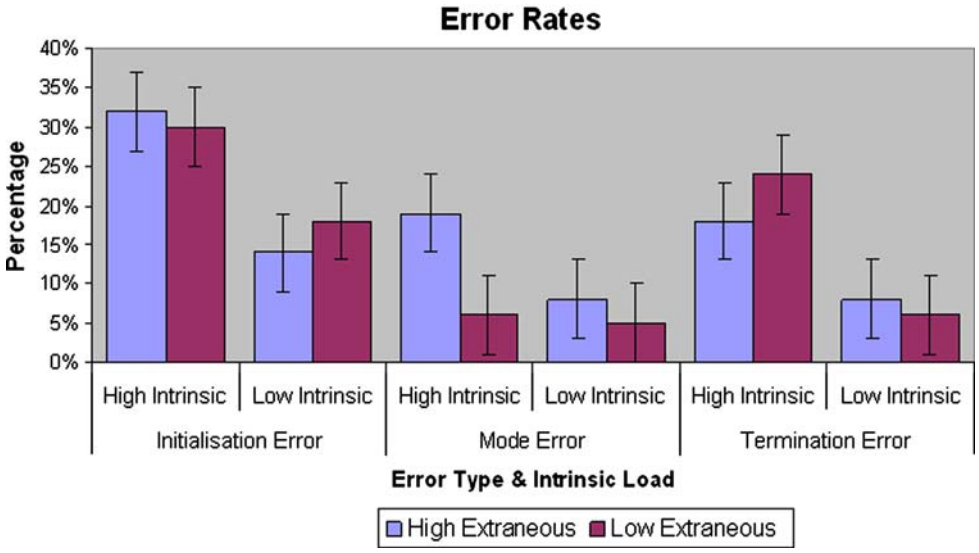

Fig. 4. Error rates in the laboratory experiments

5. Formal task modelling

We instantiated our generic architecture to give a user model for the fire engine dispatch task.[3] Using SAL, we analysed this model with respect to the errors investigated in our laboratory simulation of the same task. The comparison of the (erroneous) behaviours that emerged from the formal model with those of the human participants of the experiment allowed us to stepwise refine our formalisation of the salience and load rules.

The salience values used in the instantiation below were determined by the cognitive science specialists in our team. The suggested values stem from their analysis of the dispatch task and its interface. Since the actual experiments essentially consisted of two subtasks, we split the formal task model (and its user model) into two parts: setting call priority (top goal `PriorityGoal`) and sending route information (top goal `RouteGoal`).

5.1. Call priority

We assume that the user model for this (sub)task includes three atomic goals: `SelectPriorityGoal`, `ConfirmPriorityGoal` and `StartCallGoal`. As an example, `SelectPriorityGoal` is the following record:

```
choice := NotYetDischarged
grd := λ(inp,mem,env): inp.PrioritySelection
slc := λ(inp,mem,env): LowSLC
cue := λ(g):λ(inp,mem,env): IF g=ConfirmPriorityGoal THEN HighSLC ELSE LowSLC ENDIF
subgoals := atom(SelectPriority)
```

Thus, this goal may be selected only if the priority selection menu is displayed. The choice strategy `NotYetDischarged` is a pre-defined predicate that allows one to choose a goal only when it has not been chosen before. We assume that the sensory salience of this goal is low, since the corresponding menu choice lacks specificity, even though the visual attention presumably is in the right area at this point of task execution. The execution of `SelectPriorityGoal` cues the call confirmation goal, which therefore has high procedural salience, but no other actions are cued. The corresponding action `SelectPriority` is defined as follows:

```
tout := λ(inp,out0,mem):λ(out): out = Default WITH .PrioritySelected:=TRUE
```

Here `Default` is a record with all its fields set to false thus asserting that nothing else is done.

[3] The complete SAL sources for this example, including the SAL specification of the interface, are available at http://www.dcs.qmul.ac.uk/research/imc/hum/examples/facs-fmis07.zip .

The definitions for the other two goals are similar. Their sensory salience, however, is assumed to be high. `ConfirmPriorityGoal` serves as a procedural cue of high salience for the goal `StartCallGoal`, whereas the latter being the last step of this subtask does not cue other actions. Finally, the top goal `PriorityGoal` for the call priority subtask is defined as follows:

```
load := intrinsic
achieved := λ(inp,mem,env): inp.WaitMsg
subgoals := comp({SelectPriorityGoal, ConfirmPriorityGoal, StartCallGoal})
```

It includes all three atomic goals as its subgoals. The predicate `achieved` defines the perceived goal of the task. The latter is regarded as achieved when a wait message is displayed by the interface. This only happens once the processing of the selected call has been started. Note that the interface specification ensures that this state can never be reached, if the priority setting procedure was not properly (as described in Sect. 4) executed.

The variable `intrinsic` is a parameter of our user model. It denotes the intrinsic load associated with the subtask execution. The user model derived by the above instantiation of the generic architecture is also parametric with respect to the extraneous load. Both parameters can be manipulated in different verification runs, in a similar way to that in the empirical studies. Our immediate aim in using this user model is to check for those errors that were investigated during our laboratory work. To do that one has to define the corresponding correctness assertions in SAL.

Correctness properties: As explained in Sect. 4, all five potential errors are related in some way to the wrong sequencing of actions. Therefore, a natural choice of specifying the corresponding properties is to use the LTL operator "before", denoted `B` in SAL. Its meaning is as follows: `B(p,q)` is true if and only if, for any behaviour, `p` becomes true before `q` or `q` remains false forever.

Two potential errors are related to the subtask of setting call priority. The first one, the initialisation error, could occur when the 'Start next call' button is pressed (`out.StartCallPressed`) without prioritising calls. During model checking, this can be detected as a violation of the correct sequencing of those actions, as specified by the following SAL assertion:

```
B (out.PrioritySelected, out.StartCallPressed)                    (3)
```

It states that a call deemed to be of the highest priority is chosen (`out. PrioritySelected`) before the 'Start next call' button is pressed.

The second candidate, the confirmation error, could occur when, after selecting a call, the 'Start next call' button is pressed BEFORE the 'Confirm priority change' button. Here, the correct sequencing of actions is specified as the following SAL assertion:

```
B (out.PrioritySelected, out.StartCallPressed)
        ⇒ B (out.ConfirmPriorityPressed, out.StartCallPressed)    (4)
```

Note that those behaviours where 'Start next call' is pressed before prioritising calls are excluded from consideration in this assertion, since they are flagged by verifying property (3). Next we instantiate the user model for the second subtask.

5.2. Sending route information

We assume that the user model for this (sub)task includes the following atomic goals: `ObserveModeGoal`, `GPSGoal`, `ManualGoal`, `AddWaypointsGoal`, `SelectBackupGoal`, `ConfirmRouteGoal` and `SendRouteGoal`. For the goal of observing the mode indicator (`ObserveModeGoal`), the essential components are specified as follows:

```
grd := λ(inp,mem,env): inp.ModeDisplayed ≠ NoMode
slc := λ(inp,mem,env): LowSLC
cue := λ(g):λ(inp,mem,env): IF g=GPSGoal ∨ g=ManualGoal THEN HighSLC ELSE LowSLC ENDIF
```

Here `ModeDisplayed` denotes the required mode for route construction. It can take one of the following three values: `ModeGPS`, `ModeManual` and `NoMode`. This goal can only be selected when the mode indicator is displayed and shows the required mode. The sensory salience of `ObserveModeGoal` is low, since the mode indicator is

displayed in a different area to the route construction menus. Observing the mode indicator is a highly salient procedural cue for both automatic and manual route construction goals, `ModeGPS` and `ManualGoal`, respectively. Finally, the memory update for the corresponding `ObserveModeAction` specifies that the indicated mode of route construction is stored in memory:

```
tmem := λ(inp,mem0,out):λ(mem): mem = mem0 WITH .mode := inp.ModeDisplayed
```

Similarly, the essential components for `SelectBackupGoal` are defined as follows:

```
grd := λ(inp,mem,env): inp.BackupSelection
slc := λ(inp,mem,env): LowSLC
cue := λ(g):λ(inp,mem,env): IF g = ConfirmRouteGoal THEN HighSLC ELSE LowSLC ENDIF
```

Note that selecting a backup unit is a highly salient procedural cue for clicking the 'Route complete' button (`ConfirmRouteGoal`).

Finally, we have the following definitions for the top goal `RouteGoal` of the subtask of sending route information:

```
load := intrinsic
achieved := λ(inp,mem,env): inp.SuccessMsg
subgoals := comp({ObserveModeGoal, GPSGoal, ManualGoal, AddWaypointsGoal,
                  SelectBackupGoal, ConfirmRouteGoal, SendRouteGoal})
```

The perceived task goal is to send route information. Achieving this goal is indicated by a success message displayed by the interface. As previously, the derived model is parametric with respect to both the intrinsic and the extraneous load.

Correctness properties: There are three error candidates relevant to the subtask of sending route information. The corresponding SAL assertions about the correct sequencing of actions are as follows:

$$\mathrm{B}\,((\texttt{mem.mode} \neq \texttt{NoMode}) \wedge (\texttt{mem.mode} = \texttt{DisplayMode}),\ \texttt{out.GPSselected} \vee \texttt{out.WaypointsAdded}) \tag{5}$$

$$\mathrm{B}\,(\texttt{out.BackupSelected},\ \texttt{out.SendRoutePressed}) \tag{6}$$

$$\mathrm{B}\,(\texttt{out.BackupSelected},\ \texttt{out.SendRoutePressed}) \Rightarrow \mathrm{B}\,(\texttt{out.ConfirmRoutePressed},\ \texttt{out.SendRoutePressed}) \tag{7}$$

The first assertion is relevant to the mode error. It states that the user model attends to the mode indicator and registers the value displayed there before constructing a route (`out.GPSselected` or `out.WaypointsAdded`). The second assertion is relevant to the termination error. It states that a backup unit is selected (`out.BackupSelected`) before the 'Get/send route' button is pressed (`out.SendRoutePressed`). Finally, the third assertion is relevant to the (second) confirmation error. It states that the 'Route complete' button is pressed (`out.ConfirmRoutePressed`) before 'Get/send route' (`out.SendRoutePressed`). Note that the behaviours flagged as the termination error are excluded by this assertion.

6. Formal analysis and empirical data

In this section, we juxtapose the outcomes of our formal analysis with the behavioural data from the simulation of our 'Fire Engine Dispatch Centre'. To be able to do this, one has to relate the statistical results (error rates in Fig. 4) of our experiments and the binary outcome (true or false) of model checking.

6.1. Methodological issues

In traditional software verification, the goal is to detect all possible programming errors where a program does not satisfy required properties and modify software so that they are eliminated. This, at least in principle, is

feasible, since programs can be formalised and modified when such bugs are found. The situation is different with interactive systems, where one of the actors is a human being. The behaviour of a person cannot be fully captured formally. It also cannot be modified to the extent that all error occurrences are certainly ruled out. The kind of slip errors we focus on cannot be eliminated just by better training. Furthermore, except for most trivial interactive systems and tasks, modifications to the device part of a system cannot completely prevent users from taking actions that make a task goal unachievable in the end. The combination of these factors means that erroneous behaviours are inevitable for any realistic interactive system.

Therefore, the ultimate goal of our verification approach is to focus on errors with discernible cognitive causes, which occur in the context of rational user behaviour, in other words—*systematic errors*. Redesign in these cases can address the causes and so eliminate the potential in the system for the error to be made. As a step towards our goal, we develop an abstract model of such cognitive factors as salience and load in this paper. We assess the success of our model by comparing its predictions with our experimental results. Note that the verification of the 'Fire Engine Dispatch Centre' per se is not a matter of concern in this paper. Rather, we focus on constructing a formal model that yields behaviours correlating with our empirical results. Further, a focus of this paper is to explore how such formal analysis can provide input to the experimental research programme. The immediate point here is then not to use model checking to get absolute statements about correctness of a specific system but to improve understanding about the limits of empirical results and prompt further investigation.

An erroneous action can be taken both due to discernible cognitive causes and as a result of irrational, essentially random, behaviour. Usually, there is no completely accurate way of determining which of these caused a particular action. An approximate way of doing that is to correlate discernible cognitive causes with the frequency of error occurrence. In experimental work, the notion of a threshold is widely used to decide whether some phenomenon is systematic. We apply the same idea here by setting the threshold at around 10%. The behaviours observed in our experiments with frequencies higher than this value are considered as systematic. If they are erroneous, we expect the corresponding behaviours to emerge during formal verification as counter examples to the properties checked. On the other hand, if an erroneous behaviour has a frequency lower than the threshold, it is not considered as a systematic error and, consequently, it should not be flagged during model checking.

It is important to realise that our salience rules were derived from empirical results based on their significance rather than percentages, i.e., they represent cases where an error is significantly more likely to happen rather than percentages per se. We use a high threshold (at 10%) in this verification case study, since the task structure in our experiments was highly error prone. In the case of less error prone interactions, a lower error rate could be statistically significant. This would require a lower threshold to be set (e.g., at 5%). What we really need is ways of developing an individual's baseline measure for comparison but this has not been attempted in laboratory research yet.

Naturally, setting the threshold at any value above zero raises the issue of user errors in safety-critical systems. Our view is that a distinction must be made between the liveness (something useful can be done) and safety (nothing wrong happens) properties in this case. For liveness properties, the verification of interactive systems where any user error is flagged would be meaningless. Any non-trivial liveness property will be falsified, if completely arbitrary or malicious actions are included into a user model. Instead, our approach focuses on detecting errors with discernible cognitive causes. While this leaves out some errors, it allows one to detect systematic, and much more frequent, ones at an early design stage, replacing to an extent laboratory usability studies and saving costs and efforts involved in setting them up. Given that the need for design trade-offs in practice means that not all human error issues can be designed away, it is important that those that are systematic are the ones flagged. If they are not dealt with, then it can be predicted that someone will actually make the error, because of the known causes behind them.

On the other hand, when safety is a primary concern and the corresponding correctness properties must hold for all system behaviours, there is no need for user modelling at all. The device part of such systems must ensure that safety properties are satisfied in an environment behaving arbitrarily. However, standard software verification methods can be applied in such cases.

Our approach is not intended for making predictions about particular error rates, based on the falsification of a correctness property. We do not develop quantitative models that, for example model percentages/probabilities explicitly because our research has shown that, given our specific aims, threshold models work successfully as the basis of an analysis method to detect various common systematic errors. Part of our research programme is explicitly concerned with investigating how much can be done using a basic threshold model. So far it has worked well.

Table 2. Comparison results for the original cognitive architecture [RBC+08]

Extraneous	intrinsic	initialisation	confirmation 1	mode	termination	confirmation 2
Low	Low	✗	✓	+	✓	+
Low	High	✗	✓	+	−	+
High	Low	✗	✓	+	✓	+
High	High	✗	+	✗	✗	+

False positives are indicated by +, whereas missed errors by −. Two marks are used to indicate that verification outcomes and empirical data match: ✗ for error occurrence, ✓ for its absence

6.2. Comparison of results

SAL specification language is supported by a number of tools for system analysis. We used a symbolic (BDD-based) model checker to verify correctness properties (3)–(7). In addition, a simulator and deadlock checker were also used to analyse our specifications. In the early prototype models as described here the model checking was slow taking up to 300 seconds for a property in some cases. Subsequent experiments modifying the way the models are written (e.g., rewriting them so that quantifiers are not used) has shown this can be brought down so that the verification of any of the properties takes less than 10 seconds. The models as described here should therefore be seen as a proof of principle. It appears that reasonable computation times can be achieved with careful engineering of our generic cognitive architecture.

The outcomes of our original analysis efforts [RBC+08] for the dispatch task revealed some inconsistencies with the empirical data. In that paper, we focused on the three errors—initialisation, mode and termination—found to be systematic in our laboratory work. Here, we also analyse the correctness properties related to two other error candidates—confirmation errors in both subtasks—found to be non-systematic in the experiments.

Table 2 summarizes and compares the results of formal analysis as regards our experiments. It shows that, with respect to the initialisation error, the original user model produced behaviours that are fully consistent with the human behaviour observed during the experimental studies. On the other hand, the formal verification yielded false positives for three types of error: a single case (when both loads are high) for the first confirmation error, three load conditions for the mode error and all load conditions for the second confirmation error. Finally, it missed the termination error, systematic when the intrinsic load is high and the extraneous load is low.

The false positives produced by our analysis indicate problems that were not seen in the specific experiment when this task was performed by humans. One possible explanation for this mismatch is that our judgement about salience values for some goals was imprecise. An important factor contributing to the overall sensory salience of a cue is its *specificity*, i.e., how clearly and timely that cue indicates the action required. Hollnagel [Hol93] (p. 299) explains that the strength of a cue is relative to its specificity, and thus it is the relative not absolute strength that matters. When a task is considered routine, attention is more easily diverted. Performance becomes controlled by more error-prone generic functions such as 'look for cue which indicates a turn', rather than exact intentions such as 'look for cue-X, then turn to the right.' Therefore, the strength of a cue might be relative to its specificity rather than where visual attention is directed.

In fact, as the main factor in determining the sensory salience values for our original user model [RBC+08], we considered the positions of the relevant buttons. This was compared with the area visual attention was likely to be focussed on at that point in the execution of the task. Based on this, high sensory salience was assigned to the goals `SelectPriorityGoal`, `GPSGoal`, `ManualGoal`, `AddWaypointsGoal` and `SelectBackupGoal`. In this paper, the instantiation of the user model (Sect. 5) takes into account the specificity of the corresponding menu choices as well. As a result, the sensory salience values for the above goals were changed to `LowSLC`.

Table 3 shows that, in all cases but the termination error, our formal analysis using the new salience values still yields exactly the same results as with the original model. On the plus side, the false positive for the termination error disappears. The observed mismatches hint that their most likely explanation is that our salience and cognitive load rules are simply too coarse. In the subsequent sections, we focus on their refinement.

7. Formal modelling of cognitive cueing

The cognitive architecture we described and used so far encapsulates procedural cueing—where the performance of an action acts as an associative cue for the next action, and sensory cueing—cues built-in to the device in an

Table 3. Comparison results for the original architecture instantiated with the modified salience values

Extraneous	intrinsic	initialisation	confirmation 1	mode	termination	confirmation 2
Low	Low	✗	✓	+	✓	+
Low	High	✗	✓	+	✗	+
High	Low	✗	✓	+	✓	+
High	High	✗	+	✗	✗	+

False positives are indicated by +, whereas missed errors by –. Two marks are used to indicate that verification outcomes and empirical data match: ✗ for error occurence, ✓ for its absence. Those results that are different to our previous analysis are framed

attempt to guide the interaction. Some actions, however, seem to "spring to mind" for the performance of a task (especially if they move a user towards a goal state), whereas others do not, and the latter are much more likely than the former to feature in erroneous actions.

Our attempts to overcome the inconsistencies described in Sect. 6 led us to further experiments using the fire engine paradigm. The results of these experiments suggests the need for a third cueing mechanism—cognitive cueing—that is able to represent a knowledge-based activity enabling an appropriate action to be selected. Evidence for knowledge based cognitive cueing is based on the idea that users are sometimes reliant on 'bottom-up' cues from the environment (that correspond to intentions) when planning future actions (e.g. Payne [Pay91]). When performing a familiar interactive routine, intentions can be formulated well before the opportunity to execute the procedural steps that allow that intention to be communicated. When environmental features suggest that an intention can be communicated those features become cognitively salient.

Cognitive cues can direct interaction by identifying actions that match a user's knowledge of what they have to do to reach their task goal. A post-hoc analysis, by the cognitive science specialists in our team, of the empirical data from the 'Fire Engine Dispatch Centre' simulation identified instances where cognitive cueing was responsible for directing the interaction and the following informal rule was suggested:

cognitive cueing: When both the intrinsic and extraneous load is high, the salience of cognitive cues *may* be reduced.

As with sensory salience, the reduction is interpreted in our formalisation as cognitive cueing possibly going from high to reduced. In the cognitive architecture, the levels of cognitive cueing are specified using the following two predicates:

```
CogHigh(cog,g,status,goals)(inp,mem,env) =
   g ∈ gls(goals[s.active].subgoals) ∧
   IF goals[s.active].cue(g)(inp,mem,env) = HighSLC ∧
      status.intrinsic = HighLD ∧ extraneous = HighLD
   THEN cog = HighSLC
   ELSE goals[s.active].cue(g)(inp,mem,env) = HighSLC
   ENDIF

CogLow(g,status,goals)(inp,mem,env) =
   NOT(g∈gls(goals[active].subgoals))
```

Here, `gls(goals[s.active].subgoals)` is the subgoal set of the currently active goal (subtask) `s.active`. The parameter `cog` represents a possible level (high or reduced) of the actual cognitive cueing. This value is chosen non-deterministically as an index of the guarded command *CommitAction* (Fig. 1). In the predicate `CogHigh`, it is used instead of the default level (`goals[s.active].cue(g)`) of cueing for `g` when both loads, intrinsic and extraneous, are high. The predicate `CogLow` states that there is no cognitive cueing of `g` at all, when `g` is not a subgoal of the active subtask.

The hierarchy of salience levels from Fig. 2 is modified by using the predicates `CogHigh` and `CogLow` as shown in Fig. 5. The predicate `HighestSalience` now reflects our assumption that high cognitive cueing, in the same way as sensory cueing, may enhance the procedural salience (reduced due to the high intrinsic load), yielding thus the highest level in our hierarchy. On the other hand, high cognitive cueing alone is sufficient to guaranty the second level of overall salience (predicate `HighSalience`). Finally, the modified predicate `LowSalience` effectively introduces the fourth salience level. Now the goals with the reduced level of procedural, sensory or cognitive cueing are assigned to the third salience level (predicate `LowSalience`), whereas goals with all the cues being low are relegated to the fourth level.

```
HighestSalience(sens,cog,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (ProcHigh(g,status,goals)(inp,mem,env) ∨
    NOT(ProcLow(g,status,goals)(inp,mem,env)) ∧
        (SensHigh(sens,g,status,goals)(inp,mem,env) ∨
         CogHigh(cog,g,status,goals)(inp,mem,env)))

HighSalience(sens,cog,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (ProcHigh(g,status,goals)(inp,mem,env) ∨
    SensHigh(sens,g,status,goals)(inp,mem,env) ∨
    CogHigh(cog,g,status,goals)(inp,mem,env))

LowSalience(sens,cog,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (NOT(ProcLow(g,status,goals)(inp,mem,env)) ∨
    NOT(SensLow(sens,g,status,goals)(inp,mem,env)) ∨
    NOT(CogLow(cog,g,status,goals)(inp,mem,env)))
```

Fig. 5. Modified hierarchy of salience levels

For the user models derived in Sect. 5, the only modifications required are changes to the definitions of their top goals `PriorityGoal` and `RouteGoal`. The `cue` field of these goals is now used to specify the strength of their subgoals as cognitive cues. The definition of this field for `PriorityGoal` is as follows:

```
cue := λ(g):λ(inp,mem,env): IF g=StartCallGoal THEN HighSLC ELSE LowSLC ENDIF
```

Thus only the goal of starting the next call has high salience as a cognitive cue. All other goals are low cognitive cues.

The corresponding definition for `RouteGoal` is as follows:

```
cue :=λ(g):λ(inp,mem,env):
          IF g=ObserveModeGoal ∧
             NOT(mem.GPSselected) ∧ NOT(mem.WaypointsAdded) ∧
             NOT(mem.ConfirmRoutePressed) ∧ NOT(mem.SendRoutePressed)
          ∨  g=GPSGoal ∧
             NOT(mem.WaypointsAdded) ∧
             NOT(mem.ConfirmRoutePressed) ∧ NOT(mem.SendRoutePressed)
          ∨  g=ManualGoal ∧
             NOT(mem. GPSselected) ∧
             NOT(mem.ConfirmRoutePressed) ∧ NOT(mem.SendRoutePressed)
          ∨  g=SendRouteGoal ∧
             NOT(mem.SendRoutePressed)
          THEN HighSLC
          ELSE LowSLC
          ENDIF
```

In this case, the goals associated with observing the mode indicator, constructing a route and sending route information have high salience as cognitive cues. Note, however, that this is so only when they are specific enough, as specified above by the related conditions. For example, the condition for the goal of observing the mode indicator is that none of the four goals from the later stages of this task, `GPSGoal`, `ManualGoal`, `ConfirmRouteGoal` or `SendRouteGoal`, has been discharged. The assumption here is that executing one of those goals may suggest to the user that the task stage for checking the mode has already been passed, making the cognitive salience of `ObserveModeGoal` low in such cases.

Table 4. Comparison results for the refined architecture from Sect. 7

Extraneous	intrinsic	initialisation	confirmation 1	mode	termination	confirmation 2
Low	Low	✗	✓	✓	✓	✓
Low	High	✗	✓	✓	✗	+
High	Low	✗	✓	✓	✓	✓
High	High	✗	+	✗	✗	+

False positives are indicated by +, whereas missed errors by −. Two marks are used to indicate that verification outcomes and empirical data match: ✗ for error occurrence, ✓ for its absence. Those results that are different to our previous analysis are framed

Table 4 shows that these refinements to our cognitive architecture solve several earlier issues. Namely, for the mode error, the false positives under three load conditions are no longer observed in our formal analysis based on the refined model. The verification outcomes and empirical data are now fully consistent for this error. Also, the two false positives under the low intrinsic load are no longer observed for the confirmation error in the construction subtask. The only remaining inconsistencies between the verification and empirical results are the three false positives still observed for the confirmation errors. In the next section, we consider further refinements to our salience and load rules to address these inconsistencies.

8. Interaction between procedural and sensory cues

All three false positives still detected by the formal analysis have to do with the confirmation errors. They manifest themselves as follows. After setting call priority (selecting the back-up unit), the user model, instead of confirming the previous action and at variance with the human participants of our experiments, starts a new call (sends route information). Since both confirmation actions mostly rely on procedural cueing, our next refinements deal with this aspect of salience.

8.1. Strength of effect

One possible explanation of the observed inconsistencies is that our formal model does not sufficiently take into account the strength of the effect of procedural cueing in relation to other types of cues. In particular, the goals with the reduced procedural salience (`ReducedSLC`) are assigned only to the third salience level (`LowSalience`), unless their salience is enhanced by the high sensory or cognitive cueing. The next refinement to our salience rules is therefore to increase the effect of the reduced procedural cueing. To achieve this the hierarchy of salience levels from Fig. 5 is modified by defining the predicate `HighSalience` as follows:

```
HighSalience(sens,cog,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (NOT(ProcLow(g,status,goals)(inp,mem,env)) ∨
    SensHigh(sens,g,status,goals)(inp,mem,env) ∨
    CogHigh(cog,g,status,goals)(inp,mem,env))
```

The modification effectively raises the goals with the reduced procedural cueing from the third level (`LowSalience`) to the second level of overall salience. Unfortunately, this refinement removes only one of the above inconsistencies (see Table 5). The formal analysis based on the modified cognitive architecture still produces the other two false positives. Next we consider another likely cause of the inconsistencies observed.

8.2. Effect on sensory salience

Procedural and sensory cues do not act independently [Pay91]. For example, sensory cues can help to retrieve previously formulated procedural knowledge. Our cognitive architecture already captures this aspect of their interaction by increasing the overall salience level for those goals with the reduced procedural salience that are highly cued either sensorily or cognitively (predicate `HighestSalience`). The converse effect is also possible.

Table 5. Comparison results for the refined architecture from Sect. 8.1

Extraneous	intrinsic	initialisation	confirmation 1	mode	termination	confirmation 2
Low	Low	✗	✓	✓	✓	✓
Low	High	✗	✓	✓	✗	✓
High	Low	✗	✓	✓	✓	✓
High	High	✗	+	✗	✗	+

False positives are indicated by +, whereas missed errors by −. Two marks are used to indicate that verification outcomes and empirical data match: ✗ for error occurrence, ✓ for its absence. Those results that are different to our previous analysis are framed

Table 6. Comparison results for the refined architecture from Sect. 8.2

Extraneous	intrinsic	initialisation	confirmation 1	mode	termination	confirmation 2
Low	Low	✗	✓	✓	✓	✓
Low	High	✗	✓	✓	✗	✓
High	Low	✗	✓	✓	✓	✓
High	High	✗	✓	✗	✗	✓

False positives are indicated by +, whereas missed errors by −. Two marks are used to indicate that verification outcomes and empirical data match: ✗ for error occurrence, ✓ for its absence. Those results that are different to our previous analysis are framed

Human memory relies mostly on association, thus objects frequently seen together become linked. For example, a "green light" (sensory cue) means "go." Returning to the photocopier example presented in Sect. 1—Action x (e.g. specifying the number of photocopies) becomes a procedural cue for action y (i.e. pressing the start button). However, if the start button incorporates a sensory cue (i.e. "red light" means "stop") then this cue is likely to be noticed even if the load placed on an individual is high.

This effect of procedural cueing on sensory salience has in no way been reflected in the cognitive architecture described thus far. Furthermore, the fact that the two false positives are observed under conditions of high intrinsic and extraneous load only may hint that our formal rules underestimate the level of sensory salience in these situations. A likely explanation to this is that procedural cues actually counteract the effect of the high cognitive load, maintaining the same level of sensory salience. These considerations led us to an additional refinement of our model – the following modification of the original sensory salience rule:

$$\begin{array}{l} \textit{if } \texttt{default} = \texttt{HighSLC} \wedge \texttt{intr} = \texttt{HighLD} \wedge \texttt{extr} = \texttt{HighLD} \wedge \texttt{procedural} = \texttt{LowSLC} \\ \textit{then } \texttt{sensory} = \texttt{HighSLC} \vee \texttt{sensory} = \texttt{ReducedSLC} \\ \textit{else } \texttt{sensory} = \texttt{default} \end{array} \tag{8}$$

Intuitively, the new version of this rule states that, in the presence of the procedural cueing (either high or reduced), the sensory salience is not affected even under the high intrinsic and extraneous load. Rule (8) translates into the following SAL definition of `SensHigh`:

```
SensHigh(sens,g,status,goals)(inp,mem,env) =
   IF status.intrinsic = HighLD ∧ extraneous = HighLD ∧
      goals[g].slc(inp,mem,env) = HighSLC ∧ ProcLow(g,status,goals)(inp,mem,env)
   THEN sens = HighSLC
   ELSE goals[g].slc(inp,mem,env) = HighSLC
   ENDIF
```

Note that this refinement alone (in place of the first one) would not be sufficient. It eliminates one mismatch ('Confirmation Error 1') but leaves the other two ('Confirmation Error 2'). Also, neither of these two refinements is by itself sufficient to remove the false positive for the second confirmation error when both loads are high. However, the combination of both finally eliminates the two remaining false positives. As Table 6 shows, the behaviours generated by the cognitive architecture are now fully consistent with those of the human participants with respect to all five errors investigated.

9. Conclusion

In this paper, we described an investigation into the formal modelling of the concept of salience, based on our cognitive architecture. We formalised the connection between salience and cognitive load imposed by the complexity of the task performed. We also refined the underlying principle of non-deterministic choice of goals by introducing into our cognitive architecture a hierarchy of choices governed by the salience of goals. As a way to assess these developments, we undertook the formal modelling of our 'Fire Engine Dispatch Centre' used in our empirical studies. The goal was to check the consistency between the behaviours generated by our cognitive architecture and those exhibited by human participants of our experiments.

We started our formal modelling of salience by considering two types of cues—procedural and sensory. Formal analysis of the relevant properties using the original version of the architecture revealed several discrepancies between the verification outcomes and empirical data. Our examination of these discrepancies suggested that, for sensory cueing, the specificity of cues is as important as the location of the corresponding menu choices. The analysis also led us to a conceptualisation of the third type of cueing mechanism–cognitive cueing. We then refined our cognitive architecture by adding this concept and the relevant formal rules.

Our new verification efforts showed a lesser number of discrepancies. Their further analysis suggested new refinements to our formal models. The sensory salience rule was modified to capture the effect of procedural cueing on sensory salience. We also made changes to the hierarchy of salience levels by increasing the effect of the reduced procedural salience on overall salience level. These refinements finally yielded a cognitive architecture that generated behaviours fully consistent with those of the participants of our experiments as regards the five errors investigated. Note, however, that consistency does not necessarily mean the straightforward equivalence between error occurrence and the falsity of the corresponding correctness assertion in our approach. Error rates above the threshold (systematic error) indeed correspond to a correctness assertion that is false. At the same time, error rates below the threshold (non-systematic error) correspond to a true correctness assertion.

The most immediate result of these developments is an extension of our cognitive architecture and verification framework yielding more accurate results in the formal analysis of interactive systems using our approach. We are, however, apparently refining the model based on empirical data from one experiment. On its own this would not give great confidence, as the experiment may not provide representative data. However, these developments do rely on the results that have been obtained by manipulation of factors (cognitive and perceptual load) that are known to apply in various domains [Wic02, HaG87, LeT93]. This gives some added weight to our belief that our rules are generic. The refined version of the cognitive architecture does need further empirical validation however. This could be achieved by formally modelling a different interactive system and making predictions based on the outcomes of formal analysis. These predictions could then be tested by conducting experiments based on the simulation of that system.

Irrespective of whether the refined model is generalisable, another result (of no less importance) of our formal developments is a deeper understanding of the empirical data concerning such cognitive phenomena as salience and load. This advance was facilitated by the verification-guided iterative refinements of formal models. They allowed our team to probe the experimental results more deeply than would otherwise have been possible. It should also be stressed that these refinements were not tweaks simply aimed at producing a perfect match between the verification and empirical data. They followed from the discussions between the cognitive science and formal modelling specialists in our team and relied on the cognitive mechanisms and factors known from the literature.

The initial formalisation of the concept of salience and its dependence on cognitive load was based on empirical data. The subsequent formal development of salience and load rules however raised questions and suggest a number of new experimental hypotheses:

salience hierarchy: A new experimental paradigm needs to be designed to justify the notion of a salience hierarchy. We hypothesize that participants executing a routine procedure will make significantly less errors than participants executing a non-routine procedure (where procedural cues are absent), regardless of whether they possess a high or low level knowledge of the domain. There are three 'types of effects' that need to be tested: main effect for procedural cues absent condition (A), main effect for cognitive cues absent condition (B), and effect for the interaction of A and B.

sensory salience: Manipulation of sensory cue location to an area that is more likely to be the focus of attention will increase sensory salience and lower associated slip error rates. We hypothesize that the sensory salience of an action only captures attention if the semantic meaning of the object has been encoded or the procedure has been learnt by following spatial mappings.

cognitive cueing: We hypothesize that cognitive cueing is sensitive to both intrinsic and extraneous load providing that the semantic meaning of objects are interpreted correctly.

The work described here provides a good example of the cyclic nature of our interdisciplinary research methodology based on the mutual benefits of bridging cognitive science and formal methods in computer science.

Acknowledgments

This research has been funded by EPSRC grants GR/S67494/01, GR/S67500/01, EP/F02309X/1 and GR/S73723/01.

Appendix

A. The formal cognitive architecture

In this appendix, we give the formal SAL specifications of the final version of our cognitive architecture described in this paper. It is specified as the following SAL context:

```
User {
       Inp, Out, Memory, Rlt, Pos, GoalRange, ActionRange, GoalStateRange: TYPE;
       CogOverhead, npos, MAXtrace: NATURAL;
       extr: BOOLEAN
     }
```

A.1. Types

```
TraceRange: TYPE = [0..MAXtrace];
PossesRange: TYPE = [1..npos];
GoalSets: CONTEXT = sets{GoalRange};
ActionSets: CONTEXT = sets{ActionRange};
GoalStateSets: CONTEXT = sets{GoalStateRange};

Finish: TYPE ={ notf, abort, ok };
Commit: TYPE = { ready, committed };
Salience: TYPE = { HighSLC, ReducedSLC, LowSLC };
Load: TYPE = { LowLD, HighLD };
Reality: TYPE = [# rlt:Rlt, pos:Pos #];
Switch: TYPE = { usr, mach };

State0: TYPE = [Inp, Memory, Reality];
Pred0: TYPE = [[Inp, Memory, Reality] -> BOOLEAN];
PredOut: TYPE = [Out -> BOOLEAN];
PredM: TYPE = [Memory -> BOOLEAN];
PredR: TYPE = [Rlt -> BOOLEAN];

RelOut: TYPE = [[Inp, Out, Memory] -> PredOut];
RelM: TYPE = [[Inp, Memory, Out] -> PredM];
RelR: TYPE = [[Inp, Memory, Reality] -> PredR];

LiftGoalRange: TYPE = DATATYPE
                         none,
                         lift(goal:GoalRange)
                      END;
```

```
Status: TYPE = [# active:GoalRange,
                  trace:ARRAY GoalRange OF BOOLEAN,
                  last:GoalRange,
                  size:TraceRange,
                  intrinsic:Load
               #];

Action: TYPE = [# tout:RelOut, tmem:RelM, trlt:RelR,
                  time:[BOOLEAN->NATURAL]
               #];

Subgoals: TYPE = DATATYPE
                    comp(gls:GoalSets!Set),
                    atom(act:ActionRange)
                 END;

Goal: TYPE = [# choice: [[GoalRange,Status] -> BOOLEAN],
                slc: [State0 -> Salience],
                relevant: GoalStateSets!Set,
                grd: Pred0,
                cue: [GoalRange -> [State0->Salience]],
                achieved: Pred0,
                time: NATURAL,
                load: Load,
                subgoals: Subgoals
             #];

Possession: TYPE = [# acquire: PredOut,
                      giveup: PredOut,
                      count: [Pos -> NATURAL],
                      value: [Rlt -> NATURAL]
                   #];
```

A.2. Constants, predicates and functions

```
extraneous: Load =
     IF extr THEN HighLD ELSE LowLD ENDIF;

NotYetDischarged: [[GoalRange,Status] -> BOOLEAN] =
     λ(g:GoalRange,s:Status): NOT(s.trace[g]);

Possible: [[GoalRange,Status] -> BOOLEAN] =
     λ(g:GoalRange,s:Status): TRUE;

MakeGoal(GoalStates:ARRAY GoalStateRange OF Pred0, TaskGoals:GoalStateSets!Set): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   ∀(gs:GoalStateRange): GoalStateSets!elem?(gs,TaskGoals) ⇒ GoalStates[gs](inp,mem,env);

ProcHigh(g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality): =
   goals[s.last].cue(g)(inp,mem,env) = HighSLC ∧ s.intrinsic = LowLD;
```

```
ProcLow(g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   goals[s.last].cue(g)(inp,mem,env) = LowSLC;

SensHigh(sens:Salience, g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   IF s.intrinsic = HighLD ∧ extraneous = HighLD ∧
      goals[g].slc(inp,mem,env) = HighSLC ∧ ProcLow(g,s,goals)(inp,mem,env)
   THEN sens = HighSLC
   ELSE goals[g].slc(inp,mem,env) = HighSLC
   ENDIF;

SensLow(cog:Salience, g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   goals[g].slc(inp,mem,env) = LowSLC;

CogHigh(cog:Salience, g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   g ∈ gls(goals[s.active].subgoals) ∧
   IF goals[s.active].cue(g)(inp,mem,env) = HighSLC ∧
      s.intrinsic = HighLD ∧ extraneous = HighLD
   THEN cog = HighSLC
   ELSE goals[s.active].cue(g)(inp,mem,env) = HighSLC
   ENDIF;

CogLow(cog:Salience, g:GoalRange, s:Status, goals:ARRAY GoalRange OF Goal): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality): NOT(g∈gls(goals[s.active].subgoals));

HighestSalience(sens:Salience, cog:Salience, g:GoalRange, s:Status,
                goals:ARRAY GoalRange OF Goal, TaskGoals:GoalStateSets!Set): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (ProcHigh(g,s,goals)(inp,mem,env) ∨
    NOT(ProcLow(g,s,goals)(inp,mem,env)) ∧
        (SensHigh(sens,g,s,goals)(inp,mem,env) ∨
         CogHigh(cog,g,s,goals)(inp,mem,env))) ∧
   (∃(gs:GoalStateRange): gs∈TaskGoals ∧ gs∈goals[g].relevant);

HighSalience(sens:Salience, cog:Salience, g:GoalRange, s:Status,
             goals:ARRAY GoalRange OF Goal, TaskGoals:GoalStateSets!Set): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (NOT(ProcLow(g,s,goals)(inp,mem,env)) ∨
    SensHigh(sens,g,s,goals)(inp,mem,env) ∨
    CogHigh(cog,g,s,goals)(inp,mem,env)) ∧
   (∃(gs:GoalStateRange): gs∈TaskGoals ∧ gs∈goals[g].relevant);
```

```
LowSalience(sens:Salience, cog:Salience, g:GoalRange, s:Status,
            goals:ARRAY GoalRange OF Goal, TaskGoals:GoalStateSets!Set,
            exit:LiftGoalRange): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
   atom?(goals[g].subgoals) ∧
   goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   (NOT(ProcLow(g,s,goals)(inp,mem,env)) ∨
    NOT(SensLow(sens,g,s,goals)(inp,mem,env)) ∨
    NOT(CogLow(cog,g,s,goals)(inp,mem,env))) ∧
   (lift(g) = exit ∨ (∃(gs:GoalStateRange): gs∈TaskGoals ∧ gs∈goals[g].relevant));

ExistsSalient(sens:Salience, cog:Salience, s:Status, goals:ARRAY GoalRange OF Goal,
              TaskGoals:GoalStateSets!Set, exit:LiftGoalRange): Pred0 =
λ(inp:Inp,mem:Memory,env:Reality):
     ∃(g:GoalRange): LowSalience(sens,cog,g,s,goals,TaskGoals,exit)
```

A.3. Posessions

```
possessions(p:ARRAY PossesRange OF Possession, s0:Out, s1:Out): [Pos->[Pos->BOOLEAN]] =
λ(u0:Pos):λ(u1:Pos): ∀(idx:PossesRange):
          p[idx].count(u1) =
                 IF NOT(p[idx].acquire(s0)) ∧ p[idx].acquire(s1) ∧
                    NOT(p[idx].giveup(s1))
                 THEN p[idx].count(u0) + 1
                 ELSIF p[idx].count(u0) > 0 ∧ NOT(p[idx].acquire(s1)) ∧
                       NOT(p[idx].giveup(s0)) ∧ p[idx].giveup(s1)
                 THEN p[idx].count(u0) - 1
                 ELSE p[idx].count(u0)
                 ENDIF;

posval(idx:PossesRange) : [ARRAY PossesRange OF Possession->[Reality->NATURAL]] =
λ(p:ARRAY PossesRange OF Possession):λ(s:Reality):
     p[idx].count(s.pos)*p[idx].value(s.rlt) +
     IF idx > 1 THEN posval(idx-1)(p)(s) ELSE 0 ENDIF;

PossesVal(p:ARRAY PossesRange OF Possession) : [Reality->NATURAL] =
     λ(s:Reality): posval(npos)(p)(s);
```

A.4. Cognitive architecture

```
User [
       goals: ARRAY GoalRange OF Goal,
       actions: ARRAY ActionRange OF Action,
       GoalStates: ARRAY GoalStateRange OF Pred0,
       TaskGoals: GoalStateSets!Set,
       TopGoal: GoalRange,
       InitGoal: GoalRange,
       ExitGoal: GoalRange,
       InitOut: [Out -> BOOLEAN],
       InitMem: [Reality -> [Memory -> BOOLEAN]],
       Wait: [[Inp,Memory] -> BOOLEAN],
       pos: ARRAY PossesRange OF Possession
     ] : MODULE =
```

```
BEGIN
     GLOBAL t: NATURAL
     GLOBAL switch: Switch
     GLOBAL idle: BOOLEAN
     GLOBAL env: Reality
     INPUT inp: Inp
     OUTPUT out: Out
     OUTPUT mem: Memory
     OUTPUT finished: Finish
     LOCAL commit: ARRAY ActionRange OF Commit
     OUTPUT acommitted: BOOLEAN
     LOCAL status: Status

DEFINITION
     acommitted = ∃(a:ActionRange): commit[a] = committed;

INITIALIZATION
     switch = usr;
     idle = FALSE;
     out IN { x:Out | InitOut(x) };
     mem IN { x:Memory | InitMem(env)(x) };
     finished = notf;
     commit = [[a:ActionRange] ready];
     status = (#
                 active := InitGoal,
                 last := TopGoal,
                 trace := [[i:GoalRange] FALSE],
                 size := 0,
               #);

TRANSITION
  [](g,gg:GoalRange; cog,someSens,someCog:Salience):CommitAction:
     sens ≠ LowSLC ∧ cog ≠ LowSLC ∧
     someSens ≠ LowSLC ∧ someCog ≠ LowSLC ∧
     g ∈ gls(goals[gg].subgoals) ∧
     (HighestSalience(sens, cog, g, status, goals, TaskGoals)(inp, mem, env)
      ∨
      HighSalience(sens, cog, g, status, goals, TaskGoals)(inp, mem, env) ∧
      NOT(∃h : HighestSalience(someSens, someCog, h, status, goals, TaskGoals)(inp, mem, env))
      ∨
      LowSalience(sens, cog, g, status, goals, TaskGoals, lift(ExitGoal))(inp, mem, env) ∧
      NOT(∃h : HighSalience(someSens, someCog, h, status, goals, TaskGoals)(inp, mem, env))) ∧
     (g ≠ ExitGoal ∨ goals[TopGoal].achieved(inp, mem) ∨ NOT(Wait(inp, mem))) ∧
     NOT(acommitted) ∧ finished = notf ∧
     switch ≠ mach
     →
     commit′[act(Goals[g].subgoals)] = committed;
     status′ = status WITH .trace[g] := TRUE
                      WITH .length := status.length + 1
                      WITH .last := g
                      WITH .active := gg;
     t′ = t + CogOverhead
```

```
[]
  [](a:ActionRange):PerformAction:
      commit[a] = committed
      →
      commit′[a] = ready;
      out′ ∈ {x : Out | actions[a].tout(inp, out, mem)(x)};
      mem′ ∈ {x : Memory | actions[a].tmem(inp, mem, out′)(x)};
      env′ ∈ {x : Env | actions[a].tenv(inp, mem, env)(x) ∧ possessions(pos, out, out′)(env.pos)(x.pos)};
      t′ = t + actions[a].time;
      switch′ = mach;
      idle′ = FALSE;
[]
  ExitTask:
      goals[TopGoal].achieved(inp, mem) ∧
      NOT(acommitted) ∧ finished = notf
      →
      finished′ = ok
[]
  [](someSens,someCog:Salience):Abort:
      someSens ≠ LowSLC ∧ someCog ≠ LowSLC ∧
      NOT(ExistsSalient(someSens, someCog, status, goalsTaskGoals, lift(ExitGoal))(inp, mem, env)) ∧
      NOT(goals[TopGoal].achieved(in, mem)) ∧
      NOT(acommitted) ∧ finished = notf ∧
      switch ≠ mach
      →
      finished′ = IF Wait(in, mem) THEN notf ELSE abort ENDIF;
      switch′ = mach;
      idle′ = FALSE
[]
  Idle:
      finished = notf →

END;
```

References

[BaM95] Barnard PJ, May J (1995) Interactions with advanced graphical interfaces and the deployment of latent human knowledge. In: Design, specification and verification of interactive systems: DSV-IS'95. Springer, Berlin, pp 15–49

[BBC07] Back J, Blandford A, Curzon P (2007) Slip errors and cue salience. In: Proc. ECCE 2007: Invent! Explore!, pp 221–224. doi:10.1145/1362550.1362595

[BBC+08] Back J, Blandford A, Curzon P, Rukšėnas R (2008) Explaining mode and omission errors: a load model. Hum Factors J Hum Factors Ergonomics Soc (in press)

[BBD00] Butterworth RJ, Blandford AE, Duke DJ (2000) Demonstrating the cognitive plausibility of interactive systems. Formal Aspects Comput 12(4):237–259. doi:10.1007/s001650070021

[BFH+08] Back J, Furniss D, Hildebrandt M, Blandford A (2008) Resilience markers for safer systems and organisations. In: Computer safety, reliability, and security: SAFECOMP 2008, Springer, Berlin, pp 99–112. doi:10.1007/978-3-540-87698-4_11

[BoF99] Bowman H, Faconti G (1999) Analysing cognitive behaviour using LOTOS and Mexitl. Formal Aspects Comput 11(2):132–159. doi:10.1007/s001650050045

[ByB97] Byrne MD, Bovair S (1997) A working memory model of a common procedural error. Cogn Sci 21(1):31–61. doi:10.1016/S0364-0213(99)80018-4

[CaL07] Cartwright-Finch U, Lavie N (2007) The role of perceptual load in inattentional blindness. Cognition 102(3):321–340. doi:10.1016/j.cognition.2006.01.002

[Cam03] Campos JC (2003) Using task knowledge to guide interactor specifications analysis. In: Jorge JA, Jardim Nunes N, Falcão e Cunha J (eds) Interactive systems. design, specification and verification, 10th international workshop. Lecture notes in computer science, vol 2844. Springer, Berlin, pp 171–186

[ChB04] Chung P, Byrne MD (2004) Visual cues to reduce errors in a routine procedural task. In: Proc. 26th Ann. Conf. of the cognitive science society. Lawrence Erlbaum Associates, NJ

[CRB07] Curzon P, Rukšėnas R, Blandford A (2007) An approach to formal verification of human–computer interaction. Formal Aspects Comput 19(4):513–550. doi:10.1007/s00165-007-0035-6

[CuB01] Curzon P, Blandford AE (2001) Detecting multiple classes of user errors. In: Little R, Nigay L (eds) Proceedings of the 8th IFIP working conference on engineering for human–computer interaction (EHCI'01). Lecture notes in computer science, vol 2254. Springer, Berlin, pp 57–71. doi:10.1007/3-540-45348-2_9

[DBD+98] Duke DJ, Barnard PJ, Duce DA, May J (1998) Syndetic modelling. Human–Computer Interaction 13(4):337–394. doi:10.1207/s15327051hci1304_1

[DBM+95] Duke DJ, Barnard PJ, May J, Duce DA (1995) Systematic development of the human interface. In: APSEC'95: Second Asia-Pacific software engineering conference. IEEE Computer Society Press, pp 313–321. doi:10.1109/APSEC.1995.496980

[DuD99] Duke DJ, Duce DA (1999) The formalization of a cognitive architecture and its application to reasoning about human computer interaction. Formal Aspects Comput 11(6):665–689. doi:10.1007/s001659970004

[Fie01] Fields RE (2001) Analysis of erroneous actions in the design of critical systems. D.Phil Thesis, Technical Report YCST 20001/09. University of York, Department of Computer Science

[Gra00] Gray WD (2000) The nature and processing of errors in interactive behavior. Cogn Sci 24(2):205–248. doi:10.1207/s15516709cog2402_2

[HaG87] Hale A, Glendon I (1987) Individual behaviour in the control of danger. Industrial safety series, vol 2. Elsevier, Amsterdam

[Hol93] Hollnagel E (1993) Human reliability analysis, context and control. Academic Press, London

[LeT93] Leveson NG, Turner CS (1993) An investigation of the therac-25 accidents. IEEE Computer 26(7):18–41. doi:10.1109/MC.1993.274940

[MoD95] Moher TG, Dirda V (1995) Revising mental models to accommodate expectation failures in human-computer dialogues. In: Design, specification and verification of interactive systems: DSV-IS'95. Springer, Berlin, pp 76–92

[MOR+04] de Moura L, Owre S, Ruess H, Rushby J, Shankar N, Sorea M, Tiwari A (2004) SAL 2. In: Alur R, Peled DA (eds) Computer aided verification: CAV'04. Lecture notes in computer science, vol 3114. Springer, Berlin, pp 496–500

[New90] Newell A (1990) Unified theories of cognition. Harvard University Press, Cambridge, MA, USA

[Nor83] Norman DA (1983) Some observations on mental models. In: Stevens AL, Gentner D (eds) Mental models. Lawrence Erlbaum Associates, NJ

[PaM95] Paternò F, Mezzanotte M (1995) Formal analysis of user and system interactions in the CERD case study. In: Proceedings of EHCI'95: IFIP working conference on engineering for human–computer interaction. Chapman and Hall Publisher, London, pp 213–226

[Pay91] Payne SJ (1991) Display-based action at the user interface. Int J Man–Machine Stud 35(3):275–289. doi:10.1016/S0020-7373(05)80129-4

[Ras80] Rasmussen J (1980) What can be learned from human error reports? In: Duncan K, Gruneberg M, Wallis D (eds) Changes in working life. Wiley, London

[Rea90] Reason J (1990) Human error. Cambridge University Press, London

[RBC+08] Rukšėnas R, Back J, Curzon P, Blandford A (2008) Formal modelling of salience and cognitive load. In: Proc. 2nd Int. workshop on formal methods for interactive systems: FMIS 2007. Electronic Notes in Theoretical Computer Science 208:57–75. doi:10.1016/j.entcs.2008.03.107

[RCB+07] Rukšėnas R, Curzon P, Back J, Blandford A (2007) Formal modelling of cognitive interpretation. In: Doherty G, Blandford A (eds) Interactive systems. design, specification and verification, 13th international workshop. Lecture notes in computer science, vol 4323. Springer, Berlin, pp 123–136. doi:10.1007/978-3-540-69554-7_10

[RCB+08] Rukšėnas R, Curzon P, Blandford A, Back J (2008) Combining human error verification and timing analysis. In: Proceedings of engineering interactive systems 2007. Lecture notes in computer science, vol 4940. Springer, Berlin, pp 18–35

[Rus01] Rushby J (2001) Analyzing cockpit interfaces using formal methods. Electronic Notes in Theoretical Computer Science 43:1–14. doi:10.1016/S1571-0661(04)80891-0

[SBB07] Su L, Bowman H, Barnard PJ (2007) Attentional capture by meaning: a multi-level modelling study. In: Proc. 29th ann. conf. of the cognitive science society. Lawrence Erlbaum Associates, NJ, pp 1521–1526

[SBB+08] Su L, Bowman H, Barnard PJ, Wyble B (2008) Process algebraic modelling of attentional capture and human electrophysiology in interactive systems. Formal Aspects Comput (in press)

[Wic02] Wickens C (2002) Multiple resources and performance prediction. Theor Issues Ergonomic Sci 3(2):159–177. doi:10.1080/14639220210123806

Received 11 March 2008
Accepted in revised form 1 November 2008 by A. Cerone and D.A. Duce
Published online 15 January 2009