Formal Modelling of Salience and Cognitive Load[⋆]

R. Rukšėnas[a,1], P. Curzon[a,2], J. Back[b,3], A. Blandford[b,4]

[a] *Dept. of Computer Science, Queen Mary, University of London, London, UK*

[b] *University College London Interaction Centre, London, UK*

Abstract

Well-designed interfaces use procedural and sensory cues to increase the salience of appropriate actions and intentions. However, empirical studies suggest that cognitive load can influence the strength of procedural and sensory cues. We formalise the relationship between salience and cognitive load revealed by empirical data. We add these rules to our abstract cognitive architecture developed for the verification of usability properties. The interface of a fire engine despatch task used in the empirical studies is then formally verified to validate the salience and load rules. Finally, we discuss how our formal modelling and verification suggest further refinements of the rules derived from the informal analysis of empirical data.

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": 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 [14,11]. Often opportunities for making such errors can be reduced with good design [6]. A methodology for detecting designs that allow users, when behaving in a rational way, to make systematic errors will improve such systems.

⋆ This research is funded by EPSRC grants GR/S67494/01 and GR/S67500/01.

1 Email: `rimvydas@dcs.qmul.ac.uk`

2 Email: `pc@dcs.qmul.ac.uk`

3 Email: `j.back@ucl.ac.uk`

4 Email: `a.blandford@ucl.ac.uk`

When performing a familiar interactive routine, an intention 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 and actions can be cognitively cued. 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. However, although sensory salience can be manipulated by making an indicator bigger or louder or just in time, this does not ensure that the action will be performed since the action may not be cognitively salient. A series of empirical studies conducted by Back *et al* [1] has found that many slip errors are caused by an inappropriate "springs to mind" response to environmental features (that correspond to known actions). For example, the appearance of a text entry box may prompt a user to start typing without initialising the box (i.e., moving the mouse cursor and clicking inside the box). Accounts of many errors can be given in terms of how people are allocating attentional resources. An individuals awareness of what they have to do next can be driven by cues that are internal to the cognitive system (goals and methods) or external to the cognitive system (cued in the environment).

Non-sensory cues, known as procedural cues (internal to the cognitive system), can be used to retrieve previously formulated intentions (expert procedural knowledge) enabling the next procedural step to be performed. Remembering that, and so doing, after performing x always do y if z is true is an example of following a procedural cue rule. Sensory cues (external to the cognitive system) can also be used to retrieve intentions (expert procedural knowledge). For example, if sensory cue p is attended to then it may indicate that q should be the next step if r is true.

People make slip errors frequently, but do not make them every time. Empirical studies [1] suggest that cognitive load can influence the frequency of errors by affecting the strength of procedural and sensory cues. In this paper, we formalise the relationship between salience and cognitive load revealed by the informal analysis of empirical data. We then incorporate these rules into our abstract cognitive architecture [9,15] developed earlier from 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. This extension refines non-determinism by introducing a hierarchy of choices governed by the salience (strength) of procedural and sensory cues, and the level of cognitive load imposed by the task performed.

As a validation step for our extension, we formally model the fire engine despatch task used in the empirical studies [1]. One reason for doing this is to check whether the systematic errors identified during the experiments can also be detected by the formal verification of the same task, thus indicating that our extended cognitive architecture generates behaviours corresponding to those of real people. Another reason is that possible mismatches between the two sets of behaviours can suggest refinements to our salience and load rules and their formalisation within the cognitive architecture.

Summarising, the main contribution of this paper is the following:

- An investigation into the formal modelling of salience and cognitive load.

- A formalisation of the connection between salience and cognitive load revealed by empirical studies.
- An extension of our verification framework involving salience and load rules, and a hierarchy of salience levels.
- A formal modelling, as a validation step, of the task used earlier in our empirical studies.

Related work. There is not much related work on salience and cognitive load. Cartwright-Finch and Lavie [7] developed theory that a high extraneous load only reduces perception of a sensory cue when cognitive control functions are not available to maintain the active goal. More generally, work on human error has shown that the provision of visual cues can strengthen procedural cueing providing they manage to capture attention [8].

Duke *et al* [10], and Bowman and Faconti [4] use Interactive Cognitive Subsystems (ICS) [3] 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, the above 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.

In the related area of safety-critical systems, Rushby *et al* [17] 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 [5]. 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.

Cognitive principles

In the formal user model, we rely upon abstract cognitive principles that give a *knowledge level* description in the terms of Newell [13]. Their focus is on the internal goals and knowledge of a user. These principles are briefly discussed below.

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. 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) salience 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. 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 other than a goal hierarchy 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 [6], 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.

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`

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.

Cognitive Architecture in SAL

We have formalised the cognitive principles within the SAL environment [12]. It provides a higher-order specification language and tools for analysing state machines specified as parametrised modules and composed either synchronously or asynchronously. 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 user model is given in Fig. 1, where predicates in italic are shorthands explained later on. Below, whilst explaining this specification (SAL module `User`), we also discuss how it reflects our cognitive principles.

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 `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.* [5] User goals are organised as a hierarchical (tree like) goal–subgoals structure. The nodes of this tree are either compound or atomic:

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

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

[5] Note that we are omitting from the description of the goal structure its aspects related to the relevance of goals. They are not used in the work described here, for the omitted detail see [16].

```
TRANSITION
  [](g:GoalRange,slc:Salience):   CommitAction:
     NOT(comm) ∧
     finished = notf ∧
     (HighestSalience(slc, g, status, goals, ...)
      ∨                                                          commit′[act(Goals[g].subgoals)] = committed;
      HighSalience(slc, g, status, goals, ...) ∧                 status′ = status
      NOT(∃h : HighestSalience(LowSLC, h, status, goals, ...)    →      WITH .trace[g] := TRUE
      ∨                                                                 WITH .last := g
      LowSalience(slc, g, status, goals, ...) ∧                         WITH .length := status.length + 1
      NOT(∃h : (HighSalience(LowSLC, h, status, goals, ...) ∨
           HighestSalience(LowSLC, h, status, goals, ...))) ∧
     (g ≠ ExitGoal  ∨  MayExit)
[]
  [](a:ActionRange):   PerformAction:
                                     commit′[a] = ready;
     commit[a] = committed     →     Transition(a)
[]
  ExitTask:
     goals[TopGoal].achieved(in, mem) ∧
     BrokenState(in, mem, env) ∧
     NOT(comm) ∧                              →    finished′ = ok
     finished = notf
[]
  Abort:
     NOT(ExistsSalient(...)) ∧
     NOT(goals[TopGoal].achieved(in, mem)) ∧          finished′ = IF Wait(in, mem)
     NOT(comm) ∧                                  →               THEN notf
     finished = notf                                              ELSE abort ENDIF
[]
  Idle:
     finished = notf    →
```

Fig. 1. Cognitive architecture in SAL (simplified)

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 and aims 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:

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

choice 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 A predicate, denoted `achieved`, that specifies the main task goal when `g` is the top goal, not used for atomic goals.

salience A value, denoted `slc`, that specifies the sensory salience of `g`.

cue A function, denoted `cue`, that for each goal `h` returns the strength of `g` as a procedural cue for `h`.

subgoals 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:

$$\texttt{commit}[\mathsf{a}] = \texttt{committed} \quad \rightarrow \quad \begin{array}{l}\texttt{commit}'[\mathsf{a}] = \texttt{ready};\\ \mathit{Transition}(\mathtt{a})\end{array}$$

The left-hand side of → 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, *Transition*(`a`) represents the state updates associated with this particular action `a`.

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 that describe how two components of the user model state (outputs `out` and memory `mem`) and environment `env` are updated by executing this action. These relations, provided when the generic user model is instantiated, are used to specify *Transition*(`a`) as follows:

```
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}
```

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, since we usually assume the user remembers the actions 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*). A goal `g` *may* be selected only when one of the disjuncts specifying its salience level (see Sect. 3) is true. The last conjunct in the guard of *CommitAction* distinguishes the cases when the selected goal is `ExitGoal` or not. `ExitGoal` (given as a parameter of the `User` module) represents such options as “cancel” or “exit”, available in some form in most of interactive systems. We omit the definition of *MayExit*, 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 (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` (the latter states that there exists a goal 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. However, 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 specification 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

In this section, we discuss the connection between the salience of cues and cognitive load observed in empirical studies and expressed as salience and load rules. We then formalise these rules within our verification framework and incorporate them into our cognitive architecture.

Cognitive load. Slip errors are made frequently, but they are not made every time. The frequency of these errors is determined by causal factors internal (goals) and external to the cognitive system. After formulating goals, new information may interfere with the ability to retain previous formulations. We designed an experimental paradigm [2] that manipulated the availability (and awareness) of both procedural and sensory cues that were needed to overcome performing erroneous "springs to mind" actions. Our hypothesis was that slip errors were more likely when the salience of cues was not sufficient to actively influence attentional control. If processes are directed by a passive (off-line) attentional control system then errors associated with performing "springs to mind" actions are more likely. A simulation of a 'Fire Engine Despatch Centre' was developed. The overall objective was to send navigational information to fire engines enabling the fastest possible incident response times. Training trials were used to ensure that participants became familiar with the sequence of actions required. Cognitive load was manipulated by the complexity of routes imposed and the quantity of task irrelevant information displayed.

We found that the difficulty associated with performing a proceduralized 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 and extraneous load imposed on the individual is high. These findings are compatible with Cartwright-Finch and Lavies [7] 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 of our empirical data suggested the following connections between salience and cognitive load [2]:

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

procedural High intrinsic load reduces the salience of procedural cues.

Next we formalise these connections within our verification framework.

Formalisation. In our formalisation, salience can take one of the following three values: `HighSLC`, `LowSLC` and `NoSLC`, whereas both the intrinsic and extraneous load can be either `HighLD` or `LowLD`. First, we had to capture the meaning of "reduced salience" in the above rules. We decided to interpret this as salience going from high to low. Then the sensory salience rule is expressed as follows:

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

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 experiencied, 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 need a predicate that specifies when the sensory salience of a goal is high. Thus, rule (1) is translated into the following definition of `SensSalient`:

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

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

```
HighestSalience(arb,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) ∨
    ProcLow(g,status,goals)(inp,mem,env) ∧ SensSalient(arb,g,status,goals)(inp,mem,env))

HighSalience(arb,g,status,goals)(inp,mem,env) =
   atom?(goals[g].subgoals) ∧ goals[g].grd(inp,mem,env) ∧ goals[g].choice(g,s) ∧
   goals[last].cue(g)(inp,mem,env) = NoSLC ∧ SensSalient(arb,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

The procedural salience rule is formally expressed as follows:

$$(2)\quad \begin{array}{l} \mathit{if}\ \texttt{default} = \texttt{HighSLC}\ \wedge\ \texttt{intr} = \texttt{HighLD}\ \mathit{then}\ \texttt{procedural} = \texttt{HighSLC} \\ \mathit{else}\ \texttt{procedural} = \texttt{default} \end{array}$$

In the cognitive architecture, this is translated into two 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 ∨
   goals[status.last].cue(g)(inp,mem,env) = HighSLC ∧ status.intrinsic = HighLD
```

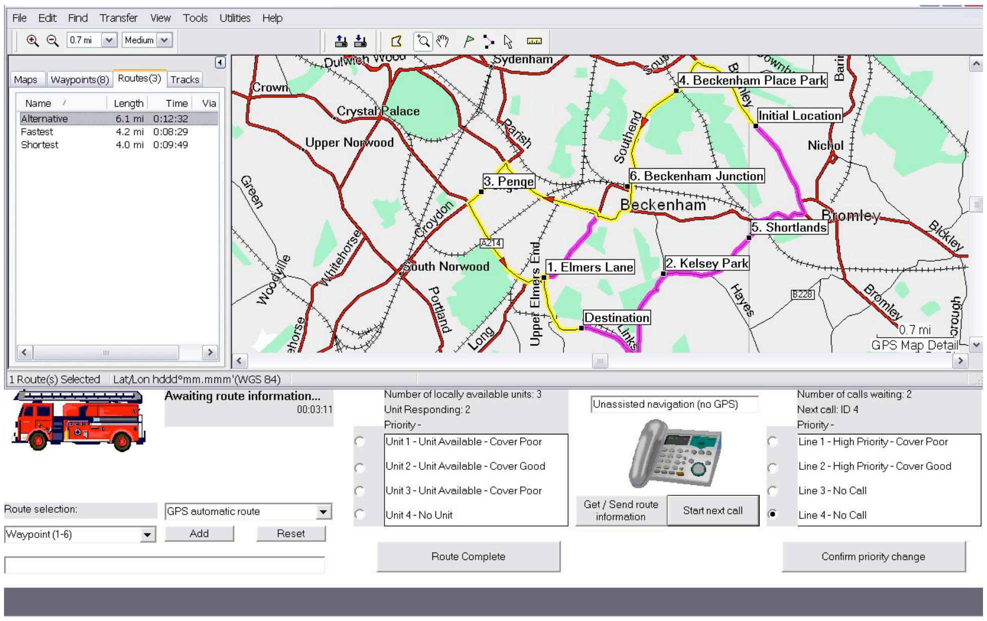
Hierarchy of choices. Next we discuss how the sensory and procedural salience influence the choice of goals in our cognitive architecture. Recall that the underlying choice principle is non-determinism – any "enabled" goal can be chosen for execution. The addition of salience refines the notion of enabledness 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 `SensSalient` 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 described in Sect. 5 produced errors (and the corresponding behaviours of the model) not observed during our empirical studies. The analysis of the counter examples suggested a refinement of the two-level hierarchy which yielded a new version specified in Fig. 1.

The new version consists of three levels of salience. Assuming the choice strategy and the guard for an atomic goal is true, its salience belongs to the highest level, if (i) its procedural salience is high, or (ii) its procedural salience is low, and sensory salience is high (see Fig. 2). It belongs to the middle level (high salience), if it is procedurally cued, but its sensory salience is high. Such a goal is only chosen, if there are no goals in the highest level. Finally, the lowest level includes all the remaining atomic goals whose choice strategy and guard are true.

4 Fire Engine Despatch Task

In this section, we describe the task we chosen to model for the validation of our development of the cognitive architecture.

The overall objective of the task was to send navigational information to fire engines enabling the fastest possible incident response times using the interface



Name	Length	Time	Via
Alternative	6.1 mi	0:12:32	
Fastest	4.2 mi	0:08:29	
Shortest	4.0 mi	0:09:49	

1 Route(s) Selected Lat/Lon hddd°mm.mmm'(WGS 84)

Awaiting route information...
00:03:11

GPS automatic route
Waypoint (1-6)

Number of locally available units: 3
Unit Responding: 2
Priority -
Unit 1 - Unit Available - Cover Poor
Unit 2 - Unit Available - Cover Good
Unit 3 - Unit Available - Cover Poor
Unit 4 - No Unit

Unassisted navigation (no GPS)

Number of calls waiting: 2
Next call: ID 4
Priority -
Line 1 - High Priority - Cover Poor
Line 2 - High Priority - Cover Good
Line 3 - No Call
Line 4 - No Call ●

Fig. 3. ‘Fire Engine Despatch’ interface

shown in Fig. 3. When commencing the 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 could 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 informs which option must be used. The leftmost drop-down menu supports manual route construction by allowing to select waypoints and add them 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. Again the backup unit is only set once the 'Route complete button has been clicked.

5 Task Verification

In this section, we instantiate our generic architecture, thus deriving a user model for the 'Fire Engine Despatch Task'. This model is then used for the verification of correctness properties for the interface described in the previous section.[6]

In this paper, we consider one usability property. It aims to ensure that, in any possible system behaviour, the user's main goal of interaction (as they perceive it) is eventually achieved. This is written in SAL as the following LTL assertion (here `F` means "eventually"):

(3) $$\mathtt{F}(\mathtt{Perceived}(\mathtt{inp}, \mathtt{mem}, \mathtt{env}))$$

The main purpose of our verification is to find out how close, with respect to the errors detected, its results are to the data of our empirical studies. Since the actual experiments essentially consisted of two subtasks, we split the task (and its user model) into two parts: setting call priority and sending route information. Next we consider the first subtask.

Call priority

We assume that the user model for this (sub)task includes three 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): HighSLC
cues := λ(g):λ(inp,mem,env):
             IF g = ConfirmPriorityGoal THEN HighSLC ELSE NoSLC 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 high, since the visual attention, at this point in task execution should be in correct area. The salience of this goal as a procedural cue for the call confirmation action is high, but it should not cue other actions. The corresponding action `SelectPriority` is defined as follows:

```
tout := λ(inp,out0,mem):λ(out): out = Default WITH .PrioritySelected := TRUE
tmem := λ(inp,mem0,out):λ(mem): mem = mem0 WITH .PrioritySelected := TRUE
```

[6] Due to the shortage of space, we omit here the formal specification of the interface. The SAL sources for this example are available at http://www.dcs.qmul.ac.uk/~rimvydas/usermodel/fmis07.zip.

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

The definitions for the other two goals are similar. Their sensory salience is assumed to be high. `ConfirmPriorityGoal` is high procedural cue for `StartCallGoal`, whereas the latter being the last step in the procedure does not cue other actions. Finally, the top goal `CallPriorityGoal` is defined as follows:

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

It includes all three atomic goals as its subgoals. Since setting call priority is a cognitively simple procedure, we assume that the intrinsic load for this task is low. Finally, the component `achieved` defines the perceived goal of the task. The latter is regarded as achieved when a wait message is displayed by the interface. The latter happens only 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 user model derived by the above instantiation of the generic architecture is parametric with respect to the extraneous load. This parameter can be manipulated in different verification runs, similarly as has been done in the empirical studies.

Verification. For both extraneous load conditions, verification of property (3) fails. The counter examples indicate that the user model starts by immediately executing `StartCallGoal`. This corresponds to the empirical studies which found this initialisation error to be systematic. Its main cognitive cause is that the required action of choosing priority is the first one in the procedure, and thus is not procedurally cued. This suggests that the initialisation error is unlikely to be eliminated by increasing the sensory salience of the relevant menu options. On the other hand, "sequencing" the interface so that the 'Start next call' button becomes available only when call priority has been set should eliminate this error. Verification of the task with a modified interface confirms this.

Next we check whether the task goal is achieved assuming the first step was taken correctly. This is expressed as a slightly modified property (3):

```
X(commit[SelectPriorityAction]=committed) ⇒ F(Perceived(inp,mem,env))
```

Here `X` is the LTL operator "next". The property states that the task goal is eventually achieved, if the first thing that the user model does is committing to `SelectPriorityAction`. Verification of the modified usability property is successful for both extraneous load conditions. Again, this corresponds to the results of our empirical studies which found that the action of confirming priority is almost never omitted presumably due to its high procedural cueing by the previous action of selecting priority. Next we consider the second subtask.

Sending route information

We assume that the user model for this (sub)task includes the following atomic goals: `DecideModeGoal`, `GPSGoal`, `ManualGoal`, `ClickAddGoal`, `SelectBackupGoal`, `ConfirmRouteGoal` and `SendRouteGoal`. For `DecideModeGoal`, 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 NoSLC 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. It is procedurally cued by the action of starting new call which ends the first subtask. We assume that this cueing is low, since the mode indicator is displayed after some delay during which a person is likely to be engaged in a complex process relevant to the route identification and construction. Because we split the whole despatch task into two subtasks, the procedural cue for `DecideModeGoal` is specified to be the top goal `SendRouteGoal`. The sensory salience of `DecideModeGoal` is low, since the mode indicator is displayed in different area than the route construction menus. Finally, the memory update for `DecideModeAction` 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
```

The essential components for `SelectBackupGoal` are defined as follows:

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

The salience of the procedural cues, `GPSGoal` and `ManualGoal`, for this goal is high. Finally, for the top goal `SendRouteGoal` we have the following definitions:

```
load := intrinsic
achieved := λ(inp,mem,env): inp.SuccessMsg
```

The variable `intrinsic` is a parameter of our user model. It denotes the intrinsic load associated with the route construction procedure and depends on the construction method. As previously, the model is also parametric with respect to the extraneous load. The perceived task goal is to send route information. Achieving this goal is indicated by a success message displayed by the interface.

Verification. As mentioned in Sect. 3, initially we used a hierarchy with two salience levels. For this version of the cognitive architecture, verification of property (3) produced erroneous behaviours that were not observed in the empirical studies. Namely, after selecting the backup unit, the user model was executing `SendRouteAction` instead of `ConfirmRouteAction` which is procedurally cued with high salience by `SelectBackupAction`. This lead to the introduction of additional salience level, prioritising the goals with high procedural salience (e.g. `ConfirmRouteAction`) over those whose sensory salience is high (e.g. `SendRouteAction`). The modified version of the cognitive architecture was used for further verification.

To simplify the analysis of the counter examples produced, we introduced two additional correctness properties:

(4) $$\mathtt{G}(\mathtt{mem.RouteConstructed} \Rightarrow (\mathtt{mem.mode} \neq \mathtt{NoMode}))$$

(5) $$\mathtt{G}((\mathtt{out.ConfirmRoute} \vee \mathtt{out.SendRoute}) \Rightarrow \mathtt{mem.BackupSelected})$$

The first one is relevant to the mode error and states that the memory component `RouteConstructed` (updated by `GPSAction` and `ClickAddAction`) can only be true, if the user model attended to the mode indicator and stored its value in `mem.mode`. The second property states that the buttons 'Confirm Route' and 'Get/Send Route' can only be clicked, if a backup unit has been selected (indicated by the memory component `BackupSelected` which is updated by `SelectBackupGoal`. It is relevant in the situations when the action of selecting backup is omitted (termination error).

We start with property (4). Its verification fails for all load conditions: the user model omits the action of attending to the mode indicator. These results are inconsistent with respect to the empirical studies which found that only both the intrinsic and extraneous load being high leads to the systematic mode error. However, these inconsistencies are false positives. Our verification did not miss erroneous behaviours. Rather, it indicated problems that did not occur when this task was performed by humans. This suggests that our salience and load rules and/or the hierarchy of salience levels need further refinements. We intend to explore this in future work.

Next we verify property (5) relevant to the termination error. In this case, the correlation with the empirical data is closer. For all load conditions but intrinsic being high and extraneous being low, verification of (5) yields the same results as the empirical studies. In the case when both the intrinsic and the extraneous load is high the omission of selecting backup is observed. When the intrinsic load is low verification of (5) is successful. This corresponds to low rates for these load conditions in our empirical studies (non-systematic error). On the other hand, the single mismatch occurring when the intrinsic load is high and the extraneous load is low is potentially more serious than previous false positives. In this case, our verification is successful, even though our empirical data indicates a systematic error.

One possible explanation for this mismatch is that our judgement about salience values for some goals was inappropriate. In fact, verification yields the termination error when, for `SelectBackupGoal`, the salience of procedural cueing is set to high and the sensory salience is set to low (in the original specification, these values were low and high, respectively). Furthermore, the new value for procedural cueing could be reasonably argued for, though admittedly the new value for the sensory salience is probably more difficult to defend. Another possible explanation is that our rules are simply too coarse. Whichever the case may be further investigations are necessary.

6 Conclusion

In this paper, we added to our cognitive architecture the concepts of procedural and sensory salience. We formalised the connection between both salience types and cognitive load imposed by the complexity of the task performed. We then refined the underlying principle of non-deterministic choice of goals by introducing a hierarchy of choices governed by the salience of goals. Verification attempts using the new version of the cognitive architecture suggested further refinements to the hierarchy of salience levels.

As a validation step for these developments, we undertook the formal modelling of the fire engine despatch task used in our empirical studies. Our goal was to check the consistency between the behaviours generated by our cognitive architecture and those exhibited by human participants of our experiments. The validation was partially successful. We found that, with respect to the initialisation error, verification yielded results that are consistent with the human behaviour observed during the experimental studies. For the mode error, our verification produced false positives in some cases. On the other hand, in the single case when the intrinsic load is high and the extraneous load is low, verification missed the termination error found to be systematic in the experiments. These inconsistencies raised questions to be answered by further refinements of our formal rules and new empirical studies.

References

[1] Back, J., A. Blandford, and P. Curzon, *Slip errors and cue salience*, To appear: Proc. ECCE 2007.

[2] Back, J., A. Blandford, P. Curzon, and R. Rukšėnas, *Explaining mode and omission errors: a load model*, Submitted for publication.

[3] Barnard, P.J., and J. May, *Interactions with advanced graphical interfaces and the deployment of latent human knowledge*, In: Design, Specification and Verification of Interactive Systems: DSV-IS'95, Springer-Verlag, 1995, 15–49.

[4] Bowman, H., G. Faconti, *Analysing cognitive behaviour using LOTOS and Mexitl*, Formal Aspects of Computing **11** 1999, 132–159.

[5] Butterworth, R., A. Blandford, and D. Duke, *Demonstrating the cognitive plausibility of interactive systems*, Form. Asp. Computing **12** 2000, 237–259.

[6] Byrne, M. D., and S. Bovair, *A working memory model of a common procedural error*, Cognitive Science **21**(1) 1997, 31–61.

[7] Cartwright-Finch, U., and N. Lavie, *The role of perceptual load in inattentional blindness*, Cognition **102**(3) 2007, 321–340.

[8] Chung, P., and M.D. Byrne, *Visual cues to reduce errors in a routine procedural task*, In: Proc. 26th Ann. Conf. of the Cognitive Science Society, Hillsdale, NJ: Lawrence Erlbaum Associates, 2004.

[9] Curzon, P., and A. E. Blandford, *Detecting multiple classes of user errors*, in: R. Little, and L. Nigay, eds., Proc. EHCI 2001, vol. 2254 of LNCS, Springer-Verlag, 2001, 57–71.

[10] Duke, D.J., P.J. Barnard, D.A. Duce, and J. May, *Syndetic modelling*, Human-Computer Interaction **13**(4) 1998, 337–394.

[11] Gray, W., *The nature and processing of errors in interactive behavior*, Cognitive Science **24**(2) 2000, 205–248.

[12] de Moura, L., S. Owre, H. Ruess, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, *SAL 2*, in: R. Alur, and D.A. Peled, eds., Computer Aided Verification: CAV 2004, vol. 3114 of LNCS, Springer-Verlag, 2004, 496–500.

[13] Newell, A., "Unified Theories of Cognition," Harvard University Press, 1990.

[14] Reason, J.: "Human Error," Cambridge University Press, 1990.

[15] Rukšėnas, R., P. Curzon, J. Back, and A. Blandford, *Formal modelling of cognitive interpretation*, in: Proc. DSVIS 2006, vol. 4323 of LNCS, Springer-Verlag, 2007, 123–136.

[16] Rukšėnas, R., P. Curzon, A. Blandford, and J. Back, *Combining human error verification and timing analysis*, To appear: Proc. EIS 2007, LNCS, Springer-Verlag.

[17] Rushby, J., *Analyzing cockpit interfaces using formal methods*, Electronic Notes in Theoretical Computer Science **43** (2001).