Electronic Communications of the EASST
Volume 45 (2011)

Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems (FMIS 2011)

Closure and Attention Activation in Human Automatic Behaviour: A Framework for the Formal Analysis of Interactive Systems

Antonio Cerone

18 pages

Guest Editors: Judy Bowen, Steve Reeves
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Closure and Attention Activation in Human Automatic Behaviour: A Framework for the Formal Analysis of Interactive Systems

Antonio Cerone[1]

United Nations University, International Institute for Software Technology (UNU–IIST)
Macau SAR China[1]

Abstract: *Automatic behaviour* can be defined as a fast processing human activity that does not require attention to occur. According to Norman and Shallice's model of attention and automaticity the majority of responses are under fairly automatic control triggered by environmental cues. In this paper we define a process algebraic framework to formalise Norman and Shallice's model and illustrate it through two case studies: Driving and using an Automatic Teller Machine (ATM). Finally we show how to use model-checking to analyse model instantiations and present the outcome of the analysis for the ATM case study.

Keywords: Formal Methods, Interactive Systems, Automaticity, Attention, Expectancy, Contention Scheduling.

1 Introduction

The use of formal methods in the analysis of interactive systems has started in the 1990's in the domain of safety-critical systems, where the relevant human component of the system is represented by operators with expected expertise and skills. Operator behaviour was often modelled as defined by the interface requirements, rather than in terms of general cognitive capabilities and limitations.

Nowadays, as a consequence of ubiquitous computing and widespread technology, issues of safety and security originated by human-machine interaction are no longer restricted to skilled operator's behaviour in traditionally critical domains, such as transportation, chemical and nuclear plants, health and defence, but actually permeate many aspect of everyday human life. Driving behaviour [Ran94] and interaction with an Automatic Teller Machine (ATM) [RCB08] are two examples of aspects of everyday life in which cognitive errors may lead to safety and security violations.

Large variations in the typology and motivations of humans interacting with computer systems make it impossible to define a predicted user's behaviour: human errors are actually the very result of an unexpected user behaviour that emerges through the interaction. To best capture such an emergent behaviour, user's models must specify the *cognitively plausible behaviour*, that is, all possible behaviours that can occur and that involve different cognitive processes [BBD00].

A number of recent works have explored the use of formal models to understand how cognitive errors can affect user performance. Curzon and Blandford [CB04] model the behaviour of a user who assumes all tasks completed when the goal is achieved but forgets to complete some important subsidiary tasks (*post-completion error)*. Cerone and Elbegbayan [CE07] set the most

pessimistic scenario, in which the user is a novice, with minimal skills, and explore alternative user behaviours corresponding to a variety of attitudes and personalities. Rukšėenas, Curzon and Blandford [RCB08] see rational behaviour as a game between planning aspects, typical of a well-trained expert user, and reactive aspects, typical of a novice user.

In this paper we consider human behaviour in performing everyday activities as fairly automatic. This is the context in which typical cognitive errors analysed in previous research, such as post-completion errors, are most likely to occur. Automatic behaviour is, however, by no means purely reactive; it actually develops throughout an extensive training process and features a high-level form of consciousness. Furthermore, in many occasions, deliberate and conscious low-level actions are still required and attention takes control. When attention is activated by the failure of expectations that the user has developed through experience and training, cognitive errors may result and emerge in the form of inappropriate deliberate responses.

Formal methods have seldom been used to analyse attentional mechanisms. Recently Su, Bowman, Barnard and Wyble [SBBW09] have used process algebraic modelling to explore the temporal attentional limitation of human operators and have then presented arguments about how this affects their ability to interact with computer systems. However, their work focuses on stimulus rich environments in which attention is continuously activated. In our work, instead, we consider attention as a sparsely activated mechanisms within everyday routine activities.

2 Attention versus Automatic Behaviour

Attention can be defined as a selective processing activity that aims to focus on one aspect of the environment while ignoring others. Modern research on *selective attention* goes back to the early 1950s, when Colin Cherry analysed a phenomenon that he called *cocktail party problem:* humans can selectively attend to one conversation among a large number of conversations going on around them [Che53]. The research question was to understand which kind of information could be perceived from unattended conversations. Cherry concluded that only low-level information, such as distinguishing a female voice from a male voice, but no information about meaning, not even the language in which the conversation is carried out, could be extracted from an unattended conversation. Although this conclusion resulted in the first information processing model of attention, the *early selection model* [Bro58], subsequent studies showed that attention is most frequently elicited by unattended stimuli that have some meaningful relation to what we are consciously doing [GW60] and that, therefore, there is some meaningful analysis of unattended information [Tre60]. In some cases, a full analysis of meaning could even precede the selection process, as is explained by *late selection theories* [DD63] and partly confirmed through experiments [Mac60].

Automatic behaviour or *automaticity* can be defined as a fast processing activity that does not require attention to occur. In some cases responses may be triggered by stimuli of which we are not yet conscious, or of which we were previously conscious but we are no longer [Bur94]. For example in the *Stroop effect*, in which the task is to name the colours in which words are presented, when shown a colour word (e.g. “red”) presented in an incongruent colour (e.g. “blue”), the participant is actually more likely to read the word rather than saying the colour in which it is presented [Str35]. Here, there is no conscious way to intervene on the automaticity

process (*unconscious automaticity*), unless we brake the rule of the experiments by defocussing our eyes. However, in most cases automaticity develops following extended consistent practise or exposition to a regular pattern [SS77a, SS77b]. For example, automaticity is essential in driving a car and, in such a context, it develops throughout a learning process: during the learning process the driver has to make a conscious effort to use gear, indicators, etc. in the right way and would not be able to do this while talking or listening to the radio. Once automaticity in driving is acquired, the driver is aware of the high-level tasks that are carried out, such as driving to office, turning to the right and waiting at a traffic light, but is not aware about low-level details such as changing gear, using the indicator and the colour of the traffic light, amber or red, while stopping at a traffic light (*conscious automaticity*).

Among several theories that attempt to explain attention and automaticity, in this paper we consider Norman and Shallice Model [NS86]. According to this model, the majority of responses are under fairly automatic control. They are triggered by environmental cues that contact specific schemata, which define the routine activities associated with a given task. For example, while driving, the noise from the engine is a cue that contacts the schema for using the gear and triggers the driver to change gear: there is no conscious decision in changing gear, and later no recollection of this specific action. Norman and Shallice propose a *contention scheduling* mechanism to solve a clash between two enabled routine activities. For example, while driving and approaching a traffic light that turns to amber, we might either stop or speed up through the crossing. In order to select the correct action, the mechanism to solve this clash needs to assess which action is more effective in achieving the goal of the current task. For example, if the driver is late and perceives driving through the crossing as safe, then speeding up may be the chosen action.

In general, behaviour is not simply a set of automatic routine activities, but in many occasions deliberate and conscious actions are required. For example, a driver normally used to drive on the right side of the road, as in continental Europe, who is driving in UK or Ireland and is approaching a roundabout, cannot use the automatic schema that dictates to look to the left and turn to the right. This schema is now inappropriate because traffic drives on the left rather than on the right. Norman and Shallice propose a *Supervisory Activating System* (SAS), which becomes active whenever none of the routine selection schemata are appropriate. Typical situations that activate the SAS, and associated examples related to the driving task, are listed as follows:

required decision as it is the case when road signs are in conflict;

expectation failure which can be assessed as

- **hazard** as it is the case for an unexpected sound from the engine;
- **novelty** as it is the case for driving a car different from the usual one or the above mentioned situation of driving on the left rather than on the right;

curiosity for something we see, which may urge us to consciously slow down to better see what is happening;

temptation such as the sight of a stall selling some food we are craving for, which may urge us to consciously stop to purchase it;

anger which may be caused by another driver honking to ask for space to overtake.

In this paper we focus on user's expectations. Therefore, we will incorporate in our model deliberate and conscious behaviour that results from the assessment of either a hazard or a novelty. This assessment is in most cases based on previous experience. For example, experience tell us that unexpected sounds from the engine are symptoms of mechanical problems. However, if there is no previous experience, as in the case of driving for the first time on the left, our mental model of the task will help. For example our mental model of traffic circulation allows us to work out that when traffic drives on the left, we need to give way to the right and turn to the left at a roundabout.

3 Formal Framework

The notation that we use throughout the paper is based upon Hoare's notation for describing Communicating Sequential Processes (CSP) [Hoa85]. We will use the following *CSP operators*:

prefix $a \rightarrow b$
defines the sequentialisation of two events a and b;

external choice $P \Box Q$
defines a choice between two possible ordering of actions, whereby the choice is driven by an external process;

parallel composition $P \,|[\mathscr{S}]|\, Q$
forces the synchronisation of those actions in set $\mathscr{S}$ (called synchronisation set) that are offered by both processes, while allowing all other actions to occur independently;

interleaving $P \,|||\, Q$
is a special case of parallel composition with empty synchronisation set:

$$P \,|||\, Q = P \,|[\emptyset]|\, Q.$$

The sort of a process is the set of events that the process may offer throughout its evolution. In the following we also use representations for a multiple choice:

$\Box_i \; P(e_i) = P(e_1) \Box \; P(e_2) \Box \; \ldots \; \Box P(e_n)$
$\Box_{i \neq j} \; P(e_i) = P(e_1) \Box \; P(e_2) \Box \; \ldots \; \Box P(e_{j-1}) \Box \; P(e_{j+1}) \Box \; \ldots \; \Box P_n$
$\Box_{e \in \mathscr{D}} \; P(e) = P(e_1) \Box \; P(e_2) \Box \; \ldots \; \Box P(e_k)$

where e_i for $i = 1, \ldots, n$ and e are events, $P(e_i)$ processes containing events e_i and $\mathscr{D} = \{e_1, e_2, \ldots, e_k\}$, with $k < n$.

3.1 Automaticity Rules

In this section we illustrate how to formalise Norman and Shallice's routine activity schemata. We define a schema through an *automaticity rule* consisting of a *guarding condition* and an *action*. Possible guarding conditions are: knowledge about the current status of the interaction, perception of a stimulus, intrinsic or extrinsic motivation, judgment of a situation, etc.

3.1.1 Closure

An important phenomenon that occurs in automatic behaviour is *closure* [DFAB98]. When some part of an ongoing task has been completed there is a tendency to flash out short-term memory to be ready to start a new task. This may cause the removal from short-term memory of some important subtasks that are still not completed and result in some form of failure of the main task, usually called *post-completion error*. Undesired closure most commonly occurs when the main goal of the task is achieved before completing some subsidiary tasks. A classical example is provided by an Automatic Teller Machine (ATM) that delivers cash before returning the card. Since the user's main goal is to get cash, once the cash is collected the short-term memory is flushed and the user may leave the interaction forgetting the card in the ATM. That is why modern ATMs return the card before delivering cash.

Closure has been formally modeled in previous work using Higher Order Logic (HOL) [CB00]. In our approach, in order to model closure, we distinguish two kinds of actions as follows.

goal action whose execution directly results in the achievement of a goal;

task action whose execution does not directly result in the achievement of a goal.

Goal actions activate a closure process, whereas task actions do not. This is modeled in CSP as follows:

$$\begin{aligned}
&\mathbf{GoalAction}(\mathbf{cond_i}, \mathbf{action_i}, \mathbf{closure_i}) = \\
&\quad \mathbf{cond_i} \rightarrow \mathbf{GoalAction} \\
&\quad \Box\ \mathit{start} \rightarrow (\mathit{leave} \rightarrow \mathbf{GoalAction} \\
&\qquad \Box_{k \neq i}\ \mathbf{closure_k} \rightarrow \mathit{leave} \rightarrow \mathbf{GoalAction} \\
&\qquad \Box\ \mathbf{cond_i} \rightarrow \mathbf{action_i} \rightarrow \mathbf{closure_i} \rightarrow \mathit{leave} \rightarrow \mathbf{GoalAction})
\end{aligned}$$

$$\begin{aligned}
&\mathbf{TaskAction}(\mathbf{cond_j}, \mathbf{action_j}) = \\
&\quad \mathbf{cond_j} \rightarrow \mathbf{TaskAction} \\
&\quad \Box\ \mathit{start} \rightarrow (\mathit{leave} \rightarrow \mathbf{TaskAction} \\
&\qquad \Box_{k}\ \mathbf{closure_k} \rightarrow \mathit{leave} \rightarrow \mathbf{TaskAction} \\
&\qquad \Box\ \mathbf{cond_j} \rightarrow \mathbf{action_j} \rightarrow \Box_k\ \mathbf{closure_k} \rightarrow \mathit{leave} \rightarrow \mathbf{TaskAction})
\end{aligned}$$

with general processes (**GoalAction** and **TaskAction**) and categories of events (conditions **cond**, actions **action** and closures **closure**) are in bold.

Each **GoalAction** or **TaskAction** process defines one *automaticity rule*. Action *start* is used to mark the interaction start and action *leave* to mark the user leaving the interaction. After starting the interaction, the user is free to leave the interaction anytime, that is, either without performing any action or after performing any action. For a goal action defined by process **GoalAction** and triggered by event $\mathbf{cond_i}$, the $\mathbf{closure_i}$ event occurs only after the $\mathbf{action_i}$ event, whereas any other $\mathbf{closure_k}$ event, with $k \neq i$, may occur anytime and is independent of the occurrence of $\mathbf{action_i}$ event. For a task action defined by process **TaskAction** and triggered by event $\mathbf{cond_j}$, any $\mathbf{closure_k}$ event may occur anytime and is independent of the occurrence of the $\mathbf{action_j}$ event. Thus, while composing in parallel processes **GoalAction** and **TaskAction** and forcing their synchronisation on all $\mathbf{closure}_k$ events as follows

$$\mathbf{GoalAction}(\mathbf{cond_i}, \mathbf{action_i}, \mathbf{closure}_i)\ |[\cup_k \{\mathbf{closure}_k\}]|\ \mathbf{TaskAction}(\mathbf{cond_j}, \mathbf{action_j})$$

such synchronisation may only occurs after a $\mathbf{action_k}$ event.

3.1.2 Contention Scheduling

If we consider the automaticity rule that controls a driver's behaviour while approaching an amber light, which exhibits a case of contention scheduling, as we pointed out in Section 2, the driver has two possible responses: (1) stop at the traffic light; (2) speed up through the crossing. Response (1) is the safer. However, as mentioned in Section 2, if the driver is in a hurry and perceives that driving through the crossing is safe, then response (2) will be chosen. Let *stop* be the event that models response (1) and *gothrough* be the event that models response (2).

Let *safe* be the event that models the perception of the driver that going through the amber light is safe and *unsafe* its complementary event. Let *hurry* be the event that models driver's hurry and *nohurry* the event that models driver's absence of hurry. A possible CSP model of the driver behaviour while approaching an amber light is as follows.

$$\begin{aligned} AmberLight \quad = \quad & hurry \rightarrow (safe \rightarrow gothrough \rightarrow AmberLight \\ & \qquad \Box \ unsafe \rightarrow stop \rightarrow AmberLight) \\ \Box \quad & nohurry \rightarrow stop \rightarrow AmberLight \end{aligned}$$

Processes **GoalAction** or **TaskAction** defined in Section 3.1.1 consider only one single guarding condition. However, we have seen in the example above that contention scheduling usually requires several nested conditions to establish priorities for contention resolution. Therefore, we define a parametric **Contention** process as follows:

$$\begin{aligned} & \mathbf{Contention}(\mathbf{Proc}, \mathbf{Proc_k}, \mathbf{cond_i}, \mathbf{Proc_i}, \mathbf{cond_j}, \mathbf{Proc_j}) = \mathbf{Proc} = \\ & \quad \mathbf{cond_i} \rightarrow \mathbf{Proc_i}(\mathbf{Proc_k}) \ \Box \ \mathbf{cond_j} \rightarrow \mathbf{Proc_j}(\mathbf{Proc_k}) \end{aligned}$$

where **Proc** is a name assigned to the **Contention** process while $\mathbf{Proc_k}$ is a name process used in the recursion that occurs within the $\mathbf{Proc_i}$ and $\mathbf{Proc_j}$ processes. In general, there will be several nestings of *contention* processes with the toplevel having $\mathbf{Proc} = \mathbf{Proc_k}$ and defining the automaticity rule. Therefore the example above can be expressed through a nesting of *contention* processes as follows:

$$\begin{aligned} & \mathbf{Contention}(AmberLight, AmberLight, hurry, InaHurry, nohurry, NotInaHurry) = \\ & AmberLight = \\ & \quad hurry \rightarrow \mathbf{Contention}(InaHurry, AmberLight, \\ & \qquad\qquad safe, \mathbf{TaskAction}(AmberLight, amber, gothrough), \\ & \qquad\qquad unsafe, \mathbf{TaskAction}(AmberLight, amber, stop)) \\ & \quad \Box \ nohurry \rightarrow \mathbf{TaskAction}(AmberLight, amber, stop) \end{aligned}$$

Note that this required to extend the **GoalAction** and **TaskAction** processes by including as argument a name for the process to be used in the recursion as follows.

$$\mathbf{GoalAction}(\mathbf{Proc}, \mathbf{cond_i}, \mathbf{action_i}, \mathbf{closure_i}) =$$

$$\begin{aligned}
&\quad \mathbf{cond_i} \rightarrow \mathbf{Proc} \\
&\Box\ start \rightarrow (leave \rightarrow \mathbf{Proc} \\
&\qquad \Box_{k \neq i}\ \mathbf{closure_k} \rightarrow leave \rightarrow \mathbf{Proc} \\
&\qquad \Box\ \mathbf{cond_i} \rightarrow \mathbf{action_i} \rightarrow \mathbf{closure_i} \rightarrow leave \rightarrow \mathbf{Proc})
\end{aligned}$$

$$\begin{aligned}
&\mathbf{TaskAction}(\mathbf{Proc}, \mathbf{cond_j}, \mathbf{action_j}) = \\
&\quad \mathbf{cond_j} \rightarrow \mathbf{Proc} \\
&\Box\ start \rightarrow (leave \rightarrow \mathbf{Proc} \\
&\qquad \Box_k\ \mathbf{closure_k} \rightarrow leave \rightarrow \mathbf{Proc} \\
&\qquad \Box\ \mathbf{cond_j} \rightarrow \mathbf{action_j} \rightarrow \Box_k\ \mathbf{closure_k} \rightarrow leave \rightarrow \mathbf{Proc})
\end{aligned}$$

Finally, to show how recursion works within nested **Contention** processes, we expand the **Contention**-based definition of process *AmberLight* above as follows:

$$\begin{aligned}
&\mathbf{Contention}(AmberLight, AmberLight, hurry, InaHurry, nohurry, NotInaHurry) \\
&= AmberLight \\
&= hurry \rightarrow InaHurry\ \Box\ nohurry \rightarrow NotInaHurry
\end{aligned}$$

where

$$\begin{aligned}
&InaHurry \\
&= safe \rightarrow \mathbf{TaskAction}(AmberLight, amber, gothrough) \\
&\quad \Box\ unsafe \rightarrow \mathbf{TaskAction}(AmberLight, amber, stop) \\
&= safe \rightarrow amber \rightarrow gothrough \rightarrow AmberLight \\
&\quad \Box\ unsafe \rightarrow amber \rightarrow stop \rightarrow AmberLight
\end{aligned}$$

$$\begin{aligned}
&NotInaHurry \\
&= \mathbf{TaskAction}(AmberLight, amber, stop) \\
&= amber \rightarrow stop \rightarrow AmberLight
\end{aligned}$$

Since including process names as arguments of the **Contention**, **GoalAction** and **TaskAction** processes, although needed for formal rigour, decreases readability, in the rest of the paper we will use simplified notation $\mathbf{Contention}(\mathbf{cond_i}, \mathbf{Proc_i}, \mathbf{cond_j}, \mathbf{Proc_j})$ rather than full notation $\mathbf{Contention}(\mathbf{Proc}, \mathbf{Proc_k}, \mathbf{cond_i}, \mathbf{Proc_i}, \mathbf{cond_j}, \mathbf{Proc_j})$ as well as the shorter notation for processes **GoalAction** and **TaskAction** defined in Section 3.1.1.

3.1.3 Composition of Automaticity Rules

In this section we show how to use the CSP parallel composition to combine automaticity rules. There are two tricky issues to take into account while composing automaticity rules:

1. identify the correct synchronisation set;

2. deal with rules that incorporate contention scheduling.

Obvious members of the synchronisation set are $\mathbf{closure_i}$ events. Moreover, an event may occur both as a condition in an automaticity rule and as an action in another automaticity rule.

Such an event needs to be included in the synchronisation set of the parallel composition of these two automaticity rules. If an event occurs as a condition in two automaticity rules then there should be no synchronisation on such event, since either of the two automaticity rules may be non-deterministically triggered by this condition. If an event occurs as an action in two automaticity rules then there should be no synchronisation on such event, since the two rules are triggered by distinct conditions.

The presence of multiple (possibly nested) conditions in the contention scheduling requires the definition of a recursive function that extracts from the layers of nestings all conditions on which synchronisation may be needed.

Let $\mathbf{Action_i}$ denote either $\mathbf{GoalAction}(\mathbf{cond_i}, \mathbf{action_i}, \mathbf{closure_i})$ or $\mathbf{TaskAction}(\mathbf{cond_i}, \mathbf{action_i})$. We can build a set $\mathscr{R}$ of triples each consisting of one automaticity rule $\mathbf{Action_{i,j}}$, one set of actions $\mathscr{A}_{\mathbf{i,j}}$ and one set of conditions $\mathscr{C}_{\mathbf{i,j}}$ as follows

- $\mathbf{Action_{i,i}} = \mathbf{Action_i}$, $\mathscr{A}_{\mathbf{i,i}} = \{\mathbf{action_i}\}$, $\mathscr{C}_{\mathbf{i,i}} = \{\mathbf{cond_i}\}$
  if $i > 0$
- $\mathbf{Action_{i,j}} = \mathbf{Contention}(\mathbf{cond_i}, \mathbf{Action_{(i+1),(k-1)}}, \mathbf{cond_k}, \mathbf{Action_{(k+1),j}})$,
  $\mathscr{A}_{\mathbf{i,j}} = \mathscr{A}_{\mathbf{i+1,k-1}} \cup \mathscr{A}_{\mathbf{k+1,j}}$,
  $\mathscr{C}_{\mathbf{i,j}} = \{\mathbf{cond_i}, \mathbf{cond_k}\} \cup \mathscr{C}_{\mathbf{i+1,k-1}} \cup \mathscr{C}_{\mathbf{k+1,j}}$,
  if $0 < i < k < j$.

The parallel composition of automaticity rules $\mathbf{AutomaticityRules} = \mathbf{AutomaticityRules_n}$ is built

1. by selecting a triple $\langle \mathbf{Action}, \mathscr{A}, \mathscr{C} \rangle \in \mathscr{R}$ and defining
   (a) $\mathbf{AutomaticityRules_1} = \mathbf{Action}$
   (b) $\mathscr{A}_\mathbf{1} = \mathscr{A}$
   (c) $\mathscr{C}_\mathbf{1} = \mathscr{C}$
   (d) $\mathscr{R}_\mathbf{1} = \mathscr{R} - \{\langle \mathbf{Action}, \mathscr{A}, \mathscr{C} \rangle\}$
2. by iteratively selecting the other triples $\langle \mathbf{Action}, \mathscr{A}, \mathscr{C} \rangle \in \mathscr{R}$ and defining
   (a) $\mathbf{AutomaticityRules_m} = \mathbf{AutomaticityRules_{m-1}} \,||[\mathscr{S} \cup (\cup_k \{\mathbf{closure}_k\})]||\, \mathbf{Action}$
   where $\mathscr{S} = (\mathscr{A}_\mathbf{m-1} \cap \mathscr{C}) \cup (\mathscr{C}_\mathbf{m-1} \cap \mathscr{A})$
   (b) $\mathscr{A}_\mathbf{m} = \mathscr{A}_\mathbf{m-1} \cup \mathscr{A}$
   (c) $\mathscr{C}_\mathbf{m} = \mathscr{C}_\mathbf{m-1} \cup \mathscr{C}$
   (d) $\mathscr{R}_\mathbf{m} = \mathscr{R}_\mathbf{m-1} - \{\langle \mathbf{Action}, \mathscr{A}, \mathscr{C} \rangle\}$

   until $\mathscr{R}_\mathbf{n} = \emptyset$

3.2 Modelling the Supervisory Activating System

We have seen in Section 2 that the Supervisory Activating System (SAS) becomes active whenever the routine selection of operations becomes inappropriate. In this paper we restrict our analysis to the case of *expectation failure*.

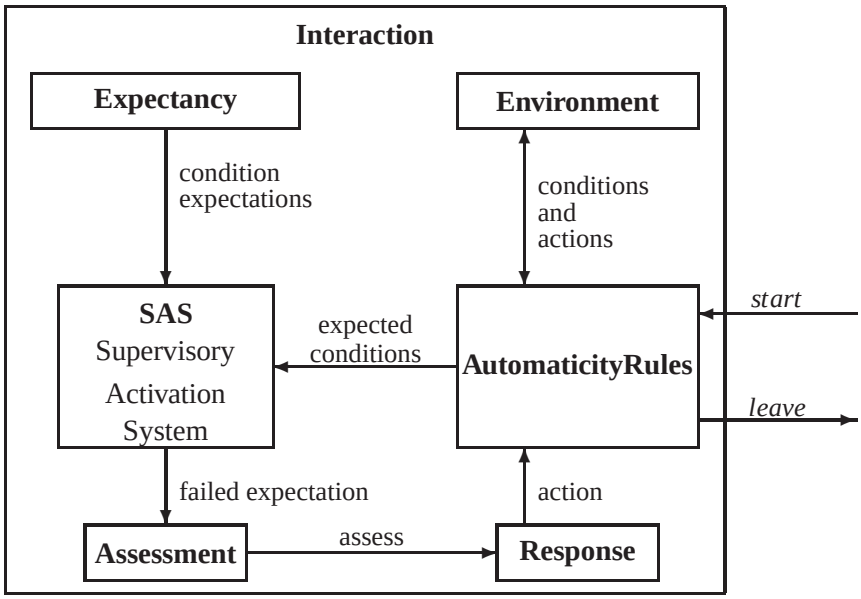

Figure 1: General Architecture

The architecture of the SAS is depicted in Figure 1. Action *start* is used to mark the beginning of the interaction and action *leave* to mark the user's act of leaving the interaction.

The **AutomaticityRules** process consists of the parallel composition of the **GoalAction** and **TaskAction** processes defined in Section 3.1, synchronising on all actions describing closures, and further synchronises on all conditions and actions involved in the human-computer interaction through the parallel composition with the **Environment** process.

The **Environment** process defines the environment in which the interaction occurs, including not only computer interfaces and possibly electrical and mechanical components, but also subjective and social contexts as well as personal, cultural and general perceptions and views. For example, the fact that a driver is in a hurry is a subjective context while the fact that the driver perceives driving through an amber light as safe is a personal perception.

The **Expectancy** process consists of a multiple external choice

$$\textbf{Expectancy} \quad = \quad \Box_i \, \textbf{Expectation}(\textbf{cond}_\textbf{i}, \textbf{expect}_\textbf{i})$$

where each $\textbf{Expectation}(\textbf{cond}_\textbf{i}, \textbf{expect}_\textbf{i})$, for $i = 1, ..., n$, is defined as follows

$$\textbf{Expectation}(\textbf{cond}_\textbf{i}, \textbf{expect}_\textbf{i}) \quad = \quad \textbf{cond}_\textbf{i} \rightarrow \textbf{expect}_\textbf{i} \rightarrow \textbf{Expectancy}$$

with $\textbf{expect}_\textbf{i}$ the event that characterises the user's expectation determined by condition $\textbf{cond}_\textbf{i}$.

In our model we assume at most one expectation associated with one condition. However, in general, one condition may lead to distinct expectations. Process $\textbf{Expectation}(\textbf{cond}_\textbf{i}, \textbf{expect}_\textbf{i})$ could be generalised to the parallel composition over i and k of processes

$$\textbf{Expectation}(\textbf{cond}_\textbf{i}, \textbf{expect}_\textbf{k}) \quad = \quad \textbf{cond}_\textbf{i} \rightarrow \textbf{expect}_\textbf{k} \rightarrow \textbf{Expectancy}$$

but this additional complexity would hinder the illustrative purpose of the model.

Let us define $\mathscr{E} = \cup_i \{\textbf{expect}_\textbf{i}\}$, the set of all expectations, and $\mathscr{EC} = \{\textbf{cond}_\textbf{i} \mid \textbf{expect}_\textbf{i} \in \mathscr{E}\}$, the set of conditions which raise these expectations. For each specific instantiation of our model we need to define the set $\mathscr{CC}$ of all conditions that occur immediately after expectations, each condition either meeting or failing the corresponding expectation.

The **SAS** process synchronises with the condition expectations ($\textbf{expect}_\textbf{i} \in \mathscr{E}$) offered by process **Expectancy** and with all conditions ($\textbf{cond}_\textbf{i} \in \mathscr{EC}$) that are offered by the **AutomaticityRules** process and raise such expectations, and then offers events that express whether each expectation ($\textbf{expect}_\textbf{i}$) is met (*met*) by condition $\textbf{cond} \in \mathscr{CC}$ or not ($\textbf{failed}_\textbf{i}$). Let $\mathscr{F}$ denote the set of all expectation failures for the specific instantiation of our model. The **SAS** process is defined as follows.

$$\begin{aligned}\textbf{SAS} = {} & \Box_{\textbf{cond} \in \mathscr{CC}}\ \textbf{cond} \rightarrow \textbf{SAS} \\ & \Box\ \mathit{start} \rightarrow \textbf{ExpectCheck}\end{aligned}$$

$$\begin{aligned}\textbf{ExpectCheck} = {} & \mathit{leave} \rightarrow \textbf{SAS} \\ & \Box_i\ \textbf{expect}_\textbf{i} \rightarrow \Box_{\textbf{cond} \in \mathscr{CC}}\ \textbf{cond} \rightarrow \textbf{outcome} \rightarrow \textbf{ExpectCheck}\end{aligned}$$

where $\textbf{outcome} = \mathit{met}$ if the expectation is met and $\textbf{outcome} \in \mathscr{F}$ otherwise.

Events $\textbf{outcome} \in \{\mathit{met}\} \cup \mathscr{F}$ are offered by the **SAS** process to the **Assessment** process, which assesses for each expectation failure ($\textbf{failed} \in \mathscr{F}$) whether this is considered by the user ($\textbf{perception}_\textbf{failed}$) as a hazard or a novelty. This is modeled in CSP as follows.

$$\begin{aligned}\textbf{Assessment} = {} & \mathit{met} \rightarrow \Box_{\textbf{failed} \in \mathscr{F}}\ \textbf{action}_\textbf{failed} \rightarrow \textbf{Assessment} \\ & \Box_{\textbf{failed} \in \mathscr{F}}\ \textbf{failed} \rightarrow \textbf{action}_\textbf{failed} \rightarrow \textbf{perception}_\textbf{failed} \rightarrow \textbf{Assessment}\end{aligned}$$

The user's conscious and deliberate response is modeled in CSP as follows.

$$\begin{aligned}\textbf{Response}(\textbf{perception}, \textbf{action}) = {} & \textbf{perception} \rightarrow \textbf{action}_\textbf{perception} \rightarrow \textbf{Response}_\textbf{i} \\ & \Box\ \textbf{action}_\textbf{perception} \rightarrow \textbf{Response}\end{aligned}$$

Let $\mathscr{P} = \{\textbf{perception}_\textbf{failed} \mid \textbf{failed} \in \mathscr{F}\}$ be the set of perceptions (hazard or novelty). We define the set of interactions $\mathscr{I}$ as the set of conditions, and possibly their negations, and actions that are also events of the sort of **Environment**.

The overall interaction process is modeled in CSP as follows

$$\begin{aligned}\textbf{Interaction} = {} & \textbf{Environment}\ |[\mathscr{I}]| \\ & \textbf{AutomaticityRules}\ |[\{\mathit{start}, \mathit{leave}\} \cup \mathscr{CC}]| \\ & \textbf{SAS}\ |[\{\mathit{leave}\}]| \\ & \textbf{Response}(\textbf{hazard}, \mathit{leave})\ |[\{\textbf{action}_\textbf{novelty}\}]| \\ & \textbf{Response}(\textbf{novelty}, \textbf{action}_\textbf{novelty})\ |[\mathscr{EC} \cup \mathscr{E}]| \\ & \textbf{Expectancy}\ |[\{\mathit{met}\} \cup \mathscr{F} \cup \mathscr{P}]| \\ & \textbf{Assessment}\end{aligned}$$

where $\textbf{action}_\textbf{novelty}$ is a specific action with which the user responds to the novelty (**novelty**).

We have assumed that the user's response to a perceived hazard (**hazard**) is to leave the in-

teraction (action *leave*). What this means depends on the specific interaction and an additional process may be requested to fully specify such response.

Furthermore, for simplicity we have considered only one novelty and one hazard, but in general a variety of novelties and hazards may occur during interaction. Therefore the parallel composition may include several **Response** processes, each associated with distinct novelty and hazard events.

3.2.1 Driving Case Study: Model

Let us consider the task of driving to office, whose goal is to park the car in the reserved bay in the company car park. For simplicity, we model the entire trip as just the task of driving through one traffic light between departure and arrival.

Using processes **TaskAction** and **GoalAction** defined in Section 3.1.1 and process **Contention** defined in Section 3.1.2 we can define process *DriveRules* that incorporates all automaticity rules for the driving task as follows.

$$\begin{aligned} DriveRules \;=\; & (\mathbf{TaskAction}(depart, light) \;|[\{arrived\}]| \\ & \mathbf{TaskAction}(green, gothrough) \;|[\{arrived\}]| \\ & \mathbf{Contention}(hurry, \mathbf{Contention}(safe, \mathbf{TaskAction}(amber, gothrough), \\ & \qquad unsafe, \mathbf{TaskAction}(amber, stop)), \\ & \quad nohurry, \mathbf{TaskAction}(amber, stop)) \;|[\{arrived\}]| \\ & \mathbf{TaskAction}(red, stop) \;|[\{gothrough, arrived\}]| \\ & \mathbf{GoalAction}(gothrough, free) \;|[\{free, arrived\}]| \\ & \mathbf{GoalAction}(free, park, arrived) \end{aligned}$$

After the car departs (*depart*) it approaches the traffic light (*light*), which can be *green*, *amber* or *red*. The car may stop at the traffic light (*stop*), or go through it (*gothrough*), after or without stopping. Once arrived at the car park, if the driver's personal parking bay is free (*free*), the driver parks the car (*park*). The task is completed and closure is attained (*arrived*).

We have only one expectation

$$DriveExpect = \mathbf{Expectation}(gothrough, expectfree)$$

in which the driver expects to find the personal bay free at the arrival at the company car park. The associated condition is *gothrough* because in our simplified model, going through the traffic light is the action that immediately precedes the arrival at the car park.

The *DriveSAS* process is modelled as follows

$$\begin{aligned} DriveSAS = {} & free \rightarrow DriveSAS \;\Box\; occupied \rightarrow DriveSAS \\ & \Box\; start \rightarrow DriveExpectCheck \end{aligned}$$

$$\begin{aligned} DriveExpectCheck = {} & leave \rightarrow DriveSAS \\ & \Box\; expectfree \rightarrow\; (free \rightarrow met \rightarrow DriveExpectCheck \\ & \qquad \Box\; occupied \rightarrow freefailed \rightarrow DriveExpectCheck) \end{aligned}$$

The set of expectations is $\mathscr{E}_{Drive} = \{expectfree\}$ and the set of associated conditions is $\mathscr{EC}_{Drive} = \{gothrough\}$. The set of conditions that occur after the unique expectation *expectfree* is $\mathscr{CC}_{Drive} = \{free, occupied\}$. If the personal bay is free (*free*) then the expectation is met (*expectmet*). If the personal bay is occupied (*occupied*) then the expectation fails (*freefailed*). The set of expectation failures is $\mathscr{F}_{Drive} = \{freefailed\}$ and the set of perceptions is $\mathscr{P}_{Drive} = \{\textbf{novelty}\}$

The assessment of expectation failures is defined as follows

$$\begin{aligned} DriveAssess = {} & met \rightarrow \Box_j \ \textbf{action}_\textbf{j} \rightarrow DriveAssess \\ & \Box \ freefailed \rightarrow otherfree \rightarrow \textbf{novelty} \rightarrow DriveAssess \end{aligned}$$

Note that since our simple model does not include an expectation failure that is perceived as a hazard, *DriveAssess* does not offer an **hazard** event. The expectation failure urges the driver to look for another bay where to park (*otherfree*) and, when this is found, the expectation is assessed as a novelty (**novelty**) and the car parked (*park*). Therefore, the only response process is **Response**(**novelty**, *park*). Notice that we assume here that there is always a free parking bay. To release this assumption would require two distinct novelty events with two different responses.

The environment part consists of the traffic light and the conditions that allow the driver to assess whether to drive through the amber light. Let us suppose that the driver perceives that it is always unsafe to drive through an amber light and that the driver is in a hurry. The environment is modelled as follows.

$$DriveEnv = DriveTrafficLight \ ||| \ Hurry \ ||| \ UnSafe$$

$$\begin{aligned} & DriveTrafficLight = light \rightarrow (GreenLight \ \Box \ AmberLight \ \Box \ RedLight) \\ & GreenLight = green \rightarrow GreenLight \ \Box \ change \rightarrow AmberLight \\ & AmberLight = amber \rightarrow AmberLight \ \Box \ change \rightarrow RedLight \\ & RedLight = red \rightarrow RedLight \ \Box \ change \rightarrow GrenLight \end{aligned}$$

$$Hurry = hurry \rightarrow Hurry$$

$$UnSafe = unsafe \rightarrow UnSafe$$

The set of interactions is $\mathscr{I} = \{light, green, amber, red, hurry, nohurry, safe, unsafe\}$. Therefore, the overall interaction process is modeled in CSP as follows

$$\begin{aligned} DriveInteraction = {} & DriveEnv \ |[\mathscr{I}_{Drive}]| \\ & DriveRules \ |[\{start, leave, free, occupied\}]| \\ & DriveSAS \ |[\{leave\}]| \\ & \textbf{Response}(\textbf{novelty}, park) \ |[\{gothrough, expectfree\}]| \\ & DriveExpect \ |[\{met, freefailed, \textbf{novelty}\}]| \\ & DriveAssess \end{aligned}$$

3.2.2 Automatic Teller Machine (ATM) Case Study: Model

We consider a simple ATM task in which the user is triggered to insert a card (*cardI*) when the interface shows (through a message on the display) that the machine is ready (*ready*), is triggered

to enter the pin (*pinE*) by the knowledge that the card has been inserted (*cardI*), is triggered to collect the card (*cardC*) when seeing the card out (*cardO*) and is triggered to collect the cash (*cashC*) when seeing the cash out (*cashO*).

$$\begin{aligned} ATMsRules \quad = \quad & \textbf{TaskAction}(ready, cardI) \ |[\{cardI, cashgot\}]| \\ & \textbf{TaskAction}(cardI, pinE) \ |[\{cashgot\}]| \\ & \textbf{TaskAction}(cardO, cardC) \ |[\{cashgot\}]| \\ & \textbf{GoalAction}(cashO, cashC, cashgot) \end{aligned}$$

Note that we have implicitly assumed that the user knows that the card is always inserted before entering the pin, which is anyway the case for any ATM we have encountered in our experience.

We can observe that user's automatic behaviour does not imply any order between the actions of collecting the card and collecting the cash; such order is dictated by the specific ATM with which the user interacts. In particular, we may consider the following ATMs.

$$\begin{aligned} ATM_1 \quad = \quad & ready \rightarrow cardI \rightarrow pinE \rightarrow cashO \rightarrow cashC \rightarrow cardO \rightarrow cardC \rightarrow ATM_1 \\ ATM_2 \quad = \quad & ready \rightarrow cardI \rightarrow pinE \rightarrow cardO \rightarrow cardC \rightarrow cashO \rightarrow cashC \rightarrow ATM_2 \end{aligned}$$

These are two possible interaction environments. Environment ATM_1 delivers the cash before returning the card, whereas ATM_2 returns the card before delivering the card.

Depending on which kind of ATM, ATM_1 or ATM_2, the user has experience with, we have respectively either expectations

$$ATMsExpect_1 = \textbf{Expectation}(pinE, expectcash) \ \Box \ \textbf{Expectation}(cashC, expectcard)$$

or expectations

$$ATMsExpect_2 = \textbf{Expectation}(pinE, expectcard) \ \Box \ \textbf{Expectation}(cardC, expectcash)$$

A user who has experience with ATM_1, which delivers cash before returning the card, expects to have the cash delivered (expectation *expectcash*) immediately after entering the pin (*pinE*) and expects to have the card returned (expectation *expectcard*) immediately after collecting the cash (*cashC*). Analogously, a user who has experience with ATM_2, which returns the card before delivering cash, expects to have the card returned (expectation *expectcard*) immediately after entering the pin (*pinE*) and expects to have the cash delivered (expectation *expectcash*) immediately after collecting the card (*cardC*).

The *ATMsSAS* process is modelled as follows

$$\begin{aligned} ATMsSAS = \ & cardO \rightarrow ATMsSAS \ \Box \ cashO \rightarrow ATMsSAS \\ & \Box \ start \rightarrow ATMsExpectCheck \end{aligned}$$

$$\begin{aligned} ATMsExpectCheck = \ & leave \rightarrow ATMsSAS \\ & \Box \ expectcard \rightarrow \ (cardO \rightarrow met \rightarrow ATMsExpectCheck \\ & \qquad \Box \ cashO \rightarrow cashfailed \rightarrow ATMsExpectCheck) \\ & \Box \ expectcash \rightarrow \ (cashO \rightarrow met \rightarrow ATMsExpectCheck \\ & \qquad \Box \ cardO \rightarrow cardfailed \rightarrow ATMsExpectCheck) \end{aligned}$$

The set of expectations is $\mathscr{E}_{ATM} = \{expectcard, expectcash\}$. The sets of associated conditions are $\mathscr{EC}_{ATM1} = \{pinE, cashC\}$ for ATM_1 and $\mathscr{EC}_{ATM2} = \{pinE, cardC\}$ for ATM_2. The set of conditions that occur after the two expectations is $\mathscr{CC}_{ATM} = \{cashO, cardO\}$. The expectations are not met if cash is delivered (*cashO*) when card is expected (*expectcard*) and if card is delivered (*cardO*) when cash is expected (*expectcash*) with resultant set of expectation failures $\mathscr{F}_{ATM} = \{cardfailed, cashfailed\}$. The set of perceptions is $\mathscr{P}_{ATM} = \{\textbf{novelty}, \textbf{hazard}\}$

The assessment of expectation failures is defined as follows

$$\begin{aligned} ATMsAssess = met \rightarrow (cashC \rightarrow ATMsAssess \\ \Box\ cardC \rightarrow ATMsAssess) \\ \Box\ cashfailed \rightarrow cardC \rightarrow \textbf{hazard} \rightarrow ATMsAssess \\ \Box\ cardfailed \rightarrow cashC \rightarrow \textbf{novelty} \rightarrow ATMsAssess \end{aligned}$$

A user who is returned the card when expecting to be delivered cash (*cashfailed*) may suspect that there was some problem with the authentication process and perceive the hazard to have the card confiscated if attempting to perform the transaction again. On the other hand, a user who is delivered cash when expecting to be returned the card (*cardfailed*) perceives the situation just as a novelty that does not activate any specific response. Therefore, the only response to expectation failure occurs when a hazard has been assessed and consists of leaving the interaction as expressed by process **Response**(**hazard**, *leave*).

The set of interactions is $\mathscr{I} = \{ready, cardI, pinE, cashO, cashC, cardO, cardC\}$. Therefore, we consider four overall interaction processes, for $i, j = 1, 2$, which are modeled in CSP as follows

$$\begin{aligned} ATMsInteraction(i,j) = ATM_i\ &|[\mathscr{I}_{ATM}]| \\ &ATMsRules\ |[\{start, leave, cashO, cardO\}]| \\ &ATMsSAS\ |[\{leave\}]| \\ &\textbf{Response}(\textbf{hazard}, leave)\ |[\mathscr{EC}_{ATMj} \cup \{expectcard, expectcash\}]| \\ &ATMsExpect\ |[\{met, cardfailed, cashfailed, \textbf{hazard}\}]| \\ &\textbf{Assessment} \end{aligned}$$

4 Model-Checking Analysis

From an analytical point of view our focus is to verify whether the design of the interface and the other environmental components addresses cognitive aspects of human behaviour such as closure phenomena and user expectations that trigger the SAS to activate attention. Although we have seen in Section 3.1.2 that our approach supports the modelling of contention scheduling, we are not dealing with this aspect in our analysis. In fact, the assessment process automatically used to adjust routine behaviour to solve contention involves many aspects related to culture, background and beliefs, which cannot be easily incorporated in our framework. Therefore, improving environmental design to make contention scheduling more effective and beneficial is beyond the scope of this paper.

Model-checking techniques [CGP99] provide an effective analytical tool to exahustively explore the system state space and capture the behaviour that emerges from the combination of

several system components. Closure, automatic behaviour, expectancy and attention are phenomena that represent distinct components of human cognition and action, and their combination results in an apparently holistic ways of performing tasks. In this context model-checking can be used to capture errors that emerge when environmental design cannot deal with closure phenomena, or when the outcome of the interaction between automatic behaviour and environment does not meet human expectations. We use Linear Time Temporal Logic (LTTL) [MP91] to specify system properties and then we use model-checking to verify such properties on the CSP model.

4.1 Automatic Teller Machine (ATM) Case Study: Analysis

A formal specification of the ATM case study presented in Section 3.2.2 is as follows.

$$G(ready \Rightarrow (\neg cashO)\,\mathscr{U}\, cardI) \tag{1}$$
$$G(ready \Rightarrow (\neg cashO)\,\mathscr{U}\, pinE) \tag{2}$$
$$G(((cardI \wedge (\neg ready)\,\mathscr{U}\, pinE) \Rightarrow (\neg ready)\,\mathscr{U}\, cashO) \vee (pinE \wedge (\neg ready)\,\mathscr{U}\, cardI) \Rightarrow (\neg ready)\,\mathscr{U}\, cashO) \tag{3}$$
$$G(cardI \Rightarrow (\neg ready)\,\mathscr{U}\, cardO) \tag{4}$$
$$G(cardO \Rightarrow (\neg ready)\,\mathscr{U}\, cardC) \tag{5}$$
$$G(cashO \Rightarrow (\neg ready)\,\mathscr{U}\, cashC) \tag{6}$$

Formulae (1) and (2) specify respectively that inserting the card and entering the pin are required to withdraw cash. Formula (3) specifies that cash can only be delivered if the card is inserted and the correct pin entered. Formulae (5) and (6) specify that the ATM can be ready for another transaction only if card and cash are collected.

If we check the conjunction of the above properties on the four systems defined in Section 3.2.2 we find out that it holds only for $ATMsInteraction(2,2)$. This is the case of an ATM returning the card first and a user having experience with this kind of ATM. As for $ATMsInteraction(1,1)$ and $ATMsInteraction(1,2)$, in which the ATM delivers cash first, a closure that flashes the short-term memory may occur after collecting the cash, that is, after achieving the goal of the task. In this case the user may forget to collect the card and therefore property 5 does not hold (post-completion error). Finally, for $ATMsInteraction(2,1)$, the user is returned the card when expecting to be delivered cash and may suspect that there was some problem with the authentication process. This is perceived as the hazard to have the card confiscated, and the user is likely to leave the interaction without collecting cash, thus violating property 6.

This case study has been implemented using the Concurrency Workbench of the New Century [CLS00]. The code is available at:

http://www.iist.unu.edu/~antonio/Research/Software/CWB-NC/ATM/.

5 Conclusion and Future Work

We have proposed a process algebraic framework for modelling cognitive activities, such as closure, contention scheduling and attention activation, that occur in automatic routine behaviour.

Although these activities are essential to deal with limitations of short-term memory, solve contentions and respond to unexpected situations, they may cause cognitive errors in interactions with inappropriate environments. The framework we propose is general enough to be applied to very different real-life context, such as driving a car and withdrawing cash at an ATM.

Our model-checking analysis is able to capture the emergence of such cognitive errors. Post-completion errors, such as the violation of property 5 in Section 4.1, may be often avoided by designing the environment in such a way to ensure that all subtasks are completed before the goal is achieved. This is the case of ATM_2, which models an ATM that returns the card before delivering cash. This approach would always work in an ideal world where all environments are designed according to the above criterion. However, in the real world humans have to frequently deal with inappropriate environments, thus building up experience that may result adverse in interacting with "correctly" designed environments. In the context of the ATM case study, although nowadays most ATMs work as modeled by ATM_2, there are still some developing countries where all ATMs work as modelled by ATM_1. Thus we can imagine that a user from one of such countries would have problems while visiting a country where all ATMs work as modeled by ATM_2.

Our qualitative analysis supports the identification of potential errors but does not give any clue about their frequency. The development of a framework for quantitative analysis is part of our future work. We also envisage a more comprehensive framework in which qualitative analysis is used to identify potential cognitive errors that may be induced by a new environment design, while quantitative analysis is used to predict the frequency, and thus the criticality, of such errors.

Bibliography

[BBD00] R. Butterworth, A. E. Blandford, D. Duke. Demonstrating the cognitive plausability of interactive systems. *Formal Aspects of Computing* 12:237–259, 2000.

[Bro58] D. E. Broadbent. *Perception and communication*. Pergamon Press, 1958.

[Bur94] J. A. Burgh. The four horsemen of automaticity: Awareness, intetion, efficiency, and control in social cognition. In Jr. and Srull (eds.), *Handbook of Social Cognition*. Volume 1, pp. 1–40. Lawrence Erlbaum, 1994.

[CB00] P. Curzon, A. Blandford. Using a Verification System to Reason about Post-Completion Errors. In Palanque and Paterno (eds.), *Participants Proc. of DSV-IS 2000: 7th Int.Workshop on Design, Specification and Verification of Interactive Systems, at the 22nd Int. Conf. on Software Engineering*. Pp. 292–308. 2000.

[CB04] P. Curzon, A. E. Blandford. Formally Justifying User-centred Design Rules: a Case Study on Post-completion Errors. In Boiten et al. (eds.), *Integrated Formal Methods*. Lecture Notes in Computer Science 2999, pp. 461–480. Springer-Verlag, Berlin, Germany, 2004.

[CE07] A. Cerone, N. Elbegbayan. Model-checking Driven Design of Interactive Systems. In *Proceedings of FMIS 2006*. Electronic Notes in Theoretical Computer Science 183, pp. 3–20. Elevier, 2007.

[CGP99] E. M. J. Clarke, O. Grumberg, D. Peled. *Model Checking*. MIT Press, 1999.

[Che53] C. Cherry. Some experiments on the recognition of speach, with one and with two ears. *Journal of Acoustical Socity of America* 25:975–979, 1953.

[CLS00] R. Cleaveland, T. Li, S. Sims. The Concurrency Workbench of the New Century. Stony Brooke, NY, USA, 2000.
URL: `http://www.cs.sunysb.edu/~cwb`.

[DD63] J. A. Deutsch, D. Deutsch. Attention: Some theoretical considerations. *Psychological Review* 70(1):51–61, 1963.

[DFAB98] A. Dix, J. Finlay, G. Abowd, R. Beale. *Human-Computer Interaction*. Pearson Education, 1998.

[GW60] J. A. Gray, A. A. Wedderburn. Grouping strategies with simultaneous stimuli. *Quarterly Journal of Experimental Psychology* 12:180–184, 1960.

[Hoa85] C. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[Mac60] D. G. MacKey. Aspects of the theory of comprehension , memory and attention. *Quarterly Journal of Experimental Psychology* 25(1):22–40, 1960.

[MP91] Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems *Specification**. Springer-Verlag, 1991.

[NS86] D. A. Norman, T. Shallice. Attention to action: Willed and automatic control of behaviour. In Davidson et al. (eds.), *Consciousness and Self-Regulation*. Advances in Research and Theory 4. Plenum Press, 1986.

[Ran94] T. A. Ranney. Models of driving behaviour: A review of their evolution. *Accid. Annal. and Prev.* 26(6):733–750, 1994.

[RCB08] R. Rukšėenas, P. Curzon, A. Blandford. Modelling rational user behaviour as games between an angel and a demon. In Cerone and Gruner (eds.), *Proceedings of SEFM 2008*. Pp. 355–364. IEEE Press, 2008.

[SBBW09] L. Su, H. Bowman, P. Barnard, B. Wyble. Process algebraic model of attentional capture and human electrophysiology in interactive systems. *Formal Aspects of Computing* 21(6):513–539, 2009.

[SS77a] W. Schneider, R. M. Shiffrin. Controlled and automatic human information processing: 1. Detection search and attention. *Psychological Review* 84:1–66, 1977.

[SS77b] R. M. Shiffrin, W. Schneide. Controlled and automatic human information processing: 2. Perceptual learning, automatic attending and a general theory. *Psychological Review* 84:127–189, 1977.

[Str35] J. R. Stroop. Studies of interference in serial verbal reactions. *Experimental Psychology* 18:643–662, 1935.

[Tre60] A. M. Treisman. Contextual cues in selective listening. *Quarterly Journal of Experimental Psychology* 12:242–248, 1960.