Department of Computer Science

Ph.D. in Computer Science

Ph.D. Thesis

A Formal Framework for Modelling and Analysing Safety-Critical Human Multitasking

Giovanna Broccia

Supervisor:
Prof. Paolo Milazzo
Università di Pisa

To my father, for his extraordinary curiosity
To my mother, for her incredible constancy
To myself, for having taken the best from both

Abstract

Nowadays people often interact with multiple devices or with a single device performing multiple tasks. In such a multitasking context, the amount of cognitive resources required from each task (cognitive load) influences the activity of the attentional mechanisms of the user. In particular, focusing attention on a "main" task (e.g. driving a car) may be impeded by a secondary "distractor" task (e.g. using a GPS navigator system) with a high cognitive load. Moreover, the cognitive resource that is mostly involved in interactions with computers and other technological devices is the human *working memory*, which is a volatile memory with a limited capacity used to store and process the information necessary for performing a task. The concurrent use of such a resource might cause memory overloading and might get the users to forget useful information for the interaction with one of the systems. When human multitasking involves safety-critical tasks, failure to devote sufficient attention to some tasks (the safety-critical ones) could have serious consequences.

To study these kinds of problem we define a formal model of safety-critical human multitasking which describes the cognitive processes involved in HCI and the switching of attention among concurrent tasks. The model describes the attention attractiveness of each task as a factor proportional to the task cognitive load, its criticality and the time it was ignored during the interaction. The model builds on classical results from applied psychology on selective attention and working memory.

We implement the model through a simulator, which allows us to get a quick feedback about whether a human can safely perform multiple such tasks at the same time, and as an executable formal framework in Real-Time Maude, which enables us to analyse safety-critical human multitasking through simulation and reachability analysis.

We validate the model against data collected from an experimental study with real users involved in the interaction with two concurrent tasks.

We show how a number of prototypical multitasking problems can be analysed in Real-Time Maude by studying: (i) the interaction with a GPS navigation system while driving, (ii) a medical operator setting multiple infusion pumps simultaneously, and (iii) some typical scenarios involving human errors in air traffic control.

Contents

1 Introduction

Nowadays we often interact with multiple devices or with a single device performing multiple tasks at the same time. Multitasking has become surprisingly present in our life and we are used to think that carrying out more tasks simultaneously is equivalent to optimise time: keeping up several instant message conversations at once, answering an e-mail while listening to a talk at a conference, hanging out with social network while watching television, "jumping" from a website to another while completing homework assignments, just to give some examples.

However, despite what most of us could believe, the multitasking performance takes a toll on productivity and psychologists who study the mental processes involved in multitasking have found that the human mind and brain are not designed for doing more than one task at a time [78, 90]. We cannot focus on more than one thing at a time, what we can do is switch from one task to another with extraordi-

nary speed, since all these tasks use the same part of the brain [55].

1.1 Cognitive Flexibility

What we usually call multitasking is actually what researchers call *cognitive flexibility*[1], and it is the mental capability to switch between different activities in a conscious or unconscious way. Cognitive flexibility requires that each task in a quotidian sequence have an appropriate configuration of mental resources [80]: each time we choose the tasks to be performed we exercise an intentional control to select and devise a suitable task-set for our goals [49].

One of the main resource to be shared in such multitasking contexts is the *working memory* – a cognitive system responsible for the transient holding and processing of pieces of information needed to perform a task – together with the *human selective attention* – a selective activity whose purpose is to focus on one element of the environment while ignoring the others. According to the Norman and Shallice theory [83], the main role of attention is in the controlling of action and such actions can be executed under two levels of cognitive control: during *automatic control* actions come in fast succession with no consciousness, namely with no need of attention; in contrast, during *deliberate control,* actions are carried out under the intentional control of the individual, who deliberately directs the attention to them.

For instance, when we buy a new mobile phone and we "learn" to use it, we need to make a conscious effort and direct attention to each action we perform on the new device: while learning how to use the new messaging application, when learning to save a new phone number, and so on. In such case we act under deliberate control. When the learning process is completed, we act in a automatic way, without overthinking the actions we want to perform; in such case we act under automatic control.

Researchers identify two different "modes" of executing multiple tasks at a time, denoted *concurrent processing* and *sequential processing* [93, 113]. The concurrent

[1] From here onward we will continue to call it *multitasking* for the sake of simplicity

processing paradigm primarily focuses on dual tasking: two tasks are performed "simultaneously" although at a degraded level (e.g., driving while conversing or mind wandering). Under such paradigm, it is possible to perform two actions at the same time since one of the two (or even both) is performed under automatic control, thus the selective attention is not actually shared among the two tasks.

In contrast, the sequential processing mode is used when more than two tasks have to be performed and one task is chosen at a time while other tasks idle. Sequential task performance is characterised, in turn, by different paradigms[117]:

- *Interruption management*: the user executes two tasks, referred as an ongoing task and an interrupting task. Researchers focus on the time to switch from one to another, the time to resume the ongoing task after an interruption, and the quality of both tasks [114].
- *Supervisory sampling and control*: the user interacts with a display or an instrument panel, where an instrument might serve different tasks. Researchers focus on the user selection strategy between tasks [52, 66, 99, 100].
- *Strategic task overload model*: tasks are characterised by attractiveness factors which drive the user attention. Researchers focus on the time spent on a task rather than in other [115, 117].
- *Voluntary task switching*: the user is allowed to switch from one task to other whenever and however he/she wants [4, 64, 89, 94]. A dominant theory based on such paradigm is the theory of threaded cognition [95, 96], where task selection is determined by the availability of resources which cannot be shared among tasks.

1.2 Safety-Critical Human Multitasking

When in a multitasking scenario some of the tasks are safety-critical, then failures to perform the tasks correctly and timely (e.g., due to cognitive overload or giving

too much attention to other tasks) could have dangerous consequences. Therefore, we call *safety-critical human multitasking* a situation where a person interacts with a safety-critical system while using other less critical devices. For instance, pilots usually reprogram the flight management system while communicating via radio and monitoring flights instruments [47]. Operators of critical medical devices, such as infusion pumps, often have to retrieve patient-specific parameters by accessing the hospital database on a different device while setting the safety-critical system. A driver often interacts with the GPS navigation system and/or the infotainment system while driving. Finally, astronauts must manage multiple (possibly safety-critical) tasks: e.g., during docking, they need to control the speed via RCS rockets while estimating the distance to the docket port, dealing with weightlessness and possibly communicating in a foreign language.

Human multitasking could cause the memory to overload (too much information to process/remember), which leads to forgetting/mistaking useful information to complete critical tasks. For instance, [74] reports that during a routine surgery, the ventilator which helps the patient to breathe was turned off to take an X-ray without blurring the picture. However, the X-ray jammed, and the anesthesiologist who went to fix the X-ray forgot to turn on the ventilator, leading to the patient's death. Again, in [32], the cause of 139 deaths while using an infusion pump are analysed, and it is found that 67 are caused by operator distractions while 10 are caused by problems with the device itself. Similar examples can be found in the context of aviation [6, 101] and car driving [46].

In addition to memory overload, human multitasking may also lead to cognitive overload when some tasks are too cognitively demanding, which could lead to ignoring the critical tasks for too long or in a crucial moment while focusing attention on less critical tasks. For instance, while reprogramming the flight management system, the pilot could miss something important on the flight instruments. Or again, if the interface of the virtual clinical folder requires too much attention, the medical operator could make some mistake while setting the infusion pump. Finally, an infotainment system which attracts the driver's attention during a road curve could cause a car accident.

1.3 Thesis Contribution

In the context presented above, there is, therefore, a clear need to analyse not only the functionality of single devices but also to analyse whether a human can safely use multiple devices/systems at the same time. Such study requires understanding how the human cognitive processes work when users concurrently interact with multiple systems, and how human attention is directed at the different tasks.

Moreover, we focus on whether some tasks properties could affect the addressing of attention to them. In particular, the *cognitive load* of a task (i.e., a measure of its complexity in terms of frequency and difficulty of the memory activities it requires to perform [13]) is a crucial parameter when deciding to which task direct the human attention. Finally, how much a user could perceive a task as safety-critical and how long he/she could ignore a task, are other crucial parameters which could affect the execution of the tasks.

Formal Model

We propose a formal model of safety-critical human multitasking which is an extension of the cognitive framework proposed by Cerone for the analysis of interactive systems [29]. As in that work, our model includes the description of the human working memory and of the other cognitive processes involved in the interaction with a device. However, Cerone only considers the interaction with a *single* device, while we focus on multitasking. Moreover, our model also captures the limitations of the working memory, enabling us to reason about hazards caused by memory overload, and includes timing features, enabling us to reason about hazards caused by cognitive overload. We selected the framework proposed by Cerone as a starting point since, given the problem of studying multitasking, we decided to extend an existing model which describes at least in part some of the cognitive mechanisms involved in human-machine interaction (e.g., the memory system and its action mechanism) and which provides a formal model that can be subjected to a set of formal analyses, in order to systematically and automatically check whether the model meets given properties.

The semantics of the safety-critical human multitasking model is defined as a probabilistic transition system, whose transition relation is defined by a set of inference rules. Each rule models a different cognitive processes involved in multitasking.

The multitasking paradigm underlying our model is the voluntary task switching. In particular, in our model each task is characterised by an attention attraction factor computed as the product of the three main quantities: the cognitive load of the task, its criticality level, and the time the task has been ignored. At each step of the interaction each task has a probability to be executed – i.e., it will be the task to which the user will direct his/her attention – proportional to such attention attraction factor.

Simulator and Model Validation

We implement a simplified version of the model as a Java simulator, that can be used to have a quick feedback on whether users can safely complete multiple tasks at the same time.

We use the simulator to validate the model against data gathered from an experimental study we conducted. Namely, a web application has been devised in collaboration with psychologists and used to administrate a test to real users. Users were asked to interact with two tasks concurrently: a critical task and a "distractor" task whit different levels of cognitive load.

The collection of the real users performance data, of the data of some of their personal characteristics, and of the data on how they perceive the criticality and the difficulty of tasks, enables us to identify 6 typologies of users. According to each typology characteristics and to each different level of cognitive load we then implement a different test with our simulator. The results of our simulations agree with the data gathered from the experimental study.

Real-Time Maude Executable Framework

We implement the multitasking model in Real-Time Maude [84, 86], a rewriting logic language and tool which extends Maude [33] to support the formal specification and analysis of real-time systems.

The Real-Time framework enables us to analyse safety-critical human multitasking through simulation, and reachability analysis. Moreover, we show that Real-Time Maude can be used to study a number of prototypical multitasking problems.

Case Studies

Finally, we illustrate our framework by modelling and analysing three case studies:

1. the interaction of a user with a GPS navigation system while driving;
2. a medical operator setting multiple infusion pumps simultaneously;
3. some typical concurrent tasks of an air traffic control operator.

We apply model checking to show that:

- the cognitive load of the GPS navigation system could cause the driver to keep the focus away from driving for too long or in a crucial moment;
- distractions and memory overload could cause an air traffic controller to make critical mistakes
- a given multitasking strategy could cause the memory overload and lead the medical operator to forget important information for the safety of the patient

1.4 Outline of the thesis

The thesis is organised in 7 chapters, plus introduction and conclusion.

Chapter 2 provides some background concepts useful for understanding the rest of the thesis. The chapter is divided into three main sections, one presenting the cognitive background, the others presenting some technical background. In particular, in Section 2.1 it is described how the human memory works and it is divided into separate types, and how human attention works in multitasking contexts. Moreover, it gives an overview of the psychological literature taken into account by the thesis. Section 2.2 recalls some definitions of transition systems. Section 2.3 presents some background concepts on Real-Time Maude by showing how the Maude modules are specified and how the formal analysis can be performed with it.

Chapter 3 presents the state of the art in the modelling of human-machine interaction, with a particular emphasis on models which focus on the cognitive aspects involved in the interaction. Some of the models presented support only simulation (Section 3.1), others provide a formal model which can be also subject to a range of formal analysis (Section 3.2). Finally, in Section 3.3, we discuss the limitations of the presented models.

Chapter 4 contains the main contribution of the thesis: the formal model of human multitasking. In particular, we formally define the syntax of the model, and its semantics given in terms of a purely probabilistic transition system.

Chapter 5 describes our Java simulator. Chapter 6 shows the model validation. The chapter is divided into three main sections: in Section 6.1 we present the experimental study, in Section 6.2 we present the design of the simulation experiments, finally in Section 6.3 we show the results obtained.

Chapter 7 presents the implementation of the model in Real-Time Maude, and it shows how all the inference rules presented in Chapter 4 are implemented as Real-Time Maude rules.

Chapter 8 shows the use of our model in three different application domains: the interaction of a user with a GPS while driving, a medical operator setting multiple infusion pumps at the same time, and some typical concurrent tasks of an air traffic control operator.

Finally, Chapter 9 presents the conclusion of the thesis and depicts possible fu-

ture works.

1.5 Publications

Part of the material presented in this thesis has appeared in some publications or has been submitted for publication in:

- Giovanna Broccia. Model-based analysis of driver distraction by infotainment systems in automotive domain. In *Proceedings of the SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2017)*, pp. 133-136. ACM (2017)
- Giovanna Broccia, Paolo Milazzo, Peter Csaba Ölveczky. An Algorithm for Simulating Human Selective Attention. In *Software Engineering and Formal Methods (SEFM 2017) Collocated Workshops*, LNCS, vol. 10729. Springer (2017)
- Giovanna Broccia, Paolo Milazzo, Peter Csaba Ölveczky. An Executable Formal Framework for Safety-Critical Human Multitasking. In *NASA Formal Methods (NFM 2018)*, LNCS, vol. 10811. Springer (2018)
- Giovanna Broccia, Paolo Masci, Paolo Milazzo. Modeling and Analysis of Human Memory Load in Multitasking Scenarios. In *Proceedings of the SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2018)*, pp. 9:1–9:7. ACM (2018)
- Giovanna Broccia, Paolo Milazzo, Peter Csaba Ölveczky. Formal Modeling and Analysis of Safety-Critical Human Multitasking. Under review in *Innovations in Systems and Software Engineering, a NASA Journal*

2 Background

In this chapter, we present the necessary background to understand the rest of the thesis. Section 2.1 introduces some basic concepts on cognitive psychology and focuses on human memory and on human selective attention. Section 2.2 presents some background concept on (probabilistic) transition systems, in particular we recall the definitions of *transition system*, *labelled transition system*, and *probabilistic transition system*. Section 2.3 presents some background concepts on Real-Time Maude.

2.1 Cognitive Background

The reflection about the human mind and its processes grows since the times of ancient Greeks: in 387 BC Plato suggests that the brain is the seat of the men-

tal processes. Over years many debates arise regarding whether human beliefs are solely experiential (empiricism) or are native (nativism). In the 20th century the development of computer science lead to reflection about the parallelism between human brain and computers. In 1959 Chomsky's critique of empiricism [30] initiates what is now known as the "cognitive revolution". With Neisser book [82] the term *cognitive psychology* comes into common use.

Cognitive psychology is the study of higher mental processes such as attention, language use, memory, perception, problem solving, creativity and thinking [1].

2.1.1 Memory System

Memory is the capability to encode, store and maintain and subsequently recall information and past experiences in human brain. The encoding stage consists in the translation of the environmental information in a meaningful entity; the storing and maintaining stage consists in the retention in time of stored information; the recalling stage consists in the recovery from the memory of information earlier encoded and stored. Memory might fail in each of these three stages.

Each of these three stages of memory are related to a different type of memory: sensory memory, short term memory and long term memory. Such model as sequence of types/stages of memory is known as the multi-store model, after Richard Atkinson and Richard Shiffrin described it in 1968 [5], and it is the most popular model for studying memory.

The main principles of the Atkinson-Shiffrin theory stated that:

- When an environmental stimulus is detected by senses it is immediately available in the sensory memory, which is actually composed of multiple registers, one for each sense.

  The sensory registers detect and hold information for using them in short-term memory. Information are only transferred to the short-term memory

[1] American Psychological Association. Glossary of psychological terms. [on line]. Available at: http://apa.org/research/action/glossary.aspx?tab=3 (Retrieved 29-08-2018)

when *attention* is addressed to it, otherwise they decay rapidly (in about seconds) and they are forgotten.

- The *short term memory* (STM) stores all the sensory information to which is given attention. Information stored in the short term memory decay in approximately 20 seconds, it is nevertheless possible to maintain them in STM if they are actively rehearsed through attention. Information in STM do not have to be of the same type as their sensory input (e.g., written text which enters in STM visually can be held as auditory information).

  There is a limit to the amount of information that can be held in STM, generally referred as *memory span*. In [77], psychologist George Miller suggested that STM has a memory span of seven items plus or minus two. Such items can be organised in higher order cognitive representation, such as when a person has to remember a telephone number and he/she gathers together digits' groups; this phenomenon is called *chunking*.

- The *long term memory* (LTM) is where all knowledge of a subject is stored. Information enters in LTM from STM through the process of consolidation, that is the processes of stabilising a memory item after the initial acquisition, involving rehearsal and meaningful association. Long-term memory is assumed to be almost limitless in its duration and capacity: generally brain structures start to deteriorate and fail before any limit of learning is reached. Over the years, several types of LTM have been recognised:

  - *Declarative Memory* is the memory of events explicitly stored in memory which can be deliberately recalled.
  - *Procedural Memory* is the memory of skills and competence (e.g. the usage of objects or body's movements) which once learned become automatic.
  - *Prospective Memory* is the memory of action to be performed in the future.

– *Retrospective Memory* is the memory of events experienced in the past.

Other models of human memory are present in literature; specifically, a number of theorists consider the memory system as levels of processing [42, 98, 103, 107]. According to such theorists, the memory activity deals with the analysis of stimuli at a number of levels. The preliminary stages are concerned with the preliminary analysis of physical and sensory features (e.g. lines, angles, brightness, pitch, loudness). The later levels are concerned with matching the input against stored abstraction from past learning. The persistence of the memory traces is a function of the depth of the analysis: the deeper is the analysis the longer lasting and stronger is the trace. According to such a theory, memory is viewed as a continuum from the transient traces of the sensory analysis to the longer lasting traces of the semantic-associative analysis.

We use the multi-store model [5] as a cognitive basis for our framework since it is the most widely accepted approach for describing and studying memory [42].

Working Memory

The term *working memory* (WM) is often used to refer to STM although some neuro-psychological studies show that the two forms of memory are distinct, in particular since they arise from different neural subsystems within the prefrontal cortex [45].

Several models have been proposed to explain how WM works, the most influential is the *multi-component model* proposed by Baddeley and Hitch in 1974 [9]. The theory proposes a model composed of three components: the *central executive* functioning as control center and two "slave systems", the *phonological loop* and the *visuospatial sketchpad*, responsible for the short term maintenance of information.

The central executive directs attention to relevant information and coordinates cognitive processes when more than one task is simultaneously performed. The phonological loop stores phonological information and the visuospatial sketchpad stores visual and spatial information. They both refresh continuously items to prevent their decay.

Therefore, WM refers to the cognitive system responsible for transient holding, processing and manipulation of information. The main difference between STM and WM is that STM is only involved in the short term storage of information, while WM is a short term memory buffer that allows for processing and manipulation of information [40, 45].

WM span has proven to be highly predictive of performance in reading comprehension [43], the processing of ambiguous syntactic constructions [79], reasoning [11, 69] and complex learning [102].

Working Memory Span Tasks

Working memory span tasks (WMST) are tasks used to measure the performance of the working memory. Such tasks are widely used in cognitive psychology [38] since WM plays an important role in a wide range of complex cognitive behaviours, such as comprehension, reasoning, and problem solving [50], and it is an important individual variable in general intellectual ability [36, 37, 51].

These tasks were designed from the perspective of Baddeley and Hitch theory [9] to measure how much the WM would allow the organism to keep active and accessible in memory task-relevant information while executing complex cognitive tasks (i.e. tasks which requires complex processing activities). Therefore, to measure the WM performance and not only the capacity of the short-term store.

WMST were created to require not only information maintenance, but also the concurrent processing of additional information [28, 43, 108]. Such tasks involve performing two sequential activities: one mnemonic activity which imposes the memorisation and recall of a set of elements (such as digits or words); and one secondary activity which imposes a processing operation (e.g. comprehending sentences, verifying equations, or enumerating an array of shapes). Participants are asked to see or hear a sequence of elements spaced by a processing operation. At the end of each trial they have to recall the sequence correctly (which means recall the correct elements and in the correct order), with increasingly longer sequences being tested in each trial (from two to five elements per trial).

As regards the score, there are two sources of data: one from the processing activity of the task and one from the mnemonic activity. However, a number of evidence from studies on adults supports the common procedure of not considering the processing activities while calculating the WM score. First, the processing precision is typically close to the ceiling since the task instructions accentuate the accuracy in the processing activity in order to ensure that the subjects commit to the secondary tasks. Second, the performance in the processing activity correlates positively with the performance in the mnemonic activity: subjects who recall the most number of elements also are most accurate in the processing activity [62, 109].

With the traditional scoring method, subjects are assigned a quasi-absolute span score [43, 109]. The task continues until the subject's accuracy falls below a certain threshold: the last item size recalled with a specific probability is the span score. A problem with this absolute scoring method is that the difficulty of an item may vary and compromise the score reliability. For instance, longer sentences in a reading span task could decrease the number of elements recalled [104–106]; similarly, the display duration for each sentence, or the semantic similarity of the stimuli, could have an influence on the mnemonic performance [39].

Another scoring method is to assign to responses to each element (to be recalled) a number (e.g. 1 for the correct answers and 0 for the incorrect answers); such a number varies according to the scoring method chosen. There exist four different scoring procedures using this method:

- Unit scoring methods:
    1. *Partial-credit unit scoring* (PCU) expresses the mean proportion of elements within an item that were recalled correctly
    2. *All-or-nothing unit scoring* (ANU) expresses the proportion of items for which all the elements were recalled correctly
- Load-weighted scoring methods:

1. *Partial-credit load scoring* (PCL) represents the sum of correctly recalled elements from all items, regardless of whether the items are perfectly recalled or not
2. *All-or-nothing load scoring* (ANL) represents the sum of the correctly recalled elements from the items in which all the elements are recalled in correct serial order.

Load-weighted scoring is rarely used, since assigning a greater weight to harder items is useless: all items within a task are supposed to measure the same underlying ability, that is the memorisation of elements in the face of concurrent processing. Therefore, unit scoring is preferred and, among them, empirical results favour partial-credit unit scoring [38].

The PCU for each user is computed as follows:

$$PCU = \frac{\sum_{i=1}^{N} \frac{b_i}{a_i}}{N}$$

where N is the number of items, b_i the number of elements correctly recalled, and a_i the number of elements to recall. In Table 2.1.1 it is presented an example of the PCU score for a WM span task.

Cognitive Load and Visual Selective Attention

One important assumption in the multi-component theory is that information in WM decays over time, unless it is prevented by rehearsal. There are several theories about the nature of such decay, the most elaborate is the "Time-Based Resource Sharing Model" [13].

The theory builds on the following principles:

1. Items stored in WM are subject to processing and maintenance activities;
2. Processing and maintenance activities both use the same cognitive resource, that is *attention*: when there are small time intervals in which the processing

item	*a*	*b*	*score*
1	2	2	1
2		2	1
3		2	1
1	3	3	1
2		2	0.66
3		3	1
1	4	4	1
2		3	0.75
3		3	0.75
1	5	3	0.6
2		5	1
3		2	0.4

PCU	**0.85** (10.16 / 12)

Table 2.1.1: Example of PCU computation for a WM span task.

activities do not require such resource, such time can be used for maintenance activities, namely to refresh memory traces.

3. When attention is drawn away from maintenance activities, items in WM suffer of a time-related decay;

4. Processing activities which require items' retrieval from LTM have a detrimental effect on maintenance activities;

5. It is not possible to perform multiple items' retrievals at the same time.

Barrouillet et al. define an indicator, called *cognitive load* (CL), for measuring the temporal density of attentional demand for each task. Specifically, CL gives a measure of the total amount of time during which maintenance of items in WM is impeded, and thus it provides a measure of how much a task is cognitively demanding.

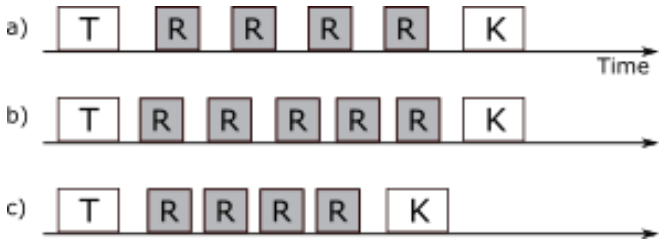

Figure 2.1.1: WM Span Task.

When different activities are performed at a constant pace during a task, the CL corresponds to the following:

$$CL = \sum a_i n_i / T \tag{2.1}$$

where n_i corresponds to the number of activities of type i, a_i is the difficulty of these activities i and T is the total duration of the task.

In a simplified situation in which all the task's activities are identical in nature, the CL corresponds to the following:

$$CL = aN/T$$

where N correspond to the total number of activities of the task.

Figure 2.1.1 shows three WMST where participants are asked to maintain letters in memory (white boxes) while reading aloud digits that are sequentially presented on the screen (grey boxes); each of them have different CL. When digits are presented at a comfortable pace, participants can use the free time between two digits to refresh the information about the letters (fig. 2.1.1 a); when the number of digits increases, the number of activities increases and the cognitive load grows(fig. 2.1.1 b); this also happens when the total time to perform the task is reduced (fig. 2.1.1 c).

As regards the human attention, it has a key role on keeping information on the three types/stages of memory and it also used by the maintenance and processing activities of the same task. Moreover, such resource has a main role also when

users perform multiple tasks concurrently: several studies show how attentional limitations could cause troubles while performing multitasking [71, 87, 111].

De Fockert et al. [44] describe the role of WM, CL and the attention mechanism in the execution of concurrent tasks. According to their experimental studies, when the CL of what they call "distractor" task increases, the performance with a "main" task is impeded since an high CL leads to reduced differentiation between high and low priority, and consequently fallacious addressing of attention.

Although there remains vigorous debate about how best to characterise the items decay, the Barrouillet et al. [13] theory remains one of the most cited and the most successful in terms of explaining experimental data [8, 35, 41, 63]. Precisely, such theory is validated through several experiments on both adults and children [12–15, 44], while the De Fockert et al. [44] theory is validated through neuroimaging, a technique which enables to analyse and study the activity of different brain areas and some specific brain functions.

2.2 Transition Systems

A transition system is a mathematical model describing the potential behaviour of systems. Essentially, it is used to describe dynamic processes with configurations representing states and transitions saying how to go from state to state.

Definition 2.1 (Transition system)**.** A *transition system* (TS) is a pair $(S, \rightarrow)$ where:

- S is the set of states ranged over by $s, s_0, s_1, \ldots$;
- $\rightarrow \subseteq S \times S$ is the transition relation.

We write $s_i \rightarrow s_j$ when $(s_i, s_j) \in \rightarrow$.

The nature of the states of a TS depends on what the system describes. For instance, if a TS is used to describe a chess match, its states will represent all the possible positions of all chess pieces in the chessboard. The transition relation, instead, represents the steps which can be performed by the system to go from a state to another: $s_0 \rightarrow s_1$ means that the system can go from state s_0 to state s_1 in one

step. In the above chess example, one step corresponds to the chess move of one of the players.

A state s is reachable from a state s_0 (denoted as $s_0 \Rightarrow s$) if either $s_0 = s$ or there exists $s_1, \ldots, s_{n-1} \in S$ such that $s_0 \to s_1 \to \ldots \to s_n \to s$. Namely, s is reachable from s_0 if a system in state s_0 can perform a finite (and possibly empty) sequence of transitions after which the system state is s.

A labelled transition system [65] is a TS where transitions are enriched with labels.

Definition 2.2 (Labelled transition system)**.** A *labelled transition system* (LTS) is a tuple $(S, L, \to)$ where:

- S is the set of states (or configurations) ranged over by $s, s_0, s_1, \ldots$;
- L is a set of labels ranged over by $l, l_0, l_1, \ldots$;
- $\to \subseteq S \times L \times S$ is the labeled transition relation.

We write $s_0 \xrightarrow{l} s_1$ when $(s_0, l, s_1) \in \to$.

In a LTS the label can represent different things depending on what the system describes. Typical uses of labels include representing expected input, conditions that must be satisfied to trigger the transition, or actions performed during the transition.

Often the set L contains a special label denoting an hidden action τ. We denote $s_0 \Rightarrow s_n$ a finite (and possibly empty) sequence of τ-labeled transitions from s_0 to s_n, i.e. $s_0 \Rightarrow s_n$ if either $s_0 = s_n$ or or there exists $s_1, \ldots, s_{n-1} \in S$ such that $s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_n \xrightarrow{\tau} s$. Moreover, we denote $s_0 \overset{l}{\Longrightarrow} s_n$ a finite (and possibly empty) sequence of transitions from s_0 to s_3 such that there exists $s_1, s_2 \in S$ such that $s_0 \Rightarrow s_1 \xrightarrow{l} s_2 \Rightarrow s_3$. Finally, we denote $\overset{\bar{l}}{\Longrightarrow}$ the relations corresponding either to $\Rightarrow$ if $l = \tau$, or to $\overset{l}{\Longrightarrow}$ if $l \neq \tau$.

A probabilistic transition system [70] is a TS where transitions are enriched with probabilities.

Definition 2.3 (Probabilistic transition system)**.** A *probabilistic transition system* (PTS) is a tuple (S, A, Can, μ) where:

- S is the set of states;
- A is a set of (observable) actions which states may perform;
- Can is an A-indexed family of sets of states, with Can_a indicating the set of states that can perform the action a;
- μ is a family of probability distributions, $\mu_{p,a} : S \to [0,1] \, \forall a \in A, p \in Can_a$, indicating the possible next states (and their probabilities) after p has performed a.

  Whenever $p \in Can_a$, we have $\sum_{p'} \mu_{p,a}(p') = 1$, since $\mu_{p,a}$ is a probability distribution.

The term $\mu_{p,a}(p') = \mu$ means that p can perform a and with probability μ becomes the state p' afterwards. Thus, p' is a possible next state after a has been performed on p just in case $\mu_{p,a}(p') > 0$.

A purely probabilistic transition system is a PTS where there is no notion of actions and transitions are associated only to a probability distribution.

Definition 2.4 (Purely probabilistic transition system)**.** A *purely probabilistic transition system* (PPTS) is a tuple $(S, \to)$ where:

- S is a finite set of states ranged over by $s, s_0, s_1, \ldots$;
- $\to \subseteq S \times [0,1] \times S$ is the transition relation.

We write $s_0 \xrightarrow{p} s_1$ when $(s_0, p, s_1) \in \to$, which means that the state s_0 makes a transition to the state s_1 with a probability p. Let $\pi(s_0, s_1)$ denotes the sum of the probabilities of all the transitions from s_0 to s_1 in $\to$. We impose the restriction $\sum_{s' \in S} \pi(s, s') = 1$ which means that all probabilities outgoing from a state must sum to 1. Essentially, a PPTS corresponds to a discrete time markov chain (DTMC).

A PPTS whose states are terms built over some signature can be specified by means of a set of inference rules. An inference rule for the specification of a PPTS (a *transition rule*) is a logical rule having the form

$$\frac{t_1 \xrightarrow{\pi_1} t'_1 \ldots t_n \xrightarrow{\pi_n} t'_n}{t \xrightarrow{\pi} t'}$$

where $t_i \xrightarrow{\pi_i} t'_i$ for $1 \leq i \leq n$, are the *premises* and $t \xrightarrow{\pi} t'$ is the *conclusion*. A transition rule states that whenever the premises are transitions of the PPTS, then also the conclusion is a transition of the PPTS. Side conditions can be associated to a transition rule with the effect of imposing that the conclusion of the rule is a transition of the PPTS whenever both the premises and the side conditions are satisfied. A transition rule without premises is called an *axiom*, and a (non empty and possibly infinite) PPTS can be specified by providing a set of transition rules with at least one axiom.

2.3 Real-Time Maude

Real-Time Maude [84, 86] is a rewriting-logic-based formal specification language and simulation and model checking tool which extends Maude [33] to support the formal specification and analysis of real-time systems.

The specification formalism is based on *real-time rewrite theories* [85], which in turn is an extension of *rewriting logic* [21, 75], and it is especially used to model distributed real-time systems in a object-oriented style. Real-Time Maude specifications are executable and the tool provides a range of formal analysis method such as timed rewriting for simulation, timed reachability analysis, untimed or time-bounded *linear temporal logic* (LTL) model checking, and *timed computational tree logic* (TCTL) model checking. The tool is available at `http://www.ifi.uio.no/RealTimeMaude`.

2.3.1 Rewriting Logic Specification

A Maude module specifies a rewrite theory [21, 75] of the form $(\Sigma, E \cup A, R)$, where:

- $(\Sigma, E \cup A)$ is a *membership equational logic* (MEL) [76] theory specifying the system's state space as an algebraic data type, where:
  - Σ is an algebraic signature, namely a declaration of sorts, subsorts, and function symbols.
  - E is a set of (possibly conditional) equations.
  - A is a set of equational axioms such as associativity, commutativity, and identity.
- R is a set of labeled conditional rewrite rules, specifying the system's local transitions, each of which has the form:

$$l : q \longrightarrow r \ \textbf{if} \ \bigwedge_i p_i = q_i \ \wedge \ \bigwedge_j w_j : s_j \ \wedge \ \bigwedge_m t_m \longrightarrow t'_m,$$

  where l is a *label*, and q, r are Σ-terms of the same kind. Such rule specifies a one-step transition from a substitution instance of q to the corresponding substitution instance of r, if the condition holds.

Sorts and subsort relations are declared by the keywords `sort` and `subsort`. The system is a term of sort `System`. A function declaration has the form `op` f : $s_1 \ldots s_n$ `->` s `[` *atts* `]` and it declares a function f with n arguments of sorts $s_1, \ldots, s_n$ respectively, which returns an element of sort s. Operators can have user-definable syntax, using underbars '_' marking the argument positions. Some function could have equational attributes *atts* which could declare, for instance, the function to be associative (`assoc`), commutative (`comm`), or/and to have an identity element. The `frozen` attribute defines arguments that cannot be rewritten by rules. An operator can also be a *constructor* (`ctor`) definying the data elements of a sort.

Equations are introduced with keywords `eq` or `ceq` for conditional equations. They have the form `eq` $t = t'$ and `ceq` $t = t'$ `if` *cond* . The terms t and t' could contain variables, declared with the keywords `var` or `vars` , or introduced on-the-fly with the form *var* : *sort*. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` (i.e. otherwise) attribute can be applied to a term $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied.

Rewrite rules have the form: `rl [l] :` u `=>` v or `crl [l] :` u `=>` v `if` *cond*. Finally, a comment is preceded by '`***`' or '`---`' and lasts till the end of the line.

Object-Oriented Specification in Real-Time Maude

A Real-Time Maude *timed* module specifies a real-time rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R)$ [85], where:

- $(\Sigma, E \cup A)$ contains an equational subtheory $(\Sigma_{TIME}, E_{TIME}) \subseteq (\Sigma, E \cup A)$, satisfying the *TIME* axioms that specifies sort `Time` as the time domain, which can be discrete or dense.

  Real-Time Maude provides some predefined modules for time domains: e.g., `NAT-TIME-DOMAIN-WITH-INF` defines the time domain to be $\mathbb{N}$, contains the subsort declaration `Nat < Time`, and the supersort `TimeInf` extends the sort `Time` with an "infinity" value `INF`.

- The rules in R are decomposed in:

  - "normal" rewrite rules specifying the system local transitions
  - *tick* rewrite rules specifying the passing of time in the system, specified as:
    `rl [l] : {` t `} => {` t' `} in time` τ
    `crl [l] : {` t `} => {` t' `} in time` τ `if` *cond*
    where τ is a term denoting the duration of the rule, t and t' are terms of sort `System`, and `{_}` is an operator encapsulating the global state, so that the form of the tick rules ensures that time advances uniformly

in all parts of the system. Namely, the tick rule says that it takes time τ to go from state $\{t\}$ to state $\{t'\}$ (if *cond* is satisfied). The form of the tick rules ensures that the time pass uniformly in all parts of the system.

As already mentioned, Real-Time Maude is particularly suitable to formally model real-time systems in a object-oriented style. The state of an object-oriented specification is a term of sort `Configuration`, which is a multiset of objects and a subsort of `System`.

A class declaration `class` C `|` att_1 `:` s_1, ..., att_n `:` s_n declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An object of `class` C is represented as a term `<` O `:` C `|` att_1 `:` val_1, ..., att_n `:` val_n `>` of sort `Object`, where O of sort `Oid`, is the object's identifier and val_1 to val_n are the current values of the attributes att_1 to att_n.

By convention, in a rewrite rule, attributes whose values do not change and do not affect the next state of other attributes, need not be mentioned. Similarly, attributes whose values influence the next state of other attributes, but are themselves unchanged can be omitted from right-hand sides of rules.

2.3.2 Formal Analysis

We summarise below some of the Real-Time Maude analysis commands.
The timed rewrite command

`(tfrew` t `in time <=` r `.)`

simulates *one* of the many possible system behaviours from the initial state t by rewriting it up to a certain duration less than or equal to the time value r. The time bound can also have the form `with no time limit`.
The search command

`(utsearch [` n `]` t `=>*` *pattern* `such that` *cond* `.)`

uses a breadth-first strategy to search for (at most n) states that are reachable from the initial state t, match the search pattern, and satisfy *cond*. The number of states to search (`[`n`]`) and the condition to be satisfied (`such that` *cond*) can be omitted.

If the arrow is `=>!` , then the command searches for final states which are states that cannot be further rewritten.
The search command

`(tsearch [` *n* `]` *t* `=>*` *pattern* `such that` *cond* `in time <=` *r* `.)`

is similar but with a time bound *r*.
The command

`(find latest` *t* `=>*` *pattern* `such that` *cond* `in time <=` *r* `.)`

explores all behaviours from the initial state *t* to find the longest time needed to reach the desired state for the first time in a behaviour.

Finally, Real-Time Maude specifications can also be subjected to unbounded and time-bounded LTL model checking which analyses whether each behaviour satisfies a linear temporal logic formula from an initial state, and to a TCTL model checking to analyse timed temporal logic properties [72, 73].

3

State of the Art

In this chapter, we discuss the state of the art in modelling the cognitive aspects involved in human-machine interaction with complex systems. Some of these models are based on a mathematical specification and implementation which supports only simulation (Sect. 3.1); others make use of formal methods and provide an executable model which can be also subject to a range of formal analyses (Sect. 3.2). We will discuss the main limitations of the presented models in Section 3.3.

3.1 Computational Models

3.1.1 ACT-R architecture

The *Adaptive Control of Thought—Rational* (ACT-R) architecture is an executable rule-based framework for modelling cognitive processes [3]. Figure 3.1.1 illus-

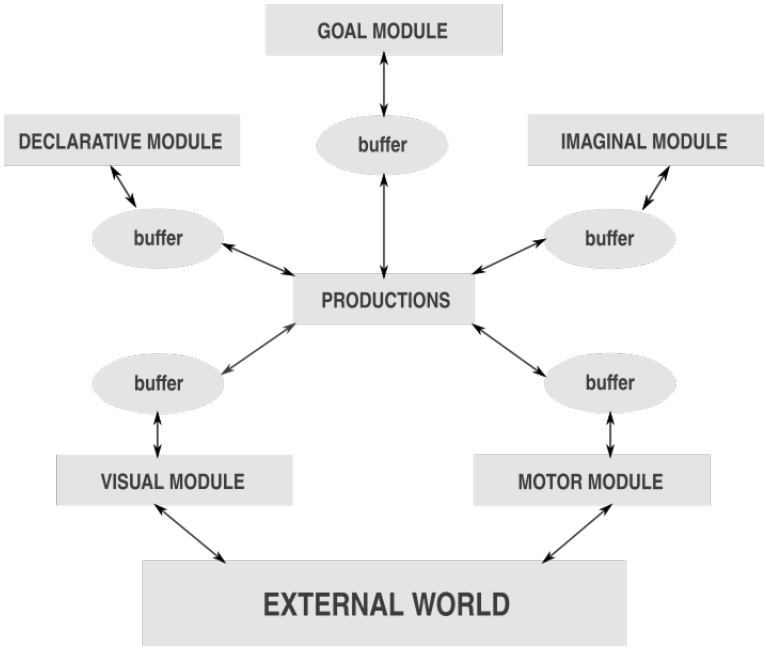

Figure 3.1.1: ACT-R architecture.

trates the basic architecture of ACT-R 6.0. It consists of a set of modules handling different aspects of cognitive activities; each module is connected with a buffer, which can be seen as a temporary storage unit which allows the information to "flow" through modules. Figure 3.1.1 contains just some of the modules in the system: a *goal module* for keeping track of current goal (for each subtask a goal is set); a *declarative module*, representing the declarative memory, storing factual information (i.e. all explicit knowledge about the world expressible in words); an *imaginal module*, representing essentially the working memory, storing temporary information; a *visual module* for identifying objects in the visual field; a *motor module* for controlling movement.

All modules are coordinated by a central production system, where production rules are stored. The production system gathers all the information from buffers and it decides the next step of action to perform. Production rules are if-then functions:

IF

goal buffer: *goal X*

declarative buffer: *information Y*

visual buffer: *information Z*

THEN

motor buffer: *action A*

means that if there is a goal X in the goal buffer, an information Y in the declarative buffer and an information Z in the visual buffer, then the action A will be performed.

Sometimes there are different production rules having the same conditions and thus different rules might apply. However, only one production rule can be selected. To solve such situation each production rule has a *utility* value: the rule with the highest value is selected. The utility of a rule i is defined as:

$$U_i = P_i G - C_i \tag{3.1}$$

where P_i is an estimate of the probability that if rule i is chosen the current goal will be achieved, G is the value of the current goal, and C_i is an estimate of the cost to achieve the goal. Both P_i and C_i are learned from experience with that rule: production rules that are successful will increase their utility, whereas rules that are more often unsuccessful will have their utility decay over time.

The ACT-R architecture has been used to model and study complex real-world tasks. In particular, in [92] it has been applied to study the effects of distraction by in-car interfaces while driving. The approach focuses on two computational behavioural models, one for the primary task (i.e. driving) and another for the secondary task (i.e. using a dialling interface for in-car cellular phones). The resulting integrated model executes both tasks and generates predictions about the resulting behaviour. Four different dialling interfaces have been modelled: *full-manual*, where user should type the entire number and then press the button "send"; *speed-manual*, where user should type a number associated with the desired contact and then press the button "send"; *full-voice*, where user should say aloud the desired

contact's full number; *speed-voice*, where user should say aloud the desired contact's name. All four interfaces require the user to activate them by pressing the button "power". The integrated model shows that both manual interfaces have significant effects on steering performance, while the voice interfaces have not significant effects.

The ACT-R version used in [92] did not provide automatic mechanisms for interleaving multiple tasks. Thus, the way used by Salvucci to model such interleaving mechanism was by having a driving system that, at some point in the execution, generates a new subgoal for dialling the cell phone and it stores it in the driving goal module. After each control, that is after each production rule is executed, the driving model chooses with a probability of 0.5 (set arbitrarily) whether to cede control to the phone dialling subgoal or to the driving subgoal.

A more recent version of ACT-R incorporates modifications in order to model more realistic internally-driven multitasking, i.e. complex tasks in which people themselves decide when to switch between tasks. In [68], Kushleyeva et al. focus on the problem of when to switch away from the current goal in the buffer. As already mentioned, ACT-R architecture provides a mechanism of conflict resolution when more than one production rule might apply, and according to such mechanism, the rule that maximizes the utility value is chosen. The G parameter in the utility function 3.1 is expressed as the amount of time the model is willing to spend on that given goal. The modification proposed in [68] is to set the initial value of G at goal creation and to decrease such value linearly with the passage of time: as the time passes and G decreases, the system will go from favouring rules with high probability P, to favouring rules with low cost C.

Moreover, a new production rule for giving up control is introduced for each goal in the model; such rule has low values of parameters P and C, which ensures that it will be activated almost at the end of (or near) the time period the model intended to spend on accomplishing the goal. In other words, such modifications balance task's execution by favouring the least recently executed task.

Concluding, ACT-R architecture is a rule-based framework used to simulate cognitive processes involved in human-machine interaction, with whom is pos-

sible to simulate the interaction with multiple interfaces through a multitasking mechanism based on the time a task has not been executed.

As already mentioned, this system supports only simulation, thus it is not possible to exhaustively and automatically check whether the model reaches a given specification. Moreover, the way a task is chosen among the other is based essentially on execution time, however, there are other task's characteristics which could affect such decision, such as how much the task is safety-critical or how much it is memory demanding.

3.1.2 SEEV model

The research on visual attention in supervisory control has identified four factors which determine where the eye is looking at any given time during a task involving the scanning of a display: salience, effort, expectancy and value [116].

Salience measures how much an *area of interest* (AOI) – a physical location where information regarding specific task can be found – attracts attention: how much a signal stands out from the rest of the workspace (background or other AOIs) due to its size, colour or contrast. *Effort* measures the cost of moving attention from an AOI to another. *Expectancy* measures how much the user expects chenges in the workspace. *Value* measures the importance of the information related to a task.

These four factors have been combined in an additive scanning model: the *Salience Effort Expectancy Value* (SEEV) model, which predict visual scanning behaviour in a display-based workspace [110]. The model predicts the distribution of attention: both when the user gives attention to a specific AOI and when he neglects it.

SEEV model is specifically designed to describe sequential visual scanning of an instrument panel where each instrument may serve different tasks: the multitasking strategy underlying such model is the supervisory sampling and control (see Chapter 1).

However, it is not possible to analyse voluntary task switching, that is the interaction where the user himself decides when to switch from a task to another.

3.1.3 STOM model

Strategic Task Overload Management (STOM) model [112, 117] is a multi-attribute decision model for task switching in overloaded multitasking conditions. According to such model, when multiple tasks have to be executed, the task that might be selected can be predicted through a rank based on four critical attributes. Such attributes are: the *priority* which refers to the relative importance of a task (e.g. a safety-critical task has a higher priority than a non critical task); the *difficulty* which is associated with the mental workload of the task; the *interest* of a task (e.g. a cell phone conversation can be more interesting than monitoring the roadway for unexpected hazards); the *salience* of a task, which is defined as the ability of a new task to attract attention (e.g. an auditory task has a higher salience than a visual task).

Each of such attributes has a polarity, governing the task attractiveness. In particular, salience, importance, and priority have a positive polarity, whereas difficulty has a negative polarity. Therefore the rank is calculated as :

$$S + P + I - D$$

However, the salience factor applies only to a new task, since it characterises its arrival. The rank of an ongoing task is calculated as the sum of priority, importance, difficulty and a fifth factor: the time on task, which is not an original member of STOM.

With STOM model it is possible to analyse multitasking interactions where the user decides which task to execute according to its rank. However, the user memory is not explicitly modelled and neither its action mechanisms, useful to analyse multitasking problems such as memory overload, prospective memory failures (i.e. the user forgets to perform a planned action) and retrospective memory failures (i.e. the user forgets that he performed an action).

3.1.4 Task modelling

Task modelling is a technique used to build a model which describes precisely the relationship among various tasks; such relationship might be both temporal and semantic.

In some cases, the task model of an existing system is created in order to better understand its underlying design and analyse its potential limitations. In other cases, designers create the task model of new applications to indicate how activities should be performed in order to obtain a new, usable system.

One of the most used notations for task modelling is Concur Task Tree (CTT) [88]. CTT has been used as a basis for the development of model-based analysis tools for user tasks. In [10] for instance, it is presented an integration framework which uses task models and system models to analyse the co-execution of tasks, and which estimates user workload as the total number of cognitive tasks.

Task models describe goals and activities that should be performed by users to reach such goals. However, their main focus is to assess the compatibility of task specifications with a user interface design, not to explicitly model users' characteristics.

3.1.5 GOMS

The GOMS models [25] – actually a family of models – describe the cognitive components of users in order to evaluate human-computer interaction in terms of task duration. A GOMS model is composed of four elements: *goals, operators, methods* and *selection rules*.

Goals are what the user has to accomplish and they are often divided into subgoals. Operators are perceptual, cognitive or motor actions whose execution is necessary to change the user's mental state or the task environment. Methods are procedures to accomplish a goal. Selection rules handle the choice of a method when more than one are available to accomplish a goal.

There are several variants of the GOMS models:

- *Keystroke-Level Model* (KLM) is the simplest version presented by Card,

Moran and Newell [27]: the analyst lists the sequence of operators and then totals the execution times to estimate the execution time of a task. The KLM model includes six operators (e.g. *K* to press a key button, *P* to point on a display with a mouse, M to mentally prepare to act, and so on). Each of these operators has an estimated execution time.

- *Card, Moran, and Newell GOMS* (CMN-GOMS) [25, 26] Card et al. do not describe these models with an explicit "how to" guide but they illustrated nine models at different levels of detail.

- *Natural GOMS Language* (NGOMSL) is a structured natural-language notation for representing GOMS models and a procedure for constructing them. The model is in a program form and provides a prediction of operators sequence, execution time and time to learn the methods.

- *Cognitive-Perceptual-Motor GOMS* (CPM-GOMS) does not assume that operators are performed serially; rather perceptual, cognitive and motor operators can be performed in parallel as the task demands. Thus, CPM-GOMS can model multitasking behaviour and makes use of a schedule chart to represent operators and the dependencies between them.

GOMS models are used to determine times for both cognitive processing and motor movements. Even if time features of tasks are modelled, these are not used to analyse whether a task is ignored for a time period or not. Moreover, GOMS does not take into account users' differences: it is only applied to skilled users.

3.2 Formal Models

3.2.1 A cognitive framework for the analysis of interactive systems

As mentioned in Chapter 1, the framework we propose is a significant modification of the framework proposed by Cerone in [29]. Cerone attempts to unify two different directions in the analysis of interactive systems – the behaviour of both

users and skilled operators – by providing a formal framework to model and analyse the interaction between the human component (user or operator) and a device. The framework is specified in Maude and makes use of a set of rules which rewrite both the interface state and the human component state until the goal of the interaction is not reached.

As regards the human components, states are represented by user/operator short-term memory, modelled as a set of information, and transitions are modelled through basic tasks. The information that can be stored in STM are the reference to the actions that lead to the achievement of the goal or maintain a correct system state, the reference to a future action to be performed, and/or the user/operator cognitive plan. Tasks can be decomposed in a hierarchy of tasks until reaching basic tasks, which are modelled as a quadruple:

$$info_i \uparrow perc_h \Longrightarrow act_h \downarrow info_j \tag{3.2}$$

where the perception $perc_h$ activates the retrieval of information $info_i$ from STM, the execution of act_h and the storage of $info_j$ in STM.

As regards the interfaces, states are modelled in terms of perceptions produced in humans by an action on the interface. Such perceptions/interface states may induce different degrees of urgency in reacting that is modelled by a timeout.

A set of rules specify the dynamic behaviour of the entire system:

1. *Interacting:* models a step in the interaction with a device. It is triggered by a perception and may add information to STM.
2. *Closure:* models the achievement of the goal.
3. *Danger:* models a situation where the user/operator perceives as dangerous (the normal response to a danger is to abandon a task).
4. *Timeout:* models an autonomous action of the interface with no involvement of the human component and it is triggered by the expiration of the timeout.

5. *Cognitive:* models a cognitive process of the human, which changes his mental plan, with no involvement of the interface.
6. *Decision:* models a decision step in the interaction.

Concluding, Cerone models a set of cognitive processes and the use of short-term memory to model the interaction between a human component (user or operator) and a device, in order to detect possible human errors using model checking. However, he only considers the interaction with a single device and he does not focus on human multitasking. Moreover, in [29] STM is modelled as a set of information without limitations, thus is not possible to reason about errors related to memory overload. Finally, in Cerone's framework, timing features are not taken into account.

3.2.2 Work Model that Compute (WMC)

Taskload is a measure of the number of tasks a user is expected to perform at a given time and it has been shown to be a good indicator of user mental workload in the avionics domain. Houser et al. [58] present a formal model for reasoning about excessive taskload and concurrency issues that can lead to errors in human-machine interaction with complex systems.

The model is specified through a *Work Model that Compute* (WMC), a simulation framework consisting of a work model that describes how a given domain works and an engine that simulates the work model. Each work model is composed of three elements: *agents*, *actions*, and *resources*. Resources are a collection of elements of the environment which can be manipulated by agents; actions which manipulate resources, are linked to a specific agent. The work model specifies the frequency and priority of actions, the duration of the resources and which agents are involved. A scenario is the engine which pulls all these elements in a simulation and generates a simulation trace. The work model, the scenario and the simulation trace are then translated into a formal model representing the simulation over a given period of time; this translation generates also a set of specifications to be analysed with the model checker in order to find interesting taskload conditions in

the model. The simulator uses a scheduler which decides which action to perform next relying on scheduling rules defined by the authors, based on actions priority and duration.

The model permits not only to simulate multitasking interaction but also to formally verify some properties with model checking. However, the way the scheduler decides which action has to be performed by agents is based on parameters defined by the authors, without a cognitively plausible scheduling algorithm based on psychological literature. Moreover, even though the resources described in [58] can be seen as memory information, they do not explicitly model the memory action mechanism and then they cannot explicitly study memory issues.

Remaining in the field of avionics, in [53] WMC are used to simulate the interaction of a pilot with a flight management system and such interaction is then analysed with SAL model checker to study *automation surprises* (i.e. the automation behaviour of the aircraft which deviates from what pilots expect). In order to detect them, a mental model of pilots is modelled to find situations where such models differ from the actual state of the aircraft. The same potential problem (i.e. automation surprise) in flight guidance system is studied in [61], where is used the PVS theorem prover and the NuSMV model checker. However, these two works do not analyse multitasking situation and do not focus on the cognitive aspects of human behaviour: although in [53] mental models of pilots are modelled, such models are more subjective and differ from the cognitive models based on psychological literature, which attempt to describe the cognitive behaviour of all users/operator.

3.2.3 Other Models

In [23, 24, 56] the IVY workbench is used to analyse interactive systems. The IVY workbench [22] is a model-based tool which supports the *modal action logic* (MAL) language [48] for modelling the system. Moreover, the tool supports the specification of properties of the device's behaviour, and their verification through model checking. When verification fails, the counterexamples produced by the

verification process act as scenarios for the analyses of the errors.

However, the models presented in [23, 24, 56] do not explicitly model WM and they do not deal with multitasking.

Rukšėnas et al. [91] present a formal framework for predicting bounds for task-completion times and detecting user-error related design issues, which is based on a model of cognitively plausible behaviour. The generic model of cognitively plausible user behaviour (GUM) is a specification of a set of assumptions about the way users interact which nevertheless, do not completely specifies cognitive behaviours of all users: real users can act outside this model, about which the framework says nothing. Such assumptions are divided into two main groups: behavioural assumptions, which focuses on the role of users' knowledge in directing their behaviour, and salience assumption, which deals with the salience of cues in choosing actions. In [56, 91] four different notions of salience have been identified:

- *Specificity* defines the "just in time" trigger of an activity provided by an information resource.
- *Cognitive salience* defines the assumed user knowledge of a task in terms of action to be performed next; the information does not deal with the implementation of the device.
- *Procedural salience* derives from training in the use of a particular device's programming sequences.
- *Sensory salience* is triggered by some visual or auditory cues to remind to the user the next action to perform.

Also in this case, such framework can be not only simulated but also subjected to a set of formal analyses. However, GUM is guided mainly by assumptions about the salience of user cues. Moreover, also in this work, they do not explicitly model WM and its action mechanism.

Finally, Combéfis et al. [34] present a formal framework used to estimate the effort needed to perform a task, in terms of workload. The framework generates automatically mental models from the specification of user interfaces, which include information about the sequence of actions and the knowledge the user needs to know to be able to interact successfully with the system. However, the main target of this work is the development of user manuals rather than the evaluation of human multitasking.

3.3 Limitations of Existing Models

Multitasking

Some of the frameworks presented do not model and analyse human multitasking but they focus on errors which might arise in the interaction between a user and a single computer interface. Moreover, some of them do not analyse the voluntary task switching multitasking paradigm, which defines a spontaneous choice of users to switch from one task to another.

Formal Analysis

Many of the presented frameworks do not provide a formal model that can be subjected to a set of formal analyses to exhaustively and automatically check whether the model meets given properties.

Time Features

Another important limitation present in different works is that time features are not explicitly modelled. When time is modelled, it is done to define actions execution duration and it is not used to analyse time issues such as a task ignored for too long or a task ignored in a given moment of time, which are some of the main time issues in human multitasking.

Memory Model

Many of the frameworks presented do not explicitly model memory. Some of them, model a set of resources/information, which can be seen as the environmental information which users use to interact with systems, but none of them describes the memory action mechanism and its capacity limit. Therefore, it is not possible to use them to analyse some memory issues such as memory overload, prospective memory failure, and retrospective memory failure, which according to literature are some of the main problems in human multitasking. Moreover, many of the models presented do not rest their works on parameters deduced from the psychological literature, but they make an assumption about tasks' characteristics which might induce their choice in multitasking interaction.

In this context, we propose a formal framework for modelling and analyse human-machine multitasking interaction, where each task is characterised by different parameters: its criticality, its cognitive load and the time it has been ignored. Every task is then characterised by an attention attractiveness factor computed as the product of its parameters. The factor value permits to simulate the voluntary task switching multitasking paradigm: the user chooses spontaneously which task to execute among the others, according to such value.

The framework is specified in Real-Time Maude which permits not only to formally specify and simulate human multitasking but also to formal analyse it with reachability analysis.

A set of timing features permit to analyse time problems. For instance, the parameter storing the time a task has not been executed permits to analyse whether a task has been ignored for too long. Or again, with Real-Time Maude commands it is possible to check whether a given action is performed at a precise time or not.

Our model includes the description of the human working memory and its action mechanism, which permits us to analyse, for instance, whether a multitasking strategies might overload the memory or whether, in the middle of the interaction, the user forgets to perform a planned action or whether, while interacting, he forgets an action he already executed.

Finally, the cognitive processes described in our framework are deduced from psychological literature and observations of experimental data on working memory and visual selective attention. Therefore, such cognitive processes have a psychological basis and it is not deduced from our assumptions. This makes it possible to give a cognitively plausible explanation of some human multitasking issues, such as why secondary tasks could be distracting: due to their high criticality or because they require a heavy mental load.

4

Formal Model of Safety-Critical Human Multitasking

This chapter presents our formal model of safety-critical human multitasking. Specifically, we present a mathematical model of human selective attention used to study situations where users concurrently interact with multiple devices and they have to voluntary choose which task to execute next.

The model is an extension and modification of the cognitive framework proposed by Cerone for the analysis of interactive systems [29]. As in that work, we describe the cognitive processes involved in HCI and the human working memory. However we focus on multitasking and not on the analysis of the interaction with single device as in [29]. Moreover, our model also describes the limitations of the working memory, enabling us to reason about memory overload, and includes

timing features, enabling us to reason about hazards caused by distractions.

We will present the syntax of the model in Section 4.1, and its semantics in Section 4.2. Finally, in Section 4.3 we will illustrate the syntax and the semantics of our model with a simple example.

4.1 Syntax

A safety-critical human multitasking model is a set of interfaces, representing the interfaces of the devices/systems with which a user interacts. Each interface is associated with one or more tasks, representing the tasks the user performs on that interface. A task is essentially a sequence of actions that the user performs on that interface to reach some goal. In order to do this, he/she has to remember and retrieve information from his/her working memory.

At each step of the interaction the user has to choose which task he/she wants to perform on one of the interfaces, namely to which task he/she wants to address his/her attention. In such a context, some of the tasks' properties could affect the addressing of the user's attention. In particular, how much a user perceives a task as critical, how much the task is cognitively demanding, as well as the time the user ignored the task.

We will give top-down definitions by presenting first the most general concepts and then the more specific ones. Moreover, we will use some standard notation of the formal languages to actually define sets.

Definition 4.1 (Model)**.** A *Safety-Critical Human Multitasking* model *Is* is a set of *Interfaces*

$$Is = \{I_i, \ldots, I_n\}$$

4.1.1 Interface

We now define the interfaces, which incorporate the set of tasks that the user will perform on it, and the description of the device behaviour.

As regards the states of the interface, we follow a user-centric approach and we model them as perceptions, namely as what the user perceives of them. For instance, if a user is interacting with an ATM and the device is ready to accept the card, a plausible interface state is *cardReady* since the user *perceives* that the device is ready to receive his/her card (e.g. by looking at the device's screen).

Some perceptions/states might be subject to a timeout, which captures the fact that they do not last forever. For instance, the ATM interface might allow typing the PIN code only for a short period of time, after that the perception/state is no more available. The state *PINReady* **for time** 10 denotes that the state *PINReady* lasts for 10 seconds.

Definition 4.2 (InterfaceStates)**.** Given a set of perceptions P, we define the set *InterfaceStates* as:

$$\mathit{InterfaceStates} ::= p \mid p \textbf{ for time } t$$

for every $p \in P$, and $t \in \mathcal{R}_{\geq 0}$.

Definition 4.3 (Interface)**.** An interface I is defined as a tuple:

$$I = \langle A, P, \mathit{Inf}, \mathit{InitialState}, \mathit{Transitions}, \mathit{Tasks} \rangle$$

where:

- A is the set of symbols representing the *actions* that the user can perform on the device;
- P is the set of symbols representing the possible users' *perceptions* about the interface states;
- *Inf* is the set of symbols representing the information that the user can memorise during the interaction with a device;
- $\mathit{InitialState} \in \mathit{InterfaceStates}$ is the initial state of the interface representing the initial user perception possibly associated with a timeout;

- *Transitions* is a set of transitions built on P and A, representing the behaviour of the interfaces, namely how the interface state changes when the user performs an action;
- *Tasks* is the set of tasks the user wants to perform on the interface.

Interface Transitions

A transition models the passage from a state of the interface to another after the user performs an action. For instance, in the ATM example, the interface state passes from *cardReady* to *PINReady* **for time** 10, as soon as the user performs the action *insertCard*.

Definition 4.4 (Transition)**.** Given a set of actions A, a set of perceptions P, and the set of interface states *InterfaceStates*, we define a *Transition* as a tuple $\langle p, a, q \rangle$ where $p \in P$, $q \in$ *InterfaceStates*, and $a \in A$. We represent a transition as:

$$p \xrightarrow{a} q$$

Task

We define the interface attribute *Tasks* as a set since users can perform multiple tasks on a single device. Actually, what we call task is what is commonly known as a *scenario* in the context of software engineering: it describes *one* of the possible sequence of interactions which a user could perform with an interface in a defined time-frame.

As previously mentioned, a task is a sequence of actions the user performs on a device, in order to reach a desired goal, during a certain period of time. In a *non*-multitasking situation – i.e. when the user performs a single task – the actions composing the task are performed sequentially without any interruption. However, in a multitasking interaction, the user needs to choose which task to perform at each time, which means that while he/she is performing a task, all the other tasks are ignored.

We model a task as a sequence of subtasks. For instance, the task of withdrawing money at an ATM consists of a sequence of five subtasks:

1. Insert the card in the ATM;
2. type the PIN code;
3. type the desired amount of money;
4. retrieve the card;
5. collect the money.

Each subtask will be actually further decomposed into a sequence of simpler "basic" tasks.

Moreover, each task is characterised by a measure of how much the user perceives it to be safety-critical, a factor which influences how much a task attracts the user's attention.

Definition 4.5 (Task)**.** Given a set of perceptions P, a set of actions A, and a set of information Inf, a task T is a tuple $\langle Subtasks, c, g \rangle$ where:

- $Subtasks$ is a sequence of subtasks, with $emptyTask$ representing an empty sequence;
- $c \in \mathbb{R}_{>0}$ models the criticality level of the task;
- $g \in A$ models the final action corresponding to the achievement of the task goal.

Thus, each task specifies the sequence of subtasks of which is composed, a measure of how much the user perceives it as critical, and an action, which is the *goal action*.

Subtask . Each subtask composing the task is a (possibly empty) sequence of basic tasks: e.g., the subtask *type the PIN code* in the ATM example shown above consists of typing a certain number of digits and then pressing a button to confirm the operation.

Definition 4.6 (Subtask)**.** Given a set of perceptions P, a set of actions A, and a set of information Inf, a subtask is a sequence of basic tasks denoted as:

$$Subtask ::= (BasicTask)^* BasicTask | emptyTask$$

BasicTask. A basic task is a single step in the task that cannot be further decomposed: e.g., in the ATM example, *typing the digit 4* is a plausible basic task of the subtask *typing the PIN code*.

Between two basic tasks, it is possible to have some time, which could correspond to the time necessary to switch from one basic task to the next, but also to the time required by the device to process the received input and to enable the execution of the next basic task. For instance, in the ATM example, after executing the last basic task of subtask number two (i.e. type the PIN code), the ATM might need some time to check if the inserted PIN code is correct and to enable the next subtask (i.e. type the desired amount of money). We call such time between two basic tasks *delay*, which represents the time before the basic task is enabled and can be executed (therefore when the delay of the first basic task of a task is not yet elapsed, we consider the task like it is pausing), and we model it as a positive real number. Moreover, we characterise each basic task with two additional parameters, i.e. the duration and the difficulty, which we model as positive real number.

Definition 4.7 (BasicTask)**.** Given a set of actions A, a set of perceptions P, and a set of information Inf,we define a basic task as a tuple $\langle j, p, a, k, t, d, \delta \rangle$ that we represent as:

$$j \mid p \implies a \mid k \textbf{ duration } t \textbf{ difficulty } d \textbf{ delay } \delta$$

where:

- $j, k \in Inf$
- $p \in P$
- $a \in A \cup \{noAction\}$

- $\delta \in \mathbb{R}_{\geq 0}$
- $d, t \in \mathbb{R}_{>0}$

The basic task indicates that when the interface is on state p and the user has inside his/her working memory the information j, he/she can perform the action a and replace the information j with the information k in his/her working memory; such a basic task has duration t and difficulty d and it is enabled – and thus it can be executed – if and only if the delay δ is elapsed.

We consider two kinds of basic tasks (both described in Definition 4.7): a *user action* to be performed on the interface, and a *cognitive basic task* carried out by the user with no involvement of the interface. In the first case, the basic task specifies the action to perform on the interface and the information to update in the memory; for instance, the basic task of collecting back the card from the ATM has the form [1]:

$$cardInside \mid cardReturning \implies getCard \mid noInfo$$

and means that when the user perceives that the ATM is giving back the card and he/she has in his/her WM the information that the card is inside the ATM, he/she can take the card and remove the information about it from his/her WM.

In the second case, the basic task only specifies that the item to update in the memory must be the mental plan of the user, and the action to perform must be equal to *noAction* (since there is no involvement of the interface). For instance, when the user perceives, by looking at the ATM screen, that the time to insert the PIN code is over, he/she inserts in his/her working memory a cognition representing the new plan: re-start the interaction with the ATM; such basic cognitive task has the form:

$$noInfo \mid PINexpired \implies noAction \mid restart$$

[1] For the sake of semplicity, we omit in these examples the parameters **duration**, **difficulty**, and **delay**.

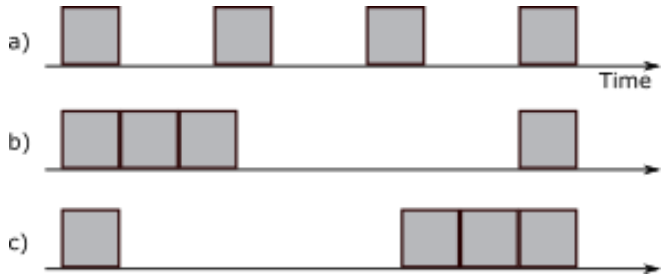

Figure 4.1.1: Three different tasks with basic tasks denoted as grey boxes.

Cognitive Load

Each task is characterised by a measure of how much it is cognitively demanding, namely the cognitive load.

In [13], the time between two basic tasks is not explicitly taken into account, since the definition of the CL of a task assumes that the single activities are performed at a constant pace (see Eq. 2.1). According to such a definition, the CL of the three tasks depicted in Figure 4.1.1 would be the same. However, if the three tasks in Figure 4.1.1 were potential "distractors" of another "main" task, they would interfere with the main task differently over time (assuming that difficulties of each basic activity are equal): the first task (Fig. 4.1.1 *a*) would attract the attention of the user constantly over time, the second task (Fig. 4.1.1 *b*) would attract the attention mostly at the very beginning, and the third task (Fig. 4.1.1 *c*) mostly after some time.

Therefore, we redefine the formula for estimating CL and we compute it on a subtask basis: the CL of a task changes every time a new subtask begins and remains the same throughout its execution.

Notation 4.1. Before presenting the definition we introduce some notations:

- $ST_{T,j}$ denotes the j-th subtask of the task T
- $difficulty_{ST,k}$ denotes the difficulty of the k-th basic task of the subtask ST;
- $duration_{ST,k}$ denotes the duration of the k-th basic task of the subtask ST;

- $delay_{ST,k}$ denotes the delay of the k-th basic task of the subtask ST;

Definition 4.8 (Cognitive Load)**.** Given a task T, we compute the cognitive load of T as the division of the *difficulty factor* of the task – namely a measure of temporal density of difficulty of the first subtask of T – (defined in Equation 4.2), by the *total duration* of the first subtask of T (defined in Equation 4.3):

$$CogLoad_T = \frac{DF_{ST_{T,0}}}{TD_{ST_{T,0}}} \tag{4.1}$$

where:

- $DF_{ST_{T,0}}$ denotes the difficulty factor of the task T, computed as:

$$DF_{ST_{T,0}} = \sum_{k=0}^{N-1} duration_{ST,k} \times difficulty_{ST,k} \tag{4.2}$$

- $TD_{ST_{T,0}}$ denotes the total duration of the first subtask of the task T, computed as:

$$TD_{ST_{T,0}} = \sum_{k=0}^{N-1} duration_{ST,k} + delay_{ST,k} \tag{4.3}$$

where N is the number of basic tasks composing the first subtask of the task T ($ST_{T,0}$).

Attention Attraction Factor

For each task T in each interface $I \in Is$ an *attention attraction factor* a_T is computed, which measures the likelihood of the task T to attract the user's attention.

Definition 4.9 (a-factor)**.** Let $T = \langle ST, c, g \rangle$ be a task, we compute the a-factor as:

$$a_T = CogLoad_T \times c \times (waitTime_T + 1) \tag{4.4}$$

where $waitTime_T$ denotes the time the task T has not been executed, information which will be computed dynamically during the task execution (defined in Section 4.2).

The attention attraction factor reflects the psychological literature presented in the background and the observation that the nature of the task usually has an influence on the user's attention.

As regards the cognitive load, we already mentioned the studies about the correlation between human attention and CL in Chapter 2.

As regards the criticality, we base on the assumption that such parameter is an intuitive attribute for task attention attraction: users tend to focus more frequently on more critical tasks than on less critical ones [57, 97]. Moreover, criticality is usually associated to the priority parameter of the STOM model (see Sect. 3.1.3), and studies on tasks' interruption suggest that tasks evaluated as primary might be disruptive since they attract more attention [59].

As regards the time, the nature of its role in the attention attraction factor is complex. On the one hand, there are good evidences that the longer users stay on a task, the more they are likely to leave them for performing other tasks due to resource depletion or for using different resources [67]. On the other hand, some tasks with high cognitive load, show the opposite effect: users tend to stay more on them because switch away would comport to restart such tasks from the beginning [54]. In our case, we don't consider how long a specific task has been executed, but we consider how long all other tasks in the multitasking scenario have been executed, and we assign to such time a positive polarity, in the sense that the more a task has been ignored, the more it attracts attention.

4.1.2 Working Memory

The working memory contains all information useful for the interaction with the interfaces. An element in working memory could be a basic information, a cognition or a goal.

A basic information is a cognitive item acquired through a procedural step in

the current task. For instance, when the user inserts his/her card inside the ATM, he/she puts inside his/her working memory the basic information *cardInside*.

A cognition is a mental plan resulting from the process of acquiring knowledge and understanding. For instance, when the user, looking at the ATM screen, understands that the device is out of order, he/she puts inside his working memory the cognition *ATMoutOfOrder*, which means that he/she has to change his/her mental plan (without interacting with the device) and find another device to achieve the goal of withdrawing money.

A goal is the aim of the interaction with the interface, which is to perform some final action (i.e., in the ATM example *withdrawMoney*).

Definition 4.10 (Information set)**.** Given a set of actions A, we define the set of information Inf as:

$$Inf = BI \cup COG \cup G$$

where:

- BI is a set of basic information;
- COG is a set of cognition;
- G is a set of goals where $g \in G$ is defined as:

  $$g ::= \mathbf{goal}(a)$$

  where $a \in A$ is the action that leads to the achievement of the goal.

We assume BI, COG, and G pairwise disjoint.

Definition 4.11 (Working Memory)**.** Working memory WM is defined as a tuple

$$\langle Memory, size \rangle$$

where:

- $size$ models the capacity of the working memory;

- *Memory* is a map assigning to each interface the set of information associated with that interface:

  $\{I_i \mapsto mem_i; \ldots; I_n \mapsto mem_n\}$

  where $mem_i \subseteq Inf$ is the set of information associated with the interface I_i and $\sum_{i=0}^{n} |mem_i| \leq size$

4.2 Semantics

We define the semantics of the safety-critical human multitasking model as a PPTS, where PPTS stands for purely probabilistic transition system. The transition relation is defined in an inductive way by a set of inference rules. Such rules model the different cognitive behaviours involved in the multitasking interaction and determine how attention is directed to the different interfaces and how this would change the state of the PPTS.

4.2.1 Configuration

Before presenting the configuration of the PPTS more in detail, we introduce some notation useful for the reading and comprehension of the configuration itself and the set of inference rules, we will present below.

Notation 4.2. Given an interface $I = \langle A, P, Inf, InitialState, Transitions, Tasks \rangle$, and given a task $T = \langle Subtasks, c, g \rangle$

- $Transitions(I) = Transitions$, denotes the set of transitions of the interface I;
- $g_T = g$, denotes the goal action of the task T.

Moreover, we assume we can index the tasks of the interface I with a natural number, so that (I, i) denotes the i-th task of the interface I.

In what follows, we define a notation of configuration that will be used as the state of the PPTS.

Definition 4.12 (Configuration)**.** Given a safety-critical human multitasking model $Is = \{I_1, \ldots, I_n\}$ where $I_i = \langle A, P, Inf, InitialState, Transitions, Tasks \rangle$, and a set *InterfaceStates* on P, the state of the PPTS is given by a *Configuration* $\mathcal{C}$ defined as follows:

$$\mathcal{C} = \langle tasks, CL, CS, WM, TS, GC \rangle$$

where:

- $tasks : Is \times \mathbb{N} \to Tasks$ is a mapping which, given an interface I_i and a natural number k, gives the k-th task of I_i;
- $CL : Is \times \mathbb{N} \to \mathbb{R}_{>0}$ is a mapping which, given an interface I_i and a natural number k, gives the cognitive load of the k-th task of I_i;
- $CS : Is \to InterfaceStates$ is a mapping which, given an interface I_i, gives its current state;
- WM is a Working Memory as defined in 4.11;
- $TS : Is \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ is a mapping which, given an interface I_i and a natural number k, gives the last time the k-th task of I_i has been executed;
- $GC : \mathbb{R}_{\geq 0}$ is a global clock, used to keep track of the execution time.

Inside each configuration $\mathcal{C}$ of the PPTS we have all the elements that possibly will be modified by the PPTS transitions. In particular, the cognitive load of each task is used to compute their attention attraction factor and is properly recomputed whenever a new subtask begins; as well as the current state of the interface which could change if a user performs an action on that interface. Again, while executing a basic task an information is deleted from WM and one is added inside it. And obviously, the last time a task has been executed changes every time a user selects one of the tasks of the model, as well as the global clock.

Now we present the initial state of the PTS, obtained by the model Is.

Definition 4.13 (Initial configuration)**.** Given a safety-critical human multitasking model $Is = \{I_1, \ldots, I_n\}$ where $I_i = \langle A, P, Inf, initialState_i, Transitions_i, \{T_1^i, \ldots T_m^i\}\rangle$, the initial configuration is

$$initConf = \langle tasks, CL, CS, WM, TS, GC\rangle$$

such that, $\forall i \in [1, n]$, $\forall k \in [1, m]$:

- $tasks(I_i, k) = T_k^i$
- $CL(I_i, k) = CogLoad_{T_k^i}$
- $CS(I_i) = initialState_i$
- $WM(I_i) = \{\mathbf{goal}(g_{T_1^i}), \ldots, \mathbf{goal}(g_{T_m^i})\}$
- $TS(I_i, k) = 0$
- $GC = 0$

In the initial configuration, each task of each interface of the model *Is* can be obtained by the mapping *tasks*, as well as the cognitive load of each task in *Is*. As regards the current state of each interface, two possibilities are considered: it can be a timed state associated with a timeout or an untimed state. The working memory *WM* is initialised by adding all the task goals inside it, for each task of each interface in *Is*. Finally the last time each task in *Is* has been executed, and the global clock, are set to 0.

Auxiliary Functions

We define a set of auxiliary functions which will be used in the inference rules defined in Subsection 4.2.2.

Notation 4.3. Given $WM = \langle Memory, size\rangle$ defined on *Is*

- $WM(I) = Memory(I)$ denotes the set of information associated with the interface *I*;

- $size(I) = size$ denotes the capacity of $Memory(I)$

Occupied. Given the Working Memory WM, the function $occupied(WM)$ gives the number of items in *Memory*:

$$occupied(WM) = \sum_{I \in Is} |WM(I)|$$

Minus. Given $mem \subseteq Inf$ and $i \in mem \cup \{noInfo\}$

$$mem - i = \begin{cases} mem \setminus \{i\}, & \text{if } i \neq noInfo \\ mem, & \text{if } i = noInfo \end{cases}$$

Plus. Given $mem \subseteq Inf$ and $i \in Inf$

$$mem + i = \begin{cases} mem \cup \{i\}, & \text{if } i \neq noInfo \\ mem, & \text{if } i = noInfo \end{cases}$$

WaitTime. Given $I = \langle A, P, Inf, InitialState, Transition, \{T_1, \ldots, T_m\} \rangle$, if the argument of the function *waitTime* is the i-th task of an interface I, the function gives the time it has not been executed, namely what we call *wait time*:

$$waitTime(I, i) = GC - TS(I, i)$$

If its argument is an interface I, the function gives the minimum wait time among all tasks of I:

$$waitTime(I) = min_{i \in [1,m]}(waitTime(I, i))$$

Enabled . Given a task $tasks(I, i)$ such that

$tasks(I, i) = \langle j | p \Longrightarrow l | k$ **duration** t **difficulty** d **delay** δ $BT_w :: STs, c, g \rangle$

the predicate $enabled(I, i)$ is true if the i-th task of I is enabled, namely if in $WM(I)$ is present its goal, if the delay of its first basic task is elapsed, and if the current state of the I has a timeout not yet expired.

$$enabled(I,i) = \begin{cases} true, & \text{if } \begin{array}{c} (CS(I) = p \textbf{ for time } t \ \wedge \ t > waitTime(I)) \\ \vee CS(I) = p \ \wedge \\ \textbf{goal}(g) \in WM(I) \wedge GC - TS(I,i) \geq \delta \end{array} \\ false, & \text{o.w.} \end{cases} \tag{4.5}$$

If the task $tasks(I, i) = \langle emptyTask, c, g \rangle$, the function $enabled(I, i)$ is false.

AllIdling . Given a model $Is = \{I_1, \ldots, I_n\}$, where the interface I_k is defined as $\langle A, P, Inf, InitialState, Transition, \{T_1, \ldots, T_m\} \rangle$, and where $T_y = \langle ST_1 :: STs, c, g \rangle$, the predicate $allIdling(Is)$ is true if in Is the delays of all the first basic tasks of each task of each interface, are higher than 0, false otherwise.

$$allIdling(Is) = \begin{cases} true, & \text{if } \forall\, I \in Is, \forall\, k \in [\,1, m] \ GC - TS(I_k) \geq delay_{0,ST_{1_y}} \\ false, & \text{o.w.} \end{cases} \tag{4.6}$$

MinDelay . Given a model $Is = \{I_1, \ldots, I_n\}$, where the interface I_k is defined as $\langle A, P, Inf, InitialState, Transition, \{T_1, \ldots, T_m\} \rangle$, and where $T_y = \langle ST_1 :: STs, c, g \rangle$, the function $minDelay(Is)$ gives the minimum time needed to reach a configuration where at least one of the tasks is not pausing.

$$minDelay(Is) = min(\sum_{I \in Is}(\sum_{k \in [1,m]} delay_{0,ST_{1_y}} + TS(I_k, y))) \tag{4.7}$$

4.2.2 Definition of the PPTS

Definition 4.14 (Semantics)**.** Given a model *Is*, the semantics is the purely probabilistic transition system $\langle \mathcal{C}, \xrightarrow{p} \rangle$ where $\xrightarrow{p} : \mathcal{C} \times [0,1] \times \mathcal{C}$ is the least probabilistic transition relation defined by the inference rules defined in what follows.

Interacting Rule

The interacting rule models the execution of what in Section 4.1 we defined a *user action*, namely an action performed on the device to which the user directed his/her attention.

$$\frac{\begin{array}{c} I \in Is \qquad enabled(I,i) \\ CS(I) = p \vee CS(I) = p \textbf{ for time } t \qquad p \xrightarrow{l} q \in Transitions(I) \\ tasks(I,i) = \langle BT_0 BT_1 \ldots BT_n :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ j \in WM(I) \qquad l \neq g \qquad l \neq noAction \\ WM' = WM\left[I \mapsto WM(I) - j + k\right] \\ occupied(WM') \leq size(WM) \end{array}}{\begin{array}{c} \left\langle tasks, CL, CS, WM, TS, GC \right\rangle \xrightarrow{\Phi_{I,i}} \\ \left\langle tasks\left[(I,i) \mapsto \langle BT_1 \ldots BT_n :: STs, c, g \rangle\right], CL, \right. \\ \left. CS\left[I \mapsto q\right], WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t' \right\rangle \end{array}} \; Interacting_1$$

The $Interacting_1$ rule models the execution and removal from $\mathcal{C}$ of the first basic task (BT_0) of the task (I,i) of the interface I, if it is not the last basic task of the current subtask. Such task is selected among the others in $\mathcal{C}$ with a probability $\Phi_{I,i}$ (presented below). The rule is applied if and only if the following conditions are satisfied:

- the interface I must be an interface of the model Is ($I \in Is$);

- the basic task to be executed must synchronise with one of the interface transitions: namely the perception and the action to perform of both transition and basic task must be the same ($p \xrightarrow{l} q$ and $j \mid p \implies l \mid k \ \ldots$);
- also, the current state of the interface I must have the same perception of the basic task and the transition ($p \vee p$ **for time** t);
- the action to perform (l) must be different from *noAction* and from the goal action g;
- the information to be retrieved from the working memory to perform the basic task (j) must be in the memory associated with the interface I;
- the number of items in the rewrote memory ($occupied(WM')$) – i.e. after the execution of the rule – must not exceed the capacity of the working memory ($size(WM)$) so that the addition of a new information (k) would not lead to the memory overload.

If all these conditions are satisfied then:

- the basic task executed is removed from the configuration ($tasks\,[(I,i) \mapsto \langle BT_1 \ldots BT_n :: STs, c, g\rangle]$);
- the current state of the interface I is updated ($CS\,[I \mapsto q]$);
- the information j is replaced in memory by the information k ($WM\,[I \mapsto WM(I) - j + k]$);
- the time the interface I has been executed for the last time is updated ($TS\,[(I,i) \mapsto GC + t']$);
- the global clock is updated with the duration of the basic task executed ($GC + t'$).

If the basic task executed is the last one of the current subtask, a new cognitive load value must be computed, it is then applied the rule $Interacting_2$, presented in what follows.

$$\frac{\begin{array}{c} I \in Is \qquad enabled(I,i) \\ CS(I) = p \vee p \,\textbf{for time}\, t \qquad p \xrightarrow{l} q \in Transitions(I) \\ tasks(I,i) = \langle BT_0 :: STs, c, g\rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \,\textbf{duration}\, t' \,\textbf{difficulty}\, d \,\textbf{delay}\, \delta \\ l \neq g \qquad l \neq noAction \qquad j \in WM(I) \\ WM' = WM\left[I \mapsto WM(I) - j + k\right] \\ occupied(WM') \leq size(WM) \end{array}}{\begin{array}{c} \left\langle tasks, CL, CS, WM, TS, GC\right\rangle \xrightarrow{\Phi_{I,i}} \\ \left\langle tasks\left[(I,i) \mapsto \langle STs, c, g\rangle\right], CL\left[(I,i) \mapsto CogLoad_{\langle STs,c,g\rangle}\right], \right. \\ \left. CS\left[I \mapsto q\right], WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t'\right\rangle \end{array}} \; Interacting_2$$

The rule has the same conditions as the $Interacting_1$, except for the task selected, whose first basic task is the last one the of the current subtask ($tasks(I,i) = \langle BT_0 :: STs, c, g\rangle$).

Since the cognitive load of the task is computed each time a new subtask begins (and it remains the same throughout its execution), after the execution of the basic tasks and its removing from the configuration it is necessary to compute a new value for the task cognitive load ($CL\left[(I,i) \mapsto CogLoad_{\langle STs,c,g\rangle}\right]$), using the function $CogLoad_T$ defined in 4.1.

Selection Probability

As previously mentioned, each task is characterised by a measure of how much it attracts the user attention, namely the a-factor, defined in Equation 4.4.

We instantiate such definition to the current configuration, considering also when the task is enabled or not (see function 4.5). We thus define $a_{I,i}$ as follows:

$$a_{I,i} = CL(I,i) \times c \times (GC - TS(I,i)) \tag{4.8}$$

We then compute for each task in the configuration, a probability to be selected

proportional to its a, called the probability Φ and defined as follows.

Definition 4.15 (Φ probability)**.** Given a model $Is = \{I_1, \ldots, I_n\}$ where $I_k = \langle A, P, Inf, InitialState_i, Transitions_i, \{T_1^i, \ldots, T_m^i\}\rangle$, the probability $\Phi_{I,i}$ is defined as:

$$\Phi_{I,i} = \frac{a_{I,i}}{\sum_{I_k \in Is}(\sum_{j \in [1,m]} a_{I_k,j})} \tag{4.9}$$

Since there are as many transition as enabled tasks, all probabilities sum to 1.

Cognitive

The cognitive rule models the execution of what in Section 4.1 we defined a *cognitive basic task*, namely a change of the mental plan of the user, without any involvement with the interface.

$$\frac{\begin{array}{c} I \in Is \qquad enabled(I,i) \\ tasks(I,i) = \langle BT_0 BT_1 \ldots BT_n :: STs, c, g\rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ l = noAction \qquad j \in WM(I) \qquad k \in COG \\ WM' = WM\left[I \mapsto WM(I) - j + k\right] \\ occupied(WM') \leq size(WM) \end{array}}{\begin{array}{c} \langle tasks, CL, CS, WM, TS, GC\rangle \xrightarrow{\Phi_{I,i}} \\ \langle tasks\left[(I,i) \mapsto \langle BT_1 \ldots BT_n :: STs, c, g\rangle\right], CL, \\ CS, WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t'\rangle \end{array}} \; Cognitive_1$$

The rule $Cognitive_1$ is applied if the cognitive basic task executed is not the last one of the current subtask. The rule models the execution and removal from C of the cognitive basic task BT_0, selected with probability $\Phi_{I,i}$. Some of the conditions to be satisfied for the application of the rule are equal to those for the rule $Interacting_1$: the updated memory must not overlead the capacity of the working memory, the information j must be present in the memory associated wiht the interface I, the

interface I must be an interface of the model Is, and the task selected (I, i) must be enabled. Nevertheless, other conditions must be satisfied, such as:

- the action to perform must be equal to *noAction*, since the rule models a change of the user memory without any involvement in the interface;
- the information to be added in memory must be a cognition $(k \in COG)$.

If all the conditions are satisfied, here too, the basic task executed is removed from the configuration, the information j is replaced in memory from the cognition k and the time the task has been executed for the last time, is updated, as well as the global clock. However, no change is done in the current state, since the user does not perform any action on the interface.

Here again, if the basic task executed is the last one of the current subtask, it is applied the rule $Cognitive_2$, which updates the new value of the cognitive load of the task (I, i) with the function $CogLoad_T$. It is defined as follows.

$$\frac{\begin{array}{c} I \in Is \qquad enabled(I,i) \\ tasks(I,i) = \langle BT_0 :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ l = noAction \qquad j \in WM(I) \qquad k \in COG \\ WM' = WM\left[I \mapsto WM(I) - j + k\right] \\ occupied(WM') \leq size(WM) \end{array}}{\begin{array}{l} \left\langle tasks, CL, CS, WM, TS, GC \right\rangle \xrightarrow{\Phi_{I,i}} \\ \left\langle tasks\left[(I,i) \mapsto \langle STs, c, g \rangle\right], CL\left[(I,i) \mapsto CogLoad_{\langle STs,c,g \rangle}\right], \right. \\ \left. CS, WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t' \right\rangle \end{array}} \; Cognitive_2$$

Closure

The rule closure models the achievement of the goal for the selected task, namely the execution of the goal action g inside the task.

$$\frac{\begin{array}{c} I \in Is \qquad enabled(I,i) \\ CS(I) = p \vee CS(i) = p \textbf{ for time } t \qquad p \xrightarrow{l} q \in Transitions(I) \\ tasks(I,i) = \langle BT_0 \ldots BT_n :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ j \in WM(I) \qquad l \neq noAction \qquad l = g \\ WM' = WM\left[I \mapsto WM(I) - j + k - \textbf{goal}(g)\right] \\ occupied(WM') \leq size(WM) \end{array}}{\begin{array}{c} \left\langle tasks, CL, CS, WM, TS, GC \right\rangle \xrightarrow{\Phi_{I,i}} \\ \left\langle tasks\left[(I,i) \mapsto \langle emptyTask, c, g \rangle\right], CL, \right. \\ \left. CS\left[I \mapsto q\right], WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t' \right\rangle \end{array}} \; Closure$$

The rule models the execution of the basic task BT_0 of the enabled task (I,i), selected with probability $\Phi_{I,i}$. Here again, some conditions are equal to those for the rule $Interacting_1$: the memory must not overload after the execution of the basic task, the information j must be present in the memory associated with I, the basic task executed and one of the transitions of I must synchronise, as well as the current state of I must have the same perception as the one of the basic task and the transition. However, in such a case, the action to perform *must* be the action that will lead to the achievement of the goal g.

After the execution, the task (I,i) is removed from the configuration (therefore the subtask sequence becomes empty); the current state of I is updated, as well as the time the task has been executed for the last time and the global clock. Finally, in the memory associated with I the information k replaces the information j, and the goal of the task just executed ($\textbf{goal}(g)$) is removed from the memory associated with I.

Forgetting

When the execution of the basic task exceeds the capacity of the memory, an item must be forgotten (i.e. deleted from WM) to make room to the new information.

Since maintaining information in *WM* requires the user's attention, and the interface with the highest wait time is the one to which the user has not given attention for the longest time, one of the information related to the interface chosen with a probability Υ proportional to its wait time, is randomly forgotten. The probability Υ will be presented below.

The memory could overload for the execution of both user action basic task and cognitive basic task: we will present two similar rules (*ForgetInteracting* and *ForgetCognitive*) to model respectively these two situations. Both rules are specified in two versions: one version for the case where the basic task executed is the last one of the current subtask, the second version otherwise. We will present all of them in what follows.

$$\frac{\begin{array}{c} I, I' \in Is \qquad enabled(I, i) \\ CS(I) = p \vee CS(I) = p \textbf{ for time } t \\ p \xrightarrow{l} q \in Transitions(I) \\ tasks(I, i) = \langle BT_0 BT_1 \ldots BT_n :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ l \neq g \qquad l \neq noAction \\ z \in WM(I') \qquad j = noInfo \qquad k \neq noInfo \\ WM' = WM\left[I \mapsto WM(I) - j + k, I' \mapsto WM(I') - z\right] \\ occupied(WM) = size(WM) \end{array}}{\begin{array}{c} \langle tasks, CL, CS, WM, TS, GC \rangle \xrightarrow{\Phi_{I,i} \Upsilon_{I'}} \\ \langle tasks\left[(I, i) \mapsto \langle BT_1 \ldots BT_n :: STs, c, g \rangle\right], CL, \\ CS\left[I \mapsto q\right], WM', TS\left[(I, i) \mapsto GC + t'\right], GC + t' \rangle \end{array}} \; ForgetInteracting_1$$

$$\frac{\begin{array}{c} I, I' \in Is \qquad enabled(I, i) \\ CS(I) = p \vee CS(I) = p \,\textbf{for time}\, t \\ p \xrightarrow{l} q \in Transitions(I) \\ tasks(I, i) = \langle BT_0 :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \,\textbf{duration}\, t' \,\textbf{difficulty}\, d \,\textbf{delay}\, \delta \\ l \neq g \qquad l \neq noAction \\ z \in WM(I') \qquad j = noInfo \qquad k \neq noInfo \\ WM' = WM\left[I \mapsto WM(I) - j + k, I' \mapsto WM(I') - z\right] \\ occupied(WM) = size(WM) \end{array}}{\begin{array}{l} \langle tasks, CL, CS, WM, TS, GC \rangle \xrightarrow{\Phi_{I,i}\Psi_{I'}} \\ \langle tasks\left[(I, i) \mapsto \langle STs, c, g \rangle\right], CL\left[(I, i) \mapsto CogLoad_{\langle STs, c, g \rangle}\right], \\ CS\left[I \mapsto q\right], WM', TS\left[(I, i) \mapsto GC + t'\right], GC + t' \rangle \end{array}} \; ForgetInteracting_2$$

$$\frac{\begin{array}{c} I, I' \in Is \qquad enabled(I, i) \\ tasks(I, i) = \langle BT_0 BT_1 \ldots BT_n :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \,\textbf{duration}\, t' \,\textbf{difficulty}\, d \,\textbf{delay}\, \delta \\ l = noAction \qquad z \in WM(I') \\ j = noInfo \qquad k \neq noInfo \qquad k \in COG \\ WM' = WM\left[I \mapsto WM(I) - j + k, I' \mapsto WM(I') - z\right] \\ occupied(WM) = size(WM) \end{array}}{\begin{array}{l} \langle tasks, CL, CS, WM, TS, GC \rangle \xrightarrow{\Phi_{I,i}\Psi_{I'}} \\ \langle tasks\left[(I, i) \mapsto \langle BT_1 \ldots BT_n :: STs, c, g \rangle\right], CL, \\ CS, WM', TS\left[(I, i) \mapsto GC + t'\right], GC + t' \rangle \end{array}} \; ForgetCognitive_1$$

$$\frac{\begin{array}{c} I, I' \in Is \qquad enabled(I, i) \\ tasks(I, i) = \langle BT_0 BT_1 \dots BT_n :: STs, c, g \rangle \\ BT_0 = j \,|\, p \implies l \,|\, k \textbf{ duration } t' \textbf{ difficulty } d \textbf{ delay } \delta \\ l = noAction \qquad z \in WM(I') \\ j = noInfo \qquad k \neq noInfo \qquad k \in COG \\ WM' = WM\left[I \mapsto WM(I) - j + k, I' \mapsto WM(I') - z\right] \\ occupied(WM) = size(WM) \end{array}}{\begin{array}{c} \left\langle tasks, CL, CS, WM, TS, GC \right\rangle \xrightarrow{\Phi_{I,i}\Psi_{I'}} \\ \left\langle tasks\left[(I,i) \mapsto \langle STs, c, g \rangle\right], CL\left[(I,i) \mapsto CogLoad_{\langle STs,c,g \rangle}\right], \right. \\ \left. CS, WM', TS\left[(I,i) \mapsto GC + t'\right], GC + t' \right\rangle \end{array}} \; ForgetCognitive_2$$

As the specification of the rules shows, the probability to go from a configuration to the other is the multiplication of the probability Φ_T that the task T will be selected and executed, for probability Ψ_I that the interface I will be selected to delete a random information from its memory.

Deletion Probability

As previously mentioned, in the forgetting rules one information related to the interface chosen with a probability proportional to its maximum wait time, is randomly forgotten. We call such probability Ψ and we defined it as follows.

Definition 4.16 (Ψ probability)**.** Given a model $Is = \{I_1, \dots, I_n\}$ where $I_k = \langle A, P, Inf, InitialState, Transitions, \{T_1, \dots, T_{m_k}\} \rangle$, the probability Ψ_I is defined as:

$$\Psi_I = \frac{waitTime(I)}{\sum_{I \in Is} waitTime(I_k)} \cdot \frac{1}{|WM(I)|} \tag{4.10}$$

Essentially, for each interface in the the model Is, the probability Ψ is proportional to their maximum wait time, moreover it is taken into account also the num-

ber of elements in the set of information associated with each interface, considering that one of that information will be randomly forgotten.

Since there are as many transitions as the number of elements in WM and the number of interfaces in Is, then all probabilities sum to 1.

All idling

Finally, if all the delays of each first basic task of each task in the configuration $\mathcal{C}$ are higher than 0 (i.e. all tasks are pausing), then the time passes until some task is no more pausing.

$$\frac{I \in Is \qquad enabled(I, i) \qquad allIdling(Is)}{\langle tasks, CL, CS, WM, TS, GC \rangle \xrightarrow{1} \langle tasks, CL, CS, WM, TS, GC + minDelay(Is) \rangle} \; AllIdling$$

The rule is applied if at least one of the interface in Is is enabled, and if all the interfaces in Is are pausing. In such case, just the global clock is updated with the earliest time when the delay of some basic task reaches 0.

4.3 Example

In order to better explain the syntax and the semantics of our model let us show a simple example of the concurrent interaction with 3 tasks.

Given a safety-critical human multitasking model $Is = \{I_1, I_2\}$ the interfaces I_1 and I_2 are defined as follows (according to Definition 4.3):

$$I_1 = \langle \{g_1, g_2\}, \{p, p', q, q'\}, \{info_1\}, p, \{p \xrightarrow{a} p', q \xrightarrow{a} q'\}, \{T_{1_1}, T_{1_2}\} \rangle$$

$$I_2 = \langle \{c, g_3\}, \{k, k'\}, \{\}, k, \{k \xrightarrow{c} k'\}, \{T_{2_1}\} \rangle$$

Where the tasks T_{1_1}, T_{1_2}, and T_{2_1} are defined as follows (according to Definition 4.5):

$T_{1_1} = \langle info_1 \mid p \;\Rightarrow\; g_1 \mid noInfo$ **duration** 3 **difficulty** 5 **delay** $0, 0.5, g_1\rangle$,

$T_{1_2} = \langle noInfo \mid q \;\Rightarrow\; g_2 \mid noInfo$ **duration** 2 **difficulty** 9 **delay** $12, 0.3, g_2\rangle$

$T_{2_1} = \langle noInfo \mid k \;\Rightarrow\; g_3 \mid noInfo$ **duration** 6 **difficulty** 4 **delay** $0, 12, g_3\rangle$

As the specification shows, the 3 tasks T_{1_1}, T_{1_2}, and T_{2_1} consist of a single basic task, whose action is the goal action (g_1, g_2, and g_3). As regards the characteristics of each task:

- Task T_{1_1} has a cognitive load equal to $CL(I_1, 1) = \frac{5\cdot 3}{5} = 5$ (computed with Equation 4.1), and a criticality of 0.5.
- Task T_{1_2} has a cognitive load equal to $CL(I_1, 2) = \frac{9\cdot 2}{9+12} = 1.2857$, a criticality of 0.5, and a delay of 12 which means that will be enabled after 12 time units.
- Task T_{2_1} has a cognitive load equal to $CL(I_2, 1) = \frac{4\cdot 6}{6} = 4$, and a criticality of 12.

In Figure 4.3.1 is shown how part of the transition system would evolve from the initial configuration *ic* defined in Definition 4.13. In figure are shown just the global clock and the tasks which have not been executed yet. The other information about the configuration, included the working memory, are omitted for the sake of simplicity.

At the beginning, the tasks enabled are T_{1_1} and T_{2_1}. Their α-factor is:

- $\alpha_{1_1} = 5 \times 0.5$
- $\alpha_{2_1} = 4 \times 12$

Computed as the product of cognitive load and criticality according to Equation 4.1.1 (note that in the computation of the α-factor, their wait time is not taken into account since it is equal to 0).

The task T_{1_1} is then chosen with probability 0.0495, and the task T_{2_1} is chosen with probaility 0.9505.

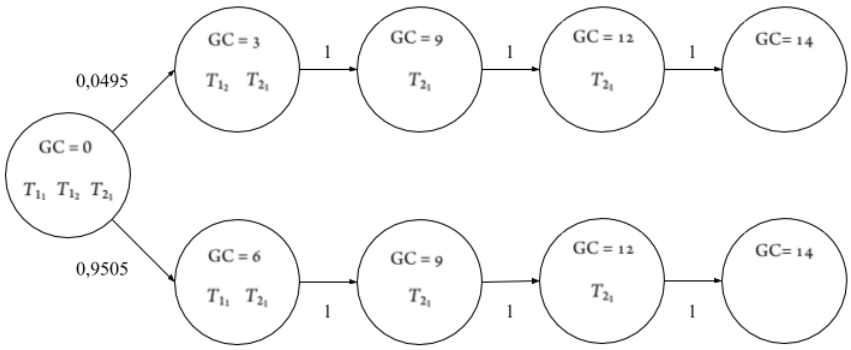

Figure 4.3.1: Part of the transition system.

Once such choice is done, at the next step it is executed the enabled task that has not been previously executed. Once both are executed, and thus when the global clock is equal to 9, in the configuration is present just the task T_{1_2}, which has nevertheless a delay not yet elapsed. Therefore, both configurations idle until the global clock is equal to 12 and the task T_{1_2} is executed.

5

Model Simulator

We implement a slightly simplified version of the formal model presented in Chapter 4 as a simulator, in order to simulate the human selective attention and the functioning of the working memory in the case of users involved in multiple tasks requiring different cognitive effort, some of which may be safety-critical [18].

Simulations allow us to get a quick feedback about whether a user can safely perform multiple tasks or whether one of the tasks can deflect the user' attention from another, possibly critical, task. We found that the proposed simulation algorithm simulates human selective attention in accordance with the description of its functioning in the psychological literature.

This chapter presents the safety-critical human multitasking simulator.

5.1 Simulator

The simulator is implemented in Java. The full specification is available at http://www.di.unipi.it/msvbio/software/AttentionSim.html

We implement a simplified version of human multitasking where users are allowed to perform a single task on each interface, we then decide not to model the interfaces.

We implement a Java class for each element of human multitasking presented in Chapter 4 (except for the interface objects). According to this, we have:

- a `BasicTask` class, where each `BasicTask` object has five parameters:
  - two parameters `information1` and `information2` of type String
  - a parameter `duration` of type Integer
  - a parameter `difficulty` of type Double
  - a parameter `delay` of type Integer

  denoting respectively the information to retrieve from the working memory and the information to replace in the working memory during the execution of the basic task, and the duration, difficulty and delay of the basic task. We do not model a parameter regarding the actions to perform on the interface since we do not model interfaces.

- a `Subtask` class, where each `Subtask` object is implemented as vector of `BasicTask` objects. The `cognitiveLoad()` function computes the cognitive load of a subtask according to the equation 4.1

- a `Task` class, where each `Task` object has five parameters:
  - a parameter `taskId` of type Integer
  - a parameter `stSequence` which models the sequence of the subtasks of the task as a vector of `Subtask` objects

  - a parameter `criticality` of type Double denoting the criticality level of a task
  - a parameter `timeStamp` of type Integer denoting the last time the task has been executed.
  - a parameter `status` of type String, which we add at the model for analysis purposes and which has value `"ongoing"` if everything in the interaction is going well, value `"completed"` when the task is completed, and value `"interrupted"` when something in the interaction goes wrong and the task is interrupted

  In the `Task` class are defined a set of auxiliary functions:

  - `update()` which removes the first element of the vector of subtasks if it is empty, and updates the cognitive load of the current subtask
  - `alpha()` which returns the attention attraction factor a of each task that is computed according to equation 4.1.1.
  - `hd()` which returns the first basic task of the task
  - `tl()` which removes the first basic task of the task (the first element of the vector of `BasicTask` objects of the first vector of `Subtask` objects of the task)
  - `completed()` returns true if the task is completed and false otherwise

- a `WorkingMemory` class, where a `WorkingMemory` object has two parameters:
  - a parameter `capacity` of type integer denoting what in Chapter 4 we called *size*
  - a parameter `memory` implemented as a hash table which maps integers (representing the identifier of each task) to a vector of strings (representing the set of information necessary for the completion of that task), denoting what in Chapter 4 we called *Memory*

- a `Configuration` class, which models the state of the simulation. An instance object of such class contains
  - a vector of `Task` objects
  - a `WorkingMemory` object
  - variable `globalClock` of type Integer

  Moreover, in the class `Configuration` are defined a set of auxiliary functions:
  - `enabled()` which returns a vector of `Task` objects whose first basic task must not have a delay higher than zero. Moreover, each task must have a goal inside the memory associated with it, it must be not completed, and its status must be `"ongoing"`
  - `waitingTasks()` which returns a vector of `Task` objects whose first basic task has a delay higher than zero. Also in this case, each task must have a goal inside the memory associated with it, it must be not completed, and its status must be `"ongoing"`
  - `completed()` which returns true if all tasks in the configuration are completed
  - `removeInformation(String information, int taskId)` which removes the `information` from the memory mapped to the `taskId`, when the information is not `noInfo`
  - `addInformation(String information, int taskId)` which adds the `information` to the memory mapped to the `taskId` , when the information is not `noInfo` . If the memory would overload with the addition of the information, it first calls the function `overload()` presented below
  - `overload()` which removes a random information from the memory associated with a task chosen with a probability proportional to the

time it has not been executed (namely the `globalClock` minus its `timeStamp`

- a `Simulator` class: where the algorithm for simulating selective attention is specified. We define it in Algorithm 1.

The algorithm performs a main loop that essentially executes one basic task in each iteration. The basic task to be executed is the first basic task of one of the enabled tasks. For each such candidate basic task, its *attention attraction factor* a_i is computed and it then has a probability of being chosen that is proportional to a_i. Once a basic task has been chosen:

- the global clock is updated with the duration of the basic task executed;
- the timestamp of the chosen task is updated;
- the second information of the basic task executed replaces its first information in the memory associated with the task chosen;
- the basic task executed is removed from the configuration;
- if the basic task executed was the last one of the current subtask, such subtask is removed from the configuration and the cognitive load of the task is updated;
- if the task chosen is completed, its status becomes `completed` .

If the algorithm reaches a configuration where no task is enabled, the main loop performs an iteration where only the global clock is updated with the minimum value needed to reach a configuration where at least one task is not pausing (namely, the minimum among all sums of the timestamp of each task and the delay of its first basic task).

In order to show that the proposed simulation algorithm simulates selective attention in accordance with relevant literature, let us first consider a variant of the

Algorithm 1 Algorithm for simulating selective attention on a configuration c

```
1:  while not c.completed() do
2:    if c.enabled() ≠ ∅ then
3:      for all Task ∈ c.enabled() do
4:        Task.alpha()
5:      end for
6:      choose Task_i ∈ c.enabled() with probability a_i / Σ_{k=0}^{c.enabled().size} a_k
7:      c.globalClock := c.globalClock + Task_i.hd().duration
8:      Task_i.timeStamp := c.globalClock
9:      c.removeInformation(T_i.hd().information1, i)
10:     c.addInformation(T_i.hd().information2, i)
11:     Task_i.tl()
12:
13:     if Task_i.stSequence[0].isEmpty() then
14:       Task_i.update()
15:     end if
16:
17:     if Task_i.completed() then
18:       Task_i.status := "completed"
19:     end if
20:   else if c.waitingTasks() ≠ ∅ then
21:     c.globalClock := min{Task_k.timeStamp + Task_k.hd().delay |
        Task_k ∈ c}
22:   end if
23: end while
```

algorithm that does not take the task wait time (namely the time the task has not been executed) into account when computing the a factor.

Let us consider two concurrent tasks with the same criticality level and each consisting of k identical basic tasks: we refer to the duration and difficulty parameters of each basic task of the first task (T_1) as t_1 and d_1; and to the duration and difficulty parameters of each basic task of the second task (T_2) as t_2 and d_2.

In order to complete both tasks, the simulation algorithm performs exactly $2k$ steps (where a step represents the execution of a single basic task). Since the two tasks have the same criticality level, the probability of completing task T_1 at step

n, with $k \leq n \leq 2k$, is

$$P(T_1, n) = \left(\frac{t_1 d_1}{t_1 d_1 + t_2 d_2}\right)^k \left(\frac{t_2 d_2}{t_1 d_1 + t_2 d_2}\right)^{(n-k)} \binom{n-1}{n-k}.$$

The expected number of steps necessary to complete task T_1 can therefore be given as $E[T_1] = \sum_{n=k}^{2k} P(T_1, n)n$, that corresponds to

$$E[T_1] = \left(\frac{t_1 d_1}{t_1 d_1 + t_2 d_2}\right)^k \sum_{n=k}^{2k} \left(\frac{t_2 d_2}{t_1 d_1 + t_2 d_2}\right)^{(n-k)} \binom{n-1}{n-k} n.$$

The formula shows that the expected number of steps for the completion of T_1 increases with the difficulty and duration of the basic tasks of T_2, namely, it increases when the CL of T_2 increases. Hence, the algorithm simulates the switching of attention in agreement with what described in [13] and [44]. However, since the task wait time is not taken into account, this variant of the algorithm could lead to unrealistic starvation cases (e.g., the algorithm could repeatedly skip a task with very low criticality level and CL).

Let us now discuss what changes when the task wait time is taken into account. Formula $E[T_1]$ becomes more complex since the repeated probabilistic events are no longer independent. However, the structure of the formula remains the same, with a result that is still increasing with the difficulty and duration of the basic tasks of T_2 (in agreement with [13] and [44]). In addition to this, the wait time tends to favour at each step the task that has not been chosen in the previous step. As a consequence, the regular alternation of T_1 and T_2 is promoted with, as a result, a reduced variance in the distribution of the number of steps necessary to complete T_1. Hence, the use of wait time reduces the probability of unnatural starvation among the tasks.

6

Model Validation

We validate the human selective attention algorithm proposed against data gathered from an experimental study performed with real users involved in the interaction with two concurrent tasks: one representing a "main" critical task, the other representing a "distractor" task with different levels of cognitive load.

As presented in Chapter 4, we compute for each task an a-factor representing the likelihood the task will attract the user's attention. At each step of the interaction the user chooses the task to be executed with a probability proportional to its a. The a-factor of a task (see Equation 4.4) is described as the product of three parameters: the cognitive load of the current subtask, the criticality level of the task, and the time elapsed since the last time the task has been executed.

Although the proposed mechanism simulating the switching of attention between tasks is consistent with psychological literature and results from experimen-

tal psychological studies, we conducted an experimental study with real users involved in a multitasking interaction on a web application with a "main" critical task and a secondary "distractor" task. Essentially, the main question we want to answer is:

Does the computation of the α-factor "mirror" the tasks prioritisation which real users perform in a safety-critical multitasking context?

The experiment and the analysis of the experimental data, together with a proper development of simulations, based on such data, allow us to fine-tune the proposed model and to validate it.

We will present the experimental study in Section 6.1, the design of the simulation experiments in Section 6.2 and the results we obtained in Section 6.3.

6.1 Experimental Study

The development of the web application for the experimental study is part of a collaboration with the psychologists Prof. Carmen Berrocal Montiel and Dr. Cristina Belviso of the University of Pisa, which lead to the definition of a set of appropriate tasks for the validation of the proposed algorithm. We defined two separated tests: one for evaluating the WM performances of the participants, and one called *shared attention test* where users were asked to interact with two tasks concurrently.

6.1.1 Working Memory Span Tasks

Before the shared attention test, we administrate to the participants two different working memory span tasks, in order to identify different inclinations to multitasking, so that we can underline common characteristics and differences to realise more precise simulation tests.

We administrate two different WM span tasks: the reading span task (RST) [43], and the operation span task (OST) [108] [1]. In both, a sequence of numbers

[1] Available at http://pages.di.unipi.it/milazzo/AppSpans/

of variable length (from 2 to 5 numbers) is presented in the screen; each number is spaced by a sentence (RST) or an equation (OST) to evaluate. When all numbers are presented to the users, they have to recall the numbers in the exact order they were presented. We administrate 3 test repetitions for each sequence length: in total 12 repetitions for both RST and OST. As regards the procedure for measuring the WM capacity, we use the partial-credit unit scoring (PCU).

6.1.2 Shared attention test

As regards the shared attention test[2], we defined two tasks, (i) a main and critical one, and (ii) a secondary distracting task (with different levels of cognitive load):

i. As shown in Figure 6.1.1, in the main task users visualise on the screen a chain of 9 rings and a black pellet which randomly moves left and right along the chain. Every time the task starts the black pellet is on the green ring and moves randomly every second.

   Users are asked to avoid that the black pellet reaches one of the red rings at the two ends of the chain, by pushing two buttons on the screen which move the pellet in the two directions: if they do not succeed, the task fails.

ii. In the secondary distracting task (shown in Figure 6.1.2), users visualise on the screen a sequence of boxes and a keyboard. At cyclic intervals, a letter appears inside a box; letters appear one by one until all boxes are full.

   Users have to find and push on the keyboard the letter corresponding to the one inside the box indicated by the arrow, until all the letters are inserted in the same order they were presented. Every time they have to insert a new letter (i.e. the previous letter has been successfully inserted and the next one has appeared) the keyboard changes.

   Such activity has a total duration expressed through a timeout, visualised by a decreasing number and a black bar whose length decreases. Once such

[2] Available at http://pages.di.unipi.it/milazzo/AppSpans2/

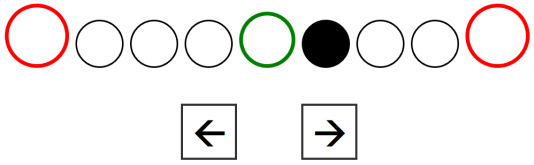

Figure 6.1.1: Main critical task.

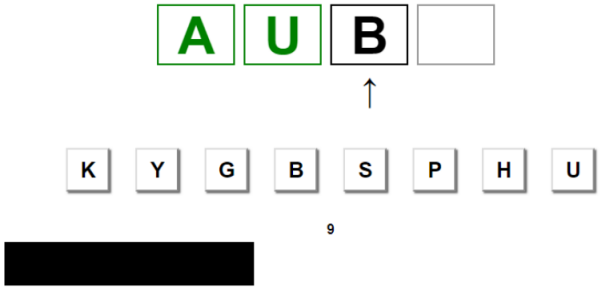

Figure 6.1.2: Secondary distracting task.

timeout expires the task is concluded: if the user did not succeed in inserting all the letters, the task is considered failed, otherwise the task succeeds.

The tasks are presented on two separate tabs of the same window: users can see only one of the two tabs at a time, and they can switch from one to another by pushing the space bar. So, the user has to perform the two tasks concurrently by interleaving them. Both tasks have to be completed successfully.

As already mentioned, the secondary task is instantiated with different levels of CL (see Equation 4.1). In order to do this, three different parameters of the secondary task are differentiated: the number of letters to insert (*letters*) , the length of the keyboard (*keys*), namely the number of keys composing the keyboard, and the total duration of the task (*duration*).

To vary the CL of a task we need to manipulate the total duration of the task and what in Chapter 4 we called difficulty factor (i.e. a measure of the temporal density of difficulty of the task). As regards the total duration, we define three possible durations of 18, 22, and 26 seconds. As regards the difficulty factor, we

letters	*2 letters*					
keys	*Single*			*Double*		
duration	18	22	26	18	22	26

letters	*4 letters*					
keys	*Single*			*Double*		
duration	18	22	26	18	22	26

letters	*8 letters*					
keys	*Single*			*Double*		
duration	18	22	26	18	22	26

Table 6.1.1: Values of parameters letters, keys, and duration.

variate:

- the parameter letters by defining three kinds of task where users have to insert 2 letters, 4 letters, and 8 letters respectively;
- the parameter keys by defining a kind of task where users should search for the right letter in a *single* keyboard (i.e. the number of keys composing the keyboard is equal to the number of letters to be inserted), and a kind of task where they have to search for the letter in a *double* keyboard (i.e. the number of keys is twice as many letters to be inserted). Every time a new letter has to be inserted the keyboard changes.

In total we have 18 different levels of CL for the secondary task (see Table 6.1.1), and the web application administrates 3 test repetitions for each level (presented randomly): in total users have to perform 54 test repetitions. All these values (both for the tasks' parameters and for the number of items to perform) have been the results of the collaboration with the team of psychologists, with which we defined the values in order that the test would be significant and valid.

It is worth to note that the criticality of the main task and the difficulty of each single action for all levels of CL of the secondary task are parameters that can change from user to user, for the way they perceive it to be.

6.1.3 Participants

The definition of the experimental study has been submitted to the ethical committee of the University of Pisa, which authorised the administration of the test. To take part in the experimental study, participants were asked to sign an informed consent form and a consent for the processing of personal data.

We performed two test sessions taking care that the environment, the provided equipment, and the test timetable were the same in both sessions. Were excluded from the study persons suffering from cognitive functions disorders or consuming drugs with an effect on such functions. Participation has been voluntary and without any incentive; participants were free to abandon the test at any time.

In total, 26 participants took part in the experimental study mother-tongue Italian, of both sexes (60% men, 40% women), aged between 18 and 40 years old, with a normal visual acuity (or correct by lenses).

6.1.4 Data Collection

WM Span Tasks. The web application is able to collect users answers for both WM span tasks for each item, and it is thus able to compute the PCU for the OST task and for the RST task. From such data, we can compute the total PCU score for each participant, calculated as the mean of the scores of both tasks.

Total PCU values go from a minimum of 0.35 to a maximum of 0.97 (see Table 6.1.2).

Shared Attention Test. For the shared attention test, the web application is able to track:

- the length of the keyboard, the number of letters to insert and the time to complete the task;
- the time a letter appears on the screen;
- the time a user pushes a letter on the keyboard;

id	OST PCU	RST PCU	TOTAL PCU
		I session	
1	0.88	0.97	0.925
2	0.795	0.78	0.7875
3	0.79	0.8	0.795
4	0.93	0.87	0.9
5	0.93	0.92	0.925
6	0.87	0.72	0.795
7	0.84	0.87	0.855
8	0.77	0.73	0.75
9	0.93	0.9	0.915
10	0.79	0.79	0.79
		II session	
11	0.80	0.71	0.7525
12	0.649	0.8	0.7245
13	0.2	0.5	0.35
14	0.92	0.93	0.925
15	0.8	0.75	0.775
16	0.89	0.86	0.875
17	0.9	0.9	0.9
18	0.88	0.93	0.905
19	0.67	0.755	0.7125
20	0.98	0.81	0.895
21	0.85	0.96	0.905
22	0.84	0.85	0.845
23	0.75	0.75	0.75
24	0.83	0.84	0.835
25	0.97	0.97	0.97
26	0.825	0.91	0.8675

Table 6.1.2: PCU scores of the 26 participants.

- each time a user fails in pushing the correct letter on the keyboard;
- each time a user succeeds in pushing the correct letter on the keyboard;
- each time a user passes from a task to the other;
- each time the black pellet reaches a red ring;
- each time a user fails in inserting the correct sequence of letters in time.

In order to identify a correlation between the users' PCU and their multitasking performance – which would be consistent with the relevant psychological literature –, we divide PCU values into 3 intervals and we divide participants into 3 different groups:

1. *lowPCU*: user total PCU ≤ 0.80;
2. *mediumPCU*: $0.80 <$ user total PCU ≥ 0.90;
3. *highPCU*: user total PCU > 0.90;

We compute for all users whose PCU is in a given interval, the mean of the errors for the main task (T1) and for the secondary task (T2). As shown in Figure 6.1.3, the number of errors for both tasks decreases as the users' PCU increases.

Moreover, we then can compute for all users, whose PCU is in a given interval, the average time to find the right letter on the keyboard and push it for each level of cognitive load.

As shown in Table 6.1.3 the higher the PCU, the faster the participants find and push the correct letter.

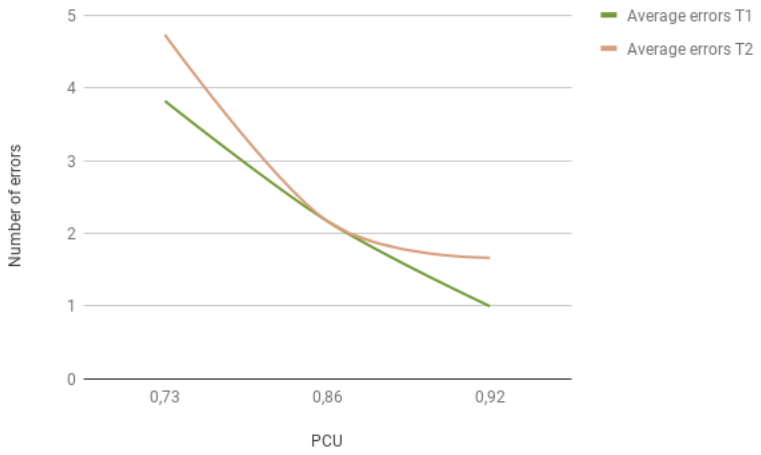

Figure 6.1.3: Average errors for the main task (T1) and the secondary task (T2).

	2L 2K 18S	2L 2K 22S	2L 2K 26S	2L 4K 18S	2L 4K 22S	2L 4K 26S
lowPCU	1,293	1,183	1,296	1,394	1,473	1,332
mediumPCU	1,094	1,356	1,141	1,258	1,281	1,248
highPCU	1,096	1,014	1,016	1,177	1,198	1,149

	4L 4K 18S	4L 4K 22S	4L 4K 26S	4L 8K 18S	4L 8K 22S	4L 8K 26S
lowPCU	1,531	1,56	1,494	1,827	1,882	1,82
mediumPCU	1,41	1,375	1,612	1,77	1,747	1,616
highPCU	1,283	1,199	1,118	1,56	1,474	1,379

	8L 8K 18S	8L 8K 22S	8L 8K 26S	8L 16K 18S	8L 16K 22S	8L 16K 26S
lowPCU	1,464	1,529	1,622	1,844	1,999	1,949
mediumPCU	1,328	1,397	1,538	1,696	1,556	1,744
highPCU	1,231	1,339	1,303	1,513	1,632	1,572

Table 6.1.3: Average duration for each PCU group and each level of cognitive load.

	2L 2K	*2L 4K*	*4L 4K*	*4L 8K*	*8L 8K*	*8L 16K*
lowPCU	1,257	1,4	1,528	1,843	1,538	1,931
mediumPCU	1,197	1,262	1,466	1,711	1,421	1,665
highPCU	1,042	1,175	1,2	1,471	1,291	1,572

Table 6.1.4: Average duration for each PCU group and each combination of numbers of letters and number of keys in the keyboard.

Observing the data we notice that there are no significant differences on the average duration for tasks with different total duration, namely it does not matter how long is the task, but how many letters the user has to find on a single or double keyboard. We then compute the average duration for each combination of number of letter to insert and number of keys on the keyboard (see Table 6.1.4).

From data collected by the web application, we can also deduce how long users stay on a task respect to the other, and according to such times, we can understand how much each participant perceives as critical the main task respect to the secondary task. However, by observing the data, we notice that about the 20% of the time a user is on the secondary task, the next letter has not yet appeared, namely such time represents the time the user goes backwards and forwards from the main task to the secondary task to check if the letter has appeared.

We call *criticality* the percentage of time a user stays on the main task respect to the secondary task. In order to check if the less a user perceives the main task as critical (i.e. the criticality is lower), the more he/she fails in such task, we divide the criticality values (which go from 48% to 66%) into 2 groups: the first groups values until 57%, the second groups values higher than 58%. We then compute the average number of errors for each of these groups, and we find that the more the main task is perceived as critical, the less the users fail in it: 3,36 errors on average for the low criticality group and 1,8 errors on average for high criticality group. In Figure 6.1.4 are shown the number of errors of each participant (denoted with a red cross) when varying the criticality. We then identify 2 more subgroups, according to how users perceive the criticality of the main task: we call them *lowCriticality*

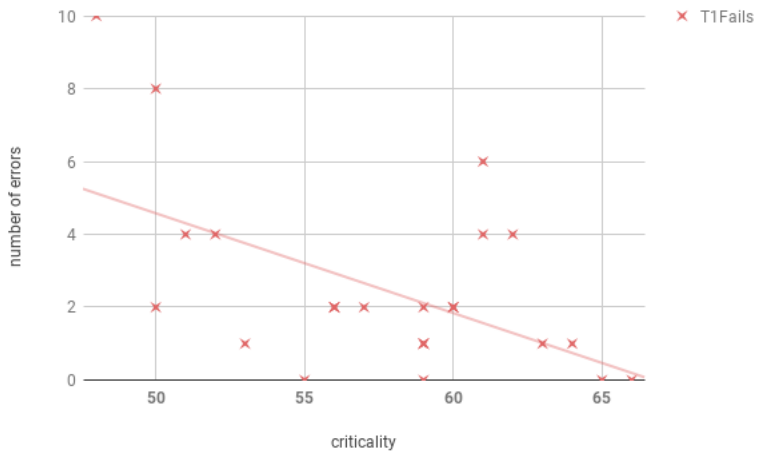

Figure 6.1.4: Number of errors of each participants when varying the calculated criticality (time spent on the main task).

and *highCriticality*.

Therefore, we consider 6 different typologies of users – by considering the 3 PCU groups and the 2 criticality subgroups – and we then identify 6 different groups:

1. *lowPCU – lowCriticality*
2. *lowPCU – highCriticality*
3. *mediumPCU – lowCriticality*
4. *mediumPCU – highCriticality*
5. *highPCU – lowCriticality*
6. *highPCU – highCriticality*

6.2 Simulation Experiments

For each group devised above, and for each level of cognitive load of the secondary task, we implement a different simulation experiment.

Main Task

As regards the main critical task (i.e. the one where users are asked to avoid that the black pellet reaches one of the two red rings), we implement it as a sequence of basic tasks, whose duration is set to 1 and difficulty is set to 0.1. We implement the same task for each PCU group, and two variants of the task for each criticality subgroup: for *highCriticality* we set the criticality of the task to 40, for *lowCriticality* we set it to 4.

Secondary Task

As explained in Section 6.1.2, in the secondary task, a letter appears inside the white boxes at a specific time: the total duration of the task is divided by the number of letters to insert, and such measure gives us the interval of time between the appearance of a letter and the next one. Therefore, the secondary task could be defined as follows (according to Definition 4.5):

$$\begin{aligned}&\langle noinfo \mid letter_1 \;\Rightarrow\; findL_1 \mid noInfo \textbf{ duration } t \textbf{ difficulty } d \textbf{ delay } \delta_1\\ &\quad\vdots\\ &\; noinfo \mid letter_n \;\Rightarrow\; findL_n \mid noInfo \textbf{ duration } t \textbf{ difficulty } d \textbf{ delay } \delta_n, c, g\rangle\end{aligned}$$

where:

- n is the number of letters to insert, and thus the number of basic tasks composing the secondary task;
- t_i is the duration of the action $findL_i$, set as the average duration for a given combination of number of letters and keys, according to the duration presented in Table 6.1.4;

- d_i is the difficulty of the action $findL_i$, which we set to 6;
- δ_i denotes the time which has to elapse so that the letter appears, namely the interval of time between the appearance of two letters minus the duration t_i.

Actually, the appearance of a letter in the secondary task is independent of the previous letter, which means that each letter in a sequence appears as soon as the given time interval has passed, whether the previous letter has been correctly inserted or not. Instead, the task presented above, implies that the delay δ_i of each basic task (namely of each letter) starts elapsing as soon as the basic task becomes the first one of the current subtask, which means that by modelling the secondary task in that way, the appearance of a letter would be related to the correct insertion of the previous letter.

We thus decide to model a different task for each letter to be inserted in the secondary task, namely to divide the *unique* task presented above into n different tasks:

$$\langle noinfo \mid letter_1 \Rightarrow findL_1 \mid info_2 \textbf{ duration } t \textbf{ difficulty } d \textbf{ delay } \delta_1, c_1, g_1 \rangle$$
$$\vdots$$
$$\langle info_n \mid letter_n \Rightarrow findL_n \mid noInfo \textbf{ duration } t \textbf{ difficulty } d \textbf{ delay } \delta_n, c_n, g_n \rangle$$

In this way, each delay of each task represents the time which has to elapse from the beginning of the simulation of the interaction with the secondary task in order that the letter appears.

Each task composing the secondary task shares a memory. In this way it is possible to ensure that all tasks are executed in the right order: each task has to put inside the memory the information to be retrieved by the next task to be executed so that a task cannot be carried out until the previous task has not been accomplished (i.e. letters have to be inserted in the correct order).

Moreover, each task composing the secondary task must have the same cognitive load, which has to be equal to the cognitive load of the unique task presented

above. Therefore, the difficulty of each task composing the secondary task is computed according to the following equation:

$$d_{NEW} = \frac{CL_{OLD} \cdot (t_{NEW} + \delta_{NEW})}{t_{NEW}} \tag{6.1}$$

where:

- CL_{OLD} is the cognitive load of the "old" unique task presented above (computed with Equation 4.1);
- t_{NEW} is the duration of the "new" (single) tasks, which equals to the duration of the unique task (set according to the average duration for each combination of the number of letters and keys);
- δ_{NEW} is the delay of the "new" (single) tasks, set according to the time which has to elapse from the beginning of the simulation in order that each letter appears.

Therefore, we implement the secondary task as a sequence of separate tasks, whose length is equal to the number of letters to be inserted. Each task is composed of a single basic task, whose duration is the average duration for a given combination of number of letters and keys, according to the duration presented in Table 6.1.4. The difficulty of the secondary tasks is computed with Equation 6.1; their criticality is set to 0.1.

For instance, a secondary task where the user (with high PCU) has to insert 4 letters with a single keyboard in 18 seconds would be defined through the following unique task:

$\langle noinfo \mid letter_1 \Rightarrow findL_1 \mid noInfo$ **duration** 1.2 **difficulty** 6 **delay** 0
$\langle noinfo \mid letter_2 \Rightarrow findL_2 \mid noInfo$ **duration** 1.2 **difficulty** 6 **delay** 3.3
$\langle noinfo \mid letter_3 \Rightarrow findL_3 \mid noInfo$ **duration** 1.2 **difficulty** 6 **delay** 3.3
$\langle noinfo \mid letter_4 \Rightarrow findL_4 \mid noInfo$ **duration** 1.2 **difficulty** 6 **delay** 3.3
$0.1, goal\rangle$

where:

- the duration 1.2 is set according to the table 6.1.4 for the tasks with 4 letters and 4 keys for high PCU users;
- the delay of the second, third and fourth basic tasks is computed as the interval of time between the appearance of two letters ($18/4 = 4.5$), minus the duration of each action (1.2).

The cognitive load of such a unique task is equal to 1.96 (see Equation 4.1).

The translation of the task depicted above would be done with the following 4 tasks:

$$\langle noinfo \mid letter_1 \Rightarrow findL_1 \mid task2 \quad \textbf{duration } 1.2 \textbf{ difficulty } 1.96 \textbf{ delay } 0, 0.1, goal_1 \rangle$$
$$\langle task2 \mid letter_2 \Rightarrow findL_2 \mid task3 \quad \textbf{duration } 1.2 \textbf{ difficulty } 9.31 \textbf{ delay } 4.5, 0.1, goal_2 \rangle$$
$$\langle task3 \mid letter_3 \Rightarrow findL_3 \mid task4 \quad \textbf{duration } 1.2 \textbf{ difficulty } 16.66 \textbf{ delay } 9, 0.1, goal_3 \rangle$$
$$\langle task4 \mid letter_4 \Rightarrow findL_4 \mid noInfo \quad \textbf{duration } 1.2 \textbf{ difficulty } 24.01 \textbf{ delay } 13.5, 0.1, goal_4 \rangle$$

where the difficulty of each task is computed with Equation 6.1

Simulation Settings

In total, we implement 108 different tests for each combination of PCU levels (3), levels of cognitive load (18), and level of criticality (2). For each of these combinations, we perform 1000 simulations and we compute the average value for the time to complete the simulated secondary task and the maximum time the simulated main task is ignored.

It is worth to note that the simulated tests are approximation of the real users performance. For instance, by modelling the secondary task as a single basic task where the user finds and push the right letter in the keyboard, we cannot simulate

the case where a user goes backward and forward from the main task to the secondary task, to check if the next letter appeared. However, we extract such time from the time passed on the secondary task, and we add it to the time passed on the main task.

6.3 Results

A *simulation* consists in the execution of both simulated tasks concurrently and we perform 1000 simulations for each of the 6 groups presented above. Namely during a simulation are executed 18 different tests (one for each level of CL of the secondary task), where the main task has a given criticality according to which of the 2 criticality subgroups we are simulating, and the secondary task has precise durations and difficulties according to which of the 3 PCU groups we are simulating. We perform each simulation in order to check if the "simulated users" behave as the real users. In particular, we observe:

1. If the time passed on the main task with respect to the time passed on the secondary task is equal to that observed in the data;

2. If the number of errors in the main task follows the same distribution of the one observed in the data;

3. If the number of errors in the secondary task follows the same distribution of the one observed in the data.

The time passed on the main task respect to the time passed on the secondary task is what we called criticality in Section 6.1.4.

As regard the main task, we know that it fails as soon as the black pellet reaches one of the red rings, and we know that the minimum number of steps for the pellet to reach a red ring (starting from the green one) is 4 steps; the longer such a task is ignored, the higher is the probability to fail. We thus consider the maximum wait time of the main task – namely the time it has been ignored – as a measure

of the probability to fail it: the higher is the maximum wait time, the higher is the probability.

On the other hand, for the secondary task we consider the time its last task has been executed and finished and we compare such time with the total duration of the task: the higher is such value, the higher is the probability that the secondary task has failed.

Regarding the number of errors observed from data, it is worth to note that it is particularly low and it can be subject to statistical noise. Therefore, we concentrate on the criticality and as regards the errors (for both the main task and the secondary task) we analyse if the simulated trend is similar to the real trend.

Criticality

As regards the criticality, we consider the final time of the secondary task as the final time of the entire test (namely the time to perform both tasks), we subtract to such time the duration of each task composing the secondary task (namely the time to perform the secondary task), and we finally compute the percentage of time the simulated test has passed on the main task. We compute the average of such measures for the entire simulation, for each of the six groups.

As shown in Figure 6.3.1 the time simulated users pass on the main task is very close to the time real users pass on the main task: errors go from -0.2% to $+3.3\%$

Tasks Fails

As regards the errors on the main task, we compute the average of the maximum wait time for the main task of the entire simulation, for each of the six groups, and we compare such measures with the average number of errors for each of the six group.

As shown in Figure 6.3.2 the probability to fail the main task decreases as the level of PCU increases, as well as the number of errors (see Figure 6.3.3) which

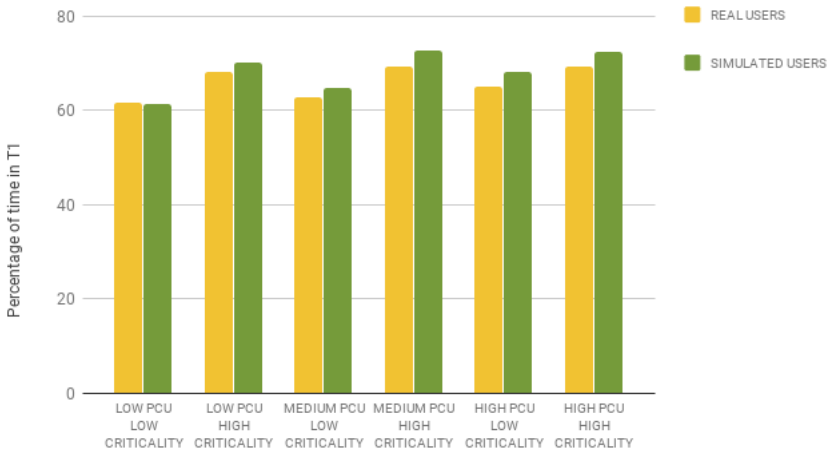

Figure 6.3.1: Time on task T1 for simulated and real users.

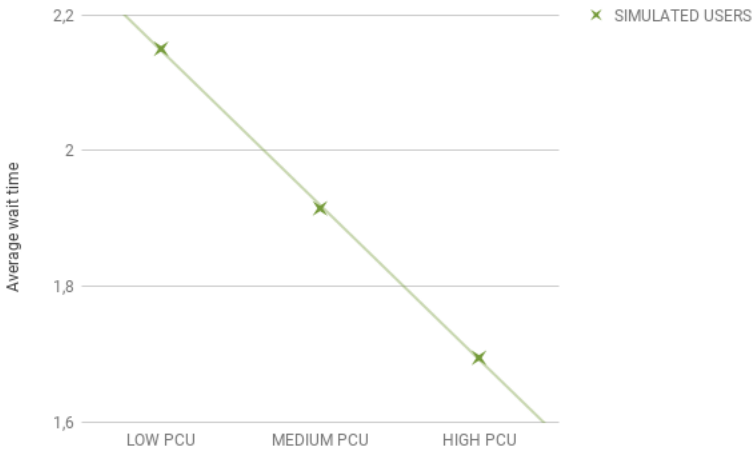

Figure 6.3.2: Average wait time when varying PCU.

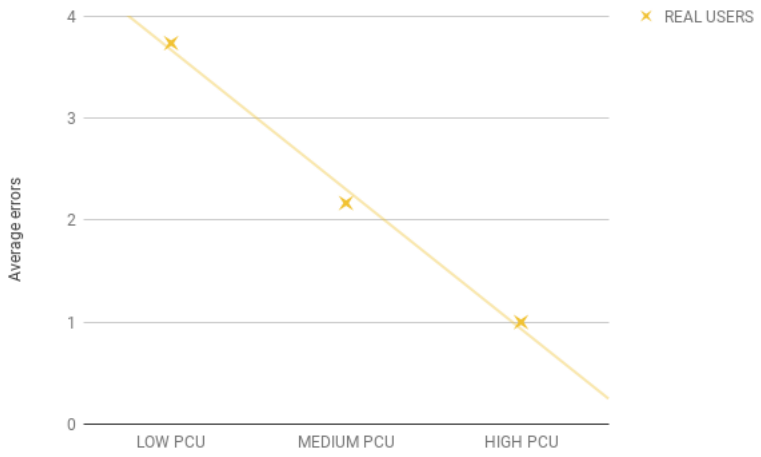

Figure 6.3.3: Average number of errors for T1.

decreases as the users' PCU increases.

The probability to fail the main task decreases as well as the criticality increases, and such trend is observed also in the data. Finally, as shown in Figure 6.3.4 and in Figure 6.3.5, the probability of fail and the average number of errors decrease for each of the six groups as the PCU decreases and the criticality passes from low to high.

Regarding the errors on the secondary task, we subtract the final duration of the secondary simulated task to the total duration of the task and we compute the average of such measures for the entire simulation, for each of the six groups.

Also in this case, we notice a decrease of the probability of errors in the secondary task when the PCU level increases, and a growth in the probability of errors when the criticality increases. We observe the same trend in the data.

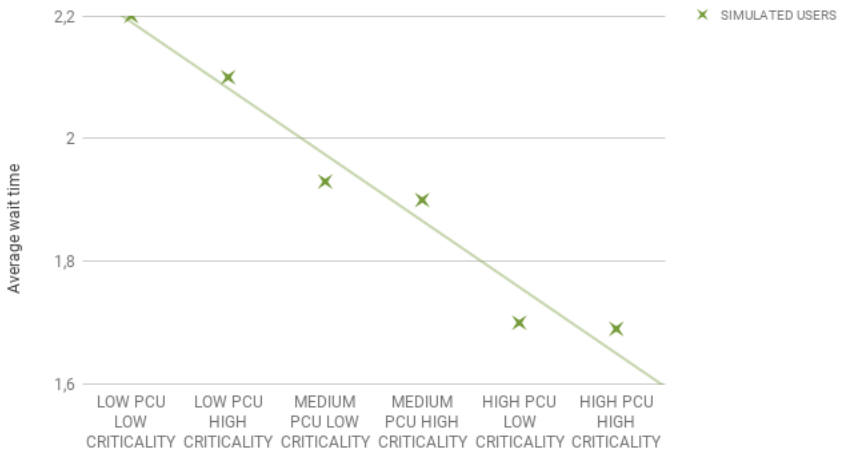

Figure 6.3.4: Average wait time for simulated tasks for each group.

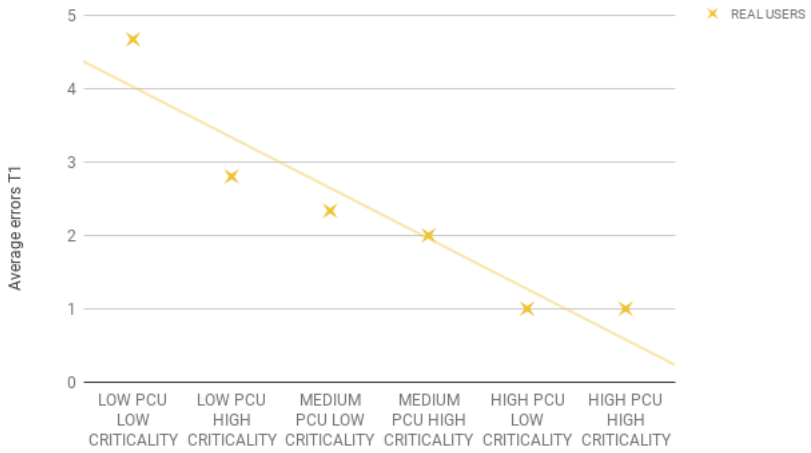

Figure 6.3.5: Average number of errors for T1 for each group.

7

Real-Time Maude Framework

This chapter presents our Real-Time Maude implementation of the formal model presented in Chapter 4 [20]. The full executable specification is available at http://www.di.unipi.it/msvbio/software/HumanMultitasking.html.

As previously mentioned in Chapter 2, Real-Time Maude is a rewriting-logic based language and simulation and model checking tool. The specification language is a high level language whose expressiveness enables us to easily define structured objects such as interfaces, tasks and working memory.

Moreover, the specification formalism is especially used to model and analyse distributed *real-time* systems, since the tool provides a range of timed formal analysis methods (such as timed rewriting for simulation, timed reachability analysis, time bounded linear temporal logic model checking, and timed computational tree logic model checking), which are particularly suitable for our purposes. Indeed, in

our analysis, we focus on different timing aspects and issues of safety-critical human multitasking, e.g. for how long a user could ignore a safety-critical task, or if it is possible that a user ignores a critical task in a crucial moment, or again how long does a user take to finish a critical task.

Furthermore, performing analysis with Real-Time Maude seems reasonable since in such safety-critical human multitasking context the states' space is tractable. Considering that the set of the states is proportional to the number of parallel tasks and their duration, and the number of tasks that a real user can perform concurrently is limited.

Unfortunately, Real-Time Maude is not a probabilistic language. We could have used the probabilistic specification language PMaude and a model checking tool such as PVeStA [2]. However, such tools would require to manage the time aspects and analyses by hand. We then leave this extension as part of the future work and we make a deterministic approximation of the probabilistic aspects of the model.

We model safety-critical human multitasking in an object-oriented style, where the configuration consists of a number of `Interface` objects, representing the interfaces of the devices with which the user interacts, a `Task` object inside each `Interface` object, defining the task that the user wants to perform on that interface, and a `WorkingMemory` object, representing the user's working memory. We present the Maude specification of such classes in Sect. 7.1.

We formalise the choice of tasks to be executed through two functions (`rank` and `bestRank`), which we present in Sect. 7.2.

Finally, we model the dynamic behaviour of the system as a set of rewrite rules, which are presented in Sect. 7.3.

Some aspects of the formal model presented in Chapter 4 have been modelled differently in Real-Time Maude:

- We model interfaces on which users can perform a single task at a time, therefore each `Interface` object can only have one `Task` object inside it.
- In Chapter 4 we model the selection of the task to be executed next with the

probability Φ, proportional to the factor a of each task (that is a measure of how much the task attracts the user attention). Since Real-Time Maude is not a probabilistic language, we model such a selection in a deterministic way: each interface is characterised by a *rank* (which is actually equal to the a factor), and the task/interface selection is performed by the function `bestRank` , which deterministically selects the interface with highest rank in the configuration.

- When the working memory overloads, in the formal model it is deleted an information from the memory associated with an interface chosen with a probability proportional to its wait time. Since Real-Time Maude is not a probabilistic language, we model such a choice in a deterministic way. Given that maintaining information in working memory requires the user attention, and the user attention is on the current task, one information related to other interfaces is nondeterministically forgotten first. If there is no information in the memories of the other interfaces, an information related to the current interface is forgotten.

- Since here we model a single `Task` object inside each `Interface` object, the memory associated with each interface is not shared among different tasks. For such a reason, every time a task goal is achieved, the memory associated with the interface on which such a task is executed is emptied, given that the interaction with that interface is completed.

- Each task in Real-Time Maude has a parameter `status` (not present in the formal model presented in Chapter 4), added for analysis purposes.

- We add the rule `timeout` to model the possible expiration of the timeout associated with an interface state. In such a case, the task is simply abandoned thus the task status becomes `abandoned`.

7.1 Classes

We first model a set of sorts that must be defined by the modeller when using the framework.

```
fmod PARAMETER-SORTS is
  sorts Action Cognition BasicInfo Perception InterfaceId .
endfm

--- ACTIONS
fmod ACTION is including PARAMETER-SORTS .
  sort DefAction .  --- actions plus "noAction"
  subsort Action < DefAction .
  op noAction : -> DefAction [ctor] .
endfm
```

Interface

We model the interfaces as transition systems, where the states are given by what users perceive them to be, and they might be subject to a timeout, capturing the fact that they can expire.

We represent interface's states as a terms of the sort `InterfaceState` presented in what follows.

```
sorts InterfaceState Perception ExpPerception .
subsort Perception < ExpPerception < InterfaceState .
op _for time_ : Perception TimeInf ->
                                InterfaceState [right id: INF] .
op expired : Perception -> ExpPerception .
```

Perception and expired perceptions are subsorts of interface states. The term p `for time` t denotes that the user will perceive p for time t, after which the perception becomes `expired(`p`)`.

As regards the interface transitions, we model them as a ';'-separated set of single interface transitions.

```
sorts InterfaceTransition InterfaceTransitions .
subsort InterfaceTransition < InterfaceTransitions .

```

```
--- single transition:
op _--_-->_ : Perception DefAction InterfaceState ->
                                          InterfaceTransition .

--- sets of transitions:
op noTransition : -> InterfaceTransitions .
op _;_ : InterfaceTransitions InterfaceTransitions ->
          InterfaceTransitions [assoc comm id: noTransition] .
```

An interface is represented as an object instance of following class.

```
class Interface | task : Object,
                  transitions : InterfaceTransitions,
                  previousAction : DefAction,
                  currentState : InterfaceState .
```

where the attribute `transitions` denotes the transitions of the interface; `task` denotes the task object representing the task that the user will perform on the interface; `previousAction` is the previous action performed on the interface (useful for analysis purposes); and `currentState` is the current state of the interface.

Task

Unlike the formal model presented in Chapter 4, in the Real-Time Maude framework, each interface can only have one task object inside it. We model a task as a '::'-separated sequence of subtasks, where each subtask is modeled as a sequence of basic tasks.

```
sorts BasicTask Subtask Task .
subsort BasicTask < Subtask < Task .

--- BasicTasks:
op _|_==>_|_duration_difficulty_delay_ :
                  Information Perception DefAction Information
                              Time PosRat Time -> BasicTask .

--- Subtask is a list of BasicTasks:
op nil : -> Subtask .
op __ : Subtask Subtask -> Subtask [assoc id: nil] .

--- Task is a list of subTasks:
op emptyTask : -> Task .
```

```
15 op _::_ : Task Task -> Task [assoc id: emptyTask] .
```

A task is represented as an object instance of the following class.

```
1 class Task | subtasks : Task,
2              waitTime : Time,
3              status : TaskStatus,
4              cognitiveLoad : Rat,
5              criticalityLevel : PosRat .
```

where the `subtasks` attribute denotes the remaining sequence of subtasks to be performed; the attribute `waitTime` denotes how long the task has not been executed; the attribute `cognitiveLoad` is the cognitive load of the current subtask; `criticalityLevel` is the task's criticality; and, finally, the attribute `status` denotes the "status" of the task, which can be `notStarted`, `ongoing`, `abandoned`, or `completed`.

It is worth to note that here we memorise for each task in the configuration, a value for the time it has not been executed; while, in the formal model we memorise the last time a task has been executed (namely its timestamp) in the map TS and we compute the same information dynamically as $GC - TS(I, i)$ (i.e. the value of the global clock minus the timestamp of the i-th task of the interface I).

Working Memory

As in the formal model (Sect. 4.1.2), we model the working memory as a map assigning to each interface a set of information useful to interact with it. We model it as an object instance of the class presented in what follows.

```
1 class WorkingMemory | memory : Memory,
2                       capacity : NzNat .
```

Elements in memory could be an basic information, a cognition or a goal: each of them are subsort of the sort `Information`.

```
1 sorts BasicInfo Cognition Goal Information .
2 subsorts Cognition Goal BasicInfo < Information .
3
4 op goal : Action -> Goal .
```

As regards the data type `Memory`, we define it as follows.

```
sort InfoSet .
subsort Information < InfoSet .

op noInfo : -> Information .
op __ : InfoSet InfoSet -> InfoSet [assoc comm id: noInfo] .

sort Memory .
op noMemory : -> Memory .
op _|->_ : InterfaceId InfoSet -> Memory .
op _;_ : Memory Memory -> Memory [assoc comm id: noMemory] .
```

7.2 Ranking Function

As mentioned in Chapter 4 (Sect. 4.1.1), each task is characterised by an attention attraction factor a, which measures the likelihood that the task will attract the user's attention. We then compute a probability Φ for each task, denoting the probability that such task will be executed (see Equation 4.9).

Since Real-Time Maude is not a probabilistic language, we simplify the choice of the next task to be executed and we model it in a deterministic way, by using two functions : `rank` and `bestRank`.

Essentially, we assign to each interface a *rank* factor, computed exactly as the a-factor (see Equation 4.4). The task associated with the interface with the highest rank in the configuration is the one that will be executed.

The function `rank` is defined in Listing 7.1.

```
var I : InterfaceId .                  var TASK : Oid .
vars INF1 INF2 : Information .         var P1 : Perception .
var DACT : DefAction .                 var NZT : NzTime .
vars PR PR2 : PosRat .                 vars T T2 : Time .
var BTL : Subtask .                    var OTHER-SUB-TASKS : Task .
var CL : Rat .                         var ACT : Action .
var INFO-SET : InfoSet .               var MEMORY : Memory .

op rank : NEConfiguration Memory -> PosRat .
eq rank(< I : Interface | task :
             < TASK : Task | subtasks : ((INF1 | P1 ==> DACT | INF2
                                duration NZT difficulty PR delay T2)
```

```
12                                                BTL) :: OTHER-SUB-TASKS,
13                                waitTime : T,
14                                cognitiveLoad : CL,
15                                criticalityLevel : PR2 > >,
16           (I |-> goal(ACT) INF-SET) ; MEMORY) =
17   if T2 == 0
18      then PR2 * CL * (T + 1)
19   else 0 fi .
20 eq rank(< I : Interface | >, MEMORY) = 0 [owise] .
```

Listing 7.1: Rank function.

It specifies that when the task is not pausing (line 17 in Listing 7.1), and it has a goal in memory associated with it (line 16), the rank is computed as the product of the criticality level PR2 of the task, the cognitive load CL of the current subtask, and the waiting time T of the task (line 18). Otherwise, the rank is equal to 0, and the task is not executed (lines 19-20).

A function bestRank is used to calculate the task with the highest rank. The function uses the max function defined in Maude (line 7 in Listing 7.2).

```
1 var OBJECT : Object .
2 var MEMORY : Memory .
3 var NEC2 : NEConfiguration .
4
5 op bestRank : NEConfiguration Memory -> PosRat .
6 eq bestRank(OBJECT, MEMORY) = rank(OBJECT, MEMORY) .
7 eq bestRank(OBJECT NEC2, MEMORY) =
8             max(rank(OBJECT, MEMORY), bestRank(NEC2, MEMORY)) .
```

Listing 7.2: bestRank function.

7.3 Rewrite Rules

This section presents a set of rewriting rules, representing the semantics of the formal model (Sect. 4.2).

Interacting

As already mentioned, the `interacting` rewrite rule models a user action performed on the interface. We present the Real-Time Maude specification of such a rule in Listing 7.3.

```
crl [interacting] :
  {OTHER-INTERFACES
  < I : Interface | task :
          < TASK : Task | subtasks : ((INF1 | P1 ==> DACT | INF2
                                duration NZT difficulty PR delay 0)
                            OTHER-BASIC-TASKS) :: OTHER-SUB-TASKS,
                            waitTime : T1,
                            cognitiveLoad : CL,
                            criticalityLevel : PR2,
                            status : TS >,
                  transitions : (P1 -- DACT --> (P2 for time TI2)) ;
                                                                TRANSES,
                  currentState : (P1 for time TI),
                  previousAction : DACT2 >
  < WM : WorkingMemory | memory : MEMORY ; (I |-> INF1  goal(ACT)
    INFO-SET),
                         capacity : CAP >}
=>
  {idle(OTHER-INTERFACES, NZT)
  < I : Interface | task :
    < TASK : Task | subtasks :
              (if OTHER-BASIC-TASKS =/= nil
                then (OTHER-BASIC-TASKS :: OTHER-SUB-TASKS)
               else OTHER-SUB-TASKS fi),
         waitTime : 0,
         status : (if TS == notStarted
                   then ongoing
                  else TS fi),
         cognitiveLoad : (if OTHER-BASIC-TASKS =/= nil
                             then CL
                           else cogLoad(first(OTHER-SUB-TASKS)) fi) >,
        currentState : (P2 for time TI2),
        previousAction : DACT >
  < WM : WorkingMemory | memory : MEMORY ; (I |-> INF2 goal(ACT) INFO
    -SET) >}
  in time NZT
if (DACT =/= ACT) /\ (DACT =/= noAction)
/\ card(MEMORY ; (I |-> INF2 goal(ACT) INFO-SET)) <= CAP
/\ rank(< I : Interface | >, (MEMORY ; (I |-> INF1  goal(ACT) INFO-
    SET))) == bestRank((< I : Interface | >) OTHER-INTERFACES, (
```

```
        MEMORY ; (I |-> INF1  goal(ACT) INFO-SET))) .
```

Listing 7.3: Interacting rule.

As the specification shows the rule models the execution of a basic task of the interface with the highest rank (line 35 in Listing 7.3), if the action to be performed is not the goal action and it is different from noAction (line 33), if the application of the rule does not cause a memory overload (line 34), and if the first basic task and one of the interface transitions synchronise (line 4, 10). When the user perceives that the interface is on state `P1` (line 4) and he has the information `INF1` in memory (line 13), then he can perform the action `DACT` (line 4). After that:

- the basic task executed is removed from the configuration (lines 18, 21);
- the `waitTime` of the task is set to 0 since it has been just executed (line 22);
- if the task had not started yet, its `status` changes from "notStarted" to "ongoing" (lines 23, 25);
- the `cognitiveLoad` is properly recomputed if the basic task just executed was the last one of a subtask (lines 26, 28);
- the `currentState` of the interface and its `previousAction` are updated (lines 29 - 30);
- the information `INF1` is replaced by the information `INF2` in the memory associated with the selected interface (line 31).

The duration of the rule is specified by the duration of the basic task executed, `NZT` (line 32). During that time, all other interfaces in configuration *idle* (line 16). The Real-Time Maude specification of the function `idle` is presented in Listing 7.4.

```
1 op idle : Configuration Time -> Configuration .
2 eq idle(none, T) = none .
3 eq idle(< I : Interface | task :
4             < TASK : Task | subtasks : ((INF1 | P1 ==> DACT | INF2
                                duration NZT difficulty PR delay T2)
```

```
5                              OTHER-BASIC-TASKS) :: OTHER-SUB-TASKS,
6                           waitTime : T3 >,
7                           currentState : IS > REST, T)
8    = < I : Interface | task :
9              < TASK : Task | subtasks : ((INF1 | P1 ==> DACT | INF2
                       duration NZT difficulty PR delay (T2 monus T))
10                              OTHER-BASIC-TASKS) :: OTHER-SUB-TASKS,
11                          waitTime : T3 + (T monus T2) >,
12                          currentState : idle(IS, T) > idle(REST, T) .
13
14 --- finished tasks: waitTime stays 0 in this case
15 eq idle(< I : Interface | task : < TASK : Task | subtasks : emptyTask
       , waitTime : T3 >, currentState : IS > REST, T)
16    = < I : Interface | task : < TASK : Task | waitTime : 0 >,
       currentState : idle(IS, T) > idle(REST, T) .
17
18 --- InterfaceState expiration time
19 op idle : InterfaceState TimeInf -> InterfaceState .
20 eq idle(P1 for time TI, T) = if T < TI then P1 for time (TI monus T)
       else expired(P1) fi .
21 eq idle(expired(P1), T) = expired(P1) .
```

Listing 7.4: Idle function.

1. The `delay` of the first basic task of all other interfaces decreases according to the time elapsed (line 9 in Listing 7.4);

2. The `waitTime` of all other interfaces increases with the time elapsed (line 11) unless the task is not finished, in such case, it is set to 0 (lines 15 - 16);

3. The remaining time until the states expiration in all other interfaces decreases according to the time elapsed (lines 20 - 21).

Cognitive

The `cognitive` rewrite rule models a cognitive basic task.

```
1 crl [cognitive] :
2   {OTHER-INTERFACES
3    < I : Interface | task : < TASK : Task | subtasks : ((INF1 | P1
       ==> DACT | COG2 duration NZT difficulty PR delay 0) OTHER-BASIC-
       TASKS) :: OTHER-SUB-TASKS,
```

```
4                                   waitTime : T1,
5                                   cognitiveLoad : CL,
6                                   criticalityLevel : PR2,
7                                   status : TS > >
8     < WM : WorkingMemory | memory : MEMORY ; (I |-> INFO-SET INF1 goal
       (ACT)),
                             capacity : CAP >}
9  =>
10   {idle(OTHER-INTERFACES, NZT)
11    < I : Interface | task : < TASK : Task | subtasks :
12              (if OTHER-BASIC-TASKS =/= nil
                    then OTHER-BASIC-TASKS :: OTHER-SUB-TASKS
                 else OTHER-SUB-TASKS fi),
13              waitTime : 0,
14          status : (if TS == notStarted
15                      then ongoing
16                    else TS fi),
17          cognitiveLoad : (if OTHER-BASIC-TASKS =/= nil
18                              then CL
19                            else cogLoad(first(OTHER-SUB-TASKS)) fi) >>
20    < WM : WorkingMemory | memory : MEMORY ; (I |-> INFO-SET COG2 goal
       (ACT)) >}
21   in time NZT
22 if (DACT =/= ACT) /\ DACT == noAction
23 /\ card(MEMORY ; (I |-> INFO-SET COG2 goal(ACT))) <= CAP
24 /\ rank(< I : Interface | >, (MEMORY ; (I |-> INF1  goal(ACT) INFO-
       SET))) == bestRank((< I : Interface | >) OTHER-INTERFACES, (
       MEMORY ; (I |-> INF1  goal(ACT) INFO-SET))) .
```

Listing 7.5: Cognitive rule.

Here again, the rule models the execution of a basic task with the interface with the highest rank (line 24 in Listing 7.5), if the action to be performed is not the goal action and it is equal to noAction (line 22), and if the application of the rule does not cause a memory overload (line 23). However, in this case the rule models a human cognition with no involvement of the interface, where the user replaces the information `INF1` with the cognition `COG2` (line 20). Also in this case, the basic task just executed is removed from the configuration (line 12), the `waitTime` is set to 0, and the `status` and the `cognitiveLoad` of the task change as in the `interacting` rule (lines 13 - 19). The duration of the rule is specified by the duration of the basic task, `NZT` (line 21), during which all other interfaces idle (line

10).

Closure

The `closure` rule models the achievement of the goal by the user.

```
crl [closure] :
  {OTHER-INTERFACES
  < I : Interface | task : < TASK : Task | subtasks : ((INF1 | P1 ==>
     ACT | INF2 duration NZT difficulty PR delay 0) OTHER-BASIC-TASKS
    ) :: OTHER-SUB-TASKS,
                                    waitTime : T1,
                                    cognitiveLoad : CL,
                                    criticalityLevel : PR2,
                                    status : TS >,
                          transitions : (P1 -- DACT --> (P2 for time
    TI2)) ; TRANSES,
                            currentState : (P1 for time TI),
                            previousAction : DACT2 >
   < WM : WorkingMemory | memory : MEMORY ; (I |-> INFO-SET INF1 goal
    (ACT)) >}
=>
   {idle(OTHER-INTERFACES, NZT)
   < I : Interface | task : < TASK : Task | subtasks : (if OTHER-
    BASIC-TASKS =/= nil then (OTHER-BASIC-TASKS :: OTHER-SUB-TASKS)
    else OTHER-SUB-TASKS fi),
                                      waitTime : 0,
                                      cognitiveLoad : CL,
                                      status : completed >,
                            currentState : (P2 for time TI2),
                            previousAction : ACT >
   < WM : WorkingMemory | memory :  MEMORY ; (I |-> INFO-SET INF2) >}
  in time NZT
if rank(< I : Interface | >, (MEMORY ; (I |-> INFO-SET INF1 goal(ACT)
    ))) == bestRank((< I : Interface | >) OTHER-INTERFACES, (MEMORY ;
     (I |-> INFO-SET INF1 goal(ACT)))) .
```

Listing 7.6: Closure rule.

The rule is quite similar to the previous ones, but in this case, the action to perform *must* be the goal action and we do not care about the memory overload since after performing the action no information useful for the interaction will be added in memory (since the interaction is completed). After the execution of the goal ac-

tion the goal is removed from memory (line 20 in Listing 7.6) and the `status` is updated to `completed`(line 17).

Forgetting

When the execution of the basic task exceeds the capacity of the working memory, an item must be forgotten, to make room to the new information.

In the *forgetting rule* presented in Section 4.2.2, it is deleted an information from the memory associated with an interface chosen with a probability proportional to its wait time, in Real-Time Maude we simplify this mechanism given that it is not a probabilistic language. Since maintaining information in WM requires the user's attention and the user attention is on the current task, one information related to the other interfaces is *nondeterministically* forgotten first. However, if there are no information related to the other interfaces, an information related to the current interface is forgotten.

We thus model two kinds of rule for the case where the WM overloads while executing a user action, and two kinds of rule for the case where the WM overloads while executing a cognitive basic task. Essentially we have 4 different situations to model:

1. The memory overloads for the addition of a basic information
   (a) one information related to other interfaces is forgotten
   (b) one information related to the current interface is forgotten
2. The memory overload for the addition of a cognition
   (a) one information related to other interfaces is forgotten
   (b) one information related to the current interface is forgotten

The rules which model the situation depicted in item 1 above and the situation depicted in item 2 above are quite similar (except for the type of information that leads to the working memory overload and for the type of basic task that will be

performed). We will show just two of these rules: the ones modelling the situations depicted in item 1 (a) and item 1 (b).

The following Listing shows how we model the case when an information related to other interfaces is forgotten [1].

```
1  crl [interactingForgetSomethingOtherInterface] :
2    { ... < I : Interface | task : < TASK : Task | ... > ... >
3    < WM : WorkingMemory | memory : MEMORY ;
4                           (I |-> INF1  goal(ACT) INFO-SET) ;
5                           (I2 |-> INF3 INFO-SET2),
6                           capacity : CAP >}
7  =>
8    { ... < I : Interface | task : < TASK : Task | ... > ... >
9    < WM : WorkingMemory | memory : MEMORY ;
10                          (I |-> INF2 goal(ACT) INFO-SET) ;
11                          (I2 |-> INFO-SET2) >}
12   in time NZT
13 if (DACT =/= ACT)
14 /\ card((I2 |-> INF3 INFO-SET2)) =/= 0
15 /\ card(MEMORY ; (I |-> INF2 goal(ACT) INFO-SET) ; (I2 |-> INF3 INFO-
       SET2) ) > CAP
16 /\ card(MEMORY ; (I |-> INF2 goal(ACT) INFO-SET) ; (I2 |-> INFO-SET2)
        ) <= CAP
17 /\ rank(< I : Interface | >, (MEMORY ; (I |-> INF1  goal(ACT) INFO-
       SET) ; (I2 |-> INF3 INFO-SET2))) == bestRank((< I : Interface |
       >) OTHER-INTERFACES, (MEMORY ; (I |-> INF1  goal(ACT) INFO-SET) ;
        (I2 |-> INF3 INFO-SET2))) .
```

Listing 7.7: Rule for forgetting information related to other interfaces.

Since the `Memory` is associative and commutative, any memory item `INF3` related to any interface `I2` different from `I` could be forgotten (lines 5, 11 in Listing 7.7). As the specification shows, the rule is applied if the cardinality of the memory related to any other interface `I2` in the configuration is different from 0, i.e. there are information related to other interfaces (line 14); if the execution of the basic task would exceed the memory capacity (line 15); and if the removal of an information would solve the memory overload (line 16).

On the other hand, the following Listing shows how we model the case where there are no information related to other interfaces in the configuration (line 12

[1] some parts of the rule are replaced by '. . .'

of Listing 7.8), in this case an information is forgotten from the current interface (line 9).

```
1  crl [interactingForgetSomethingCurrentInterface] :
2    { ... < I : Interface | task : < TASK : Task | ... > ... >
3    < WM : WorkingMemory | memory : MEMORY ;
4                           (I |-> INF1  goal(ACT) INF3 INFO-SET),
5                           capacity : CAP >}
6  =>
7    { ... < I : Interface | task : < TASK : Task | ... > ... >
8    < WM : WorkingMemory | memory : MEMORY ;
9                           (I |-> INF2 goal(ACT) INFO-SET) >}
10   in time NZT
11 if (DACT =/= ACT)
12 /\ card(MEMORY) == 0
13 /\ card((I |-> INF2 goal(ACT) INF3 INFO-SET)) > CAP
14 /\ rank(< I : Interface | >, (MEMORY ; (I |-> INF1  goal(ACT) INF3
       INFO-SET))) == bestRank((< I : Interface | >) OTHER-INTERFACES, (
       MEMORY ; (I |-> INF1  goal(ACT) INF3 INFO-SET))) .
```

Listing 7.8: Rule for forgetting information related to the current interface.

All idling

If all tasks in configuration are pausing (i.e. the first basic tasks of each task have a `delay` higher than 0), the time advances until almost one of the `delays` reaches 0. The following Listing shows how we model such situation.

```
1  crl [tickAllIdling] :
2    {ALL-INTERFACES
3    < WM : WorkingMemory | memory : MEMORY ; (I |-> goal(ACT) INFO-SET)
       >}
4  =>
5    {idle(ALL-INTERFACES, MIN-DELAY)
6    < WM : WorkingMemory | >}
7    in time MIN-DELAY
8  if MIN-DELAY := minDelay(ALL-INTERFACES) .
```

Listing 7.9: Rule for idling all interfaces.

The function `minDelay` (line 8 of Listing 7.9, line 6 of Listing 7.10) returns the lowest delay in the configuration, using the `min` function defined in Maude (line 8 in Listing 7.10).

```
1 --- OPERATOR delay
2 op delay : Object -> TimeInf .
3 eq delay(< I : Interface | task : < TASK : Task | subtasks : (((INF1
      | P1 ==> DACT | INF2 duration NZT difficulty PR delay T4) OTHER-
      BASIC-TASKS)) :: OTHER-SUB-TASKS > >) = T4 .
4 --- OPERATOR minDelay
5 op minDelay : NEConfiguration -> TimeInf .
6 eq minDelay(OBJECT) = delay(OBJECT) .
7 eq minDelay(NEC1 NEC2) = min(minDelay(NEC1), minDelay(NEC2)) .
```

Listing 7.10: Function minDelay.

Timeout

Finally, our last rule models the expiration of the timeout which sometimes is associated to the `interfaceState`.

```
1 rl [timeout] :
2  {REST
3  < I : Interface | task : < TASK : Task | >,
4                       currentState : expired(P1) >}
5 =>
6  {REST
7  < I : Interface | task : < TASK : Task | status : abandoned > >} .
```

Listing 7.11: Timeout rule.

The rule concerns only the interface with no involvement in the user's WM: when the `currentState` of the current interface `I` is expired, the `status` of the task becomes `abandoned`(line 8 in Listing 7.11).

8

Case Studies

This chapter illustrates the application of our Real-Time Maude modelling and analysis framework to three case studies to formally verify different safety-critical multitasking situations: a person interacting with a GPS navigation device while driving (Section 8.2), a medical operator setting multiple infusion pumps at the same time (Section 8.3), and an operator of an air traffic control system (Section 8.4). These three case studies show different kinds of errors which might arise in a safety-critical multitasking interaction, and they show that our approach can be applied in different contexts. However, it is worth to note that we do not use real data to define the parameters values, but values that are plausible in such a context used to demonstrate that some situation can be reproduced.

Before presenting such case studies, we explain in Section 8.1 how to use Real-Time Maude to analyse safety-critical human multitasking.

8.1 Analysing Human Multitasking with Real-Time Maude

This section shows how Real-Time Maude can be used to analyse whether a user or an operator is able to interact successfully with a set of devices for performing multiple tasks. The properties analysed are of course highly dependent on the initial assumptions about the system that is being examined. In the thesis, we focus on the following potential problems that could arise in multitasking situations:

1. A critical task might be ignored for too long since the user's attention is addressed to other tasks;

2. One of the tasks, or a critical action, might be ignored in a crucial moment since the user's attention is addressed to other tasks/actions;

3. The concurrent use of the user's WM might cause a memory overload and consequently might lead the user to forget some useful information for completing successfully the interaction.

It is worth to remark that although at every step the task to be executed is deterministically chosen according with its rank, a model might still exhibit nondeterminism since:

- If two or more transitions of an `Interface` object are defined for the current state and action (possibly leading to different next states) one of them is chosen nondeterministically (see *interacting rule* in Sect. 7.3);

- At a certain step, more than one interface might have the same best rank, in which case the task to be given attention is selected nondeterministically among those best-ranked tasks (see *interacting rule* in Sect. 7.3);

- When memory overloads, the memory item that is forgotten is selected nondeterministically (see *forgetting rule* in Sect. 7.3);

Therefore, we need to analyse whether the desired properties hold for a set of interfaces/tasks, by checking all possible behaviours that might nondeterministi-

cally take place from the initial state. To do so, we use Real-Time Maude reachability analysis and analyse whether it is possible to reach a (possibly final) state in which a desired property is violated.

8.1.1 Initial State

The initial state should have the following form:

```
{initializeCognLoad(
 < wm : WorkingMemory | memory :
                              interface_1 |-> goal(act_1) otherItems_1 ;
                              ... ;
                              interface_n |-> goal(act_n) otherItems_n ,
                          capacity : capacity >
 < interface_1 : Interface | task :
   < task_1 : Task | subtasks : (b_1_1_1 ... b_1_1_1) :: ... :: (b_1_m_1 ... b_1_m_j),
                     waitTime : 0, cognitiveLoad : 0,
                     criticalityLevel : cl_1, status : notStarted >
                           transitions : trans_1 ,
                           previousAction : noAction,
                           currentState : perc_1 >
    ...
 < interface_n : Interface | task :
   < task_n : Task | subtasks : ..., waitTime : 0,
                      cognitiveLoad : 0, criticalityLevel : cl_n ,
                      status : notStarted >
                           transitions : trans_n ,
                           previousAction : noAction,
                           currentState : perc_n >)}
```

Listing 8.1: Initial state.

where $interface_k$ is the name of the k-th interface in the configuration; $task_k$ is the task to be performed with/on $interface_k$; $b_{k_{ij}}$ is the j-th basic task of the i-th subtask of $task_k$; cl_k is the criticality level of $task_k$; $trans_k$ are the transitions of $interface_k$; $action_k$ is the goal action to be achieved with $interface_k$; $otherItems_k$ are other items initially present in the memory for $interface_k$; $perc_k$ is the initial state of $interface_k$; and *capacity* is the number of items that can be stored in working memory. The `cognitiveLoad` attributes of all interfaces in the configuration are initialised by the function `initializeCognLoad` , which computes the cognitive load of the first subtask of each task.

8.1.2 Model Checking the Properties

The first key property to analyse is:

Is it possible that an enabled task t is ignored continuously for at least time Δ?

This property can be analysed in Real-Time Maude by checking whether it is possible to reach a "bad" state where the `waitTime` attribute of task *t* is at least Δ:

```
(utsearch [1] initialState =>*
  {REST:Configuration
   < I:InterfaceId : Interface | task :
      < t : Task | waitTime : T:Time, A:AttributeSet > >}
  such that T:Time >= Δ .
```

where the variable `REST:Configuration` matches the other objects in the state and the variable `A:AttributeSet` captures the other attributes in *inner* objects.

The second key property is:

What is the longest time needed to perform a certain action a?

This can be analysed using Real-Time Maude's `find latest` command, by finding the longest time needed to reach a state where `previousAction` is *a*:

```
(find latest initialState =>*
  {REST:Configuration
   < I:InterfaceId : Interface | previousAction : a >}
  with no time limit .)
```

We can also use the `find latest` command to find out if a task *t* is guaranteed to finish before time Δ:

```
(find latest initialState =>*
  {REST:Configuration
   < I:InterfaceId : Interface | task :
      < t : Task | status : completed, A:AttributeSet > >}
  with no time limit .)
```

Another key property to analyse is:

Is it possible that the memory overload leads a task t to not be completed?

We can analyse it by searching for a "bad" *final* state where the status of the task *t* is not `completed`:

```
(utsearch [1] initialState =>!
  {REST:Configuration
   < I:InterfaceId : Interface | task :
    < t : Task | status : TS:TaskStatus, A:AttributeSet > >}
  such that TS:TaskStatus =/= completed .)
```

If we want to analyse whether it is guaranteed that *all* tasks can be completed, we just replace *t* in this command with a variable `T:Oid`.

If a safety-critical task cannot be completed, or completed in time, we can check whether this is due to the task itself, or the presence of concurrent "distractor" tasks, by analysing an initial state *without* the distractor tasks.

8.2 Using GPS while Driving

Our first case study models a user who interacts with a GPS navigation system while driving [16, 20]. The case study suggests that the interaction with the GPS system could be too much demanding from a cognitive point of view, therefore the GPS interface could be modified in order to have a less demanding task or a mechanism which, under certain conditions (busy urban context, high speed of the car), does not permit to the user to interact with the interface.

After presenting the Real-Time Maude specification of the case study, we present its analysis in Subsection 8.2.1.

We model two interfaces: the car and the GPS. The task of driving is formalised by the `Task` object presented in Listing 8.2.

```
1  < driving : Task | subtasks :
2   ((noInfo | carOff ==> insertKey | keyInserted duration 1 difficulty
      3/10 delay 0)
3    (noInfo | carOn ==>  turnKey | noInfo duration 1 difficulty 2/10
      delay 0)
4    (noInfo | carReady ==> startDrive | noInfo duration 1 difficulty
      2/10 delay 2)) ::
5   ((noInfo | straightRoad ==> straight | noInfo duration 1 difficulty
      1/10 delay 3)
6    (noInfo | straightRoad2 ==> straight | noInfo duration 1 difficulty
       1/10 delay 3)
7    (noInfo | curveLeft ==> turnLeft | noInfo duration 1 difficulty
      4/10 delay 3)
8    (noInfo | curveRight ==> turnRight | noInfo duration 1 difficulty
      2/10 delay 3)
9    (noInfo | straightRoad3 ==> straight | noInfo duration 1 difficulty
       1/10 delay 3)
10   (noInfo | straightRoad4 ==> straight | noInfo duration 1 difficulty
       1/10 delay 3)) ::
11  ((noInfo | destination ==> stopCar | noInfo duration 2 difficulty
      2/10 delay 2)
12   (keyInserted | carStopped ==> pickKey | noInfo duration 2
      difficulty 1/10 delay 0)),
13                       waitTime : 0, status : notStarted,
14                       criticalityLevel : 6/10, cognitiveLoad : 0 >
```

Listing 8.2: Driving task.

The driving task consists of the three subtasks:

1. Start driving (lines 2 - 4 in Listing 8.2) consisting of the basic tasks of inserting the car key, turning on the ignition, and start driving;

2. Drive to destination (lines 5 - 10) described as a short trip during which the driver wants to perform a basic driving action at most every three time units;

3. Park and leave the car (line 11 - 12) consisting in stopping the car and removing the key when the driver is arrived at the destination.

The interface of the car is formalised by the `Interface` object presented in Listing 8.3.

```
< car : Interface | transitions :
    (carOff -- insertKey --> carOn) ;
    (carReady -- startDrive --> straightRoad) ;
    (carOn -- turnKey --> carReady) ;
    (straightRoad -- straight --> straightRoad2) ;
    (straightRoad2 -- straight --> curveLeft) ;
    (curveLeft -- turnLeft --> curveRight)\,;
    (curveRight -- turnRight --> straightRoad3) ;
    (straightRoad3 -- straight --> straightRoad4) ;
    (straightRoad4 -- straight --> destination) ;
    (destination -- stopCar --> carStopped) ;
    (carStopped -- pickKey --> carOff),
                          task : ... ,
                          previousAction : noAction,
                          currentState : carOff >
```

Listing 8.3: Car interface.

For the GPS navigator, we assume that to enter the destination the user has to type at least partially the address. The navigator then suggests a list of possible destinations, among which the user has to select the right one. Therefore, the GPS task consists of three subtasks: (i) start and choose city; (ii) type the initial k letters of the desired destination; and (iii) choose the right destination among the options given by the GPS.

If the user types the entire address of the destination, the navigator returns a short list of possible matches; if he/she types fewer characters, the navigator returns a longer list, making it harder for the user to find the right destination. We consider two alternatives: (1) the driver types 13 characters and then searches for the destination in a short list; and (2) the driver types just four characters and then searches for the destination in a longer list. The GPS task for case (1) is modelled by the `Task` object presented in Listing 8.4.

```
< findDestination : Task | subtasks :
 ((noInfo | gpsReady ==> typeSearchMode | noInfo duration 1
    difficulty 1/10 delay 0)) ::
 ((noInfo | chooseCity ==> selectCity | noInfo duration 2 difficulty
    5/10 delay 2)) ::
 ((noInfo | typing1 ==> typeSomething | noInfo duration 1 difficulty
    3/10 delay 3)
```

```
  (noInfo | typing2 ==> typeSomething | noInfo duration 1 difficulty
    3/10 delay 0)
                          ...
  (noInfo | typing13 ==> pushSearchBtn | noInfo duration 1 difficulty
     3/10 delay 0)) ::
 ((noInfo | searching ==> chooseAddress | noInfo duration 2
    difficulty 2/10 delay 0)),
                        waitTime : 0, status : notStarted,
                        criticalityLevel : 3/10, cognitiveLoad : 0 >
```

Listing 8.4: GPS task.

Case (2) is modelled similarly, but with only four typing actions before pushing the search button. In that case, the last basic task (choosing destination from a larger list) has duration 5 and difficulty $\frac{6}{10}$.

The GPS interface in case (1) is defined by the `Interface` object presented in Listing 8.5:

```
< gps : Interface | transitions :
    (gpsReady -- typeSearchMode --> chooseCity) ;
    (chooseCity -- selectCity --> typing1) ;
    (typing1 -- typeSomething --> typing2) ;
    (typing2 -- typeSomething --> typing3) ;
                    ...
    (typing13 -- pushSearchBtn --> searching) ;
    (searching -- chooseAddress --> gpsReady),
                        task : ... , previousAction : noAction,
                        currentState : gpsReady >
```

Listing 8.5: GPS interface.

The initial state of the working memory is presented in Listing 8.6.

```
< wm : WorkingMemory | capacity : 5, memory :
                        (car |-> goal(pickKey)) ;
                        (gps |-> goal(chooseAddress)) >
```

Listing 8.6: Initial state of the working memory.

8.2.1 Analysis

We use the techniques in Section 8.1 to analyse our models. We first analyse whether an enabled driving task can be ignored for more than six seconds:

```
Maude> (utsearch [1] {initState} =>*
         {< car : Interface | task :
           < driving : Task | waitTime : T:Time, A:AttributeSet > >
          REST:Configuration}
        such that T:Time > 6 .)
```

Real-Time Maude finds no such bad state when the driver types 13 characters:

```
No solution
```

However, when the driver only types four characters, the command returns a bad state:

```
Solution 1
   T:Time --> 7
   ...
```

The driver types the last two characters and finds the destination in the long list without turning his/her attention to driving in-between.

Sometimes even a brief distraction can be dangerous. For instance, when the road turns, a delay of seven time units in making the turn could be dangerous. We check the longest time needed for the driver to complete the `turnLeft` action:

```
Maude> (find latest {initState} =>*
         {< car : Interface | previousAction : turnLeft >
          REST:Configuration }
        with no time limit .)
```

Real-Time Maude shows that the left turn action is completed at time 24:

```
Result:
{< car : Interface | currentState : curveRight,
     previousAction : turnLeft,
     task : < driving : Task | cognitiveLoad : 1/24,
              criticalityLevel : 3/5, status : ongoing,
              subtasks :…, waitTime : 0 >,
     transitions :… >
 < gps : Interface | currentState : gpsReady,
```

```
     previousAction : chooseAddress,
     task : < findDestination : Task | cognitiveLoad : 3/5,
              criticalityLevel : 3/10, status : completed,
              subtasks : emptyTask, waitTime : 0 >,
     transitions : … >
 < wm : WorkingMemory | capacity : 5,
     memory :  car |-> keyInserted goal(pickKey) ;
               gps |-> noInfo >}
in time 24
```

However, the same analysis with an initial state *without* the GPS interface object and task shows that an undistracted driver finishes the left turn at time 17:

```
Result:  { ... } in time 17
```

8.3 Interacting with Multiple Infusion Pumps

This section illustrates the application of the framework to a case study based on an experiment described in [7], where users were asked to interact with two medical devices [19]. The aim of the experiment was to study multitasking strategies adopted by clinicians, to assess whether particular strategies could induce omission errors, e.g. forgetting to perform a procedural step required to complete the task.

In [56], Harrison et al. model and analyse the same case study. The results they provide by the analysis of the case study are similar to our results, however their analysis explains omission errors in terms of salience of information, resulting in redesign solutions related to the visibility of specific user interface elements. Our framework, on the other hand, provides a different view on the problem, showing that certain omission errors could be interpreted in terms of CL – in these cases, redesign solutions that simply enhance visibility of certain user interface elements might not be sufficient to prevent the problem.

After a brief introduction to the case study, we give its Real-Time Maude specification in Subsection 8.3.1, we present the case study's analysis in Subsection 8.3.2,

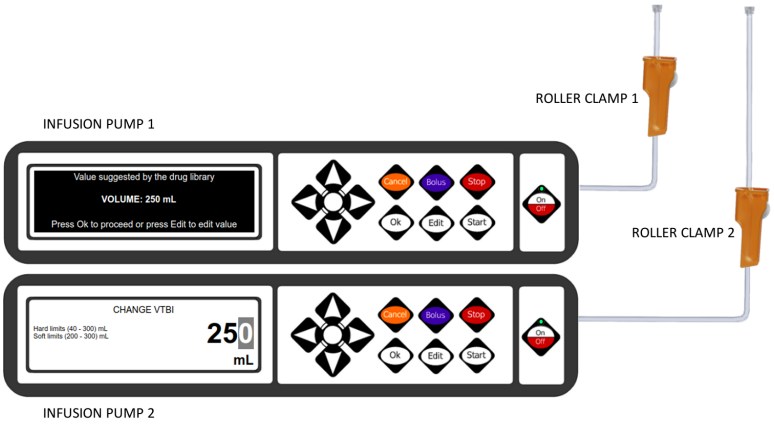

Figure 8.3.1: Example scenario with two infusion pumps.

and we finally present the model and the analysis of a redesign solution in Subsection 8.3.3.

The original experiment involved the use of two simulated infusion pumps (see Figure 8.3.1). Infusion pumps are medical devices routinely used in hospitals to inject fluids (e.g., drugs or nutrients) in the bloodstream of a patient in precise amount and at controlled rates. The devices under consideration provide a front panel with a display and a number of buttons used by clinicians to configure, operate, and monitor the pump. To set up an infusion pump, clinicians are usually required to perform five main steps:

- *Step 1.* Read infusion parameters, typically volume to be infused (vtbi) and infusion duration or infusion rate, from a prescription chart;
- *Step 2.* Enter the infusion parameters using the data entry system provided by the pump;
- *Step 3.* Connect the pump to the patient using a "giving set" (a transparent plastic tube with a needle at one end, and a bag with fluid at the other end);
- *Step 4.* Open the roller clamp to allow the fluid to circulate;
- *Step 5.* Start the infusion.

Time	*Prescription Chart*	*Pump 1*	*Pump 2*
1	read vtbi1		
2	read vtbi2		
3		enter vtbi1	
4			enter vtbi2
5	read time1		
6		enter time1	
7		open clamp1	
8	read time2		
9			enter time2
10			open clamp2
11		start infusion1	
12			start infusion2

Table 8.3.1: A possible multitasking strategy for setting up two infusion pumps.

Intensive care patients may be connected to more than one infusion pump at the same time. When multiple infusion pumps need to be configured, clinicians may choose to interleave the steps necessary for setting up the two pumps. This is usually done to optimise cognitive resources (e.g., memory load), or time (e.g., to perform operations on one pump while waiting that the other pump complete an operation) [7].

Different multitasking strategies may produce different memory loads. An example multitasking strategy for setting up the two pumps is shown in Table 8.3.1. The question we consider is *"What is the memory load necessary to complete a given task successfully using a particular multitasking strategy?"* An answer to this question could help manufacturers design devices that are simpler to use. It could also be used by hospitals, to develop better training material for clinicians. Academic researchers would also gain benefits, e.g., to test cognitive hypotheses before running an experimental study.

8.3.1 Modelling of multitasking strategies

The model includes interfaces representing each pump. The concurrent tasks relate to the procedure for setting vtbi and time values for the two pumps. To set the values, clinicians must read and memorise the values provided by the prescription chart, and then use the pumps' data entry system to enter the values.

The specification of the task for setting up *Pump 1* is shown in Listing 8.7 (the task for setting up *Pump 2* is specified the same way).

```
1 < settingPump1 : Task | subtasks :
2   ((noInfo | prescriptionFormVtbiP1 ==> noAction|vtbi300 duration 1
      difficulty 2/10 delay 0)
3    (vtbi300 | setVTBIP1 ==> type300 | noInfo duration 1 difficulty 2/10
       delay 0)) ::
4   ((noInfo | prescriptionFormTimeP1 ==> noAction|time3 duration 1
      difficulty 2/10 delay 0)
5    (time3 | setTimeP1 ==> type3 | noInfo duration 1 difficulty 2/10
      delay 0)) ::
6   ((clampOpeningP1 | clampP1 ==> openClampP1|noInfo duration 1
      difficulty 2/10 delay 0)) ::
7   ((noInfo | infusionReadyP1 ==> startInfusionP1|noInfo duration 1
      difficulty 2/10 delay 0)),
8         status : notStarted, cognitiveLoad : 0,
9         criticalityLevel : 6/10, waitTime : 0 >
```

Listing 8.7: Specification of the task for Pump1.

The task for setting *Pump 1* consists of six basic tasks, grouped into four subtasks:

1. Read and memorise the vtbi value for pump 1 from the prescription chart (line 1 in Listing 8.7);
2. Enter vtbi in pump 1 (line 2);
3. Read and memorise the infusion duration for pump 1 from the prescription chart (line 3);
4. Enter infusion duration in pump 1 (line 4);
5. Open clamp 1 (line 5);

6. Start infusion (line 6).

The basic task at line 2 in Listing 8.7 models a cognitive basic task: the operator reads from the prescription chart the vtbi value for the pump 1, he/she finds out that the value to be inserted is 300 and inserts into his/her working memory the cognition `vtbi300`.

8.3.2 Analysis

To perform the analysis, two initial states are required which contain hypotheses about the memory load necessary to complete the task. An example initial state is in Listing 8.8, where x is the capacity of the WM (line 4).

```
1 < wm : WorkingMemory | memory :
2     (pump1 |-> goal(startInfusionP1) clampOpeningP1);
3     (pump2 |-> goal(startInfusionP2) clampOpeningP2),
4   capacity : x >
```

Listing 8.8: Example initial state of WM.

Lines 2 and 3 represent the hypotheses about the initial content of the WM. Two goals are specified (i.e., starting the infusion), and two memory items to remember to open the roller clamps before starting the infusion (`clampOpeningP1` and `clampOpeningP2`).

Real Time Maude is then used to check whether a given WM capacity is sufficient to achieve the goal. This helps to obtain a quantitative evaluation of the complexity of the task (in terms of memory load) and to identify situations where the multitasking strategy could exceed the WM capacity of the operator.

The following search command in Real Time Maude checks whether there is any such situation where the user forgets a piece of information that is relevant to the tasks and is therefore unable to complete the tasks successfully:

```
Maude> (utsearch [1] {initState1} =>!
        {< I:InterfaceId : Interface |  task :
            < T:Oid : Task | status : TS:TaskStatus,
                    A:AttributeSet > > REST:Configuration}
        such that TS:TaskStatus =/= completed .)
```

For the considered case study, the model checker finds interleaving strategies where the user is not able to complete the tasks when the capacity of the WM is set to 5. One such example is in Table 8.3.1: with WM capacity set to 5, the user can perform correctly the concurrent tasks up to *enter time 2* (i.e., an omission error occurs for action *open clamp 2*). If the WM capacity is set to 6, on the other hand, the analysis indicates that the user is always able to reach the goal successfully, using any multitasking strategy.

The results of our analysis are in line with the experimental results obtained in [7], and provide an explanation to the omission error in terms of CL.

8.3.3 Modelling and Analysing a Redesigned Interface

To check whether a design solution could be adopted to reduce memory load, the pump design was modified using the Next-Action Cueing technique [31]. A set of cues is presented in the user interface of the system at appropriate moments, to remind the operator what action should be performed next. For example, when the clamp needs to be opened, the operator does not need to retrieve this information from WM if there is a visual cue on the pump screen that indicates what needs to be done (e.g., a simple message "OPEN CLAMP" on the display of the pump). By using this approach the operator is not required to remember all information from the beginning of the interaction, as information can be gathered just by looking at the cues on the front panel of the injector device.

This design solution was added to the model, by introducing a cognitive basic task in the subtask for setting up an infusion pump: perceiving the cue will trigger the activation and execution of a certain action. Such redesigned task is presented in Listing 8.9.

```
1 ((noInfo | prescriptionFormVtbiP1 ==> noAction|vtbi300 duration 1
      difficulty 2/10 delay 0)
2  (vtbi300 | setVTBIP1 ==> type300 | noInfo duration 1 difficulty 2/10
      delay 0)) ::
3 ((noInfo | prescriptionFormTimeP1 ==> noAction|time3 duration 1
      difficulty 2/10 delay 0)
4  (time3 | setTimeP1 ==> type3|noInfo duration 1 difficulty 2/10 delay
      0)) ::
```

```
5  ((noInfo | clampP1 ==> noAction|clampOpeningP1  duration 1 difficulty
       2/10 delay 0)
6  (clampOpeningP1 | clampP1 ==> openClampP1|noInfo duration 1 difficulty
        2/10 delay 0)) ::
7 ((noInfo | infusionReadyP1 ==> startInfusionP1|noInfo duration 1
       difficulty 2/10 delay 0))
```

Listing 8.9: Modified task for setting up Pump1.

The basic task at line 5 in Listing 8.9 models the cognitive basic task mentioned above: the operator, by looking at the pump interface, notices a signal and understands that he/she has to open the clamp. He/she then inserts into his/her working memory the cognition `clampOpeningP1`, which he/she uses in the next basic task to perform the action `openClampP1` with the `pump1` interface.

Analysis of this new version of the task indicates that the user is always able to complete the tasks with a WM capacity of 5 for all possible interleaving strategies.

8.4 Air Traffic Control Operator

This Section illustrates the application of our framework to the study of an air traffic control (ATC) operator [17]. We present the Real-Time Maude specification in Subsection 8.4.1, and the analysis of different ATC scenarios in Subsections 8.4.2 and 8.4.3.

ATC operators are personnel responsible for monitoring and controlling air traffic. They usually work in ATC centres and control towers on the ground, by monitoring the position, speed, altitude, and route of aircraft in their assigned sector, visually and by radar. In addition, they also deal with radio communication with pilots to give them instructions and to receive useful information about the flights, which they report on *flight progress strips* (FPSs or strips). Such FPS are paper strips used to record basic information for each aircraft, such as callsign, aircraft type, destination, altitude, planned route, flight level (FL), etc. Controllers update strips dynamically as they control the associated aircraft through their sectors. One of the main tasks of ACT operators is to avoid flight collisions, i.e. to avoid that the distance between aircraft goes below a minimum prescribed distance. When this

happens, they say that the aircraft violates *separation*. Air traffic controllers can also transfer an aircraft to the next sector controller when they are too busy, or assign one of their tasks to an assistant.

Despite the availability of advanced radar and technological support, strips and other aids, ATC operators heavily rely on working memory, by encoding, storing and retrieving information recently perceived about aircraft and the environment, such as pilot requests, information about the flights, weather reports, and so on [101].

8.4.1 Modeling ATC tasks

We focus on the following three tasks that controllers have to carry out concurrently:

1. *Monitoring a radar sector* to: (i) keep the distance between aircraft under control and avoid possible collisions; (ii) move an aircraft to the next sector controller when they are too busy; and (iii) visually perceive new information about flights.

2. *Managing pilots' calls* to update information about flights on strips.

3. *Checking that an assistant carries out assigned tasks*.

For the first of these tasks, we model a different monitoring task and consequently, a different radar interface, for each critical zone of the radar sector to be monitored. The monitoring task is essentially a set of three different subtasks; we show each subtask of this task separately:

i Controlling parts of the screen, possibly adding information about flights to the memory.

```
1 < monitoring : Task | subtasks :
2  ((noInfo | Screen1 ==> lookAtScreen1 | noInfo duration 4
      difficulty 4/10 delay 0)
3   (noInfo | Screen2 ==> lookAtScreen2 | updFL duration 4
      difficulty 4/10 delay 0)
```

```
4   (noInfo | Screen3 ==> lookAtScreen3 | noInfo duration 4
      difficulty 4/10 delay 0)) ::
5   ...
```

Listing 8.10: First subtask of the monitoring task.

This subtask consists of three basic tasks of looking at different parts of the screen. The second basic task (Line 3 in Listing 8.10) models that, while looking at a section of the screen, the ATC operator notices that one of the aircraft has changed its flight level, and the operator adds this basic information (updFL) to his/her memory.

ii. Monitoring a possible collision.

```
1   ... ::
2  ((noInfo | Screen3 ==> noAction | possibleClsn1 duration 3
      difficulty 4/10 delay 0)
3   (possibleClsn1 | Collision1 ==> monitorClsn1 | noInfo duration 4
       difficulty 5/10 delay 0)) ::
4   ...
```

Listing 8.11: Second subtask of the monitoring task.

This subtask consists of two basic tasks. The first (line 2 in Listing 8.11) is a cognitive basic task which models that the ATC operator, while looking at the screen, understands that a collision could happen. He/she then adds the cognition `possibleClsn1` to his/her memory. She therefore changes his/her mental plan by recovering such a cognition from his/her working memory and monitors the possible collision in the second basic task (line 3).

iii. Moving an aircraft to the next sector controller, activated by a cognition about the presence of too many aircraft on the screen.

```
1   ... ::
2   ((noInfo | Screen4 ==> lookAtScreen4 | noInfo duration 4
      difficulty 4/10 delay 0)
3   (noInfo | Screen4 ==> noAction | movingAircraft duration 3
      difficulty 4/10 delay 0)
```

```
4   (movingAircraft | Screen5 ==> move | noInfo duration 4
      difficulty 5/10 delay 0)),
5   waitTime : 0,
6   status : notStarted,
7   criticalityLevel : 8/10,
8   cognitiveLoad : 0 >
```

Listing 8.12: Third subtask of the monitoring task.

This subtask consists of three basic tasks. The first one (line 2 in Listing 8.12) is explained above. The second one (line 3) is a cognitive basic task which models the ATC operator realising that there are too many aircraft in his/her sector, and he/she then adds the cognition `movingAircraft` to his/her working memory. In the third basic task (line 4), he/she retrieves this cognition from his/her working memory and moves an aircraft to the next sector controller.

The end of the code above shows the remaining attributes of the `Task` object `monitoring`.

The criticality level of the monitoring tasks could vary, depending on the number of aircraft present in the sector or their type: some of them could require little active control, such as overflights, "lows and slows," and aircraft on the pilots' own navigation or on a radar route [101].

The radar interface associated to such monitoring task is defined by the `Interface` object presented in Listing 8.13.

```
1  < radar : Interface |
2    transitions :
3      (Screen1 -- lookAtScreen1 --> Screen2) ;
4      (Screen2 -- lookAtScreen2 --> Screen3) ;
5      (Screen3 -- lookAtScreen3 --> Collision1) ;
6      (Collision1 -- monitorClsn1 --> Screen4) ;
7      (Screen4 -- lookAtScreen4 --> Screen5) ;
8      (Screen5 -- move --> stop),
9    task : < monitoring : Task | ... >,
10   previousAction : noAction,
11   currentState : Screen1 >
```

Listing 8.13: Radar interface.

For the second task above, we model a radio communication task, and consequently a radio interface, for each communication with a different pilot. It consists of a sequence of subtasks modelling the pilot's calls and the updating of strips with the information received, and is formalised by the `Task` object presented in Listing 8.14.

```
1 < radioCommunication : Task | subtasks :
2  ((noInfo | call1 ==> communicating1 | updatingAltitude duration 3
      difficulty 4/10 delay 7)) ::
3  ((updatingAltitude | strip1 ==> updating1 | noInfo duration 2
      difficulty 3/10 delay 0)) ::
4  ((noInfo | call2 ==> communicating2 | updatingRoute duration 3
      difficulty 4/10 delay 5)) ::
5  ((updatingRoute | strip2 ==> updating2 | noInfo duration 2 difficulty
      3/10 delay 0)) ::
6  ((noInfo | call3 ==> communicating3 | updatingFL duration 3 difficulty
       4/10 delay 2)) ::
7  ((updatingFL | strip3 ==> updating3 | noInfo duration 2 difficulty
      3/10 delay 0))
8   waitTime : 0, criticalityLevel : 4/10,
9   cognitiveLoad : 0, status : notStarted >
```

Listing 8.14: Radio communication task.

For example, first subtask (line 2 in Listing 8.14) models a call from a pilot, who tells the ATC operator about an update of the altitude (`updatingAltitude`), and the following addition of this information to the ATC operator's working memory. The operator then uses this information in the second subtask (line 3) where he/she retrieves the information from memory and updates the altitude on the flight's strip.

The criticality level of the radio communication task could vary according to the type of aircraft the controller is receiving information from. The radio interface associated to such a communication task is formalised by the object presented in Listing 8.15.

```
1 < radio : Interface |
2      transitions :
3         (call1 -- communicating1 --> strip1) ;
4         (strip1 -- updating1 --> call2) ;
5         (call2 -- communicating2 --> strip2) ;
```

```
6          (strip2 -- updating2 --> call3) ;
7          (call3 -- communicating3 --> strip3) ;
8          (strip3 -- updating3 --> stop),
9       task : < radioCommunication : Task | ... >,
10      previousAction : noAction,
11      currentState : call1 >
```

Listing 8.15: Radio interface.

Finally, the third task, checking that an assistant carries out an assignment, is formalised by the object presented in Listing 8.16.

```
1 < checkingAssignedTask : Task | subtasks :
2     ((noInfo | assistantReady ==> checkingTask | noInfo duration 3
      difficulty 3/10 delay 0)),
3      waitTime : 0,
4      status : notStarted,
5      criticalityLevel : 3/10,
6      cognitiveLoad : 0 >
```

Listing 8.16: Task of checking that an assistent carries out an assignment.

We model the assistant as an interface with which the controller has to interact: he/she is formalised by the `Interface` object presented in Listing 8.17

```
1 < assistant : Interface |
2      transitions :
3         (assistantReady -- checkingTask --> stop),
4      task : < checkingAssignedTask : Task | ... >,
5      previousAction : noAction,
6      currentState : assistantReady >
```

Listing 8.17: Assistent interface.

8.4.2 Analysing Urgency

As in the car/GPS example, some of the tasks are not only characterised by high criticality but also by urgency. If two aircraft violate separation, the controller has to monitor the identified conflict situation as soon as he/she perceives it. The fact that many such tasks may have to be performed at the same time could lead to

a dangerous situation where some urgent actions are not performed when they should.

To analyse urgency, we model a situation where an air traffic controller monitors his/her radar sector while communicating with a pilot via radio. During the monitoring activity, the ATC operator finds three possible collisions (we model three subtasks with actions `monitorClsn1`, `monitorClsn2`, and `monitorClsn3`); however, the radio communication task might prevent him/her from monitoring such collisions at a specific time.

We check the longest time needed for the controller to complete all `monitorClsn`*X* actions:

```
Maude> (find latest {initState1} =>*
     {< radar : Interface | previousAction : monitorClsn >
      REST:Configuration }
    with no time limit .)
```

The result of this analysis shows that the `monitorClsn1` action is completed at time 15, the `monitorClsn2` action is completed at time 42, and the `monitorClsn3` action is completed at time 57. However, the same analysis with an initial state with just the radar interface object and task shows that when the ACT operator is not distracted, he/she finishes the `monitorClsn2` action at time 50 – 8 time units *later* than in the the multitasking scenario – and the `monitorClsn3` action completes at time 72, i.e. 15 time units later than in the multitasking scenario.

8.4.3 Analysing Memory Failures due to Multitasking

Analysis of interviews with air traffic controllers in [101] indicate that memory errors are associated with working memory overload and distraction. We focus on three kinds of errors presented in the report by modelling, simulating, and analysing them, and we try to give a plausible explanation of these errors in term of cognitive causes: *prospective memory failure*, *retrospective memory failure*, and *forgetting temporary information*.

Prospective Memory Failure

Prospective memory is the form of memory involved in remembering to perform a planned action. Sixteen errors presented in [101] involve prospective memory failure.

To analyse prospective memory failures, we model a situation where the controller monitors three different zones in her radar sector while communicating with two different pilots and checking that an assistant carries out an assigned task. The checking assignment task has a delay of 20 time units, since controller plans to perform this task in the future.

The initial state of the working memory is presented in Listing 8.18.

```
< wm : WorkingMemory | capacity\,: 7,
          memory : (radar1 |-> goal(lookAtScreen3)) ;
                   (radar2 |-> goal(lookAtScreen12)) ;
                   (radar3 |-> goal(lookAtScreen7)) ;
                   (radio1 |-> goal(updating1)) ;
                   (radio2 |-> goal(updating2)) ;
                   (assistant |-> goal(checkingTask)) >
```

Listing 8.18: Initial state of the working memory.

We check if all tasks are guaranteed to be completed:

```
Maude > (utsearch [1] initState1 =>!
           {< I:InterfaceId : Interface | task :
               < T:Oid : Task | status : TS:TaskStatus,
                                A:AttributeSet > >
            REST:Configuration }
         such that TS:TaskStatus =/= completed .)
```

and we find a bad state: the goal associated to the assistant interface is deleted from memory and the controller cannot complete the checking assignment task.

Retrospective Memory Failure

Retrospective memory is the memory of people, words, or events encountered or experienced in the past. Three of the errors presented in [101] involve retrospective memory failure: controllers lose track of task progress since they forget the

action previously performed. Task interruptions have disruptive effects on task performance, although several studies show that they have different consequences when performed at different moments [1, 60].

To analyse retrospective memory failures, we model a situation where the air traffic controller monitors the radar and decides to move an aircraft to the next controller's sector when he/she perceives that she is too busy. At the same time, he/she has to answer three different calls from pilots and annotate the information received by them on strips.

We show that interrupting the monitoring task at different moments can have different consequences. We model two initial states, `initState1` and `initState2`. In the first one the controller receives three calls after a delay of 7, 8 and 9 time units, respectively; in the second one the controller receives three calls after a delay of 8, 9 and 10 time units, respectively. The difficulties of the radio communication tasks have been set in order to have the same cognitive load for each task.

We check whether or not all tasks are guaranteed to complete for both initial states. Our analysis found an undesired state for `initState1`, and no such state for `initState2`. In the first case the main task is interrupted after the operator decides to transfer the aircraft: the addition of new information from calls lead to memory overload which resulted in forgetting this decision. In the second case the monitoring task is not interrupted after the controller's decision to move the aircraft.

Forgetting Temporary Information

Many of the errors reported in [101] concern forgetting temporary information. Some of these errors involve forgetting about the presence of aircraft that require little active control. To analyse such errors, we model a situation where the air traffic controller monitors different zones of the radar screen at the same time. Some of these zones have highly critical situations to monitor, while one of them have aircraft that require low control and thus have lower criticality. We show that the controller can forget the less critical sector because he/she is distracted by other

sectors, and too much information is added to his/her memory.

We model five interfaces representing the different zones of the screen. Each zone has a different cognitive load and a different criticality level, depending on the number of flights and the type of aircraft flying there.

We analyse whether all tasks are guaranteed to be completed. The usual command shows that the task associated with the zone with low criticality, representing a zone with flights which require little control, does not complete because that task's goal is deleted from the working memory. We check if the task associated with that zone is at least started with commands:

```
Maude> (find latest {initState1} =>*
    {< zone2 : Interface | previousAction : lookAtScreen4 >
     REST:Configuration }
    with no time limit .)
```

and

```
Maude> (find latest {initState1} =>*
    {< radar : Interface | previousAction : lookAtScreen12 >
     REST:Configuration }
    with no time limit .)
```

The first command shows that the first action of the task is performed at time 33, while the second command shows that the last action of the task is never performed.

9 Conclusion

In this thesis we have developed a new model of safety-critical human multitasking which describes the cognitive processes involved in HCI, and the human working memory according with results from applied psychology on human selective attention and on working memory.

The model consists of a set of interfaces representing the interfaces of the devices with which the user interacts performing tasks, which are described as sequences of subtasks (which in turn are sequences of basic tasks) representing single *scenarios*. Tasks are characterised by a measure of how much they are cognitively demanding, and since we focus on safety-critical multitasking, they also include a measure of how much they are critical. Since we consider structured task where each basic task can idle waiting to be enabled, we define a new function to compute the cognitive load of each task, based on psychological literature; the cog-

nitive load of each task changes every time a new subtask begins and remains the same throughout its execution.

The semantics of the model is defined as a purely probabilistic transition system (PPTS), whose configuration includes a set of tasks, a map assigning to each task its cognitive load, a map storing the current state of the interfaces, a map storing the last time each task was executed, a global clock, and a working memory modelled as a map assigning to each interface the information useful for the interaction with it. The transition relation is defined in an inductive way by a set of inference rules, each of which models the different cognitive behaviour involved in multitasking interaction and determine how attention is directed to the different tasks and how this would change the state of the PPTS. The switching of attention from a task to another has been modelled through a new algorithm based on the tasks' cognitive load, their criticality and the time they were ignored.

We implemented the model as a simulator which enables us to get a quick feedback about whether a human can safely perform multiple tasks concurrently. Moreover, although the selective attention algorithm is consistent with the relevant psychological literature, we validate our algorithm against data collected from an experimental study with real users involved concurrently in one critical task and one "distractor" task with different levels of cognitive load. We validate the capability of the algorithm to produce relevant results, given adequate parameters. It should be clear that concrete models will have to be fine tuned with the help of domain and human factors experts as their validity is dependent on the values used.

Moreover, we implemented the model as an executable framework in Real-Time Maude, which enables us to model, simulate and analyse safety-critical human multitasking through reachability analysis. We showed how the Real-Time Maude framework is able to analyse different problems in safety-critical human multitasking such as errors caused by user distractions, cognitive overload and memory overload. We finally illustrated our framework with three case studies from different application domains: a user interacting with a GPS navigation system while driving, a medical operator setting multiple infusion pumps simultaneously, and an air traffic control (ATC) operator dealing with some typical ATC scenarios in-

volving memory errors.

Our modelling and analysis approach is intended for developers of interactive critical systems to identify plausible human multitasking strategies that are likely to be adopted by users when using multiple interactive systems at the same time and to estimate the memory load necessary to complete concurrent tasks. It can also be used to identify plausible problems in the interaction with the systems due to cognitive overload. This type of analysis is especially useful at the early stages of system design, to better understand the effort necessary to operate the system when an implementation or a prototype of the system is unavailable. The analysis can also be used retrospectively, to analyse already implemented systems and complement results from user studies.

Given a specific case study to analyse, whose real data and results are held, with our analysis approach the analyst can create a model using the real values held for some of the parameters of the model (i.e. duration and delay), and some valid values for the not measurable parameters (i.e. difficulty and criticality). Then, the analyst has to fine-tune the non-measurable parameters by running the simulations until the results agree with the held case study results. Once all values are held, he/she can use the Real-Time Maude framework to perform analyses.

It is worth to note that the analysis needs to be done knowing the system the developers want to check, in order to identify the criticalities observed by the analysis and in order to distinguish real criticalities from not-problematic ones.

The proposed framework could be further developed in different directions. Since Real-Time Maude is not a probabilistic rewriting logic language, some aspects of the framework (e.g. the selective attention algorithm) are essentially deterministic.

Even if such simplification exists, it is worth to note that the set of possible behaviours analysed by the model checker is a subset of all the possible correct behaviours: the Real-Time Maude framework considers just the more likely one, which is, nevertheless, one of the correct behaviours of the system.

However, the framework could be extended to a probabilistic setting, according to the formal model proposed. Such probabilistic real-time models could then be

subjected to statistical model checking analysis using tools such as PVeStA [2].

From data collected from the experimental study (presented in Chapter 6), we observed that when users interact with a timed task (i.e. a task with a strict deadline), usually they tend to stay more on the task when such a deadline is close. Therefore, we could modify the computation of the a factor, taking into account such aspect. For instance, the criticality parameter could be defined as a function of the time left for the expiration of the task deadline.

Moreover, we could add to our model the definition of other cognitive systems: for instance, by adding the long term memory we would be able to model tasks where information are not volatile but have to be retrieved from the long term memory of the user. Moreover, in [81] is presented an hybrid automata model of the dopamine system; the dopamine is a neurotransmitter responsible for the human perception of the satisfaction, which strongly stimulates the attention, the memory and the learning. We could integrate in our framework the model developed in [81] in order to simulate and analyse the satisfaction of the user in a multitasking interaction.

We could also extend some existing model (e.g. the ACT-R model) with our multitasking model. Finally, we could compare our model with related models of human multitasking on selected case studies, and apply it to other safety-critical multitasking applications.

References

[1] Piotr D Adamczyk and Brian P Bailey. If not now, when?: the effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–278. ACM, 2004.

[2] Musab AlTurki and José Meseguer. Pvesta: A parallel statistical model checking and quantitative analysis tool. In *International Conference on Algebra and Coalgebra in Computer Science*, pages 386–392. Springer, 2011.

[3] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004.

[4] Catherine M Arrington and Gordon D Logan. The cost of a voluntary task switch. *Psychological science*, 15(9):610–615, 2004.

[5] Richard C Atkinson and Richard M Shiffrin. Human memory: A proposed system and its control processes1. In *Psychology of learning and motivation*, volume 2, pages 89–195. Elsevier, 1968.

[6] *Dangerous distraction. Safety Investigation Report B2004/0324*. Australian Transport Safety Bureau, 2005.

[7] Jonathan Back, Anna Cox, and Duncan Brumby. Choosing to interleave: Human error and information access cost. In *SIGCHI Conference on Human Factors in Computing Systems*, CHI'12, pages 1651–1654. ACM, 2012. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208289. URL http://doi.acm.org/10.1145/2207676.2208289.

[8] Alan Baddeley. Working memory. *Current biology*, 20(4):R136–R140, 2010.

[9] Alan D Baddeley and Graham Hitch. Working memory. In *Psychology of learning and motivation*, volume 8, pages 47–89. Elsevier, 1974.

[10] Eric Barboni, Jean-François Ladry, David Navarre, Philippe Palanque, and Marco Winckler. Beyond modelling: an integrated environment supporting co-execution of tasks and systems models. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 165–174. ACM, 2010.

[11] Pierre Barrouillet. Transitive inferences from set-inclusion relations and working memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(6):1408, 1996.

[12] Pierre Barrouillet and Valérie Camos. Developmental increase in working memory span: Resource sharing or temporal decay? *Journal of Memory and Language*, 45(1):1–20, 2001.

[13] Pierre Barrouillet, Sophie Bernardin, and Valérie Camos. Time constraints and resource sharing in adults' working memory spans. *Journal of Experimental Psychology: General*, 133(1):83, 2004.

[14] Pierre Barrouillet, Sophie Bernardin, Sophie Portrat, Evie Vergauwe, and Valérie Camos. Time and cognitive load in working memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 33(3):570, 2007.

[15] Pierre Barrouillet, Nathalie Gavens, Evie Vergauwe, Vinciane Gaillard, and Valérie Camos. Working memory span development: a time-based resource-sharing model account. *Developmental psychology*, 45(2):477, 2009.

[16] Giovanna Broccia. Model-based analysis of driver distraction by infotainment systems in automotive domain. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 133–136. ACM, 2017.

[17] Giovanna Broccia, Paolo Milazzo, and Peter Csaba Ölveczky. Formal modeling and analysis of safety-critical human multitasking. *Under review*.

[18] Giovanna Broccia, Paolo Milazzo, and Peter Csaba Ölveczky. An algorithm for simulating human selective attention. In *International Conference on Software Engineering and Formal Methods*, pages 48–55. Springer, 2017.

[19] Giovanna Broccia, Paolo Masci, and Paolo Milazzo. Modeling and analysis of human memory load in multitasking scenarios: Late-breaking results. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, page 9. ACM, 2018.

[20] Giovanna Broccia, Paolo Milazzo, and Peter Csaba Ölveczky. An executable formal framework for safety-critical human multitasking. In *NASA Formal Methods Symposium*, pages 54–69. Springer, 2018.

[21] Roberto Bruni and José Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 360(1-3):386–414, 2006.

[22] J Creissac Campos and Michael D Harrison. Systematic analysis of control panel interfaces using formal tools. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 72–85. Springer, 2008.

[23] José C Campos and Michael D Harrison. Interaction engineering using the ivy tool. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 35–44. ACM, 2009.

[24] José Creissac Campos, Manuel Sousa, Miriam C Bergue Alves, and Michael D Harrison. Formal verification of a space system's user interface with the ivy workbench. *IEEE Trans. Human-Machine Systems*, 46(2):303–316, 2016.

[25] SK Card, TP Moran, and A Newell. The psychology of human-computer interaction. hillsdale, new jersey: Lawerence erlbaum associates, 1983.

[26] Stuart K Card, Thomas P Moran, and Allen Newell. Computer text-editing: An information-processing analysis of a routine cognitive skill. *Cognitive psychology*, 12(1):32–74, 1980.

[27] Stuart K Card, Thomas P Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, 1980.

[28] Robbie Case, D Midian Kurland, and Jill Goldberg. Operational efficiency and the growth of short-term memory span. *Journal of experimental child psychology*, 33(3):386–404, 1982.

[29] Antonio Cerone. A cognitive framework based on rewriting logic for the analysis of interactive systems. In *International Conference on Software Engineering and Formal Methods*, pages 287–303. Springer, 2016.

[30] Noam Chomsky. A review of bf skinner's verbal behavior. *Readings in philosophy of psychology*, 1:48–63, 1980.

[31] Phillip H. Chung and Michael D. Byrne. Cue effectiveness in mitigating postcompletion errors in a routine procedural task. *Int. J. Hum.-Comput. Stud.*, 66(4):217–232, April 2008. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2007.09.001. URL http://dx.doi.org/10.1016/j.ijhcs.2007.09.001.

[32] T Clark et al. Impact of clinical alarms on patient safety. Technical report, ACCE Healthcare Technology Foundation, 2006.

[33] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All about maude-a high-performance logical framework: how to specify, program and verify systems in rewriting logic*. Springer-Verlag, 2007.

[34] Sébastien Combéfis, Dimitra Giannakopoulou, Charles Pecheur, and Michael Feary. A formal framework for design and analysis of human-machine interaction. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 1801–1808. IEEE, 2011.

[35] Andrew Conway, Chris Jarrold, and Michael Kane. *Variation in working memory*. Oxford University Press, 2008.

[36] Andrew RA Conway, Nelson Cowan, Michael F Bunting, David J Therriault, and Scott RB Minkoff. A latent variable analysis of working memory capacity, short-term memory capacity, processing speed, and general fluid intelligence. *Intelligence*, 30(2):163–183, 2002.

[37] Andrew RA Conway, Michael J Kane, and Randall W Engle. Working memory capacity and its relation to general intelligence. *Trends in cognitive sciences*, 7(12):547–552, 2003.

[38] Andrew RA Conway, Michael J Kane, Michael F Bunting, D Zach Hambrick, Oliver Wilhelm, and Randall W Engle. Working memory span tasks: A methodological review and user's guide. *Psychonomic bulletin & review*, 12(5):769–786, 2005.

[39] David E Copeland and Gabriel A Radvansky. Phonological similarity in working memory. *Memory & Cognition*, 29(5):774–776, 2001.

[40] Nelson Cowan. What are the differences between long-term, short-term, and working memory? *Progress in brain research*, 169:323–338, 2008.

[41] Nelson Cowan, Emily M Elliott, J Scott Saults, Candice C Morey, Sam Mattox, Anna Hismjatullina, and Andrew RA Conway. On the capacity of attention: Its estimation and its role in working memory and cognitive aptitudes. *Cognitive psychology*, 51(1):42–100, 2005.

[42] Fergus IM Craik and Robert S Lockhart. Levels of processing: A framework for memory research. *Journal of verbal learning and verbal behavior*, 11 (6):671–684, 1972.

[43] Meredyth Daneman and Patricia A Carpenter. Individual differences in working memory and reading. *Journal of verbal learning and verbal behavior*, 19(4):450–466, 1980.

[44] Jan W De Fockert, Geraint Rees, Christopher D Frith, and Nilli Lavie. The role of working memory in visual selective attention. *Science*, 291(5509): 1803–1806, 2001.

[45] Adele Diamond. Executive functions. *Annual review of psychology*, 64:135–168, 2013.

[46] Thomas A Dingus, Feng Guo, Suzie Lee, Jonathan F Antin, Miguel Perez, Mindy Buchanan-King, and Jonathan Hankey. Driver crash risk factors and prevalence evaluation using naturalistic driving data. *Proceedings of the National Academy of Sciences*, 113(10):2636–2641, 2016.

[47] RK Dismukes and Jessica Nowinski. Prospective memory, concurrent task management, and pilot error. In *Attention: From Theory to Practice*. Oxford Univ. Press, 2007.

[48] David J Duke, Philip J Barnard, Jon May, and David A Duce. Systematic development of the human interface. In *Software Engineering Conference, 1995. Proceedings., 1995 Asia Pacific*, pages 313–321. IEEE, 1995.

[49] John Duncan. Goal weighting and the choice of behaviour in a complex world. *Ergonomics*, 33(10-11):1265–1279, 1990.

[50] Randall W Engle. Working memory capacity as executive attention. *Current directions in psychological science*, 11(1):19–23, 2002.

[51] Randall W Engle, Stephen W Tuholski, James E Laughlin, and Andrew RA Conway. Working memory, short-term memory, and general fluid intelligence: a latent-variable approach. *Journal of experimental psychology: General*, 128(3):309, 1999.

[52] Michael Freed. Reactive prioritization. In *Proceedings of the 2nd NASA international workshop on planning and scheduling in space*, 2000.

[53] Gabriel Gelman, Karen M Feigh, and John Rushby. Example of a complementary use of model checking and human performance simulation. *IEEE Transactions on Human-Machine Systems*, 44(5):576–590, 2014.

[54] Robert S Gutzwiller, Christopher D Wickens, and Benjamin A Clegg. The role of time on task in multi-task management. *Journal of Applied Research in Memory and Cognition*, 5(2):176–184, 2016.

[55] John Hamilton. Think you're multitasking? think again. *Morning Edition*, 2008.

[56] Michael D Harrison, José C Campos, Rimvydas Rukšėnas, and Paul Curzon. Modelling information resources and their salience in medical device design. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 194–203. ACM, 2016.

[57] John R Helleberg and Christopher D Wickens. Effects of data-link modality and display redundancy on pilot performance: An attentional perspective. *The International Journal of Aviation Psychology*, 13(3):189–210, 2003.

[58] Adam Houser, Lanssie Mingyue Ma, Karen M Feigh, and Matthew L Bolton. Using formal methods to reason about taskload and resource conflicts in simulated air traffic scenarios. *Innovations in Systems and Software Engineering*, 14(1):1–14, 2018.

[59] Cristina Iani and Christopher D Wickens. Factors affecting task management in aviation. *Human factors*, 49(1):16–24, 2007.

[60] Shamsi T Iqbal and Brian P Bailey. Investigating the effectiveness of mental workload as a predictor of opportune moments for interruption. In *CHI'05 extended abstracts on Human factors in computing systems*. ACM, 2005.

[61] Anjali Joshi, Steven P Miller, and Mats PE Heimdahl. Mode confusion analysis of a flight guidance system using formal methods. In *Digital Avionics Systems Conference, 2003. DASC'03. The 22nd*, volume 1, pages 2–D. IEEE, 2003.

[62] Michael J Kane, David Z Hambrick, Stephen W Tuholski, Oliver Wilhelm, Tabitha W Payne, and Randall W Engle. The generality of working memory capacity: a latent-variable approach to verbal and visuospatial memory span and reasoning. *Journal of Experimental Psychology: General*, 133(2): 189, 2004.

[63] Michael J Kane, Andrew RA Conway, Timothy K Miura, and Gregory JH Colflesh. Working memory, attention control, and the n-back task: a question of construct validity. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 33(3):615, 2007.

[64] Ioanna Katidioti and Niels A Taatgen. Choice in multitasking: How delays in the primary task turn a rational into an irrational multitasker. *Human factors*, 56(4):728–736, 2014.

[65] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[66] Roberta L Klatzky. When to inspect? recurrent inspection decisions in a simulated risky environment. *Journal of Experimental Psychology: Applied*, 6(3):222, 2000.

[67] Robert Kurzban, Angela Duckworth, Joseph W Kable, and Justus Myers. An opportunity cost model of subjective effort and task performance. *Behavioral and Brain Sciences*, 36(6):661–679, 2013.

[68] Yelena Kushleyeva, Dario D Salvucci, and Frank J Lee. Deciding when to switch tasks in time-critical multitasking. *Cognitive Systems Research*, 6(1): 41–49, 2005.

[69] Patrick C Kyllonen and Raymond E Christal. Reasoning ability is (little more than) working-memory capacity?! *Intelligence*, 14(4):389–433, 1990.

[70] Kim G Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28, 1991.

[71] Nilli Lavie, Aleksandra Hirst, Jan W De Fockert, and Essi Viding. Load theory of selective attention and cognitive control. *Journal of Experimental Psychology: General*, 133(3):339, 2004.

[72] Daniela Lepri, Erika Ábrahám, and Peter Csaba Ölveczky. A timed ctl model checker for real-time maude. In *International Conference on Algebra and Coalgebra in Computer Science*, pages 334–339. Springer, 2013.

[73] Daniela Lepri, Erika Ábrahám, and Peter Csaba Ölveczky. Sound and complete timed ctl model checking of timed kripke structures and real-time rewrite theories. *Science of Computer Programming*, 99:128–192, 2015.

[74] Ann S Lofsky. Turn your alarms on. *APSF Newsletter: The Official Journal of the Anesthesia Patient Safety Foundation*, 19(4):43, 2005.

[75] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1):73–155, 1992.

[76] José Meseguer. Membership algebra as a logical framework for equational specification. In *International Workshop on Algebraic Development Techniques*, pages 18–61. Springer, 1997.

[77] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63 (2):81, 1956.

[78] Victor Mittelstädt and Jeff Miller. Separating limits on preparation versus online processing in multitasking paradigms: Evidence for resource models. *Journal of Experimental Psychology: Human Perception and Performance*, 43(1):89, 2017.

[79] Akira Miyake, Marcel Adam Just, and Patricia A Carpenter. Working memory constraints on the resolution of lexical ambiguity: Maintaining multiple interpretations in neutral contexts. *Journal of memory and language*, 33 (2):175, 1994.

[80] Stephen Monsell. Task switching. *Trends in cognitive sciences*, 7(3):134–140, 2003.

[81] Lucia Nasti and Paolo Milazzo. A hybrid automata model of social networking addiction. *Journal of Logical and Algebraic Methods in Programming*, 100: 215–229, 2018.

[82] Ulric Neisser. *Cognitive psychology: Classic edition*. Englewood Cliffs, NJ: Prentice Hall., 1967.

[83] Donald A Norman and Tim Shallice. Attention to action. In *Consciousness and self-regulation*, pages 1–18. Springer, 1986.

[84] Peter Csaba Ölveczky. Real-time maude and its applications. In *International Workshop on Rewriting Logic and its Applications*, pages 42–79. Springer, 2014.

[85] Peter Csaba Ölveczky and José Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285(2): 359–405, 2002.

[86] Peter Csaba Ölveczky and José Meseguer. Semantics and pragmatics of real-time maude. *Higher-order and symbolic computation*, 20(1-2):161–196, 2007.

[87] Harold Pashler. Dual-task interference in simple tasks: data and theory. *Psychological bulletin*, 116(2):220, 1994.

[88] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *Human-Computer Interaction INTERACT'97*, pages 362–369. Springer, 1997.

[89] Stephen J Payne, Geoffrey B Duggan, and Hansjörg Neth. Discretionary task interleaving: heuristics for time allocation in cognitive foraging. *Journal of Experimental Psychology: General*, 136(3):370, 2007.

[90] Thomas S Redick, Zach Shipstead, Matthew E Meier, Janelle J Montroy, Kenny L Hicks, Nash Unsworth, Michael J Kane, D Zachary Hambrick, and Randall W Engle. Cognitive predictors of a common multitasking ability: Contributions from working memory, attention control, and fluid intelligence. *Journal of Experimental Psychology: General*, 145(11):1473, 2016.

[91] Rimvydas Rukšėnas, Paul Curzon, Ann Blandford, and Jonathan Back. Combining human error verification and timing analysis: a case study on an infusion pump. *Formal Aspects of Computing*, 26(5):1033–1076, 2014.

[92] Dario D Salvucci. Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies*, 55(1):85–107, 2001.

[93] Dario D Salvucci. Multitasking. In *The Oxford handbook of cognitive engineering*. 2013.

[94] Dario D Salvucci and Peter Bogunovich. Multitasking and monotasking: the effects of mental workload on deferred task interruptions. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 85–88. ACM, 2010.

[95] Dario D Salvucci and Niels A Taatgen. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological review*, 115(1):101, 2008.

[96] Dario D Salvucci and Niels A Taatgen. Toward a unified view of cognitive control. *Topics in cognitive science*, 3(2):227–230, 2011.

[97] Paul C Schutte and Anna C Trujillo. Flight crew task management in non-normal situations. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 40, pages 244–248. SAGE Publications Sage CA: Los Angeles, CA, 1996.

[98] Oliver G Selfridge and Ulric Neisser. Pattern recognition by machine. *Scientific American*, 203(2):60–69, 1960.

[99] Thomas B Sheridan. On how often the supervisor should sample. *IEEE Transactions on systems science and cybernetics*, 6(2):140–145, 1970.

[100] Thomas B Sheridan, AF Kramer, DA Wiegmann, and A Kirlik. Attention and its allocation: Fragments of a model. *Attention: From theory to practice*, pages 16–26, 2007.

[101] Steven T Shorrock. Errors of memory in air traffic control. *Safety Science*, 43(8):571–588, 2005.

[102] Valerie J Shute. Who is likely to acquire programming skills? *Journal of educational Computing research*, 7(1):1–24, 1991.

[103] NS Sutherland. Outlines of a theory of visual pattern recognition in animals and man. *Proc. R. Soc. Lond. B*, 171(1024):297–317, 1968.

[104] JN Towse and GJ Hitch. Is there a relationship between task demand and storage space in tests of working memory capacity? *The Quarterly Journal of Experimental Psychology Section A*, 48(1):108–124, 1995.

[105] John N Towse, Graham J Hitch, and Una Hutton. A reevaluation of working memory capacity in children. *Journal of memory and language*, 39(2):195–217, 1998.

[106] John N Towse, Graham J Hitch, and Una Hutton. On the nature of the relationship between processing activity and item retention in children. *Journal of Experimental Child Psychology*, 82(2):156–184, 2002.

[107] Anne Treisman. Monitoring and storage of irrelevant messages in selective attention. *Journal of Memory and Language*, 3(6):449, 1964.

[108] Marilyn L Turner and Randall W Engle. Is working memory capacity task dependent? *Journal of memory and language*, 28(2):127–154, 1989.

[109] Gloria S Waters and David Caplan. The measurement of verbal working memory capacity and its relation to reading comprehension. *The Quarterly Journal of Experimental Psychology Section A*, 49(1):51–79, 1996.

[110] CD Wickens. Noticing events in the visual workplace: The seev and nseev models. *Handbook of applied perception*, 2014.

[111] Christopher D Wickens. Processing resources and attention. *Multiple-task performance*, 1991:3–34, 1991.

[112] Christopher D Wickens and Robert S Gutzwiller. The status of the strategic task overload model (stom) for predicting multi-task management. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 61, pages 757–761. SAGE Publications Sage CA: Los Angeles, CA, 2017.

[113] Christopher D Wickens and Jason S McCarley. Applied attention theory. 2008.

[114] Christopher D Wickens, Amy Santamaria, and Angelia Sebok. A computational model of task overload management and task switching. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 57, pages 763–767. SAGE Publications Sage CA: Los Angeles, CA, 2013.

[115] Christopher D Wickens, Robert S Gutzwiller, and Amy Santamaria. Discrete task switching in overload: A meta-analyses and a model. *International Journal of Human-Computer Studies*, 79:79–84, 2015.

[116] Christopher D Wickens, Juliana Goh, John Helleberg, William J Horrey, and Donald A Talleur. Attentional models of multitask pilot performance using advanced display technology. In *Human Error in Aviation*, pages 155–175. Routledge, 2017.

[117] Christopher Dow Wickens, Robert S Gutzwiller, Alex Vieane, Benjamin A Clegg, Angelia Sebok, and Jess Janes. Time sharing between robotics and process control: Validating a model of attention switching. *Human factors*, 58(2):322–343, 2016.

Acknowledgement

These years of PhD have been strenuous, demanding, gratifying, and satisfying. During such time I had the privilege to work with Paolo Milazzo, who taught me to do research and to love this work. Thank you for being such a great mentor, this thesis would be poorer without your support, your help and your suggestions.

I'd like to thank Antonio Cerone, for inspiring such thesis and for his precious suggestions. A great part of this work is fruit of the collaboration with Peter Csaba Ölveczky from the University of Oslo; thank you for your hospitality in the freezing Oslo and to have taught me how to use Maude. During my visit at the University of Braga I had the opportunity to work with Paolo Masci; thank you for the interest shown for my work, your suggestions and your incredibly hospitality. The visit at INRIA Saclay (Paris) let me knew Catuscia Palamidessi and Valentina Castiglioni, which have been truly kind and helpful. During my visit at the University of Edinburgh I had the pleasure to meet Vashti Galpin; thank you for your help and suggestions. I wouldn't have unable to complete this thesis without the work and the help of Carmen Berrocal Montiel, Cristina Belviso and Luca Vitrini; thank you all.

I'd like to thank Roberto Bruni and Fabio Paternò, the thesis committee members, and José Creissac Campos and Michael Harrison, the referees of this thesis, for their helpful comments and suggestions, the PhD coordinator Paolo Ferragina, and the Prof. Pierpaolo Degano, whose reproofs made this work possible.

My life here wouldn't had be the same without the support, help and laughter of my academic family: among others Rita "Donno" Pucci, Lucia (my little academic sister), Marco Ponza (to remind me every day I have tattoos), Marco Cornolti (I will never forgive you to have preferred Google to us), Tiziano and Daniele, Veronica, Ottavio, Andrea Marino, Andrea Michienzi, the "Brogi's boys". Thank you all!

Thank to my family for their continuous support, and to the family I chose for their unconditional love. Finally, thank to BC to be always by my side.