When a person is concurrently interacting with different devices, the amount of cognitive resources required (cognitive load) could be too high and might prevent some tasks from being completed. When such human multitasking involves safety-critical tasks, failure to devote sufficient attention to the different tasks could have serious consequences. To study this problem, we define a formal model of safety-critical human multitasking which describes the cognitive processes involved in human-computer interaction and the switching of attention among concurrent tasks. The model describes 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.

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 subtasks, which in turn are sequences of basic tasks which cannot be further decomposed. Each basic task is characterised by 3 parameters: its duration, its difficulty and its delay (namely the time which has to pass before the basic task is enabled). Such parameters are used to compute the cognitive load of the task (according to relevant psychological literature). Moreover, each task is characterised by a measure of how much it is critical.

The user performs the basic tasks on the 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. Therefore, for each task is computed an attention attraction factor (called α-factor), which measures the likelihood of the task to attract the user's attention. Such measure is computed as the multiplication of the task cognitive load, its criticality and the time it was ignored during the interaction. We then compute for each task a probability to be selected proportional to its α-factor.

We implement the model through a Java simulator, which allows 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. The simulator is a slightly simplified version of the formal model 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 the formal model, we thus have:

- a `BasicTask` class;
- a `Subtask` class;
- a `Task` class;
- a `WorkingMemory` class;
- a `Configuration` class where the state of the simulation is defined;
- a `Simulator` class where the algorithm for simulating selective attention is specified.

Although the proposed mechanism simulating the switching of attention between tasks is consistent with psychological literature and results from experimental psychological studies,

we validate the model against data gathered from an experimental study involved in a multitasking interaction on a web application with two concurrent tasks: one representing a “main” critical task, the other representing a “distractor” task with different levels of cognitive load.

We defined two separated tests: one for evaluating the working memory performances of the participants, and one called shared attention test where users were asked to interact with two tasks concurrently.
As regards the shared attention test, we model two tasks: a main critical one and a secondary distracting task.
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 central 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.
In the secondary distracting task, 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 an 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 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 secondary task is instantiated with different levels of cognitive load by varying the number of letters to insert, the number of keys on the keyboard and the total time to perform the task.

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.

Observing data we can identify 6 different typologies of users divided into 6 different groups, according to their working memory performance (i.e. their PCU), and according to how they perceive the criticality of the main task:

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

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

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.

As regards the secondary task, as explained above, a letter appears inside the white box 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:

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

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 deducted from data;
• 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:

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

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, 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 deducted from data. The criticality of each task is set to 0.1.