5
Drawing the behavior

To define the behavior of the prototype, the designer must think about the event associated to the behavior and which element triggers it, the conditions under which the behavior will happen and which actions will be executed. This **ECA** (event, conditions and actions) represents a single case of the element's behavior, since another set of conditions can trigger a different set of actions. This chapter explains both the ECA logic (Section 5.1) and how to define an ECA in UISKEI (Section 5.2).

5.1
ECA

To define the interface behavior, we needed a language to express how to react to elements' events (e.g. mouse clicks, text input), empowering the designer to choose what to do when the interface is at a given state. We found that ECA (Event Condition Action) languages could be an intuitive and powerful paradigm to this situation (Alferes, Banti, & Brogi, 2006). They are used in many applications — such as in active databases, workflow management, network management, etc. Also, they have a declarative syntax being more easily analyzed when compared to implementing with a programming language (Bailey, Poulovassilis, & Wood, 2002).

In UISKEI, an ECA represents a behavior case, following the idea of "**when** an <event> happens, **if** all <conditions> are satisfied, **then** the <actions> take place". It has a name that the user may enter to make the association to the behavior case defined clearer (the default name is "<element associated to event> - ECA <number>", the event which will trigger it, a list of conditions and a list of actions. All the ECAs are shown in the **ECAMan** sidebar (ECAs Manager), allowing the user to rapidly see them, as shown in Figure 21.

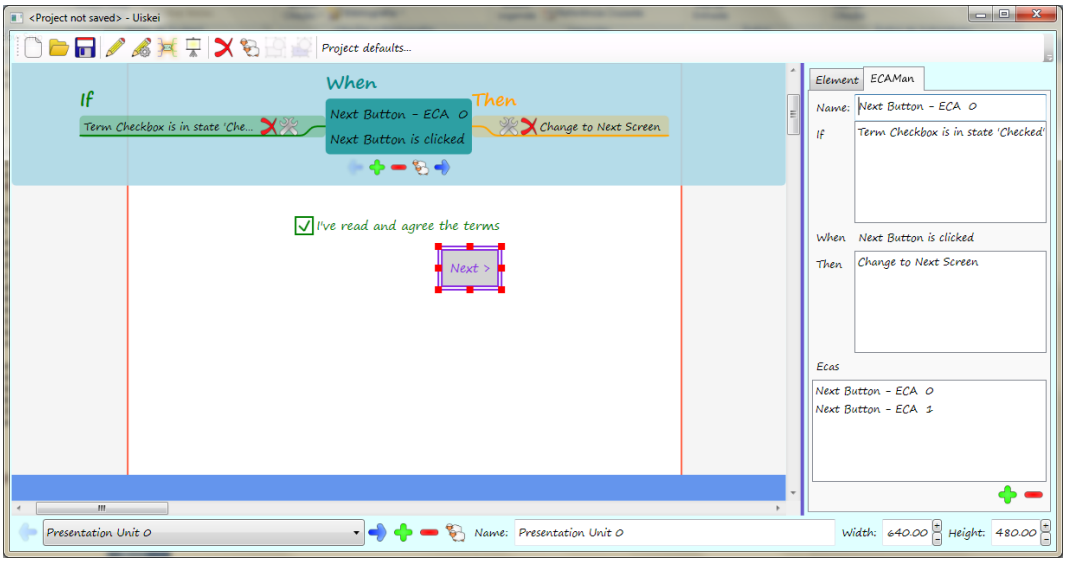

Figure 21: ECAMan interface.

As can be seen in the right pane, the ECAMan sidebar shows all ECAs in the project (lower list) and gives the details of the current selected ECA, allowing users to edit the ECA name and see the list of conditions and actions. The events will be discussed in the following section, while conditions will be discussed in Section 5.1.2, and the actions in Section 5.1.3. In Section 5.1.4, the concept of a "valid ECA" is introduced.

5.1.1. Events (When)

An **`event`** is the action which may trigger an `ECA`. Each element has its own pre-defined events. While buttons, checkboxes and radio buttons can handle the "`Clicked`" event, textboxes can handle the "`Text Changed`" event and the dropdown lists can handle the "`Selection Changed`" event. For now, an element can only have one `event`, but future work may overcome this limitation.

5.1.2. Conditions (If)

`Conditions` are expressions that must be satisfied in order to trigger an `ECA`. The conditions currently supported by UISKEI are related to the status of an element and are described below:

- Element conditions
  - If the element is in state `<state name>`

- If the element is <visible / invisible>
- If the element is <enabled / disabled>

An ECA's set of conditions has an internal operator of "AND", so all conditions must be met to activate the ECA. The "OR" operator can be achieved by defining a new ECA. Ideally, the ECAs of an element should be mutually exclusive, since if two or more of them can be activated in a given situation, only the first one in the list will do so. Therefore, the ECAs' order in the ECAMan is important to the simulation.

5.1.3. Actions (Do)

An **action** is an operation that will be performed if activated, changing the simulation state. While conditions may only refer to elements, an action can refer to an element, to a presentation unit or to a default kind of message. The operations available to each group can be seen in the following list:

- Element actions
  - Change element to state <state name>
  - Make the element <visible / invisible>
  - Toggle the element's visibility status
  - Make the element <enabled / disabled>
  - Toggle the element's enabled status
- Presentation unit actions
  - <Change to / Pop up / Modal pop up> the presentation unit
- Default message actions
  - Show a <information / warning / error> message with <text>

When an ECA event is triggered, if the set of conditions is satisfied, all of its actions are executed.

5.1.4. Valid ECAs

When an element is removed from the project, all ECAs triggered by the element's event are also removed. However, it could also be associated to conditions or actions of other ECAs, but the remaining of these ECAs could still be valid. The same issue may happen to removed presentation units and removed states. To avoid greater impact, such as also removing the ECAs with the changed conditions or actions, the concept of ECA validity was created.

When a condition or action references a removed object, it becomes invalid, invalidating the ECA that contains it. An invalid ECA continues in the ECAMan, so the user can later make changes to make it valid again, but it is ignored during simulation, to avoid undesired or unplanned behavior.

5.2 Drawing ECAs

As could be seen in Figure 13, the first version of UISKEI had a form-based approach to handle ECAs, forcing users to switch the interaction paradigm between the interface design mode (pen-based) and the interaction design mode (form-based). One of the new version's biggest challenges was making this step more adequate to pen-based interaction.

The first idea was based on the DENIM approach of navigation between pages, by connecting lines between an element and a page. This is effective when there is only one kind of behavior (when <element> is clicked, go to <page>) and two parameters (the beginning of the line defines the anchor <element> and the end, the target <page>. UISKEI, however, offers 3 different kinds of actions, and both conditions and actions have additional parameters to be defined, so a more complex approach was necessary.

The proposed solution was to create a specific mode of interaction, Eca Mode (as opposed to the Drawing Mode / Recognition Mode used to create the interface). While in this mode, the user must focus on an element – which will be the one that raises the ECA's event – and add an ECA to it. The manipulation of ECA's (adding, removing, duplicating, activating the next or the

previous ECA) is done by clicking buttons that appear on the canvas when an element is focused, as shown below:

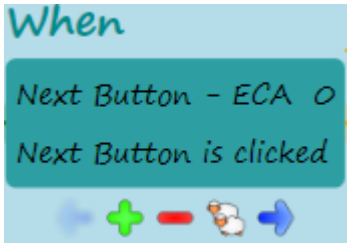

Figure 22: ECA buttons.

Conditions and actions can be added to the active ECA by drawing lines, with the start and the end point determining what is being added. The first idea was simple: if the line began on the focused element (the element associated with the ECA event), it was a condition, if it ended on it, an action. However, this solution demanded too much effort from the user, since it was necessary to draw long lines; was much too complex if an element of a different presentation unit was involved and made it impossible to have an action or condition associated to the focused element itself, since the start and end points would always be in the element, making the user's goal unknown. Another solution was proposed, following the table below:

Table 3: Condition and action creation.

Start point	End Point	Creates
Element	Anywhere	Element condition
Canvas	Element	Element action
	Presentation Unit	Presentation unit action
	None of the above	Default message action

Analyzing the start and end points we can only define which "higher level" type of operation is being added, i.e., if it is an element condition, an element action, a presentation unit action or a default message action. The specific operation and its parameters are still undefined. For example, if an element condition is being created, we do not know if it is one of kind "is in state <state name>" or "is <visible / invisible>" with only the start and end points. So, we need to ask the user what operation he/she wants (e.g. is in state <>) and the parameters (e.g. <state name>).

To make the process more fluent and pen-like, it was decided to use a pie menu approach, since it reduces target seek time, "lowering error rates by fixing the distance factor and increasing the target size in Fitts's Law" (Callahan, Hopkins, Weiser, & Shneiderman, 1988). The pie menu pops up once the pen stops for a given amount of time, determining the end point of the line. Without raising the pen, the user chooses the last parameters of the condition or the action being created. Therefore, one single stroke can define a condition or an action, as can be seen in the following image:

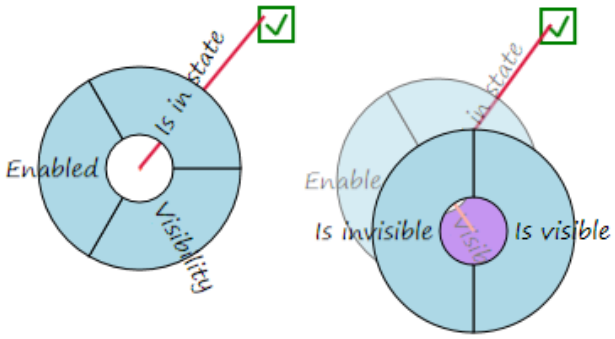

Figure 23: Pie menu allows a single stroke to define all parameters.

The default message action is defined with only one pie menu, but an additional dialog is necessary to define the text that will be shown.

Every element condition and action is defined through a two-level pie menu. Since an element can have a variable number of states, the choice of state was postponed to the second level of the pie menu. This decision was made in order to guarantee that the first level of the pie menu would always look the same, making it more recognizable to the user and, thus, more efficient. The visibility and activity status, although having a fixed set of operations, has different numbers of operations depending if it is a condition (2 options: is visible / is invisible, is enabled / is disabled) or an action (3 options: make visible / make invisible / toggle, make enabled / make disabled / toggle). So, to keep the menu coherent and make the first level the same between conditions and action, the options were organized in the second level. The final pie menu hierarchy when creating a condition or an action can be seen in the following figure.

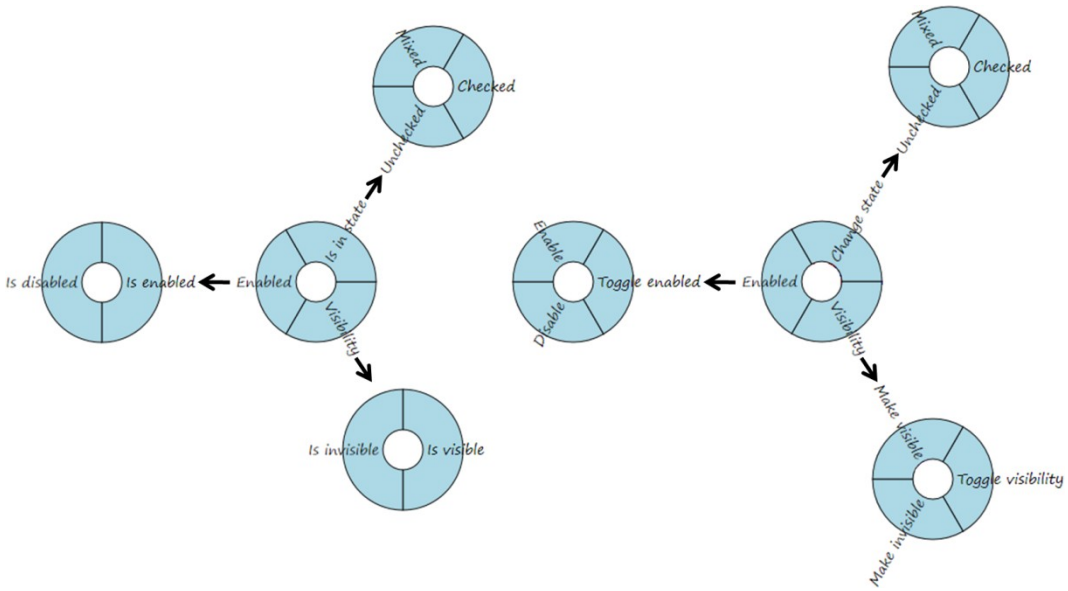

Figure 24: Element condition (left) and action (right) pie menus.

The presentation unit action is defined with only one pie menu, but the problem was that the list of presentation units didn't appear in the canvas, being a drop-down list in the lower bar of the window. A first idea was to follow the DENIM approach and have multiple zoom levels, but this would mean that the user should have a two-dimensional space awareness of where his/hers presentation units were. To make it similar to the dropdown list available off the canvas, a one-dimensional filmstrip was added to the canvas, showing a thumbnail of every presentation unit and allowing the user to scroll among them. The presentation unit filmstrip can be seen in Figure 25, along with the presentation unit action pie menu.

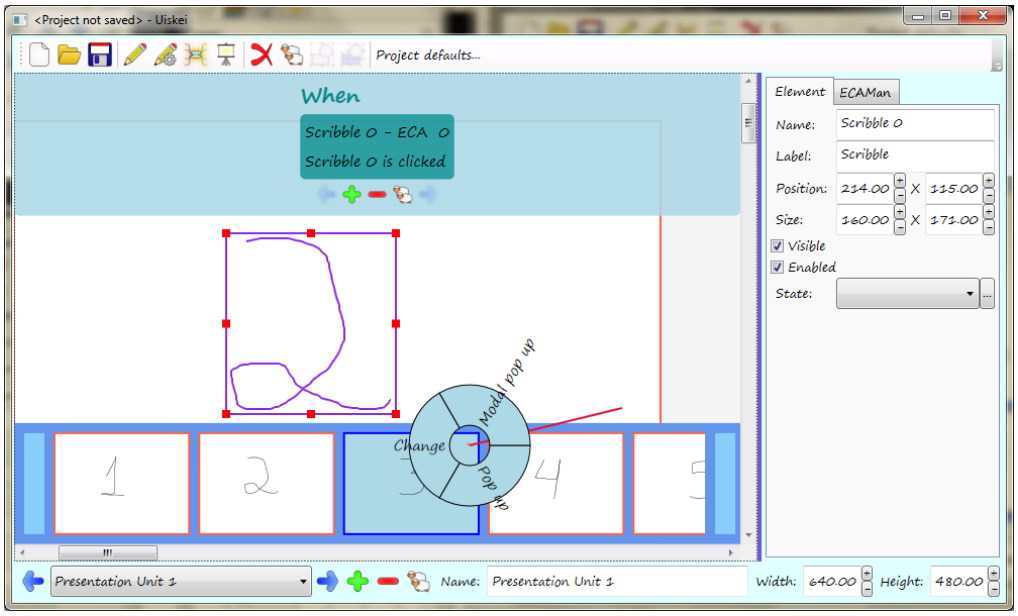

Figure 25: Presentation unit filmstrip.

The conditions and actions were added, but how should they be represented on the canvas? DENIM showed that multiple conditions could lead to a combinatorial explosion, as can be seen in Figure 26 taken from (Lin, Thomsen, & Landay, 2002). Note that there are multiple copies of a page, each one representing a combination of its possible states. Also, the usage of the same representation to indicate either interaction or navigation adds to the confusion and may further hinder the design process.

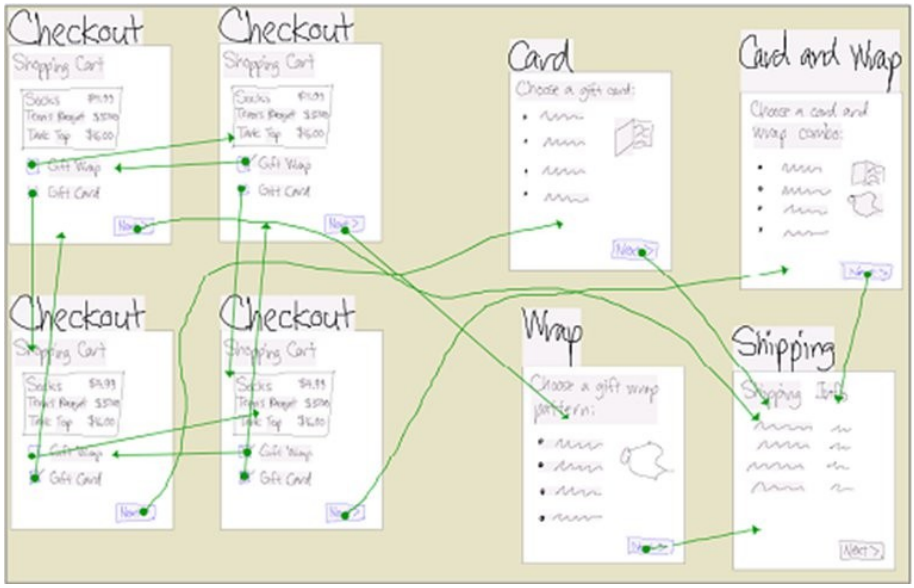

Figure 26: DENIM combinatorial explosion.

The first decision to overcome this problem was to show only one ECA at a time, avoiding screen pollution with the cost of losing sight of the "big picture". The initial idea was to display the same lines used in the creation of conditions and actions, but this idea was abandoned. First, because it would also make the screen polluted, as there could be many lines, connecting different presentation units or distant elements. Also because a single element could be associated with different conditions and actions, so it was necessary to make a distinction between them.

Instead of lines connecting objects, the proposed solution uses a mind-map-like representation. Reflecting the "if / when / do" of an ECA, the "mind-map" shows a block of connections on the left to represent the conditions and a block of connections on the right for the actions. In the middle, the "when" block, displaying the active ECA and its associated event. By doing so, it is possible to see all the ECA's conditions and actions in an ordered and centered manner, making it easier to view, comprehend and edit.

Originally, these blocks appeared over the focused element, but this occluded the elements behind it, making it difficult or even impossible to associate conditions or actions to the occluded elements. So, a special area is now reserved for the active ECA on the top of the canvas, making it always visible even when the canvas is scrolled.

The approach of drawing lines to define conditions and actions raised an issue: while in ECA mode, the user should be able to manipulate (move and resize) the focused element and to make it part of a condition or an action. If the user clicks inside the selected element and drags it, the element should move. Since the language to create an element condition is to "draw a line from the element to the canvas" and this results in moving the element, the solution was to use the "when" block as a hotspot for the focused element. So, if he/she clicks inside the "when" block and drags to its outside, a condition related to the selected element is created, thus solving this impasse.

Finally, to give users a feedback of which objects are related to the active ECA, they are painted with a color code to specify if they are related to a condition (green), an action (orange) or both (brown), as shown in Figure 27. The command button is in green (indicating it is part of a condition), the radio button and the target presentation unit are in orange (indicating they are part of an action), and the checkbox is in brown (indicating that it is both part of a condition and part of an action).

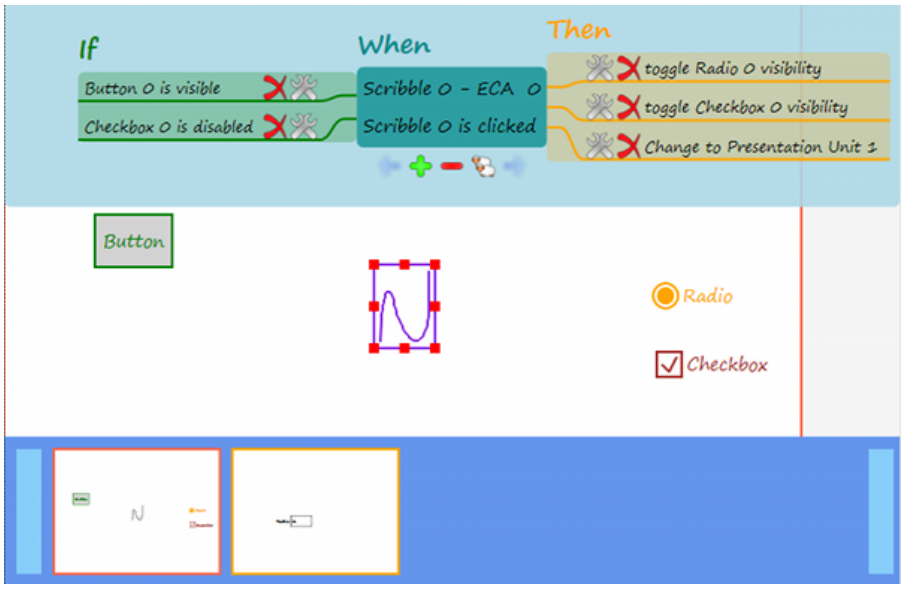

Figure 27: ECA mind-map-like representation.