9
Conclusion and Future work

UISKEI is a promising "paperless" user interface prototyping application. It explores the pen input paradigm to allow the user to draw the interface elements and their behaviors. Besides the drawing component, the behavior definition also features the power of conditionals and actions that go beyond the navigational scope. This gives to the designer the possibility to easily create an interactive prototype that he/she can later evaluate with an end user and then gather invaluable feedback.

The interface building step relies on the drawing recognition and the evolution of elements. The recognition uses the Douglas-Peucker simplification algorithm and the Levenshtein string edit distance, which obtained good recognition results and is a promising technique for customizable sketch recognition.

The interface behavior is defined in an innovative way, expressing the behavior using an ECA language and visualizing it with a mind-map like representation. The conditions and actions are added by drawing lines on the canvas and defining the parameters with the aid of a pie menu, allowing the user to define them with a single pen stroke.

The evaluation study also showed that the participants had a very positive opinion about UISKEI, giving higher grades to it than the other tools. UISKEI's overall final score was 4.5, while Balsamiq's was 3.2 and Paper's was 3.4. Not only the average score was one point higher than the other tools, but also the participants were more likely to give higher grades to UISKEI in the final cycle (with more complex interactions), since the standard deviation of the UISKEI's scores was 0.65 and for the other tools, it was 1.51.

Besides all the good feedback, UISKEI is still a work in progress. Comparing UISKEI to the related work described in Chapter 2, we can say that, while the core functionality of the program is already available, some basic features, such as undo/redo and cut/copy/paste (only duplicate is available), are

missing. Advanced functionalities, like smart guidelines to aid in the drawing phase and smart grouping, would also be great add-ons. The complete comparison table, now with UISKEI, can be seen in Table 8, where the lines in bold highlights the features that were the focus of this dissertation.

Table 8: UISKEI comparison with related software.

	Microsoft Visio	Axure RP Pro	Denim	SketchiXML	Balsamiq	CogTool	UISKEI
Free	n	n	y	y	n	y	y
UI-prototype exclusive (vs generic diagrammatic tool)	n	y	y	y	y	n	y
UI components for multiple environments (vs web-page prototype only)	y	y	n	y	y	y	y
Drawing widgets	**n**	**n**	**n**	**y**	**n**	**n**	**y**
>> Evolution of widgets	**x**	**x**	**x**	**n**	**x**	**x**	**y**
Element manipulation	**y**		**n**	**n**	**y**	**y**	**y**
Undo/Redo	y		y	y	y	y	n
Group/Ungroup	y		y	n	y	n	y
>> Select internal objects	y		x	x	n	x	n
Cut/Copy/Paste	y		y	y	y	y	n
>> Copies the action	n		n	n	y	n	x
Zoom levels	y		y	y	y	y	n
Guidelines	n		n	n	y	n	n
Layer ordering	y		n	n	y	y	n
Sketchy visual	n	n	y	y	y	y	y
Actions	**n**	**y**	**y**	**y**	**y**	**y**	**y**
>> Beyond navigation	**x**	**y**	**n**	**n**	**n**	**n**	**y**
>> Sketchy interaction	**x**	**n**	**y**	**y**	**n**	**y**	**y**
>> Event	**x**	**y**	**y**	**n**	**n**	**y**	**y**
>> Conditions	**x**	**n**	**y**	**n**	**n**	**n**	**y**
Prototype evaluation	**x**	**y**	**y**	**n**	**y**	**y**	**y**
Save	y	y	y	n	y	y	y

Regarding the implemented features, some features may be further developed. A brief discussion of them will follow in the next sections, organized by topic.

9.1 Shapes and element descriptors

At the moment, the shape descriptor only considers shapes with a single segment. Shapes with multiple segments would greatly enhance the possibility of having a larger element set.

UISKEI could also have more than one library of shapes and/or elements, so that the user can choose which ones to use and customize the element recognition. This would allow for greater software flexibility, for example, having a library for a desktop application prototype and another for a mobile application. However, if the system becomes highly customizable, a tool to check for consistency (such as not having two shapes being recognized in the same way, elements that cannot be created due to the lack of an ancestor element or required shape or even because another element is created instead of given a condition) will be needed.

9.2 Recognizer

The eight directions limit can be changed to consider more directions, but the impact in the shape descriptor and in the recognition rate should be evaluated. Also, still considering the string edit distance basis, other implementations can be tried, such as the ones described in (Schimke & Vielhauer, 2007). For example, (Coyette, Schimke, Vanderdonckt, & Vielhauer, 2007) uses a recognition method that, instead of using the Douglas-Peucker simplification algorithm, uses a uniform grid approximation approach.

If the recognition rate is still unsatisfactory, other approaches to recognition can be tried, such as using a neural network. Since recognition tends to be ambiguous and error-prone, the application could give the users feedback about the recognized shape as it is being drawn, allowing the users to make necessary changes while drawing (Tandler & Prante, 2001), or at least give the possibility to

Also, a beautification approach of user's rough sketches can be implemented, such as the one proposed in (Paulson & Hammond, 2008).

9.3 ECA

Since the `ECAs` are evaluated in the order they appear in `ECAMan`, it is necessary to give the user the option to change this order. Also, to avoid the repetition of `ECAs`, a way to express `AND` and `OR` clauses in conditionals could be

looked into. Another improvement could be to trigger more than one ECA at a time, and the visualization of such possibility should be considered.

The limitation of an element having only one possible event should be reconsidered, since some elements may have more than one event. For example, a spinbox can have its value changed by the user clicking on the up or down button or even typing the value. The `ECAs` should be able to handle multiple events, giving the option of choosing a specific event (e.g., `clicked` or `text entered`) or a more generic one (`value changed`, for example).

Finally, for bigger projects, keeping track of `ECAs` can be difficult, so a tool to validate them and search for unreachable `ECAs` (because of errors in the conditions set or because another `ECA` is always activated first) or duplicated ones may be necessary.

9.4 Prototype evaluation

The prototype evaluation could take advantage of the Wizard of Oz technique, with the final user executing the simulation in a computer and a member of the designing team accompanying the session in another computer. With this, the end user can have more freedom or be more constrained, since the designer can mediate his/her interaction (Dahlbäck, Jönsson, & Ahrenberg, 1993).

Besides the usual logging possibility, this would also allow for a definition on-the-fly of presentation units, elements and `ECAs`, which could be reused in future tests.