Automated Usability Testing Using HUI Analyzer

Simon Baker, Fiora Au
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand
{sbak030, fau001} @ ec.auckland.ac.nz

Gillian Dobbie, Ian Warren
Department of Computer Science
University of Auckland
Auckland, New Zealand
{gill, ian-w} @ cs.auckland.ac.nz

Abstract

In this paper, we present an overview of HUI Analyzer, a tool intended for automating usability testing. The tool allows a user interface's expected and actual use to be captured unobtrusively, with any mismatches indicating potential usability problems being highlighted. HUI Analyzer also supports specification and checking of assertions governing a user interface's layout and actual user interaction. Assertions offer a low cost means of detecting usability defects and are intended to be checked iteratively during a user interface's development. Hotspot analysis is a feature that highlights the relative use of GUI components in a form. This is useful in informing form layout, for example to collocate heavily used components thereby reducing unnecessary scrolling or movement. Based on evaluation, we have found HUI Analyzer's performance in detecting usability defects to be comparable to conventional formal user testing. However the time taken by HUI Analyzer to automatically process and analyze user interactions is much less than that for formal user testing.

1. Introduction

Usability is universally considered to be an important quality attribute of software. This is because the success of a software product is heavily dependent on its usability. In many cases, however, usability remains divorced from "core" software development activities, partly because of the lack of cost-effective tool support to develop and validate a system's usability. The result of treating usability as a second class activity is software with compromised usability.

The International Organization for Standardization defines usability as "*The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*" [1]. In other words, how easy it is for a user to meet their goals in using the software for a particular function. Software usability involves the following factors [2,3,4]:

- *Learnability* How easy it is for a user to gain proficiency with the software interface.
- *Understandability* How easy it is for the user to interpret and comprehend the interface.
- *Efficiency* How productive an experienced user is with the interface.
- *Memorability* How easy it is for the user to regain proficiency with the software after not using it for a reasonable period of time.
- *Satisfaction* How comfortable the user is with the software.

Developing software that is usable is difficult because the above factors are subjective in nature. They are also dependent on other factors, such as hardware constraints, the type of application being developed, and the target user. In tackling these difficulties, a general consensus has emerged that involving users in the development and testing of an application's usability leads to improved usability. In practice, however, this approach is not always viable because of the costs and overheads from involving users [2,4].

In this paper, we present an overview of HUI (Handheld User Interface) Analyzer, a tool that has been developed to find usability defects and which automates much of the gathering, analysis and reporting of usability data. The tool was originally intended to examine GUIs that run on handheld devices, but HUI Analyzer is applicable to more general GUIs. HUI Analyzer offers design-time support by evaluating static properties of an application's GUI according to a configurable set of usability heuristics. The tool also automates usability testing by reducing much of the overhead associated with user testing. HUI Analyzer retains end-user involvement as a means of determining usability, thus retaining the value from involving users in usability testing, but does so at a reduced cost than that associated with formal user testing.

This paper is structured as follows. In Section 2 we overview established techniques concerned with finding usability defects. In Section 3, we highlight a number of existing tools that aim to automate some aspect of usability testing. We proceed in Section 4 by introducing HUI

1530-0803/08 $25.00 © 2008 IEEE
DOI 10.1109/ASWEC.2008.40

Analyzer and its functionality for finding different classes of usability defects. In Section 5, we present further details of EAS semantics, the means for developers to specify how they expect their application to be used. We continue in Section 6 by describing the tool's architecture and in Section 7 we report on an evaluation of HUI Analyzer. Finally, we draw conclusions in Section 8.

2. Usability evaluation

Established approaches to evaluating an application's usability include performance measurement [5], heuristic evaluation [6] and formal user testing [2,7].

2.1 Performance measurement

Performance measurement is a technique for evaluating usability based on measuring certain parameters and comparing them with a mathematical model that aims to minimize or maximize variables such as distance, and graph depth [5].

Being quantitative, performance measurement is inappropriate for evaluating some properties of a user interface, such as aesthetics. For example, color contrast ratio is difficult to evaluate whereas the distribution of GUI components on a form is better suited to performance measurement. Furthermore, when using performance evaluation, the context of measured values must be taken into account. For example, in carrying out a particular task a user might click the mouse some number of times; whether or not this value is too high depends on the task performed.

Given the limitations of performance measurement, studies have reported that this technique has a relatively low hit rate in detecting usability problems. Consequently, performance measurement is often supplemented with other usability evaluation methods.

2.2 Heuristic evaluation

Heuristic evaluation involves critiquing an application's user interface according to a set of usability guidelines [6], some of which are abstract and leave much room for interpretation. Neilsen's 10 Heuristics, Shneiderman's 8 Golden Rules and Norman's 7 Principles are examples of such guidelines and focus on producing interfaces with minimalist design [2]. Other guidelines are much more detailed; some are based on specific platforms and offer implementation level suggestions [3,8].

Heuristic evaluations are generally performed by usability experts, many of whom have a strong background in cognitive psychology and ethnographic sciences. Much of the approach evaluators take is subjective and based on their own personal experience.

Studies have shown that heuristic evaluation tends to have a 50% hit rate in detecting usability problems [2]. However, this technique is also associated with a high hit rate of false positives – falsely diagnosed usability problems.

2.3 Formal user testing

Formal user testing requires the participation and observation of real-life candidates within a controlled environment. The candidates perform a preset and well designed set of tasks that aim to exploit usability issues under normal usage. The performance of the candidates is analyzed, sometimes in a similar way to a performance measurement type of evaluation, with the exception that the context of their action is taken into account [2,7]. Some of the data analyzed include the time taken to perform tasks, number of clicks made, number of errors encountered, any back tracking to previous states and abandonment of tasks [2,5,7].

Sometimes, as part of usability testing, the candidates are prompted for open-ended information through opinion polls or post-test questionnaires in order to obtain information regarding subjective areas of usability such as aesthetics and user satisfaction [2,7]. In general, usability testing is the most comprehensive method of usability evaluation as it approaches usability from the user's perspective [2,7].

Although a formally inducted usability test usually attains the highest hit rate of "real" usability problems, a lot of time and money must be invested in selecting appropriate candidates, simulating the right test environment, setting the appropriate software environment and, notably, collecting and interpreting test data. This overhead often renders this technique too expensive to be viable.

Beyond cost, human psychology and ethical sociology can be matters of concern when carrying out formal user testing. For example, candidates may act differently when they are aware that their actions are being monitored and recorded. Such changes in behavior can compromise the value obtained from this form of usability testing.

3. Tools for usability testing

Tools that automate some aspect of usability testing have attracted much interest in recent years. Such tools have typically been developed for desktop applications and web-based user interfaces. The tools generally fall into two categories: *guideline checkers* and *interaction analyzers*.

Guideline checkers are a family of tools that focus on critiquing a user interface with respect to usability guidelines. Some of these tools are used on finished products, while others are intended for use during product development.

WebMetro is one example of a contemporary tool that is aimed at website designers. WebMetro allows a designer to input usability guidelines and have the tool use them to automatically evaluate a website. WebMetro allows the designer to specify the guidelines at multiple levels of detail [9].

Interaction analyzers are tools that focus on user interaction with an interface, by either recording or simulating user actions. SimUI [10] is one such tool that consists of a recording module, a modification module, a playback and data-gathering module, and an analyzer. Data-gathering and recording involves collecting status information from the application being run in addition to user interactions. The analyzer performs multi-step matching of different user data and detects any difference; this is later interpreted by human observers to detect usability issues. SimUI uses an inter-process messaging mechanism that is transparent to the user and application under test.

MRP (Maximal Repeating Pattern) analyzer [11] is another interaction analyzer. This tool works by inputting a user session transcript as a character string where each character corresponds to a command. The tool then detects the maximal repeating pattern in a string. Essentially, the pattern represents repetitive actions that the user makes. The string is then mapped back to the application context to determine if repetition indicates usability problems.

We refer the interested reader to Hilbert and Redmiles' survey [12] of earlier interaction analyzer tools that were developed to extract and process user interface events.

4. HUI Analyzer

HUI Analyzer is a tool that was originally developed to automate usability testing of graphical user interfaces designed for handheld devices. The tool aids usability testing by inducting various types of analysis, each targeting a subset of usability problems.

Prior to describing the different kinds of analysis supported by the tool, we outline the basic goals the tool was designed to meet:

- *Non intrusive to users*. During usability testing, the tool's operation should be transparent to test candidates.
- *Provision for static analysis*. The tool should detect usability problems relating to an interface's static properties, such as its composition from controls, spacing between components, use of color and fonts.
- *Provision for dynamic analysis*. Static analysis should be complemented with support for detecting differences in the way an interface is *intended* to be used from the way it is *actually* used to perform specified tasks.
- *Process model independence*. The tool should not be specific to any particular software development process, for example extreme programming.
- *Analogous to unit testing*. While process independent, the tool should offer a familiar model to unit testing. It should be possible to use the tool, at minimal cost, to incrementally analyze an interface during its development.
- *Extensibility*. The range of analysis checks performed by the tool should be extensible; furthermore the tool should be easily integrated with other tools.
- *Application independent*. HUI Analyzer should run on any application developed using Microsoft's .NET Compact Framework, without requiring modification.

HUI Analyzer accepts three inputs:

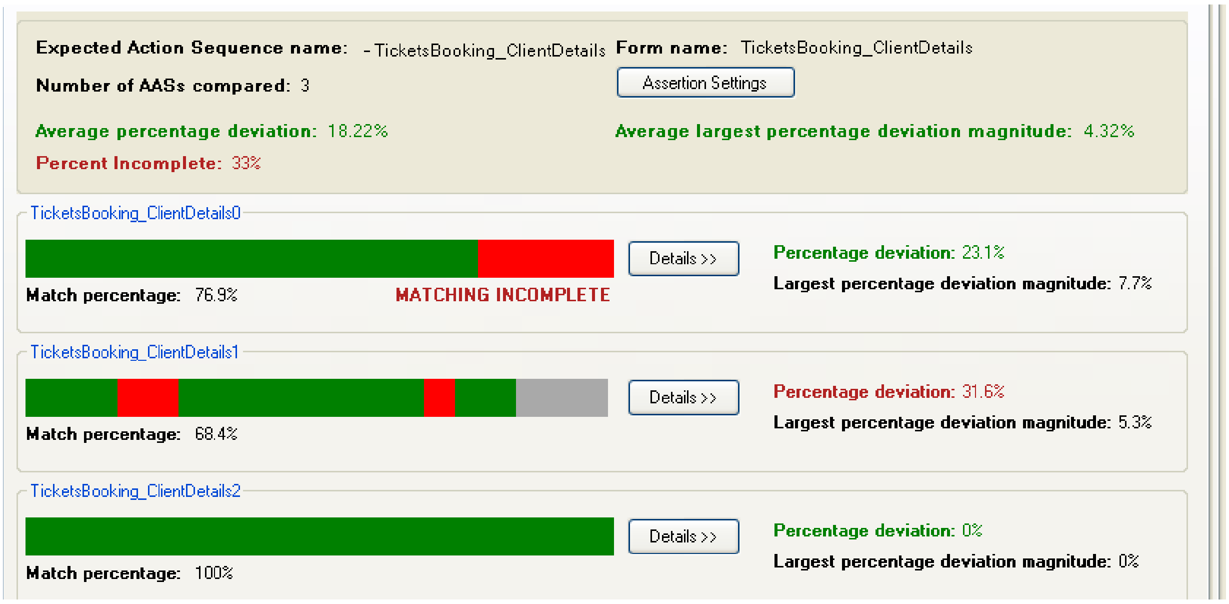
1. The compiled assembly for the GUI software developed using the .NET compact framework.
2. A description of the interaction that a developer *expects* of a user in performing a specified task. This is known as an EAS (Expected Action Sequence).
3. A description of the *actual* interaction that resulted from an independent user carrying out the task described by the corresponding EAS. The description of actual interaction is referred to as an AAS (Actual Action Sequence).

An AAS logs information about the type of each action, e.g. a mouse click, text input, or list item selection, the GUI component to which the action applies, the state of the component, and a timestamp.

Action sequences have been used to represent user interface events in much of the earlier work concerned with usability testing. In particular, Expectation Agents [13] and EDEM [13] are techniques that are based on event sequences and which identify patterns and differences in expected and actual use. HUI Analyzer's EAS/AAS representation and associated processing, described more fully in Section 5, are heavily influenced by these earlier works.

HUI Analyzer provides an EAS recorder for preparing EAS descriptions. The recorder can be used to simply record actions (interactions) made on a GUI that represent the sequence of actions a developer expects users to follow in completing a task. The EAS recorder also allows EAS descriptions to be constructed by dragging and dropping EAS constructs onto a script. The latter approach is more suited to developing non-trivial descriptions, for example those involving repetition of actions, mandatory/optional actions, and cases where one of many possible actions is required. The grammar for expressing an EAS is thus suitably rich, being discussed further in Section 5.

Drawing on the three inputs, HUI Analyzer performs three types of analysis, each focusing on distinct areas of usability: comparison checking of EAS and AAS descriptions, assertion checking, and hotspot analysis.



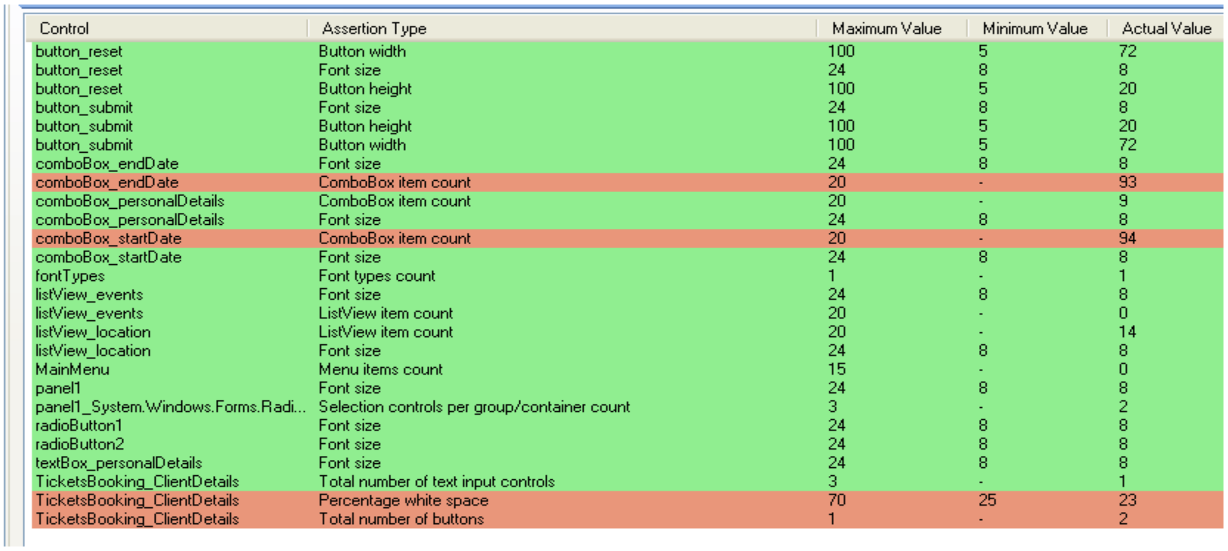
Expected Action Sequence name: - TicketsBooking_ClientDetails
Form name: TicketsBooking_ClientDetails
Number of AASs compared: 3
Assertion Settings
Average percentage deviation: 18.22%
Average largest percentage deviation magnitude: 4.32%
Percent Incomplete: 33%

TicketsBooking_ClientDetails0
Details >>
Match percentage: 76.9% MATCHING INCOMPLETE
Percentage deviation: 23.1%
Largest percentage deviation magnitude: 7.7%

TicketsBooking_ClientDetails1
Details >>
Match percentage: 68.4%
Percentage deviation: 31.6%
Largest percentage deviation magnitude: 5.3%

TicketsBooking_ClientDetails2
Details >>
Match percentage: 100%
Percentage deviation: 0%
Largest percentage deviation magnitude: 0%

Figure 1 Multiple AAS matching to a single EAS



Control	Assertion Type	Maximum Value	Minimum Value	Actual Value
button_reset	Button width	100	5	72
button_reset	Font size	24	8	8
button_reset	Button height	100	5	20
button_submit	Font size	24	8	8
button_submit	Button height	100	5	20
button_submit	Button width	100	5	72
comboBox_endDate	Font size	24	8	8
comboBox_endDate	ComboBox item count	20	-	93
comboBox_personalDetails	ComboBox item count	20	-	9
comboBox_personalDetails	Font size	24	8	8
comboBox_startDate	ComboBox item count	20	-	94
comboBox_startDate	Font size	24	8	8
fontTypes	Font types count	1	-	1
listView_events	Font size	24	8	8
listView_events	ListView item count	20	-	0
listView_location	ListView item count	20	-	14
listView_location	Font size	24	8	8
MainMenu	Menu items count	15	-	0
panel1	Font size	24	8	8
panel1_System.Windows.Forms.Radi...	Selection controls per group/container count	3	-	2
radioButton1	Font size	24	8	8
radioButton2	Font size	24	8	8
textBox_personalDetails	Font size	24	8	8
TicketsBooking_ClientDetails	Total number of text input controls	3	-	1
TicketsBooking_ClientDetails	Percentage white space	70	25	23
TicketsBooking_ClientDetails	Total number of buttons	1	-	2

Figure 2 Assertion processing output

4.1 Comparison checking

Comparison analysis indicates the extent of similarity between how a developer thinks an application's functionality should be accessed, as expressed in an EAS description, and how a user actually accesses it, described by an AAS. This analysis is performed by comparing EAS and AAS descriptions and indicating any deviations between them. These deviations can provide valuable clues as to how the user interprets the interface and what kind of errors or mistakes the user makes.

To illustrate comparison checking, let action sequences be represented by letter sequences:

EAS =	A	A	A	B	C	D						
AAS =	A	A	A	C	D	C	D	B	C	D	E	C
Resultant =	✓	✓	✓	✖	✖	✖	✖	✓	✓	✓	-	-

✓ Correct match ✖ Deviation - Ignore

The Resultant sequence shows the matching between the EAS and the AAS. If an action in the AAS is not the same as what is expected in the EAS then it is considered a deviation. For example, the fourth action expected in the EAS is a 'B' but the AAS contains a 'C'. If all of the EAS actions are matched then the comparison is considered complete and any remaining AAS actions are ignored ('E' and 'C' in the example above). A comparison is considered incomplete if the EAS is not entirely matched; this can happen if the EAS is impossible to achieve, or if there is a major accessibility problem with the interface.

The criterion for an expected action to be matched to an actual action is that all of the expected action attributes must equal that of the actual action. In other words, the event, component (name and type), and the component's state must be lexographically identical to that of the actual action. To cater for unpredictable yet valid attribute values, for example the actual text entered into a text field where the text constitutes the state of the text field component, an EAS can be prepared that generalizes actions. Such abstract actions are discussed further in Section 5.

HUI Analyzer allows multiple AAS descriptions to be compared with a single EAS, as shown in Figure 1. This is particularly useful when many users undertake the same task; making it possible to contrast any relative differences in their use of the application under test.

Furthermore, HUI Analyzer makes it possible to extract additional information from the comparisons results. Specifically, repeating patterns in deviations can be detected. For example, from the EAS and AAS descriptions given above, a repeating pattern "C D" will be detected, which has the frequency of two and a magnitude of two. Such patterns represent iterative sequences of actions that a user inputs and which the developer has not anticipated. The tool exposes these unnecessary actual actions, thus indicating a likely usability problem.

4.2 Assertion checking

Assertions are checks that evaluate to either true or false in a similar fashion to that of unit tests. Assertions use a predefined set of usability metrics as thresholds for certain aspects of usability. These thresholds are set by developers according to the usability guidelines they aim to follow. The tool includes default values for all assertion thresholds and, in addition, allows developers to define their own assertions.

There are three types of assertions, *form assertions*, *action sequence assertions* and *comparison assertions*; each of which have value at different stages of the development process.

4.2.1 Form assertions

Form assertions check *static* properties of the user interface under test. Being static, form assertions do not require an EAS or an AAS; rather they operate only on the compiled code for the user interface. Form assertions are intended to be used in the same way that unit tests are during software development, i.e., they are ideal for iterative development where by the assertions can be run on each iteration to expose any breach of usability guidelines as an interface is developed. Table 1 lists the standard form assertions with their default values.

Form assertions aim to detect usability problems relating to aesthetics. Through explicit use of metrics such as component dimensions, user interfaces that preclude high precision interaction with a stylus can be developed. This is important since studies have shown that high precision stylus-based interaction can lead to RSI conditions [2,3]. Knowledge of other metrics such as font size, clutter (percentage of free space) and color can be used to improve readability of an application's user interface.

Table 1 Form assertions

Total number of buttons	4
Total number of text input controls	2
Button height	40
Button width	80
Font types count	2
Font size	10
Selection controls per group/container count	5
ListBox item count	5
ListView item count	5
Menu items count	5
Menu depth	3
Percentage of free space	20%
Number of background colors used	2

4.2.2 Action sequence assertions

Action sequence assertions are checks on the recorded interaction using either an AAS or EAS description; they automate collection of classical usability metrics such as counting mouse clicks. Action sequence assertions are most useful when they are used in a formal user testing setting; they automate much of the counting overheads associated with user testing.

Table 2 shows action sequence assertions and their default values. To be in context of a particular user task, HUI Analyzer allows the default settings to be adjusted for particular tasks.

Table 2 Action sequence assertions

Number of clicks	15
Amount of scrolling	10
Number of help entities activated	3
Number of context menus activated	2
Amount of resizing	2
Number of text inputs	5

4.2.3 Comparison assertion

Comparison assertions check the results produced by comparison checking, discussed earlier in Section 4.1, by asserting key statistics over these results.

Table 3 Comparison assertions

Percentage deviation	20%
Percentage of incomplete comparisons (within an EAS suite)	10%
Largest percentage deviation magnitude	10%
Average percentage deviation	10%
Average largest percentage deviation magnitude	10%
Maximum time taken to complete an EAS	N/A
Repeating pattern frequency	5
Repeating pattern magnitude	5
Average repeating pattern gap	5

Comparison assertions are useful when developing complex applications that require a large number of EAS and AAS comparisons. In such cases, it is impractical for developers to inspect the results of every comparison in detail. Rather, developers need only examine results that indicate potential usability problems. Comparison assertions can thus be used to flag troublesome cases for developers to inspect in more detail.

4.2.4 User defined assertions

It is also possible for developers to extend HUI Analyzer's assertions with their own custom assertions. This can be useful when developers need to implement new ways of measuring usability, or when predefined metrics should be constrained to account for certain business rules.

Developers can define new assertions that fall under any of the abovementioned categories by simply implementing their corresponding interfaces: *IFormAssertion, IActionSequenceAssertion,* and *IComparisionAssertion*. These interfaces can be implemented as part of the application under test by importing the framework as an external package. The tool then reflectively detects all of the classes that implement any of these interfaces when the application is loaded by the tool. HUI Analyzer automatically recognizes these assertions and categorizes them with their predefined counterparts respectively.

In implementing new assertions, the tool allows developers to make use of functions that are used to implement the tool's predefined assertions. Commonly used functions include those for counting actions and components that meet particular criteria.

4.3 Hotspot analysis

Hotspot analysis presents a color map portraying the relative use of each component for a given type of interaction, such as a mouse click or text input. Darker colors indicate hotspot components that are frequently used. For example, Figure 3 shows that the large text area and lower-left drop down list components are used relatively heavily and that the Reset button is never used. Hot Spot analysis can be applied to either EAS or AAS descriptions.

Hotspot analysis can be used to optimize the location of components within a form. For example, developers can group frequently used components together, or aim to distribute the interaction evenly across the form. Hotspots can also indicate poorly chosen UI components; for example, if a dropdown list is being heavily used it could be substituted for a table or list box, thus offering a more efficient means of interaction.

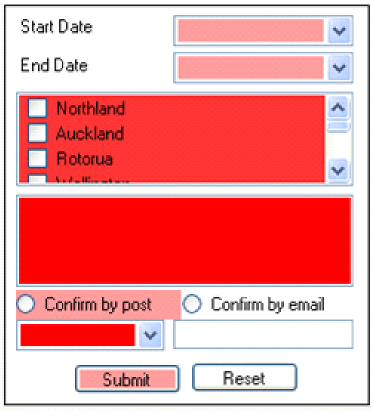


Figure 3 Hotspot analysis

5. EAS semantics

Specifying the expected interaction of a user in the form of an EAS can be complex and error-prone. There are many factors to consider such as specifying the correct ordering of actions, different alternatives of actions, granularity of each action, and filtering out irrelevant actions that have no effect on what is being tested. HUI Analyzer provides helpful EAS constructs that address the aforementioned issues, in addition to making the process of constructing an EAS efficient by reusing these constructs across multiple task descriptions.

5.1 Abstract actions

As mentioned in Section 4, there are some situations where developers cannot specify particular attributes of an action. Common scenarios include not being able to specify at design time the actual text that will be entered into a text field at run-time, or item selected from a dropdown list. In such cases, developers can specify that these details are omitted from particular actions comprising an EAS description. Where some attribute of an EAS action is omitted, the action will match a corresponding AAS action regardless of the value for the same attribute of the AAS action.

5.2 Multiplicity indicators

For every action in an EAS description, a multiplicity indicator is associated with the action. A multiplicity indicator specifies a lower/upper bound on the number of times that an action is expected to appear at a given position within an EAS. For example, a multiplicity of (2,5) specifies that an action must appear at least twice and at most five times. Where the value for the lower and upper bounds are equal, the action must appear exactly that many times. In cases where a multiplicity indicator is omitted, the associated action is permitted to appear zero or many times.

Multiplicity indicators can also be negative, for example -(1,1), indicating that a particular action should not appear at a given position within an EAS. In other words, any actual action *other* than that specified is permitted. As another example, a multiplicity indicator of -(2,5) indicates that action should not appear for at least the next two and up to the next five actions.

5.3 Action lists

A complex EAS description can be lengthy with many repetitive segments that represent commonly occurring action sequences, such as entering credentials for login screens. HUI Analyzer provides reusable constructs, which include *action lists*, to concisely compose EAS descriptions that involve common action structures.

An action list, similarly to an action sequence, is an ordered sequence of expected actions. However, an action list must be instantiated before use. Action lists essentially provide a way to refactor and reuse common EAS sequences. Once specified, an action list can be instantiated and reused multiple times in an EAS description. Furthermore, action lists can be nested to arbitrary depths to describe more complex behavior.

5.4 Action sets

The implicit ordering of action list elements is not sufficiently flexible to describe many common task scenarios. In particular, a task may involve an action that can be realized through many means, *any one* of which is valid.

Consider a task that involves the user copying the contents of a text field to the clipboard. Having selected the text in the text field, there are three ways the user could perform this action: 1) click a Copy button, 2) access the Edit menu and select the Copy item, 3) select Copy from the text field's context menu. To specify within an EAS description that any one of these three options is valid, an action set should be used. Like action lists, action sets must be instantiated prior to use. When used in an EAS description, an instance of the action set, associated with a multiplicity indicator of (1,1), would mandate that text is copied. An AAS that used any one of the three means to copy text would match the EAS description.

5.5 EAS suites

Depending on the size and complexity of an application being developed, the number of EAS descriptions could grow to be unmanageable in number. Particular problems that can arise in this situation are naming conflicts and construct duplication.

An EAS suite is a packaging construct used to organize EAS elements, including entities such as action sequences, action lists and action sets in addition to other EAS suite entities. EAS suites also have semantic significance as they are used to manage scope for widely used constructs. EAS suites are useful in refactoring large EAS descriptions and promote reuse of EAS elements.

6. Framework architecture

Architecturally, HUI Analyzer is based on two distinct components, a recorder component located on a handheld device and the tool itself, comprising a recorder, analyzer and associated user interface, hosted on a desktop machine.

The role of the recorder component on the handheld device is simply to record an AAS using a recorder module. The recorder uses little memory so as not to incur delays on the observed application. Given that handheld devices often perform poorly at context switching, the recorder shares the same address space as the application, thereby avoiding the need for multitasking and inter-process communication.

To satisfy the goal of being application independent, the recorder uses run-time reflection to discover the GUI component instances used by the application's GUI. Once discovered, listeners are registered on the components to intercept interaction events and to write these to a stream whose destination is an XML file.

On the desktop machine, a recorder component allows developers to construct EAS descriptions. Developers can record interactions made on the handheld application's user interface, in a similar manner to users creating AAS descriptions. Alternatively, and more commonly, developers access HUI Analyzer's functionality for constructing EAS descriptions using the tool's GUI to drag and drop the constructs described in Section 5.

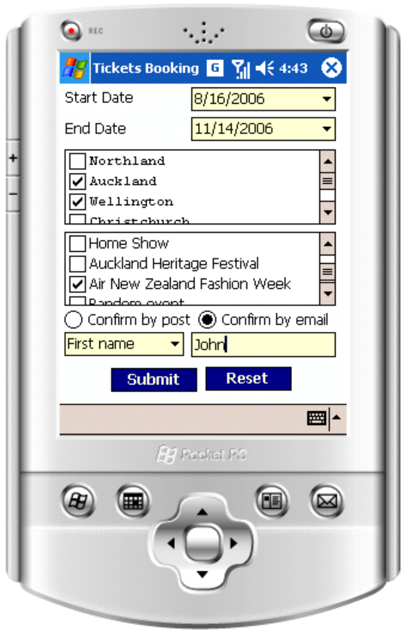
XML is used to describe data, notably AAS and EAS descriptions, produced and consumed by components. The use of XML allows for extensibility in that new components or external tools can work with AAS and EAS data simply by using a standard XML parser. Furthermore, AAS and EAS extension is also straightforward as these descriptions conform to a well-defined XML schema.

7. Evaluation

To evaluate the effectiveness of HUI Analyzer in detecting usability problems, we carried out two parallel experiments. Each experiment involved a separate group of eight users and a common application, an event booking system. One of the experiments took the form of formal user testing, described in Section 2, while the other used HUI Analyzer. Using these two distinct techniques to discover usability defects, we have been able to compare the relative effectiveness of each.

Figure 4 shows the GUI for one of the forms comprising the event-booking application. The GUI typifies the design of many user interfaces designed for handheld devices. In particular, such interfaces tend to be structured using form hierarchies with each form containing multiple controls (GUI components) that are operated using a stylus.

The application shown in Figure 4 is used by tour guides to book events and travel details for their customers. The form shown in the figure allows the user to make a reservation for one or more events for a given client. Intended interaction with the form is that the user selects start and end dates followed by locations of interest. With these inputs, the form is populated with a list of events being held between the given dates and at the locations specified. Events can then be selected for booking, which requires input of the customer's personal and contact details.



Start Date 8/16/2006
End Date 11/14/2006
[ ] Northland
[✓] Auckland
[✓] Wellington
[ ] Christchurch
[ ] Home Show
[ ] Auckland Heritage Festival
[✓] Air New Zealand Fashion Week
[ ] Random event
Confirm by post [✓ Confirm by email]
First name John

Figure 4 Event booking application

Our evaluation included tasks based on the form described above. According to best practice [5,11] the tasks had well defined objectives. In the case of the above form, each user was asked to make a reservation for specific dates, locations and events, and to input specified personal details. While the objectives of this task were made clear to users, they were not given any information as to how to achieve the objectives, neither did they have prior knowledge of the application.

Table 4 Quantitative results summary

	HUI Analyzer	User Testing
Average percentage of deviation	28.5%	31%
Average percentage of failures	50%	37.5%

Both quantitative and qualitative data was obtained from the formal user testing experiment. Quantitative data included the total number of actions made by each user, the number of slips (recoverable mistakes) made during each user's interaction with the application, for each slip the number of actions required to recover from the slip, and the number of irrecoverable mistakes (failures). Following the task, each user was asked a number of questions, answers to which served as the basis of qualitative data. These questions focused on issues including aesthetics and ease of understanding of the event-booking application.

Table 5 Comparison of usability issues detected

Usability Issue	User Testing	HUI Analyzer		
		Assertions	Comparisons	Hotspots
Aesthetics				
Color scheme Issues	✖	✓	✖	✖
Inappropriate font types	✖	✓	✖	✖
Inappropriate font sizes	✖	✓	✖	✖
Clutter	✓	✓	✖	✖
Understandability				
Selecting tour events for corresponding locations	✓	✖	✓	✓
Filling personal details	✓	✓	✓	✓
Confirming by both email and post	✓	✖	✖	✖
Efficiency				
Selecting tour events for corresponding locations	✓	✖	✓	✓
Filling personal details	✓	✖	✓	✓
Significant amount of scrolling required	✓	✖	✖	✖
Many items in combo-boxes to scan	✓	✓	✖	✓

Table 4 summarizes the results obtained from the two experiments. For the formal user testing experiment, the average percentage of deviation figure represents the number of actions users have spent attempting to recover from slips over the total number of actions performed by users. In the case of the experiment involving HUI Analyzer, this figure is obtained from the tool's comparison analysis capability that measures difference between the task's EAS and the AAS description for each user. In the case of the average percentage of failures figures, for the HUI Analyzer experiment this is derived from the percentage of incomplete EAS/AAS comparisons.

The figures in Table 4 show that HUI Analyzer has reported a similar volume of usability problems to that of using the formal user testing technique. However, these results do not confirm whether the tool has detected the correct set of usability problems that are likely to be detected through formal user testing.

Table 5 presents the breakdown of all usability issues detected by the two experiments. The table reveals that HUI Analyzer detected all but two of the usability issues detected by formal user testing. One issue concerns confirmation options, the choices being email and/or post. While the task requested users to select both options, the application allows users to select only one of these options. Interestingly, this issue was detected in the survey that followed the formal user testing exercise, not the formal testing activity itself. This suggests that a mix of usability testing approaches is most effective in detecting usability defects. The second issue concerns scrolling. It was not possible to check scrolling with the .NET framework that was used but this would be a welcome addition in the future.

Another important observation is that HUI Analyzer detected three false positives – falsely diagnosed usability problems that are actually of no concern. These false positives are the first three issues in Table 5. The reason for these issues being raised as usability problems was due to the assertion thresholds being set too strictly. This demonstrates that thresholds need to be set appropriately and in context of the application under test. Furthermore, HUI Analyzer does not flag deviations from a rigid usability standard, but flags deviations from a calibrated usability model.

8. Conclusions

In this paper we have presented a new tool, HUI Analyzer, intended to automate much of usability testing. HUI Analyzer is novel in that it offers a cohesive set of functions to measure different aspects of an application's usability at relatively low cost. These functions have been presented under the following names: *comparison checking*, *assertion checking*, and *hotspot analysis*.

Comparison checking, influenced by earlier work such as Expectation Agents, exposes any differences in the way a developer *expects* their application to be used and the way users *actually* use the application. To perform comparison checking, a developer uses HUI Analyzer to prepare an expected action sequence (EAS) for a particular task supported by the application. HUI Analyzer then records an actual action sequence (AAS) representing a user's interaction with the application and automatically compares this with the EAS description. Any differences are reported to the developer, for example highlighting tasks that the user did not complete or that were completed in a manner involving unnecessary repetition.

HUI Analyzer's assertion checking functions have been inspired by those of popular unit testing tools. Form assertions focus on the static properties of a user interface and include the size of GUI components, font sizes, and depth of menu structures. HUI Analyzer's assertion mechanism also allows assertions to be run on action sequences and the results of comparison checking. Assertions can be run iteratively over the development of a user interface to offer a fast and low-cost means of detecting the presence of usability defects.

Hotspot analysis is a novel way of illuminating the relative use of GUI components on a form. Hotspot analysis is a useful means of informing form layout, for example to collocate heavily used components thereby reducing scrolling or movement around a form. This is of particular value for handheld applications where scrolling and movement should be minimized.

Although designed for usability testing of GUIs running on handheld devices, where hotspot analysis is particularly useful, comparison and assertion checking are more general applicable to a wider class of GUI.

Beyond the functionality offered by HUI Analyzer, the tool satisfies several design goals. First, the tool is unobtrusive to test users. The recorder component that is deployed on a handheld device shares the same process as the application and transparently intercepts interactions between the user and the application. The recorder imposes negligible processing overhead and is invisible to test users, thus an application being tested behaves identically to when it is being executed normally. Second, the tool is application independent and requires no modification to a .NET Compact Framework application. Reflection is used to plant the necessary interception points. Third, while the tool can be used in an XP-like environment involving iterative assertion checking, HUI Analyzer is independent of a particular process model and can be used at many stages of software development. Finally, HUI Analyzer is extensible in that new types of assertions can be introduced by developers. Furthermore, based on well-defined XML schemas, the tool's outputs can easily be consumed by other tools.

In evaluating HUI Analyzer, we found the tool to perform comparably with formal user testing in terms of identifying usability defects. Similarly to formal user testing, with HUI Analyzer test users still have a key role to play, but the time taken by HUI Analyzer to automatically analyze and report on user interactions is much less than that for traditional formal user testing. HUI Analyzer thus quickly highlights deviations in a user interface's expected and actual use. With respect to HUI Analyzer's support for static analysis, it is important for assertion thresholds to be correctly calibrated in order to avoid the tool reporting false positives. Finally, our evaluation demonstrates the value of supplementing quantitative usability testing methods with techniques like surveying to gather qualitative data.

9. References

[1] ISO 9241-11 (1998) *Guidance on Usability,* International Organization for Standardization

[2] Dix, A, Finlay, J., Abowd, G. D., and Beale, R. *Human-Computer Interaction*. Prentice Hall, Harlow, England, third edition, 2004.

[3] Gong, J. and Tarasweich, P. *Guidelines for Handheld Mobile Device Interface Design*. In proceedings of the Decision Sciences Institute, annual meeting, Northeastern University, Boston, Massachusetts, 2004.

[4] Kurosu, M. and Kashimura, K. *Determinants of the Apparent Usability*. In proceedings of the international IEEE Systems, Man and Cybernetics conference, 1995.

[5] Charlton, S.G. and O'Brien, T.G. *Handbook of Human Factors Testing and Evaluation.* Lawrence Erlbaum Associates, 2002.

[6] Nielsen, J. *Finding Usability Problems Through Heuristic Evaluation*. In proceedings of the SIGCHI conference on Human factors in computing systems, Monterey, California, USA, 1992.

[7] Sullivan, P. *Beyond a Narrow Conception of Usability Testing*. IEEE Transactions on Professional Communication, vol. 32, no. 4, 1989.

[8] Qiu, Y.F., Chui, Y.P. and Helander, M.G. *Usability Analysis of Mobile Phone Camera Software Systems*. In proceedings of the IEEE conference on Cybernetics and Intelligent systems, Bangkok, 2006.

[9] Mariage, C., Vanderdonckt, J. and Chevalier. *Using the MetroWeb Tool to Improve Usability Quality of Web Sites*. In proceedings of the IEEE Latin American Web conference, Buenos Aires, 2005.

[10] Kishi, N. *SimUI: graphical user interface evaluation using playback.* In proceedings of the 16th annual COMPSAC Conference, Chicago, USA, 1992.

[11] Hix, D., Radin, D., Siochi, A.C. and Benel, D. *Computer analysis of user session transcripts for evaluation of the human-computer interface.* In proceedings of IEEE SoutheastCon, Williamsburg, VA, USA, 1991.

[12] Hilbert, D. M. and Redmiles, D. F. *Extracting usability information from user interface events*. In ACM Computing Surveys, Vol. 32, Issue 4, 2000.

[13] Girgensohm, A., Redmiles, D. F. and Shipman, F. M. *Agent-Based Support for Communication between Developers and Users in Software Design*. In proceedings of the Knowledge –Based Software Engineering Conference, Monterey, California, USA, 1994.