Developing a Dynamic Usability Evaluation Framework using an Aspect-Oriented Approach

Slava Shekh and Sue Tyerman

School of Computer and Information Science, University of South Australia, Australia

Abstract. Recent work in usability evaluation has focused on automatically capturing and analysing user interface events. However, automated techniques typically require modification of the underlying software, preventing non-programmers from using these techniques. In addition, capturing events requires each event source to be modified and since these sources may be spread throughout the system, maintaining the event capture functionality can become a very arduous task. Aspect-oriented programming (AOP) is a programming paradigm that separates the concerns or behaviours of a system into discrete aspects, allowing all event capture to be contained within a single aspect. Consequently, the use of AOP for usability evaluation is currently an area of research interest, but there is a lack of a general framework. This paper describes the development of an AOP-based usability evaluation framework that can be dynamically configured to capture specific events. The configuration is controlled through a frontend, which adds to the ease of use, and helps support non-programmers in conducting automatic usability evaluation.

1 Introduction

Usability evaluation is a technique used in the area of human-computer interaction (HCI) to assess how easily a user can interact with an interface. Software applications generate user interface events as part of their operation, and these events are used to assist in usability evaluation. Recent work in this area has examined techniques for automating usability evaluation by automatically capturing and analysing user interface events [4]. Automation allows the events to be captured with minimal human intervention and the event capture is unobtrusive from the perspective of the user, so the act of observation does not affect the activities being observed.

However, in order to automatically capture these events, the underlying software needs to be modified, requiring further preparation time for the usability evaluation and the involvement of a programmer to make the modifications. In comparison, a non-automated method, such as heuristic evaluation, can be performed by a non-programmer [6].

Additionally, user interface events occur in different components of the application, making it necessary to modify the system in each of these places to facilitate event capture. These modifications become increasingly difficult to manage in larger systems and extending the software becomes a very arduous task, because the soft-

Shekh S. and Tyerman S. (2009).
Developing a Dynamic Usability Evaluation Framework using an Aspect-Oriented Approach.
In *Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering - Evaluation of Novel Approaches to Software Engineering*, pages 203-214
DOI: 10.5220/0001860202030214

ware developer needs to find and change each individual part. For instance, consider a situation where the current system captures events by displaying them on the screen. If the need arises for these events to also be stored in a database, then each component of the system needs to be individually modified in order to implement the change. If the system contains hundreds or thousands of components, then modification can become infeasible.

Nonetheless, the limitations described can be overcome using aspect-oriented programming (AOP). AOP is a programming paradigm that focuses on separating the cross-cutting concerns of a system. A *concern* is a function or behaviour of the application, and can be categorised as either a *core concern* or a *cross-cutting concern*. A core concern is a primary behaviour, while a cross-cutting concern is a behaviour that spreads across multiple parts of the system [8].

Consider a simple banking application, which supports three functions: *withdraw money*, *view balance* and *log event*. An event is logged when either *withdraw money* or *view balance* is executed. *Withdraw money* and *view balance* are core concerns, because they provide the primary behaviour of the system, while *log event* is a cross-cutting concern, as its functionality needs to be integrated into both *withdraw money* and *view balance* in order for it to be able to log events generated by those components.

Traditional paradigms, such as object-oriented programming (OOP) provide limited support for separation of concerns. For example, developing a software system with OOP is based on the idea of *objects*. A single object can be used to handle a single core concern, such a system function, but is not able to encapsulate a cross-cutting concern, such as event logging. Conversely, AOP addresses separation of concerns using *aspects,* which are able to effectively encapsulate both core and cross-cutting concerns. Each aspect is designed separately from the rest of the system and all aspects are then integrated into the system using an *aspect weaver*. A weaver adds the aspect code into the base code of the application at specific points that are defined by the programmer, called *join points* (see Figure 1).

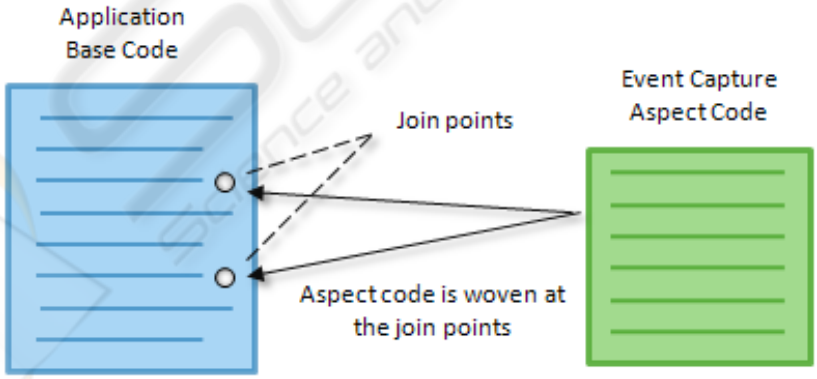

Fig. 1. Basic operation of an aspect weaver.

2 Related Work

AOP offers a number of opportunities that could benefit human-computer interaction.

2.1 Event Tracing

The use of AOP for event tracing was demonstrated by Low [7] in the development of Toolkit for Aspect-oriented Tracing (TAST), which handles all trace-related activities in Java applications. TAST is a support tool for the Test and Monitoring Tool, which had previously been developed at Siemens.

In addition to Low's work on event tracing, AOP has been used in the development of a trace monitor [2]; a tool that observes a system to detect sequences of events and takes action when events occur in a particular pattern. The tool was developed using a combination of AspectJ (a popular Java-based implementation of AOP) [1], and another language called Datalog. However, AspectJ can only trigger code at the occurrence of a single event, so the authors introduced a new language feature to AspectJ called "tracematches", which allows code to be triggered when an event sequence matches a pattern.

2.2 Event Capture

Hartman and Bass [3] described and implemented an event logging system for graphical applications to capture additional event logging information at the architectural boundaries of the application. The system was implemented using Java and Swing, and cross-cutting concerns, such as logging, were handled by AspectJ. The system was then applied to two applications and tested with a small user group. The results showed that the new system addressed many of the log-related problems that were described by earlier authors, including Hilbert and Redmiles [4], in their survey of usability evaluation techniques.

Tao further explored AOP-based event capture by using AOP as a means of capturing user interface events that occurred in different parts of the application [8], [9]. Aspects were added to the application, with the responsibility of capturing different events, along with contextual information. The use of an aspect-oriented approach allowed system-wide events to be captured in a single location.

2.3 Usability Evaluation

Extending on aspect-based event capture, Tarta and Moldovan [11] used AOP to automate the process of usability evaluation by developing a usability evaluation module, which captured events using an aspect-oriented approach. This module was integrated as part of a working application and then tested with a small user group. The testing showed promising results when compared to traditional usability evaluation techniques, such as event log analysis. Their research also provides ample oppor-

tunities for future work, such as the need for a usability evaluation framework based on AOP. The implementation of such a framework is the primary focus of this paper.

Although AOP is a suitable approach for conducting usability evaluation, there are other techniques that may also be useful. One group of researchers compared two approaches for conducting usability evaluation: aspect-oriented programming and agent-based architecture [10]. The comparison described both techniques as being effective for usability evaluation, and showed that AOP could support usability testing without the need to modify the original application code. The comparison provides valuable insights into different techniques for conducting usability evaluation and highlights the validity of an aspect-oriented approach.

3 Framework

The primary aim of the research presented in this paper was to develop a usability evaluation framework using AOP, with no impact on the source code. If AOP was not used, in the worst case scenario, twenty duplicate sets of logging code would be needed to record mouse events alone, leading to potential mistakes and difficulties with maintenance. With over 100 different events being observed, AOP has a clear advantage over traditional approaches.

The main purpose of the framework is to capture user interface events and assist in usability evaluation. At the same time, the framework aims to be as re-usable as possible, so that it can be used in different applications, following the recommendations of Tarta and Moldovan [11].

In order to capture events, the framework needs a software application to act as a source of events; an event being any change of state in the system. In this implementation, a Java-based application called the ACIE Player has been used as the event source. The Player provides an integrated environment for viewing synchronised data streams, including audio, video and transcribed text (see Figure 2).

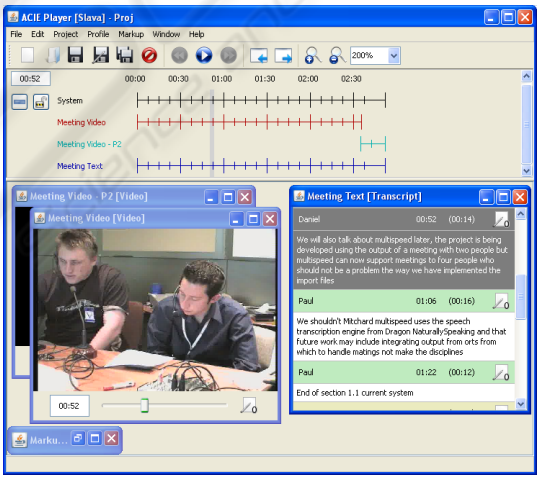

Fig. 2. Screenshot of the ACIE Player.

In addition to this playback environment, the Player also allows the user to modify the transcribed text and annotate the different data streams. Since the ACIE Player is a Java-based application, a popular Java AOP implementation called AspectJ has been used for developing the framework.

The framework consists of four main components: *aspect interface*, *event capture module*, *framework frontend* and *output module*. These components and their interactions are shown in Figure 3, and described in more detail below.

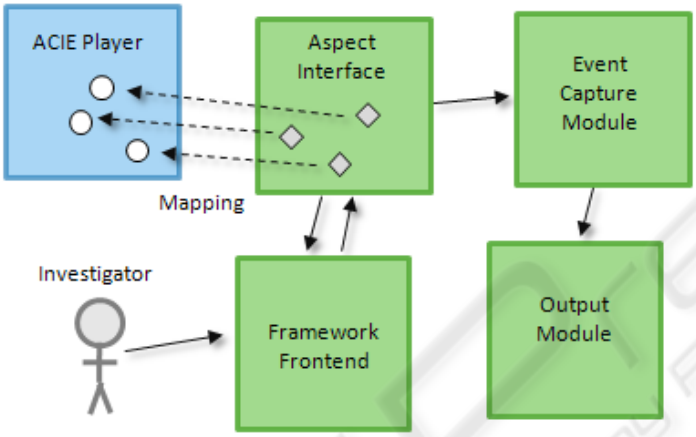

Fig. 3. Overview of the framework architecture.

3.1 Aspect Interface

The aspect interface is specific to the ACIE Player, while the other components can be reused by other applications that adopt the framework. Initially, a programmer needs to create the aspect interface for the target application. The aspect interface contains a series of join points that map to GUI components in the target application.

For instance, the work area is a GUI component in the ACIE Player and contains a method for cascading windows. The following join point in the aspect interface maps to this method and thus, captures the event of the windows being cascaded:

```
after() : target(WorkArea) && execution(* cascadeWindows()) {
  String action = "Windows Cascaded";
  Event event = new Event(action);
  CaptureModule.logEvent(event);
}
```

Although the joint point mapping needs to be done by a programmer, each unique application only needs to be mapped once.

3.2 Framework Frontend

Once the programmer has created the aspect interface, an investigator can configure the interface and perform a usability evaluation without any more input from the

programmer. In order to configure the interface, the investigator uses the framework frontend. This component provides a GUI with a series of checkboxes (see Figure 4), which are automatically generated based on the join points present in the aspect interface. The investigator then selects checkboxes based on the GUI components and specific events that they are interested in logging. In addition, the investigator can select their preferred output formats, and their overall selection is saved to a configuration file.

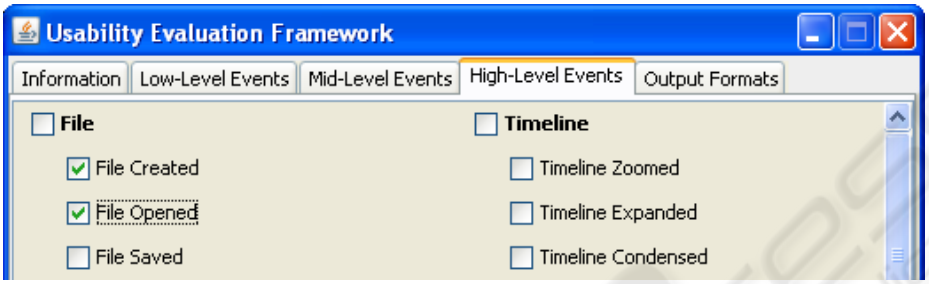

Fig. 4. Screenshot of part of the framework frontend.

3.3 Capture and Output Modules

Once the configuration file has been set up, the framework can begin logging events. While a user is interacting with the ACIE Player, the framework runs in the background and records their actions. However, the framework only logs the events that are specified in the configuration file, which is determined by the investigator's selection in the framework frontend. For each of the logged events, the aspect interface requests the event capture module to log the event. The event capture module then stores the event and passes it on to the output module, which can process the event into different output formats.

4 Case Study

The framework was tested and validated using a case study. The study was targeted at the Information Systems Laboratory (InSyL) of the University of South Australia, which is a specific user group of the ACIE Player [5]. The case study consisted of an investigation experiment and a usability experiment. The investigation experiment evaluated the use of the framework itself, while the usability experiment tested the effect of AOP on system performance to identify whether AOP was a suitable driving technology for the framework.

The participants selected for both experiments were a combination of programmers and non-programmers, which helped evaluate the general effectiveness of the framework for all user types. All participants completed the usability experiment first, and thus, this was their initial point of contact with the ACIE Player and framework.

4.1 The Case

Consider the following scenario, which illustrates how the ACIE Player is used by members of InSyL and is the target case for the study.

An analyst is interested in observing a recorded meeting and analysing the behaviour and interaction of the meeting participants. The analyst has access to the video and audio streams of the recorded meeting, and a transcript that is automatically generated from the audio. The analyst observes the video and transcript in small sections (i.e. 10 seconds at a time), and makes textual annotations based on the behaviour of participants. The analyst records the annotations using a standard syntax to make the information easier to categorise. For instance, if a participant stands up and says something, the analyst might record "P1: stand, speak".

Performing this task without an integrated tool is problematic, because the analyst needs to run multiple applications concurrently, and manually synchronise them. For example, they might run a video player to watch the meeting, a transcription program to read the meeting transcript and a text editor to record their annotations. Since all of these are separate applications, the analyst will have to constantly check that the data from each application is synchronised.

The ACIE Player is preferable for performing the task, because it is able to provide all of the required capabilities in an integrated and synchronised environment. Using the ACIE Player minimises the cognitive overhead for the analyst in performing their task, since the work is managed within a single application. This also reduces the likelihood of errors and inconsistencies in the analysis, because the analyst no longer needs to spend time managing and synchronising different media and software applications, and is able to focus more on their actual task.

4.2 Usability Experiment

In the usability experiment, all participants took on the role of a user. The main purpose of this experiment was to evaluate whether or not users could detect any degradation in system performance due to the presence of the framework. This helped to identify whether AOP was a suitable technology for capturing events and running the framework.

Each user carried out two simple tasks using the ACIE Player, based on a set of instructions. In both tasks, the overall flow of activities was as follows:

1. Launch the ACIE Player
2. Create a project (a project is simply a collection of related files)
3. Add some files to the project
4. Customise the work environment
5. Play the video
6. Edit the transcript
7. Create, edit and remove an annotation
8. Close the application

During one of the tasks, the participant used the ACIE Player by itself, while during the other task they used the Player with the framework enabled. The presence of

the framework was randomised during each task to minimise the existence of bias, and to prevent learning effects from affecting survey results. Participants were not told whether the framework was enabled during a particular task, and care was taken to ensure that there were no clues to reveal this information.

When enabled, the framework was configured to capture all available events in three different output formats: plain text, CSV and XML. This produced a large amount of event data and was therefore an accurate way of testing the user's ability to detect the framework.

Each user completed a short survey upon completing the first task of the experiment and another survey at the end of the experiment. Each survey presented the user with two questions:

Question 1 – were you aware of the presence of the framework?
Question 2 – did the framework interfere with your task?

It was assumed that the participants had basic computer skills and experience with Window-based applications, but they were not required to have any prior experience in using the ACIE Player or the framework. Each user was simply required to follow a set of instructions as best as they could, without any practice or training.

4.3 Investigation Experiment

In the investigation experiment, participants were randomly divided into three groups, where each group consisted of a user and an investigator. All participants had already completed two tasks with the ACIE Player in the usability experiment, so each person had the same amount of experience in using the Player and the framework.

The user of each group carried out the task from the first experiment, while the investigator observed the user and conducted a simple usability evaluation. The investigator performed this evaluation by following a set of instructions and completing an objective sheet – counting the occurrence of specific events and comparing this to other events. For example, one of the objectives asked the investigator to identify whether *tile window* events or *cascade window* events occurred more often.

The overall flow of activities during the investigation experiment was as follows:

1. Investigator receives an instruction sheet and an objective sheet
2. Investigator launches the usability evaluation framework
3. Investigator configures the framework to capture the events described on the objective sheet
4. User launches the ACIE Player and performs the activities from the usability experiment
5. User closes the ACIE Player
6. Investigator examines the event logs that were generated by the framework
7. Investigator completes the objective sheet

Upon completion of the experiment, the investigator completed a survey that was answered by circling a number on a 7-point Likert scale. The number 1 represented a very negative answer, while 7 represented a very positive answer. For instance, the

first question was “how would you rate your computer skill level?” The number 1 matched to the verbal answer of “beginner”, while 7 matched to “expert”.

Once again, participants were not required to have any prior experience in using the ACIE Player or framework.

5 Experimental Results

The same group of six participants was involved in both experiments of the study. All participants were members of the InSyL group.

5.1 Usability Experiment

All six participants took on the role of a user in the usability experiment. The primary source of data in this experiment was the two surveys that users were required to complete, in which they provided responses to the questions described in Section 4.2. Almost all users indicated that they did not notice the presence of the framework, and that it did not interfere with their task.

In one case, a user thought that they noticed the framework running during both of their tasks, but it was only running during one task. This suggests that what the user noticed was not actually the framework. Thus, the results indicate that AOP does not affect software usability and is a suitable technology for driving the framework.

5.2 Investigation Experiment

In the investigation experiment, the six participants were divided into three groups. In each group, one participant took on the role of a user and the other participant became the investigator. At the end of the experiment, the investigator filled in a survey by answering questions on a 7-point Likert scale.

The graph in Figure 5 shows the results obtained from the investigator survey. The survey questions have been abbreviated to improve the readability of the graph. The full questions are as follows:

- *Computer Skill* – how would you rate your computer skill level?
- *Programming* – how much experience have you had with computer programming?
- *Ease of Use* – how easy was the framework to use?
- *Information* – did the framework provide you with the information you needed to complete your objectives?
- *Enjoyment* – overall, did you enjoy using the framework?
- *Usefulness* – do you see the framework as something that you could use in your own work?

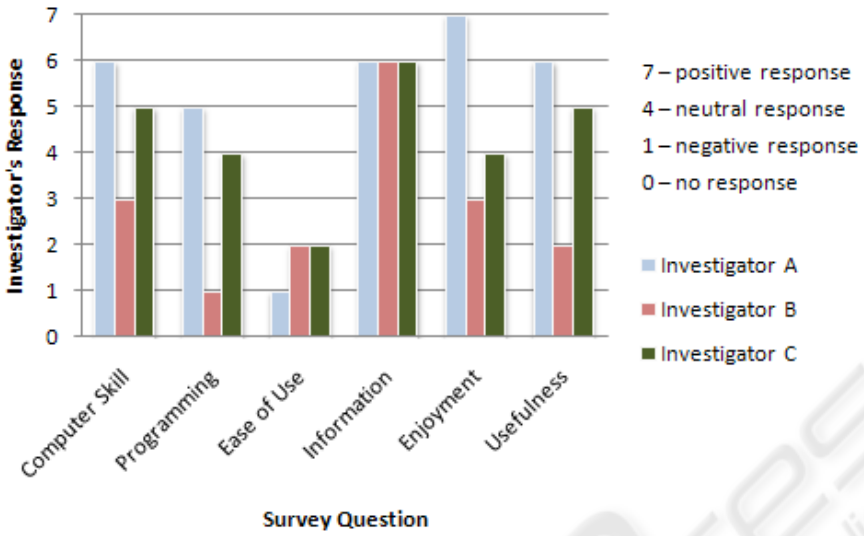

Fig. 5. Investigator survey results.

The graph shows that the investigators varied greatly in computer skill and programming ability, based on their self-assessment. However, all of the investigators found the framework difficult to use, even those with prior computer and programming experience.

Investigators described that the main difficulty was encountered while configuring the framework. The instruction sheet and the framework provided a large amount of information, which was very overwhelming for the investigators. One user explained that the framework GUI was "cramped" with information and that they were confused about what they needed to do to meet the requirements of their task.

Nonetheless, after thoroughly explaining the task to the investigator and describing how the framework should be configured, all investigators were able to complete the task successfully, even those with limited computing / programming experience.

Although configuring the framework generally proved to be a difficult task, the final task of interpreting the event data and completing the objective sheet was considered much easier. All investigators were able to fill in the objective sheet without encountering any major problems, and all of their answers proved to be 100% correct, when compared to the generated event data.

6 Future Work

The framework does not log every single event generated by the ACIE Player. Currently, only the user-initiated events are captured, but internal application events are ignored. The framework could be extended to capture all events generated by an application, which may prove to be useful. For example, the logging of internal events could provide the programmer with a trace of program execution to assist in debugging.

Some members of InSyL are not only interested in observing the participants of recorded meetings, but also in observing those who are observing the meeting. Since AOP has proven to be particularly useful in capturing a user's actions as they interact with an application, it may also be possible to capture and analyse the actions of an observer.

One of the current limitations of the framework is that a programmer needs to create an aspect interface for each unique application. This involves examining the application source code, discovering the join points and adding these join points to the aspect interface. The framework could be extended to automatically discover the join points and generate the aspect interface, potentially saving time for the programmer.

The case study showed that investigators found the framework difficult to use. One reason for the difficulties was the large amount of information presented in the framework frontend. Therefore, further analysis needs to be done in improving the GUI or creating a completely new one. For instance, the framework could run the ACIE Player in a *framework configuration mode*, where the investigator clicks on various GUI components in the Player and the framework then enables event logging of those particular components. This approach may be more intuitive, because the investigator would be clicking on the actual components and performing the actual operations that they want to log, as opposed to selecting checkboxes with the names of those operations.

Although the framework has been evaluated using a case study, the study only examined a single scenario of usage and the framework has only been tested with a single application (the ACIE Player). Since the framework is designed to be reusable, it needs to be tested with other applications to evaluate its effectiveness in different environments.

7 Conclusions

Aspect-oriented programming allows usability evaluation concerns, such as event logging, to be separated from the rest of the system. Researchers have already begun to explore this area, and in particular, Tarta and Moldovan [11] suggested the development of an AOP-based usability evaluation framework. Implementing this framework has been the accomplishment of this research project.

The framework was developed as an extension of the ACIE Player. The use of AOP enables the framework to be dynamically configured to capture a specific subset of all of the mapped events within the Player. The configuration is provided through a frontend, making the framework accessible to both programmers and non-programmers.

A case study, consisting of a usability and investigation experiment, was used to evaluate the implementation. The results showed that an aspect-oriented approach could assist in performing usability evaluation. Furthermore, the data from the usability experiment revealed that AOP did not create any performance constraints on the working environment, suggesting that AOP is a suitable technology for driving the framework.

Acknowledgements

This research project was in part funded through the UniSA/DSTO Research Agreement “Human Interaction Studies on the use of Speech Technology and Speaker Localisation in support of LiveSpaces: 2006/1170285: ACIE Player v2.0 Development Project – Honours Scholarship 2008”. We also thank Ahmad Hashemi-Sakhtsari and Michael Coleman for their assistance.

References

1. AspectJ Team 2008, *The AspectJ Project*, The Eclipse Foundation, viewed 25 April 2008, <http://www.eclipse.org/aspectj/>.
2. Avgustinov, P, Bodden, E, Hajiyev, E, Hendren, L, Lhotak, O, Moor, O, Ongkingco, N, Sereni, D, Sittampalam, G, Tibble, J & Verbaere, M 2006, 'Aspects for Trace Monitoring', *FATES/RV '06: Proceedings of Formal Approaches to Testing and Runtime Verification,* Springer, Seattle, pp. 20-39.
3. Hartman, GS & Bass, L 2005, 'Logging Events Crossing Architectural Boundaries', *INTERACT 2005: Proceedings of the 11th International Conference on Human-Computer Interaction,* Springer, Las Vegas, pp. 823-834.
4. Hilbert, DM & Redmiles, DF 2000, 'Extracting Usability Information from User Interface Events', *ACM Computing Surveys,* vol. 32, no. 4, pp. 384-421.
5. InSyL 2008, *Information System Laboratory*, University of South Australia, viewed 9 October 2008, <http://www.insyl.unisa.edu.au/>.
6. Ivory, MY & Hearst, MA 2001, 'The State of the Art in Automating Usability Evaluation of User Interfaces', *ACM Computing Surveys,* vol. 33, no. 4, pp. 470-516.
7. Low, T 2002, 'Designing, Modelling and Implementing a Toolkit for Aspect-oriented Tracing (TAST)', *AOSD 2002 Workshop on Aspect-Oriented Modeling with UML*, Enschede.
8. Tao, Y 2007a, 'Capturing User Interface Events with Aspects', *HCII 2007: Proceedings of the 12th International Conference on Human-Computer Interaction,* Springer, Beijing, pp. 1170-1179.
9. Tao, Y 2007b, 'Toward Computer-Aided Usability Evaluation for Evolving Interactive Software', *RAM-SE '07: Proceedings of ECOOP 2007 Workshop on Reflection, AOP and Meta-Data for Software Evolution*, University of Magdeburg, Berlin.
10. Tarby, J, Ezzedine, H, Rouillard, J, Tran, CD, Laporte, P & Kolski, C 2007, 'Traces Using Aspect Oriented Programming and Interactive Agent-Based Architecture for Early Usability Evaluation: Basic Principles and Comparison', *HCII 2007: Proceedings of the 12th International Conference on Human-Computer Interaction,* Springer, Beijing, pp. 632-641.
11. Tarta, AM & Moldovan, GS 2006, 'Automatic Usability Evaluation Using AOP', *2006 IEEE International Conference on Automation, Quality and Testing, Robotics,* IEEE Computer Society, Los Alamitos, pp. 84-89.