Chapter #

A LIGHTWEIGHT EXPERIMENT MANAGEMENT SYSTEM FOR HANDHELD COMPUTERS

Phil Gray, Joy Goodman and James Macleod
Department of Computing Science, University of Glasgow, Glasgow, UK

Abstract: This paper describes a system that helps HCI practitioners and researchers manage and conduct experiments involving context-sensitive handheld applications, particularly related to navigation assistance. The system provides a software framework in which application, user interface and interaction monitoring components can be plugged, offering a simple interconnection protocol and minimising the programming overheads of implementation. We have focused our attention on dealing with the challenges presented by the limited memory and processing of handheld devices and the variety of data sources for mobile context-sensitive applications. In this paper we give an overview of the system's functionality and architecture, discuss key challenges of supporting field-based experiments on handhelds and consider further developments of the system.

Key words: evaluation of user interfaces and tools, mobile applications, usage monitoring

1. INTRODUCTION

The use of mobile technologies has been growing rapidly, primarily of mobile telephones but also of other handheld devices. Together with the exploitation of new technologies such as GPS and 3G and improvements in wireless capabilities, this has inspired the development of a wider and more ambitious range of mobile applications. Additional challenges and opportunities are posed by context-awareness, which can be used to provide information related to the user's current location, e.g., to aid navigation.

However, less is known about how to make the resulting applications and devices usable. If this is to be achieved, usability experiments on handheld

devices are of key importance, both to test and improve the usability of existing and developed devices, and to gain information on how such devices can be designed in general, for example, by comparing alternative interfaces. Such experiments are important for groups that can particularly benefit from these devices but are most likely to be excluded by them - disabled people and older age groups. The design of handhelds for the older population is especially poorly understood and more work needs to be done in this area. Experiments with older users themselves on different handhelds and user interfaces are important to improving this understanding. It is within this context that the work described in this paper takes place [3, 4].

Sadly, the management of such experiments can often be a time-consuming and complicated task, as indicated in Table 1, meaning that evaluations are often limited. However, this process can be improved using software tools. Such tools have been shown to be of use in designing and testing desktop applications in the past, and similar tools could prove useful in mobile situations.

Table 1. Issues in managing experiments

Category	Example	Examples of management issues	Examples of particular issues with handhelds
Participants	16 participants aged 18-40 who don't know the area, gender-balanced	Obtaining and keeping track of participants, assigning them to conditions, coping with them dropping out	Changes may have to be made to the list of participants in the field
Conditions	Interface A is tested in condition C1 and interface B in condition C2	The right interface should be brought up at the right point in the experiment	Limited screen space and memory, rendering storage and selection of interfaces less easy
Tasks	1. Find your way from the library to the butchers 2. Find your way from the supermarket to the school	The right tasks must be matched with the right conditions for each participant to prevent order and other effects	Familiarity with the area in the 1st task should affect the 2nd as little as possible. Tasks may need to be changed due to external conditions, e.g., roadworks
Equipment	A handheld computer with the application, a GPS receiver, consent form, questionnaires, notebook and pen	Ensuring that the right equipment for each participant and set of conditions is available	Equipment must be carried. This may include equipment for several participants if the experimenter cannot return to base between trials
Data Collection	Start and end times for each condition, notes of when the participant got lost and which interface elements were used and how often	The data must be collected, collated and stored	Limited storage space on the device. Difficulties taking notes while on the move and trying to manage several pieces of paper at once.

There are many issues that such tools can support. Our eventual aim is to create a tool that would support the entire experiment process, including all the issues described in Table 1, as well as support for activities before and after the actual experiments. This paper, however, describes a prototype of an experiment management tool for mobile devices, incorporating support for managing the user interfaces and collecting usage data. In Section 2 we describe and discuss other experiment management tools and how this work relates to them. In Section 3 we then describe in more detail what our system does and how it works. We map out and discuss key challenges of supporting field-based experiments on handhelds and consider further developments of the system in Section 4, before concluding in Section 5.

2. RELATED WORK

Several experiment management tools exist for desktop applications and, while they are not generally suitable for mobile devices, some of the techniques used within them can be adapted to this setting. These tools have usually focused on either support for generating the user interface (UI) or on capturing data from the participants.

The system described in this paper does not provide support for generating interfaces *per se*, such as in the work on automatic interface generation. Rather it supports the process of generating different interfaces for the same data and then swapping between them so that they can be easily compared in an experiment. The emphasis is on aiding the running of experiments, rather than on generating good interfaces for a finished product. Worth mentioning here is the TAE Plus system [12] which, while not aimed at supporting experiments, separates the user interface and the program code making it easier to swap between interfaces.

Previous work on capturing experimental usage data has followed two main avenues – video and screen capture and event logging.

Video capture (e.g., [9]) involves videoing the participant and/or the screen during the experiment. This method is not easily transferred to the mobile domain without specialist equipment and/or wireless communication [10]. Screen capture (e.g., [1]) may be more feasible as it stores the images appearing on the computer screen at regular intervals and therefore does not require additional equipment. However, the resulting files are large and are likely to take up too much memory for a mobile device. In addition, it does not capture the context of use. Taken together with the simpler UIs on mobile devices, this renders it not much more useful than event logging if the latter is done at an appropriate level of detail.

Event logging systems [8] are potentially more useful for mobile studies, but existing logging systems place heavy demands on processing, storage, and communications. However, our understanding of interaction with mobile devices is rather poor and many opportunities remain for carrying out useful studies on or with relatively simple handheld user interfaces. Given the limitations of handhelds, these opportunities depend on keeping processing, memory and communication demands to a minimum. In addition, although some current systems do allow the experimenter to choose the user actions to be logged (e.g., [1]), this is complicated, and a simpler system is needed and indeed possible for a mobile device.

In addition, mobile devices have a greater need for an integrated experiment tool providing support for easily generating and swapping between experimental conditions as well as easy data capture and analysis. The limited memory and processing power of these devices and the difficulties associated with moving the program and data around means it is best to have a single program managing the experiment as a whole.

Although data capture and analysis have been integrated (e.g., [1, 7]), less work has been done on combining support for the interface with support for data capture. One example is UsAGE [13], which added event logging to the UI development system, TAE Plus [12], although not as a single unified program, and not for mobile devices.

3. SYSTEM FEATURES

3.1 An Example

The motivating example for the system described in this paper is the navigation aid, a typical mobile application that provides directions to the user to enable him or her to find a location. Such directions can be provided in a wide variety of formats, including maps, photographs and arrows, as well as using different modalities, but little work has been carried out comparing these different approaches [2].

Let us imagine that we want to evaluate and compare three such interfaces, shown in Figure 1. One way of doing this would be to write three applications, one with each interface, and create data sets for each for the test and pilot routes. We would then have to run the experiment, ensuring that the right interface with the right route was brought up at the right time. Code would have to be included in each application to monitor any usage data we wanted collected, such as timings and button clicks or alternatively these could be noted by hand by the experimenter.

Although this process is possible, it is rather complicated, and our system aims to simplify and support it.

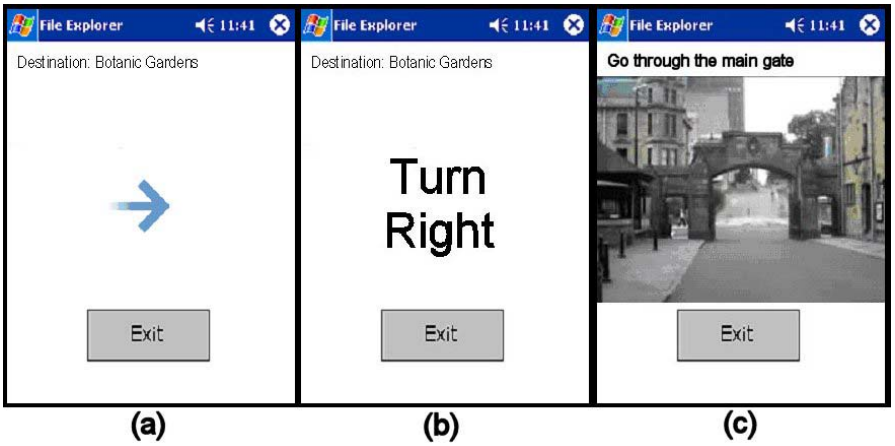

Figure 1. Three possible navigation aid interfaces

3.2 Supporting Adaptation

Our system supports the adaptation of a context-aware handheld application to different experimental conditions, by separating application data and operations, such as geographical information about the location or context (the Model) from the user interface components used to present this information to the user (the View). For example, in Figure 1, the same Model is used (information is presented about the same route) but using different Views (different interface methods).

It is possible to create very general models that contain enough information for a variety of different views. In addition, the framework supports more limited models, if less extensive surveying of the environment is desired. However, these models may only be suitable for some views. For example, the view shown in Figure 1(c) requires a model with images or pointers to images of locations, while those shown in Figure 1(a) and (b) only need to contain the directions to turn at particular coordinates. The same reduced model could therefore be used for (a) and (b) but not for (c).

Our system matches model and view as well as possible, leaving spaces in the resulting interface rather than crashing if they do not match completely, so that reasonable interfaces can still be produced if the model and view do not match but are not far apart.

This separation of model and view makes it easier to:

– create all of the experimental conditions. In context-aware applications such as navigation aids, the conditions typically consist of all possible

combinations of the different test locations with the different interfaces being tested. Using the method above, models can be shared between views rather than having to create a separate application for each combination;

– select the experimental condition to be tested. Rather than having to keep track of which application corresponds to which combination of conditions, the model and view can be chosen separately but at the same time. Currently this is done through XML configuration files, as shown in the example in Figure 2, which chooses a model called ArrowModel and a view called ArrowView, generating the interface shown in Figure 1(a);
– move the experiment to different locations, which may be necessary due to constraints outside the experimenter's control, such as roadworks. In this case, only the models need to be changed;
– test new interfaces by creating a new view for an existing model.

This process is further supported by the use of templates and C# interfaces for the models and views, reducing the amount of coding necessary to create new sets of location data and new interfaces.

Since we are working with complex and potentially unusual navigation and map-based user interfaces, we chose to represent our design options in term of parameterisable components. In particular, components can be linked by identifying data values in models to be listened to (and potentially updated by) view components. An alternative approach would be to employ a user interface specification language, like UIML [14], from which the actual user interface components could be generated by a "renderer". While this would increase genericity, there would be too high a cost in terms of the complexity and usability of the specification and specification language, especially given the potentially complex and non-standard character of interaction in our target applications.

```
<?xml version="1.0" ?>
  <configuration
xmlns="http://tempuri.org/configuration.xsd">
     <Model>ArrowModel</Model>
     <View>ArrowView</View>
     <Data>
       <Item>ArrowChange</Item>
       <Item>LocationChange</Item>
       <Item>DirectionChange</Item>
     </Data>
  </configuration>
</xml>
```

Figure 2. XML Configuration file for the data and view shown in Figure 1(a)

3.3 Supporting Observation

The system supports experimental observation by collecting usage runtime information. This may include, for example, information on which buttons or other interface elements were selected, when they were selected, when other important events occurred and the length of time taken for the whole experiment. We log information at this level of complexity, rather than lower-level actions and events, because we consider it to be the most useful level for analysing the results of the experiment and because lower-level events are of little use due to the reduced number of UI elements in a handheld interface and the simplicity of the standard input methods.

Each item of loggable information is given a label in the code for the model or the view. The experimenter can then use these labels to indicate which information is to be logged, thus customising the experiment and only collecting information of interest to that experiment. This reduces the sizes of the logs and simplifies their later analysis.

The selection of items to be logged is given in the experiment's XML configuration file, as shown in the example in Figure 2. This example generates the interface shown in Figure 1(a) and logs three events in addition to the application's starting and closing time. It logs when changes occur in the displayed arrow, the sensed GPS location and the direction. This particular interface doesn't contain any interactive UI components. If it did, their use could also be logged by generating suitable logging events in the code for the view, labelling them and including their labels in the configuration file.

3.4 System Architecture

We have created an implementation of our system written in C#, using Microsoft's .Net Compact Framework, which runs on PocketPC devices.

A runnable application consists of a single model object[1], a single view object, and an optional data collection object connected together and managed by an overall manager component (see Figure 3). The interconnection of the model, view and data collector is carried out with the aid of event generation interfaces made available via class methods: getDataItems() and getSchema(), which both return a set of identifiers from the model and view. getDataItems() returns a set of identifiers of active values that can be logged - "loggables" - and getSchema() returns a set of

[1] The model object can also accommodate additional components, such as a GPS proxy object.

identifiers of active values that form the model-view link (i.e., values that the model reveals to the view and values that the view is capable of presenting to a user for interaction) - "linkables". The manager creates a working application by:

– connecting the model and view by finding name matches in the value sets returned by the model and view via getSchema() and using the results to instantiate the actual user interface;
– determining what will be logged by finding name matches between the value sets returned by the model and view via getDataItems() on the one hand and the names of desired data to be logged located in the configuration file on the other hand.

On startup the framework reads in an experiment configuration file, such as that shown in Figure 2. The model and view classes specified in this file are then loaded and instantiated, and a data collection object is also instantiated. Using the name matching algorithms described above, event listeners are created for each matched loggable and linkable, with a predefined logging callback (see Figure 3).

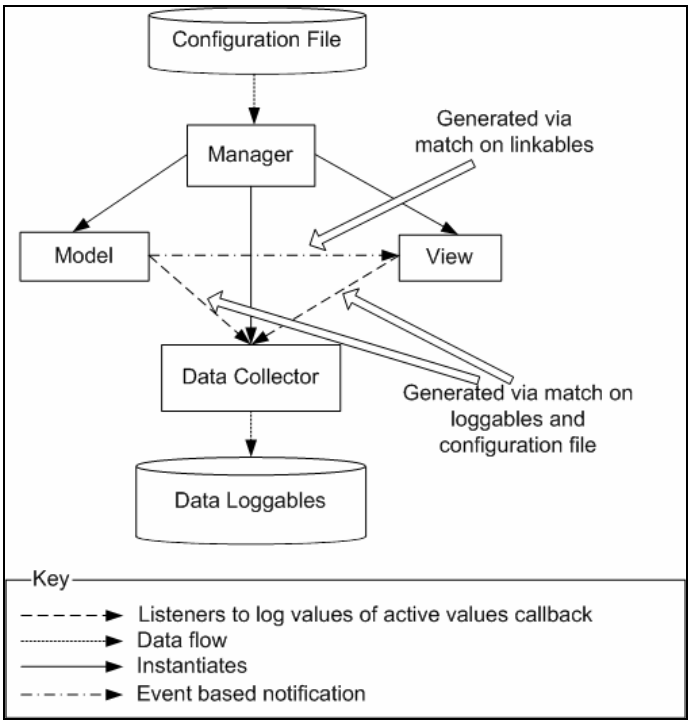

Figure 3. The architecture of the system

The ability to construct running, loggable applications from simple configuration files removes the need to pre-construct the several application variants necessary for a comparative study. The cost to the developer lies in

the need to add into the source code the information used by the framework manager to connect the components together, viz., the names of data items that notify changes to their state. Such data items can be used to update views or can be logged by a data collector component. Furthermore, model and view operations that can change the state of these data items must include in the relevant method a call to a method to fire an appropriate event.

Automatic linking of data items between model and view also demands that the system can determine a unique and sensible mapping between model and view. In the simple applications we currently envisage testing, such a mapping is possible and not costly to embed in the source code. However, this limits the generality of our automatic generation system and future versions may have to explore semi-automated approaches [5, 6, 11].

4. KEY CHALLENGES AND SUGGESTIONS FOR FURTHER DEVELOPMENT

The system as described above has been implemented in prototype form. We have constructed several alternative models and views and used these with configuration files to construct and run the simple navigation applications described in section 3. We have yet to use it "in anger", however, as part of a usability study. This trial will be taking place in the near future. Our study will investigate the relative effectiveness of several different methods of providing navigation information to older users, such as maps, sequenced landmarks and step-by-step directions. Consequently there will be a number of combinations of models and views that must be trialled, and it will also be necessary to change the configuration in the field with each individual participant.

There are many ways in which our current relatively primitive system might be enhanced, including making it easier to specify an experiment and adding tool support for other aspects of the process of conducting an experiment.

XML is a useful data interchange language, but not very easy to generate or read. A tool is needed to support the initial specification of an experimental platform that will hide the XML and that can present to an experimenter lists of model and view components that can be combined and the type of events that can be logged from each. Given that this information is available from the components via reflection, it would be possible to build a running example of the experimental application at design test. This example could be used to test the configuration before using the application in the experiment itself.

There are several additional aspects of the conduct of an experiment that it would be useful to add to our system. Currently, the system only handles a single trial. Typically data will be collected during a number of trials, with different user participants and different conditions (e.g., counterbalanced combinations of user, location and user interface version). We intend to add to the framework an Experiment Manager component that holds this information, read from an augmented experiment configuration file, so that trials can be set up and run either automatically, in sequence, or via experimenter selection. The Experiment Manager is distinct from the current Manager component in the framework that can only handle a single trial.

As the amount of logged data increases, e.g., via multiple trials, one might run into storage difficulties due to the limited memory of a handheld device and the space occupied by other application data, such as a geographic database. Our system will have to take appropriate action in such cases, including compressing the logfiles during creation, transferring to other devices if possible or alerting the experimenter to a possible loss of data prior to data loss. In the latter case, this should occur between trials based on an analysis of the amount of data logged in previous trials and the current space available. This would give the experimenter adequate warning to take action to make more memory available.

More ambitiously, we would like to add the ability to combine the logged data with data collected concurrently by one or more observers. For example, an observer might use a separate hand-held, entering time-stamped notes or experimental protocols, or taking photos or videos or audio recordings. These could be combined with the logged data later, if the timed data can be suitably synchronized.

Indeed, if the experimenter is using a separate device in the field, such as another handheld or a laptop, additional experimental support is possible. For example, using a peer-to-peer wireless connection between the participant's handheld and the experimenter's device, the experimenter may be able to monitor the handheld application (see real-time logged data, view a copy of the participant's screen) or modify the application if necessary. Also, it may be possible to shift data to the experimenter's device as a backup or to free memory on the participant's handheld.

It would also be desirable to integrate additional tools for data archiving, preparation and analysis in to the overall system. However, these operations are unlikely to be performed on the handheld device and thus are not a particular issue for the support of mobile-oriented experiments.

5. CONCLUSIONS

Interaction with mobile devices, such as handhelds, and the user interfaces that support such interaction, remain less well understood than with desktop applications. Ironically, it is more difficult to collect logged data from handheld applications than from workstations. In addition, although several experiment management tools exist for desktops, little has been done in this area for the evaluation of handheld devices, with its different characteristics and challenges. Experiments in the mobile domain have a greater need for an integrated experiment environment and for methods for managing multiple data sources and for coping with limited memory and resources.

Our approach, as reported in this paper, has been to provide a relatively simple tool that makes it easy for evaluators to construct experimental prototypes and to log data from them. Although this system is in its early stages, it provides a useful framework for managing experiments on handhelds and a useful basis on which to build other features and tackle the other challenges of this area.

ACKNOWLEDGEMENTS

This work was funded by SHEFC through the UTOPIA project (grant number: HR01002), which is investigating the design and development of usable technology for older people. We would also like to thank Kartik Khammampad who built and evaluated a navigation system for us using an interface similar to that shown in Figure 1(c) and Professor Steve Brewster for his useful comments on an earlier draft.

REFERENCES

[1] Al-Qaimari, G. and McRostie D., *KALDI: A computer-aided usability engineering tool for supporting testing and analysis of user performance*, in Blanford, A., Gray, P. and Vanderdonckt, J., editors, *Proceedings of IHM-HCI'2001*, Lille, France, 2001, People and Computers XV, Springer Verlag, pp. 153-169.

[2] Bradley, N.A. and Dunlop, M.D., *Understanding contextual interactions to design navigational context-aware applications,* in Paternò, F., editor, *Proceedings of Mobile HCI 2002*, Springer, LNCS 2411, 2002, pp. 349-353.

[3] Eisma, R., Dickinson, A., Goodman, J., Syme, A., Tiwari, L., Newell, A.F., *Early User Involvement in the Development of Information Technology-Related Products for Older People,* Universal Access in the Information Society, to appear, 2003.

[4] Goodman, J., Gray, P.D., *A Design Space for Location-Sensitive Aids for Older Users,* in Schmidt-Belz, B. and Cheverst, K., editors, *Proceedings of HCI in Mobile Guides*, workshop at Mobile HCI 2003, Sep 2003, pp. 12-16.

[5] Gray P.D., Draper, S.W.D., *A Unified Concept of Style and its Place in User Interface Design,* in Sasse, M.A., Cunningham, J. and Winder, R.L., editors, *Proceedings of HCI'96*, People and Computers XI, Springer, 1996, pp. 49-62.

[6] Griffiths,T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., Pinheiro da Silva, P., *Teallach: a model-based user interface development environment for object databases,* Interacting With Computers, Vol. 14, No.1, 2001, pp. 31-68.

[7] Hammontree, M., Hendrikson, J. and Hensley B., *Integrated Data Capture and Analysis Tools for Research and Testing on Graphical User Interfaces,* in Bauersfeld, P., Bennett, J. and Lynch, G., editors, *Proceedings of CHI'92: Human Factors in Computing Systems*, ACM Press, 1992, pp. 431-432.

[8] Hilbert, D.M. and Redmiles D.F., *Extracting Usability Information from User Interface Events,* ACM Computing Surveys, Vol. 32, No. 4, Dec 2000, pp. 384-421.

[9] Macleod, M., Bowren, R., Bevan, N., and Curson, I., *The MUSiC Performance Measurement Method,* Behaviour and Information Technology, Vol. 16, No. 4-5, 1997, pp. 279-293.

[10] Noldus Information Technology, Website accessible at http://www.noldus.com.

[11] Pribeanu, C., Vanderdonckt, J., *Exploring Design Heuristics for User Interface Derivation from Task and Domain Models,* in Kolski, C. and Vanderdonckt, J., editors, *Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, 15-17 May 2002), Kluwer Academics Pub., Dordrecht, 2002, pp. 103-110.

[12] Szczur, M. and Sheppard, S., *TAE Plus: Transportable Applications Environment Plus: A User Interface Development Environment*, ACM Transactions on Information Systems, Vol. 11, No. 1, Jan 1993, pp. 76-101.

[13] Uehling D. and Wolf K., *User Action Graphing Effort (UsAGE),* in Katz, I., Mack, R. and Marks, L., editors, *Proceedings of CHI'95: Human Factors in Computing Systems*, Companion volume, ACM Press, 1995, pp. 290-291.

[14] UIML website, http://www.uiml.org