NOKIA

Research Center

NRC-TR-2007-005

Framework for Comparative Usability Testing of Distributed Applications

Kari Kostiainen[1], Ersin Uzun[2], N. Asokan[1], Philip Ginzboorg[1]

[1]Nokia Research Center, Helsinki

[2]University of California, Irvine

http://research.nokia.com

20-Mar-2007

Abstract:

Comparative usability tests are highly useful when the optimal method to perform a certain user task needs to be selected from several alternatives. However, organizing comparative usability tests is laborious, since either complete application implementations or partial user interface mockups need to be implemented for each tested method. Testing distributed applications requires even more work, since implementing mere user interface mockups is not enough; networking functionality needs to be implemented as well to achieve the real user experience. In this paper we present a usability test framework for one distributed application type: pairing methods. Our framework supports automated test sessions, automatic test data collection and error condition simulation. Developing new pairing method mockups for user testing is easy and fast using the framework, and existing pairing method implementations can be plugged to the framework for testing with minimal changes.

Index Terms:

usability testing tool, comparative usability testing, distributed applications, pairing methods

1. INTRODUCTION

When an optimal method to perform a certain user task needs to be selected from more than one alternative proposals, e.g. for standardization or product development, two separate issues should be considered: 1) what is the best *user interaction model* for this task, and 2) what kind of *user interface* presents this model in an optimal way to the user. A comparative usability test is useful to find out both of these issues.

Traditionally, usability tests are done with either complete application implementations or with user interface mockups created just for the usability test at hand. Testing functionalities, such as automatic logging and measurement of timings, are typically not part of existing application implementations and the test organizer needs to do the data collection either manually or with an external tool. Support for testing functionalities can also be added to each existing implementation, but this requires a lot of repetitive work.

Organizing usability tests for *distributed applications* is even more laborious. If existing implementations are not available, implementing mere user interface mockups is not enough, since at least a partial implementation of the networking part is required as well, because the real user experience of a distributed application depends on the interplay of software in two or more devices, and thus synchronization of user interfaces on different devices is needed.

In this paper we present a *usability test framework* for one particular distributed application type: *pairing methods.* Pairing method is a distributed application using which a person can set up a new and secure wireless connection between two devices. Using this framework a usability test organizer can easily develop new *pairing method mockups* for usability testing without having to do any network programming; the framework handles device discovery and synchronization of user interfaces on different devices. The framework also takes care of test event logging, supports automated test sessions and enables easy error condition simulation. Existing pairing method implementations can also be plugged in to the framework for testing with minimal changes. Our current framework implementation is limited to testing pairing methods only, but we discuss the possibility of extending it into a more generic test framework for all kinds of distributed applications.

The rest of the paper is organized as follows. In Section 2 we review related work. Section 3 identifies problems related to usability testing of distributed applications and defines requirements for an automated usability test solution. In Section 4 we introduce the device pairing problem. Section 5 describes our usability test framework in detail and Section 6 presents the current implementation. Section 7 contains analysis and Section 8 presents ideas for future work. Section 9 concludes the paper.

2. RELATED WORK

Several software tools have been developed to automate certain aspects of usability testing. These tools typically provide support for one or more of the following features:

- **Automatic event capturing:** the events that the user generates during the test, such as key presses and mouse events, are automatically captured and logged for later analysis.
- **Automatic analysis:** certain usability aspects of the tested software are automatically analyzed using either a capture log created earlier by the same or different software or by analyzing the source code of the tested software.
- **Automatic user feedback:** some tools also offer the possibility to automate the user feedback collection by presenting questionares after each completed task automatically.
- **Remote testing:** the usability testing is done over network connection and the usability expert organizing the test can de located at different place (and maybe even in different time).

The existing tools can be also categorized by their target application type. Most of the existing tools are developed to analyze either websites or WIMP (window, icon, menu and pointing device) applications.

2.1 Website testing tools

Automated Summative Evaluation (ASE) [1] is an automated system for remote testing of websites. ASE has a separate control window besides the normal browser window which is used to present tasks to the test user. ASE captures events when the user performs these tasks and automatically collects user feedback for each task. The collected data is saved into a database, analyzed and then made accessible through an interactive webpage.

Test Environment Automation (TEA) [2] is a somewhat similar tool for website testing. Using TEA the test organizer can define automated test sessions that consist of tasks that the test person must perform. TEA displays desired websites to test persons, captures browsing events, and automates the collection of survey data by presenting questions to test persons after each task.

WebQuilt [3] is a proxy-based tool for browsing event capturing which can be used to test any website using any client browser. The user events are captured to the central proxy and analyzed there. WebLogger [4] is

another tool for website browsing logging. WebLogger is a standalone Windows application that starts an instance of Internet Explorer browser and records browsing events to a log file.

WebSAT [5] is a static analyzer tool that inspects HTML files for potential usability problems. WebSAT has its own set of usability rules that identify problems related to e.g. readability, maintainability and form use. WebXACT [6] is another HTML analyzer tool.

2.2 WIMP testing tools

GUITESTER [7] is a usability testing tool for Windows GUI applications. This tool records user-application interactions into log files, generates usability analysis data from these log and visualizes the resulting data. GUITESTER uses hooks to Windows operating system for collecting data and it analyzes the log files by searching predefined patterns.

KALDI [8] is Java based tool that records user actions and captures a video-like recording of the user interface being tested. It also allows for the detailed analysis of the recorded user actions through visualization of logged data.

User Action Graphic Effort (UsAGE) [9] records the actions that a user makes while performing a predefined application task. Prior to a usability testing session, an *expert user* is recorded performing a task. Later, during actual usability testing, the real test user is recorded performing the same task. The action recordings of the two users are compared by the tool and the comparison results are shown graphically.

3. PROBLEMS AND REQUIREMENTS

As discussed in the previous section, many tools have been developed to automate the user event capturing, log data analysis and user feedback collection for both websites and WIMP applications. However, none of the existing tools address

1. the problems related to organizing *comparative usability tests* in which several alternative methods are tested, and
2. the problems related to organizing usability tests for *distributed applications*.

In this section we discuss these problems and identify requirements for an automated usability test solution for distributed applications.

Implementing several new methods for comparative usability testing is laborious, especially if the application at hand is distributed application which requires network programming to achieve the real user experience. The test organizers are typically usability experts who may know how to develop user interfaces, but implementing networking functionality, such as device discovery and synchronization of user interfaces on two or more devices, might be very time consuming for them.

The solution should make implementing new *distributed application mockups* for usability testing as easy and fast as possible by providing all networking functionality. The test organizer developing the user interfaces should be able to concentrate on the user interface itself and spending a great amount of time in developing network related code should not be needed.

Besides making implementation of new distributed application mockups easier and faster, the solution should support utilizing existing implementations of distributed applications. If changes or updates to existing applications are needed, these changes should be minimal.

When evaluating different methods, especially for security related tasks, it is important to understand how the test users behave in *error conditions*. Unfortunately, testing error conditions is often difficult with existing implementations. One approach is to implement a real source of errors, such as a functional man-in-the-middle attacker for security software, but this requires complicated programming and maybe even some special hardware. Another approach is to add error condition simulation functionality to each existing implementation, but this requires lot of repetitive work, since the same simulation functionality needs to be added to each tested implementation separately.

The usability test solution should provide some way of simulating error conditions without having to make changes to the existing implementations or mockups, or having to implement a complicated real source of errors.

Another problem in comparative usability testing is that several different applications are tested. The existing tools typically automate the usability testing of a single application (or website), but if more than one application is tested, each application has to be started manually either by the test user or the test organizer.

The solution should provide automated test sessions and take care of the transition from one tested application to the next one. Also the order in which different applications are tested affects the opinions of test persons, and thus randomized test order should be possible.

Besides the above discussed features the solution should support traditional automatic event capturing and data analysis. As a summary, we list the main requirements for an automated usability test solution for distributed applications:

1. Easy implementation of new distributed application mockups without any network programming
2. Possibility use existing distributed application implementations with minimal changes
3. Easy simulation of error conditions

4. Automated test sessions with possibility of random order of tested applications
5. Automated event capturing and data analysis

4. DEVICE PAIRING

Setting up a secure wireless connection between two devices, such as two Bluetooth enabled phones, is a common and challenging task for end users. This process of setting up the secure connection is often called *device pairing*. Recently, various different methods have been proposed both in research literature [11], [12], [16] and standardization [13], [15], [17] to tackle this pairing problem. In all of the proposed methods the end user needs to perform some user interaction, such as

- enter a short numeric value into both of the devices,
- compare two short numeric values shown on the devices, or
- move the two devices next to each other so that needed initialization data, such as keys and device identifiers, can be exchanged using an out-of-band channel, such as NFC [14] or infrared.

In some of the pairing methods, the user's inability to carry out the needed user interaction leads merely to unsuccessful pairing attempt. If the user for example fails to enter the same short numeric value into both of the devices, no connection is established. However, in other methods the failed user interaction might lead to a pairing with an unintended and possibly malicious device. If, for example, the user reports that the two numeric values shown on the devices are the same even when they really are not, the user could be pairing with an attacker.

Thus, when evaluating different pairing methods, e.g. when selecting a pairing method in standardization, an extensive comparative usability study is needed to find out 1) which user *interaction model* is the easiest for users and 2) what kind of *user interface* presents this model in the least error prone way to the user. Without an extensive comparative user study it is impossible to know which method is the best.

This problem was our motivation to start developing a usabilty test framework for distributed applications.

5. USABILITY TEST FRAMEWORK FOR PAIRING METHODS

In this section we describe a *usability test framework,* an automated solution for usability testing of different pairing methods.

5.1 Test framework architecture

The usability test framework architecture is illustrated in Figure 1. The framework is used to test pairing methods running simultaneously on two devices. The test framework is installed into both test devices: one of the devices acts as a *master device* and the other is called *remote device*.

The framework in the master device is responsible for orchestrating the test session. First, it discovers the remote device using a *control channel*, such as Bluetooth or WLAN link, and establishes a connection to the remote device. Then, the master framework determines the order in which different pairing methods are tested and starts pairing methods on both devices by sending commands over the control channel to the remote device.

Once the user interaction with the test user is finished on the remote device, the remote framework sends the results to the master framework which analyzes the results together with the results obtained from the local pairing method and writes an entry to a log file.

5.2 Creating new pairing method mockups

To create a new pairing method mockup that is compatible with the framework, the developer needs to create a normal functional user interface and additionally implement two extra software modules: *glue* and *evaluator* (see Figure 1).

The glue is a simple wrapper that ties the user interface of a pairing method to the usability test framework. The glue must implement a few functions that the framework uses for

- starting the pairing method,
- sending data to the pairing method from the pairing method on the other device, and
- closing the pairing method.

Likewise, the framework provides a set of functions for pairing methods. With these functions, the pairing method can:

- indicate that the user interaction is finished and give the results to the framework, and
- send data to the pairing method on the other device.

Typically, distributed applications have certain roles. For example pairing application user interfaces act differently depending on whether this device started the pairing or responded to pairing attempt started by another device. The framework informs the glue about the role when the pairing method is started and the glue may adapt the user interface of the pairing method accordingly.

The second extra module that each user interface implementation should provide is an evaluator object. The master framework uses the evaluator object to analyze the test results from both devices. Test result evaluation is explained in more detail in Section 5.4.

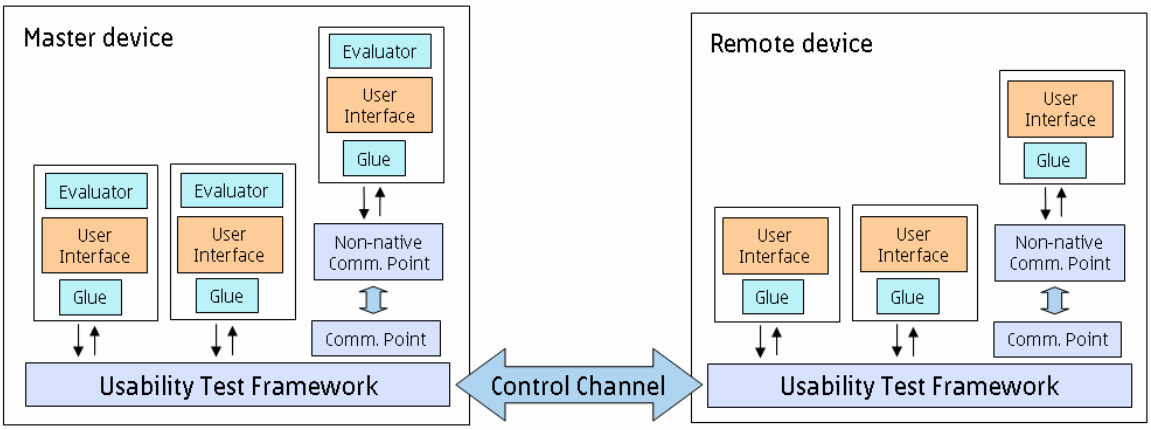

Figure 1: Usability test framework architecture

5.3 Plugging in existing implementations

The framework also supports plugging in existing pairing method implementations. If the existing implementation has been written using the same language as the framework, the pairing method implementation only needs to be updated to implement a glue and evaluator object. If the existing implementation has a functional networking part, this can co-exist with the framework's control channel. Alternatively, the existing implementation can be modified to use the communication functions provided by the framework.

Existing implementations written in a different programming language than the framework can also be plugged in to the framework. The framework has a module called *communication point* that is able to communicate with a *non-native communication point* written in another programming language (see Figure 1).

When the framework needs to communicate with a pairing method implementation written in another programming language, it sends a message to the non-native communication point over a local socket connection and the non-native communication point converts this received message to a function call for the non-native pairing method implementation. When a function call is made on the non-native pairing method implementation, it is converted to a local socket message and sent in the reverse direction. The communication point receives the message and transforms it to the corresponding function call for the framework.

Only one non-native communication point needs to be implemented for all the existing implementations that are written using the same non-native programming language.

5.4 Test input generation and result analyzing

In a test session the framework starts both pairing methods with the same randomly generated numeric *start value*. The framework does not need to know what each pairing method does with the start value. One pairing method might show this value to the user and ask the user to compare this value with the value shown on the other device, while another pairing method might ask the user to type in the value to the other device.

Once the user interaction is finished, the pairing method returns the test results to the framework. The test results contain the *user input* from the test person and possibly some other data specific to the pairing method. The user input could be e.g a PIN code typed into the user interface or YES/NO answer selected by the user, depending on the type of the pairing method. The framework measures the amount of time that the user interaction took. The elapsed time, the original start value and the test results containing the user input are combined into a *test result data structure*.

The framework does not know how to interpret the result data structure, since it is dependent on the pairing method type, but it just sends the whole structure to the master device over the control channel. On the master device the framework uses the evaluator object to determine whether the test was successful or not (the user interactions on both of the devices determine the overall result of the test).

Once the test results are analyzed, the master framework creates a log entry for this test round. This log is written in human readable form for the test organizer.

5.5 Control user interface

The usability test framework provides a *control user interface* that the test organizer can use for configuring test session parameters. The configurable parameters are the order in which different pairing methods are tested (random order is one alternative) and whether or not errors conditions, such as those that would occur during a man-in-the-middle attack, should be simulated in the test session. In practice, simulation of error conditions means that the two pairing methods are

simply started with different start values. The start values are part of the result data structure and thus the evaluator object may determine if the attack was successful or not.

The test organizer uses the control user interface also for starting a test session before giving the devices to the test user. During the start up, the test organizer configures the roles of the devices (master or remote) from the control user interface.

After the test session, the test organizer may use the control user interface to read the test session results from the recorded log on master device.

6. IMPLEMENTATION

We have implementated the usability test framework for pairing methods using Java MIDP [18]. In this section we describe our current implementation.

6.1 Framework implementation

We selected Java MIDP as the implementation platform for the framework, since it is supported by most of the current mobile phones and PDAs, and thus the framework implementation can be used in wide range of portable devices. We have tested the framework with Series 60 3rd edition [20] phones, such as Nokia E60 and Nokia N80, but the framework should work with any device that has a relatively new Java MIDP runtime installed. Porting Java MIDP code to standard Java is straightforward, so our framework can be ported to PCs as well.

The control channel in our implementation is Bluetooth connection and we use the Java Bluetooth API (JSR-82). For the communication between the master and the remote devices we use a simple text based protocol.

6.2 Implemented user interfaces

We implemented several new pairing method mockups for usability testing using the framework. Some of these mockups are presented in more detail in [10]. Each implemented mockup contains a simple glue wrapper and evaluator object in addition to the normal user interface code.

We also had one existing pairing implementation [19] which was written in C++. We wanted to plug in this existing implementation to our framework and therefore we implemented a non-native communication point in C++ for Symbian.

The non-native communication point is a simple Symbian application that starts waiting for local socket connections when started. When the test framework needs to start a user interface implemented in C++ it creates a local socket connection to the C++ communication point and communicates by sending and receiving messages over the local socket connection. Also for this local socket communication we use a simple text based protocol.

6.3 Using the framework

The test organizer starts a test session using the control user interface of the framework (illustrated in Figure 2) before giving the devices to the test person. The test organizer selects that one device should act as the master device (a) and another device should act as the remote device (b). Then, the test organizer starts the framework from both devices (c) and the framework on the master device finds the remote device using the control channel (d) which is Bluetooth in our implementation. At this point, the devices are ready to be given to the test user.

A typical test session is illustrated in Figure 3. The test user takes the devices and starts the first tested pairing method by following the instruction on the screen (a). The first pairing method user interface is presented to the user (b) and the test user performs the required user interaction, such as comparing values shown on both devices. After the first test is completed, the user is guided to proceed to the next tested pairing method (c). The user interface of the second method is shown (d) and the test user performs the required user interaction, such as entering a value shown in one device into the other device. This sequence is repeated until all pairing methods have been tested.

Once the whole test session has been completed, the test organizer may take the devices back from the test user and read the results from the log file that the master device generated (illustrated in Figure 4).

6.4 Limitations of the current implementation

Our current implementation has one limitation: configuring test session parameters from the control user interface is currently not possible. Instead, the order in which different methods should be tested and whether or not error conditions simulation should be included are currently configured at the test framework startup code and the test session organizer has to compile a new version of the framework each time these settings are updated.

6.5 Framework source code

We have open sourced our framework implementation and the latest version is available online [21].

7. ANALYSIS

7.1 Evaluation of the framework

In Section 3 we presented requirements for an automated usability testing solution for distributed applications. In this section we analyze our framework against those requirements.

Easy implementation of new distributed application mockups: Using our framework implementation of new pairing method mockups is simple, since the framework handles all networking issues.

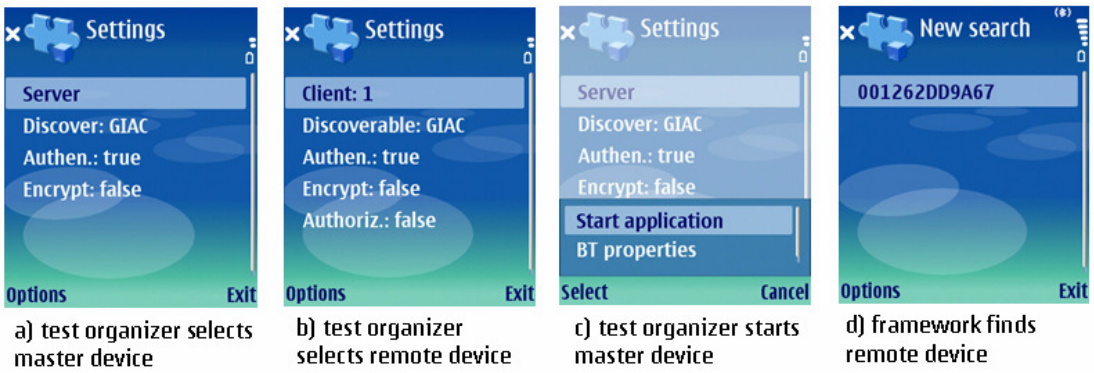

Figure 2: Usability test framework control user interface. The test organizer uses this user interface to setup the device ready for usability test session before giving the devices to the test person.

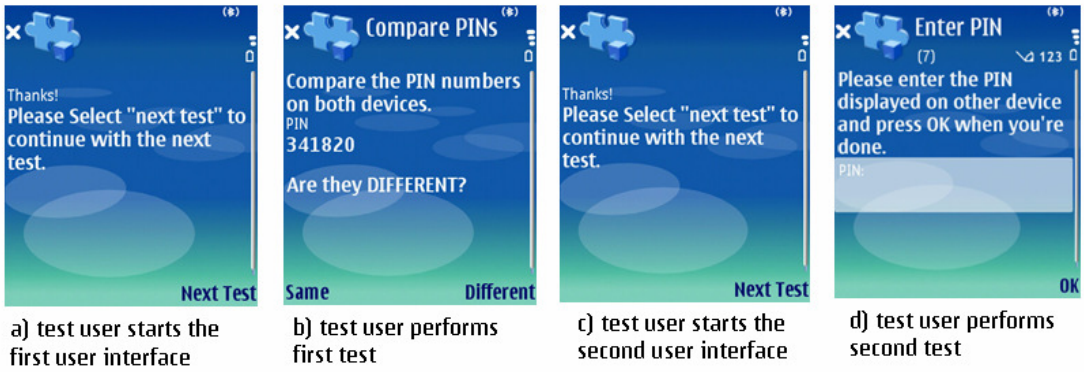

Figure 3: Usability test session screenshots. The usability test framework guides the test user through the tests session.

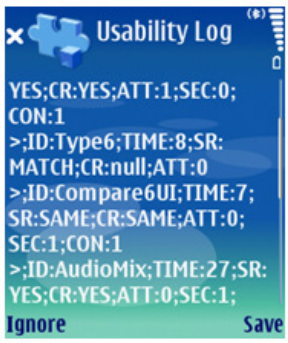

Figure 4: The test organizer may read the test session results from the log the master framework generated.

Because of the framework two additional modules, glue and evaluator object, have to be implemented in addition to the normal user interface code. This brings some additional work for the test organizer, but it is considerably easier and faster than implementing the networking functionality for each tested method.

Possibility to use existing distributed application implementations: The framework also supports plugging in existing pairing method implementations. If the existing implementation is also written in Java, then modifying it to be compatible with the framework is easy. If the existing implementation is written on a different programming language, then a bit more work is required. We implemented a non-native communication point for Symbian C++ pairing methods which required some effort. However, now that the non-native communication point is implemented existing Symbian C++ pairing method implementations can be modified to be compatible with the framework easily.

Easy simulation of error conditions: The framework provides a convenient way to simulate error conditions by starting pairing methods with different input values. As compared to implementing a real functional man-in-the-middle attacker for each tested pairing method separately, using our framework for attack simulation is clearly more convenient.

Automated test sessions: Our framework provides automated test sequences which makes organizing usability tests convenient for both the test organizer and the test person. The test organizer does not have to start each tested application for the test person separately, since the framework takes care of the transition from one tested method to another automatically.

Automated event capturing and data analysis: The automated test event capturing provided by the framework eliminates the error possibility related to human observer and makes organizing test easier for the test organizer. The systematic data collection also enables convenient post processing of usability test results.

7.2 Experiences from real usability testing

We have used this framework to organize a comparative usability test of three alternative pairing methods. This test and the results are presented in more detail in [10].

Using the framework in our usability study helped us to develop realistic pairing method mockups quickly in the preparation phase and improved the testing experience considerably. The users were comfortable with the automatic transition between different methods and usability expert could just concentrate on observing the participant without worrying about accurate timing or logging. One person was enough to conduct the tests and the systematic data collection made post processing the results easy.

7.3 Comparision with related work

We are not aware of any other software tool that would address the problems related to organizing comparative usability tests of distributed applications. Probably the most related work is Automated Summative Evaluation (ASE) [1] which also provides automated test sessions, event capturing and analysis. In addition, ASE automates the survey data collection which is not supported by our framework. However, ASE is limited to testing of websites only, and thus it cannot be used for testing distributed applications.

8. FUTURE WORK

Two directions for continuing this work could be considered: 1) providing better support for long-lasting unattended user tests and 2) generalizing the framework for all kinds of distributed applications. In this section, these aspects are discussed respectively.

8.1 Better support for unattended user tests

The fact that user tests are performed at the presence of usability test organizer may affect behaviour of the test users. This is especially significant for testing security software, such as pairing methods. The security software might show some warning to the user or ask the user to perform a small task, such as compare two numbers. At the presence of the usability test organizer the test user is more likely to be alert and carefully read the instructions. However, in real life the user has another goal (such as synchronizing his phone calendar with his PC) in mind while interacting with the security software. If the tests would be performed in more realistic context, e.g. at the home of the test person when the user is trying to achieve another goal, the results would be more reliable.

Our framework could be modified to provide support for this kind of unattended user tests. The framework could be running in the background of the participant's device e.g. one week. During that time, the framework could ask the user to perform one of the device pairing methods each time user does some operation that normally requires device pairing, such as synchronization. The framework could be configured to upload the collected results to a central server in the Internet or to user's PC over a local link on regular basis.

8.2 Generalizing the framework

Currently, our framework can only be used to test different pairing methods. How the framework could be extended into a more generic usability test framework for all kind of distributed applications remains an open research question.

Depending on the tested user task and method, the data that needs to be collected may change drastically. In the current implementation, each pairing method has to provide its own evaluator object since the framework does not know how to analyze the result data. If the framework was generalized for all kinds of distributed applications, also the start value generation would have to be isolated from the framework. Each method implementation (or a group of implementations) would have to provide their own *start value generator* that the framework could use for generating a suitable start values for testing these method.

9. CONCLUSION

In this paper we have presented a usability test framework for one distributed application type: pairing methods. The framework makes implementing new pairing method mockups simple and existing implementations can be plugged into the framework with moderate effort as well. The framework supports automated test sessions, simulation of error conditions and automated test event capturing and analysis.

REFERENCES

[1] Ryan West, Katherine Lehman. Automated Summative Usability Studies: An Empirical Evaluation. In Proceedings of CHI 06: Conference on Human Factors in Computing Systems. Pages 631-639. Montreal, Canada. April 2006.

[2] Hartmut Obendorf, Harald Weinreich, Torsten Hass. Automatic Support for Web User Studies with SCONE

and TEA. In Proceedings of CHI 04: Conference on Human Factors in Computing Systems. Pages 1135-1138. Wien, Austria. April 2004.

[3] Jason Hong, Jeffrey Heer, Sarah Waterson, and James A. Landay: WebQuilt: A Proxy-based Approach to Remote Web Usability Testing. ACM Transactions on Information Systems. Volume 19. Issue 3. Pages 263-285. July 2001.

[4] Robert Reeder, Peter Pirolli, Stuart Card. WebEyeMapper and WebLogger: Tools for analyzing eye tracking data collected in Web-use studies. Extended Abstracts of CHI 01: Conference on Human Factors in Computing Systems. Pages 19-20. Seattle, WA. March-April 2001.

[5] Web Static Analyzer Tool (WebSat). Available at: http://zing.ncsl.nist.gov/WebTools/WebSAT/overview.html

[6] WatchFire. WebXact. Available at: http://webxact.watchfire.com/

[7] Hidehiko Okada and Toshiyuki Asahi. Guitester: A log-based usability testing tool for graphical user interfaces. IEICE Transactions on Information and Systems. Volume E82-D. Number 6. Pages 1030-1041. 1999.

[8] Ghassan Al-Qaimari, Darren McRostie. KALDI: A computer-aided usability engineering tool for supporting testing and analysis of human-computer interaction. Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces. Louvain-la-Neuve, Belgium. October 1999.

[9] Dana Uehling, Karl Wolf. User Action Graphing Effort (UsAGE). CHI 95: Conference on Human Factors in Computing Systems. Pages 290-291. Denver, Colorado. May 1995.

[10] Ersin Uzun, Kristiina Karvonen and N. Asokan. Usability Analysis of Secure Pairing Methods. To appear in USEC 07: Usable Security. Trinidad and Tobago. February 2007.

[11] Serge Vaudenay. Secure Communications over Insecure Channels Based on Short Authenticated Strings. In Proceedings of the Advances in Cryptology, Lecture Notes in Computer Science. Volume 3621. Pages 309-326. 2005.

[12] Sven Laur, N. Asokan, Kaisa Nyberg. Efficient Mutual Data Authentication Using Manually Authenticated Strings. Cryptology ePrint Archive. Report 2005/424. 2005.

[13] Bluetooth Special Interest Group. Simple Pairing White Paper. August 2006. Available at: http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf

[14] Near Field Communications Forum. Available at: http://www.nfc-forum.org/home

[15] WiFi Alliance Protected Setup. Announcement August 2006. Available at: http://www.wi-fi.org/news/pressrelease-081606-WiFiProtectedSetup/

[16] Dirk Balfanz, Glenn Durfee, Rebecca Grinter, Diana Smetters and Paul Stewart. Network-in-a-Box: How to set up a secure wireless network in under a minute. In Proceedings of 13th USENIX Security Symposium. Pages 207-222. San Diego, CA. August 2004.

[17] Wireless USB Specification Revision 1.0. Association Models Supplement Revision 1.0. March 2006. Available at: http://www.usb.org/developers/wusb/wusb_2006_0302.zip

[18] Java Mobile Information Device Profile (MIDP). Available at: http://java.sun.com/products/midp/

[19] Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen, N. Asokan. Secure Device Pairing based on a Visual Channel. IEEE Symposium on Security and Privacy. Oakland, CA. May 2006.

[20] Series 60. Available at: http://www.s60.com/

[21] Comparative Usability Framework Project. Available at: http://sconce.ics.uci.edu/CUF

