SmartVNC: An Effective Remote Computing Solution for Smartphones*

Cheng-Lin Tsao, Sandeep Kakumanu, and Raghupathy Sivakumar
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
{cltsao,ksandeep,siva}@ece.gatech.edu

ABSTRACT

While a remote computing solution such as VNC is an effective technology for PC users to access a remote computer, it is not as effective while being used from smartphones. In this paper, we propose techniques to improve remote computing from smartphones that help deliver near-PC level experience to users. We introduce a key building block called *smart-macros* that have the robustness of application macros but at the same time possess the generality of raw macros. Using *smart-macros* we design and prototype *SmartVNC*, a remote computing solution for smartphones. We show using experimental studies and a trace based analysis of real user activity, that SmartVNC can improve user experience considerably.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication-Networks**]: Distributed Systems—*client/server*

General Terms

Algorithm, Design, Experimentation, Performance

Keywords

Smart-macros, Remote computing, Remote desktop, VNC, Smartphones

1. INTRODUCTION

Remote computing software allows users to remotely access their desktops. The need for such remote access is driven by a wide variety of application scenarios ranging from simple remote access of files and data, to mobile workers who have access to certain applications only on their work PC, as well as to remote IT support. The more recent emergence of virtual desktop infrastructures (VDIs) that almost exclusively rely on remote computing software for access has further elevated the importance of the latter. There are currently over *sixty* different remote computing software offerings in the market [1].

The rapid adoption of smartphones by mobile users has led to users relying on remote computing software on their smartphones to access their PC. Users being able to access their PC effectively from their smartphone has both convenience and productivity implications. Unfortunately, remote computing software available for smartphones are ported on an as-is basis from their PC counterparts and hence do not provide good user experience. The unique limitations of smartphones in terms of a constrained form-factor impose severe overheads on the user, making even mundane tasks burdensome.

To improve the performance of remote computing from smartphones, we present a solution called *SmartVNC*. SmartVNC incorporates the following four novel design elements: 1) A new class of application agnostic macros called *smart-macros* that have the generality of traditional raw macros, but at the same time possess the robustness of application macros[1]; 2) A smartphone friendly interface for the playback of the *smart-macros* that allows for seamless mixed use of raw input and macros, and provides several acceleration techniques for navigation of the remote computing view on the smartphone; 3) Macro parameterization and preemption capabilities that allow users to accomplish tasks that are variations of the tasks for which the *smart-macros* were created for; and 4) An offline macro recommender that transparently monitors the user behavior on a PC and suggests useful *smart-macros* that represent tasks frequently performed by the user.

While we present SmartVNC as a two-ended solution atop the VNC remote computing software for the Microsoft Windows operating system, the solution design can be applied to other remote computing protocols such as RDP [2] and other operating systems. On the smartphone, the SmartVNC client is tightly integrated with the VNC client, but on the desktop, the SmartVNC server requires no integration with the VNC server and co-exists in a fully transparent manner. The SmartVNC solution allows users to *create extensible and robust* smart-macros *on the PC for any application, and invoke them from the smartphone using a simple interface overlayed on the default VNC client.* Through real user ex-

*This work was supported in part by the National Science Foundation under grant CNS-1017234.

MobiCom'11, September 19–23, 2011, Las Vegas, Nevada, USA.

[1]Definitions of raw and application macros are provided in Section 3.3.

periments with the SmartVNC prototype, we show that the user experience on smartphones can be improved to match PC levels both in terms of time and effort required to accomplish remote tasks from the smartphone. We also show that the overheads incurred by SmartVNC are minimal.

The rest of the paper is organized as follows: Section 2 presents background on remote computing software. Section 3 establishes the motivation and case for SmartVNC. Section 4 presents the design highlights of SmartVNC while Section 5 presents the system details. Section 6 presents performance results, and Section 7 discusses the related work. Section 8 concludes the paper.

2. BACKGROUND

There are two basic forms of remote computing software. The first type is where information is passed between the server and the client in the form of raw pixel data. The VNC remote computing application is an example [3]. The VNC server encodes the pixel data of the remote computer and sends the encoded bit stream to the VNC client, which then decodes bitstream and renders the screen display on the local computer. To reduce the amount of data transferred, the VNC server periodically polls the pixel data of the full screen of the remote computer to detect the regions that are updated, and sends only the changed portion of the screen instead of continuously sending the full screen frames of the remote computer. VNC is a cross-platform solution and can work across multiple operating systems for both the remote and local computers. Examples of solutions based on VNC are UltraVNC and RealVNC [3]. The second type is based on graphical primitives, which are basic drawing commands provided by the operating system. RDP [2] is a Microsoft application and protocol that falls under this category.

In the rest of the paper, our discussions of the high level design elements apply to both types of remote computing software. However, most of our experiments and our prototype rely on VNC as the baseline for the remote computing server, and AndroidVNC as the remote computing client [4]. The presented results and proposed solutions could be applied to other smartphone remote computing clients such as MochaSoft RDP, iTeleport, TeamViewer, and LogMeIn ignition, but such an exploration is out of scope of this work.

3. MOTIVATION

While VNC and other remote computing solutions are mature and effective solutions to provide remote access to a PC from *another PC*, they are not explicitly designed for remote access from *a smartphone*. Remote access from a PC is intuitive to a user since the local PC provides a user interface that is equivalent to that of the remote PC. The full screen display is shown on the local monitor, and the user controls the remote PC using a mouse and a keyboard at the local PC. In such a scenario, the overall user experience of remote computing is close to that of using a local PC. However, remote computing from a smartphone is significantly more difficult because of device constraints including a small form-factor and screen size, and the lack of a physical mouse/keyboard. We characterize the degree of such difficulty with a metric called *task effort*, which is defined as the number of operations needed to accomplish a computing task. We define an operation as a mouse click or a keystroke in the PC, and as a touch (including panning and tapping) on the touch screen in the smartphone. When a user wishes to accomplish a task from a smartphone, the user performs the same sequence of operations that would be required on the PC but now via the VNC interface. Each PC operation would require one or more operations when performed in the smartphone. The task effort in accomplishing a task from a smartphone via VNC can be represented with the following equation:

$$TaskEffort^{VNC} = TaskEffort^{PC} \times Inflation \qquad (1)$$

where *Inflation* is the factor representing the additional burden imposed on the user by the limitations of the smartphone, and $TaskEffort^{PC}$ is the number of operations required to accomplish the same task directly on the PC. In the rest of this section, we first provide qualitative reasons that cause such *Inflation*, and then we analyze PC usage traces collected from different users and show that there are considerable amounts of usage redundancy to be leveraged to reduce $TaskEffort^{PC}$. In Sections 4 and 5, we show how both factors are addressed by SmartVNC.

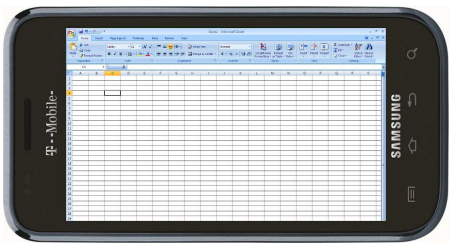


Figure 1: Screenshot of a smartphone VNC client shows an intricate interface

3.1 Inflated Effort in Remote Computing from Smartphones

The inflation in task effort for remote computing from a smartphone is attributable to the following reasons: (i) *The zoom problem:* By default, the full desktop screen is squeezed into the small screen on the smartphone (see Figure 1 that shows a screenshot of MS Exel via AndroidVNC [4]). This renders it very cumbersome to directly access or manipulate any single GUI element on the desktop. Doing so requires the user to zoom in, which is commonly done using *pinching* on newer smartphones. However, such zooming requires the user to perform additional operations. (ii) *The pan problem:* Zooming comes with a by-product problem. Once the screen is zoomed in, not all GUI elements on the original desktop screen is now visible to the user, and panning is required to navigate the full screen. This typically requires the user to swipe across the screen and further increases the number of operations the user has to perform. (iii) *The keyboard problem:* Many smartphones do not support a full fledged keyboard. Hence, to access keys not available in the default layout, users will have to press additional buttons on the keyboard. For example, on the stock Android keyboard, users will have to press the '123' button to access the number keys. Some keys are specific to PC and are unavailable in smartphone keyboard at all, such as 'Ctrl' and 'F1' that requires a custom implementation in the remote computing client to perform. (iv) *The error problem:* Users tend to make more mistakes on the smartphone when performing operations. Undoing such mistakes

and re-performing the operations would require extra operations. SmartVNC is explicitly designed to reduce *Inflation* but we defer discussions on how it accomplishes the reduction till later in the paper.

3.2 Measurement of Redundancy in User Activity

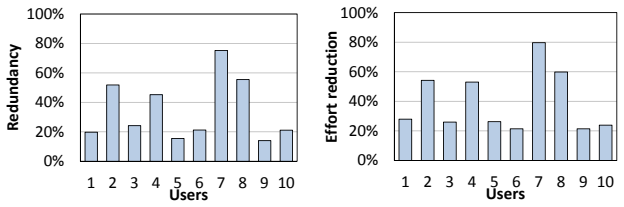
While reducing the *Inflation* factor is one approach to reduce $TaskEffort^{VNC}$, another complimentary approach we consider is the reduction of $TaskEffort^{PC}$ by enabling operations to be performed in aggregates. Such *operation aggregation* would specifically be relevant if users naturally tend to perform *redundant aggregates of operations*. Before we delve any further into ways to accomplish this, we now briefly present results from a user activity analysis to study whether there is redundancy in user operations.

The PC user activity trace is collected from ten different volunteer users spanning both academia and the industry. Each user is provided with a custom-built monitor utility that captures all operations performed by the user and exports them onto a stored file. The monitor transparently runs in the background on the PC so that it captures the true activity of the users in their routine usage of the applications. Each user periodically emailed back the stored file and the usage analysis was performed offline. The volunteers collected the traces for an average of 12.5 days.

The usage analysis first involved analyzing the redundancy in user activity, or in other words the amount of repetitive activity that can be reduced by operation aggregation. We define the degree of redundancy as follows: Consider an operation sequence "ABCABCDABE". This contains two repetitive substrings: "ABC" and "AB". Each repetitive substring can be replaced with a new *code*, and the resultant string will reveal the upperbound of the redundancy elimination possible. For example, "ABC" can be replaced with the code X and "AB" can be replaced with the code Y. The original string can thus be reduced to "$XXDY$E", which reduces the length from 10 to 5. The redundancy in the sequence is then calculated to be $\frac{10-5}{10} = 50\%$.

To quantify the redundancy in user activity, we need to first discover repetitive tasks. We define a repetitive task to be a sequence of operations that has appeared in the history at least twice. To analyze the redundancy, we use the user history to perform trace-based evaluation. We greedily match the history with the repetitive tasks that have been identified, and we assume that each repetitive task can be replaced with a single new operation. We consider only repetitive tasks of at least length two for such replacement. Figure 2(a) shows the redundancy of different users participated in the analysis. The redundancy in user activity ranges from 20% to 40% for most users, and the average is 30.96%. *This indicates that an intelligent operation aggregation technique can reduce task effort in a significant fashion.*

There are two factors that have to be considered when translating the above degree of redundancy into effort reduction achievable on the smartphone: (i) The above analysis assumes that the replacement code is always of length one, which obviously is not realistic. Hence, a notion of *lookup overhead* for the newly introduced codes has to be incorporated. We consider the lookup overhead to be approximated as $log_k N$ where k is the number of aggregate operations that can be presented to the user at any given point in time, and N is the total number of such aggregate operations. Although such a lookup overhead can be achieved only if the tasks are structured with a balanced tree with k children at each intermediate node, it serves as a reasonable approximation of the achievable effort reduction. (ii) The analysis also does not account for the mobile inflation factor discussed earlier. Using the remote computing experiments conducted later in Section 6 where users were asked to perform the same tasks on both the PC and the smartphone, we empirically determine the mobile inflation factor to be 3.31. Figure 2(b) shows the effort reduction after adjustment for both the lookup overhead and the mobile inflation factor. The potential effort reduction on the smartphone ranges from 20% to 80%. The effort reduction is related to the redundancy in user activity, which is intuitive. The interesting observation is that the effort reduction can be higher than the redundancy that appears in the user activity in PC because of the potential for reduction in the inflation factor.



(a) Redundancy observed in user activity

(b) Potential effort reduction on a smartphone by user

Figure 2: Real-user activity shows redundancy and potential effort reduction

3.3 Macros and Limitations

The notion of operation aggregation is related to the concept of a macro, which is a sequence of instructions that has been recorded and can be replayed by the user. In this subsection, we discuss two types of macro solutions that exist today and their limitations.

The first type is application macros provided by the application developers in certain application software. One of the most popular application macros is the Microsoft Excel macro [5].

Excel allows users to record their operations in the form of a Visual Basic script. The other example is iMacros [6], which is a browser plug-in that allows users to record their operations when browsing the web. However, each application macro only works for that specific application. The user has to rely on the application developer to provide such functionalities. The scope of the application macros is also limited. An application macro cannot work across multiple applications. Some functions of an application may not be captured by the macro system. For example, iMacros does not record the operation of printing a web page.

The second type is raw macros, which records and replays the raw activities such as mouse clicks at a certain coordinate and keystrokes. An example of raw macro systems is AutoHotkey [7]. Although raw macros work for generic applications in PC, it cannot replay the intended tasks robustly. Since raw macros are defined by the raw system variables, it would fail to replay the recorded task if the system environment is not the same as the recorded state. For example, moving/resizing the window would cause the

mouse click at the same coordinate to activate a different function. Raw macros cannot respond to adaptive user interfaces, such as the truncated menu that shows frequent items in Microsoft Office and other applications. The pacing of the macro replay is also a big challenge with raw macros since the availability of next function to invoke may be delayed due to the current computational load, and raw macros have no information regarding the application context.

4. DESIGN PRINCIPLES

4.1 Overview

SmartVNC is a software solution designed to improve the experience of a user accessing a remote PC from a smartphone. It is a two-ended solution, with presence at both the PC and the smartphone (see Figure 3). The SmartVNC server at the PC co-exists transparently with a remote computing server. The SmartVNC client at the smartphone is integrated as an overlay with the remote computing client. SmartVNC provides a powerful framework for *users to create robust, general and extensible macros* on the PC, *name them*, and *invoke them easily at the smartphone within the context of the remote computing client*. In the rest of this section we focus on the fundamental design elements in SmartVNC: *application agnostic smart-macros* that allow SmartVNC to provide the robustness of application macros but with the generality of raw macros; *task effort reducing front-end* on the smartphone that is presented as an overlay within the context of the remote computing client; *parameterization and pre-emptability* of macros that provide a high degree of flexibility and extensibility; and *offline macro recommender* that analyzes user activity and provides recommendations for macros to be created. Note that the rest of the discussion in this section is not meant to be an exhaustive description of SmartVNC, but rather an overview of the principal design elements. We defer the complete description to the next section.

4.2 Application-Agnostic *Smart-Macros*

Operation aggregation through the creation of *macros* is a desirable capability for task effort reduction. However, as explained in Section 3, traditional approaches to creating macros have been severely limited by one of two problems: (i) raw macros suffer from robustness issues and are notoriously erroneous if the playback environment differs even slightly from the recording environment; and (ii) application macros suffer from a lack of generality and users are dependent on application developers to provide the capability of macros on a per-application basis.

SmartVNC uses a new type of macros that we introduce called *smart-macros* that has the robustness of application macros but provides the generality of raw macros. The design of *smart-macros* is derived from a combination of key trends and observations on how operating systems today work. Most operating systems (OS) today provide application developers with higher order primitives called *GUI elements* than simplify the ability to create graphical objects. Thus, application developers can readily use GUI elements such as text-boxes, buttons, and forms instead of creating them from scratch. Because the OS also assumes the responsibility of providing callbacks to the application when GUI elements are invoked or manipulated, applications register

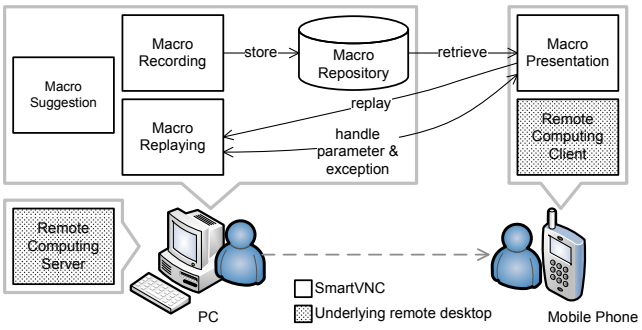


Figure 3: SmartVNC system overview

each GUI element with well identifiable information ranging from the parent application to the specifically recognizable name of the element, as well as to the state of the element. *Smart-macros tap into the GUI element framework directly to facilitate the recording of robust macros in an application agnostic fashion.* Thus, a *smart-macro* at a high level is a sequence of operations with each operation represented as an addressable GUI element with its appropriate state and the raw user input to be delivered to that GUI element.

Figure 4 shows how *smart-macros* compare to the other types of macros. As we explain in Section 5, the UI automation and the .NET frameworks available in MS Windows are used in tandem for tapping intelligently into the GUI element framework and capturing raw user activity. We also discuss how such frameworks are also readily available on other OSes.

4.3 Task Effort Reducing Front-end

Once *smart-macros* are created, they have to be presented to the user on the smartphone. SmartVNC uses a push technique to update the list of macros available on the smartphone, but we defer further discussion of the update mechanism to the next section. More importantly, the actual front-end on the smartphone has to be designed carefully with the following considerations: (i) the front-end has to be *non-intrusive* and should ideally seamlessly co-exist with the remote computing client front-end; (ii) the front-end must be *non-limiting* in terms of what the user can accomplish independent of whether relevant macros are available or not; and (iii) the front-end should be heavily tailored toward reducing task effort.

The SmartVNC front-end is designed to address the above considerations. It is designed as a collapsible transparent overlay to the remote computing client on the smartphone (see Figure 8). The user continues to be able to view and use the regular remote computing client, but can *opportunistically invoke macros from the overlay macros panel*. The user can collapse the macros panel if required. Creation of the front-end in this fashion is straightforward as the SmartVNC

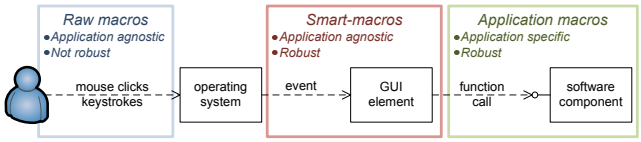


Figure 4: Types of macro solutions

client is integrated with the remote computing client software. When a macro is invoked, the invocation is carried over to the SmartVNC server on the PC out-of-band of the actual remote computing session. However, the playback of the macro at the PC is presented back to the user in real time through the remote computing session. Thus, the user can actually observe the execution of the macro. *More importantly, the user can seamlessly intersperse the use of macros with raw input to the remote computing client interface.*

Finally, the front-end is also designed to minimize the task effort when invoking macros. When the replay of macros requires user-input (see discussion on Parameterization later in the section), the front-end *automatically pans and zooms* to the concerned GUI element that needs manipulation by the user. This eliminates the need for the user to navigate to that element. Note that because the SmartVNC server has the exact information about the GUI element on the PC, identifying its coordinates on the smartphone through the API of the remote computing client is straightforward. Finally, if the GUI element can take default values (e.g. text-boxes, check boxes, radio buttons, etc.) SmartVNC pre-populates the GUI element with the default values derived from the state of the GUI element when the macro was recorded or last replayed whichever was later. The front-end provides users with the option of replaying an entire macro with default values in which case the user is not prompted for input during the replay. Finally, the replay of the macro itself is done faster than real-time minimizing the task-time for users.

4.4 Parameterization and Preemptability

Operation aggregation using macros is useful when the user wants to replay the exact sequence of operations represented by the macros. However, in reality users could want minor variations in tasks every time a macro is replayed. SmartVNC accommodates such variations by supporting *parameterization of the smart-macros.* At a high level when the user records a macro, certain operations can be classified as parameters. The macro is then recorded as a sequence of only the operations with appropriate indicators for when user input should be sought for the parameters. The original input for the parameters is also preserved as default input values for the macro. This provides a powerful abstraction to users as macros can now be used with variations on the fly. To the best of our knowledge, this is the first approach to enable parameterization of generalized macros.

The classification of an operation into either a parameter or not is done by distinguishing *whether the operation merely changes the state of the associated GUI element or invokes the GUI element respectively.* During playback of a parameterized macro, SmartVNC provides users with the option of executing the macro *in continuous mode* with just the default values and not pausing for input. If a user does choose to replay the parameterized version of a macro, an option is still provided to resume the macro in continuous mode after every parameter input.

Another form of macro extensibility SmartVNC supports is the ability to pause macros, perform raw input, and then resume the original macro. This provides a further degree of extensibility than parameterization as macros now can be extended with arbitrary introduction of new operations as well. By default, the pausing of macros is allowed in SmartVNC only when the playback is stopped for user input. However, it is possible to instrument the playback to occur slower than real-time and allow users to pause the macro at any given point in the playback.

4.5 Offline Macro Recommender

The effectiveness of the SmartVNC solution depends on the user creating useful macros. While the user is very likely to know the most important aggregates of operations it is challenging for a user to be able to create all useful macros. Hence, one of the important design elements in SmartVNC is the offline macro recommender. The SmartVNC server monitors user activity even when a macro is not being recorded and logs the activity after classifying parameter operations from others. The macro recommender component of SmartVNC then, on demand, parses the log to generate a list of repetitive tasks observed in the user activity. For each such task, the application, the length, the exact sequence of operations, and the frequency of occurrence of the task are presented. The report is generated simply as readable text. The tasks are first filtered based on user specified length and frequency thresholds, and rank ordered based on decreasing values for $length * frequency$. Users may then choose to explicitly record a subset of the macros specified in the report[2].

The tracking of user activity is done intelligently to account for users switching from one application to another. Thus, if a user performance activity $A1$ at time $t1$ in application $App1$, moves on to another application, and returns back to $App1$ to perform activity $A2$, SmartVNC will track the concatenation of $A1$ and $A2$ as a continuous task. Also, the Macro Recommender only finds the longest matched sequences to eliminate redundantly identifying sub-strings of tasks as also repetitive tasks. Briefly, this is accomplished using a suffix tree for building task patterns from the user activity log. The Macro Recommender takes the history of user activity since the last time a recommendation report was generated, and it inserts all suffixes of the history into the suffix tree. After inserting all suffixes, the Macro Recommender traverses the suffix tree to identify repetitive tasks, which are nodes that satisfy both the length (node depth) and the frequency thresholds. We defer the detail mechanism to the next section.

5. SOLUTION

In this section we present the system realization of the SmartVNC solution. We present the system architecture, the details of our implementation and the various components involved in the building the system.

Figure 5 shows the software architecture and the components that reside in four functional blocks described in Section 4. The solution is designed as a server-client architecture, where the server resides on the desktop and the client resides on the smartphone. Since the SmartVNC design is centered around operation aggregation that is orthogonal to the core remote computing functionality, the solution is implemented as an overlay to an existing remote computing solution but does not change any of the native behavior of the remote computing software. The SmartVNC client requires integration with the remote computing client on the smartphone, but the integration is merely to gain access to the UI and the remote computing protocol is left untouched.

[2]Future work could facilitate users simply choosing an identified task and the macro automatically being recorded.

The SmartVNC server on the other hand is fully decoupled from the remote computing server and has no direct interactions with it. The arrows in the figure show the interaction between components in terms of function calls.

5.1 SmartVNC Server on Desktop

The SmartVNC server resides on the desktop and implements three functional blocks of the solution - Macro Recording, Macro Replaying, and Macro Suggestion. The desktop also maintains a persistent database for all the recorded macros. All the desktop components are developed using C#. The persistent database is an SQL-based relational database called HyperSQL [8]. We use an unmodified realVNC [3] server on the desktop as an independent process. While a number of components constitute the SmartVNC server we explain the key ones below and only briefly summarize the others.

Macro Recorder Frontend: This is a simple GUI application that allows the user to start/stop/abort the recording of a macro and also allows playback for verification. Figure 6 shows a screenshot of the frontend.

GUI Element Extractor: This component has two responsibilities: converting raw operations into GUI elements and retrieving a unique identity for each element. The GUI element extractor uses the APIs provided by the accessibility frameworks for extracting the handle for the GUI element. In the context of MS Windows, the UIA framework provides functions FromPoint() and FocusedElement() to determine the AutomationElement for a mouse entry and a keyboard entry, respectively. Next, the GUI element extractor has to retrieve a unique identity for the GUI element so that it can be reliably located while executing a macro. The AutomationElement has several properties that could be used to identify it, such as name or automation ID. However, even a combination of these properties is not sufficient to uniquely identify an element. Automation ID is not provided by all GUI elements, and multiple GUI elements in a GUI application window can have the same name. SmartVNC, instead, traces the GUI tree hierarchy from the target GUI element back to the root and uses the full ancestor list as the unique identity.

Parameter and State Identifier: To allow parameterization, we categorize GUI elements as a parameter type based on the control type. Table 1 show the classification of parameter and non-parameter operations. We record the operation performed during the recording of the macro as the default value for the parameter. We also maintain the state of those GUI element which function is stateful, such as setting/unsetting a check box.

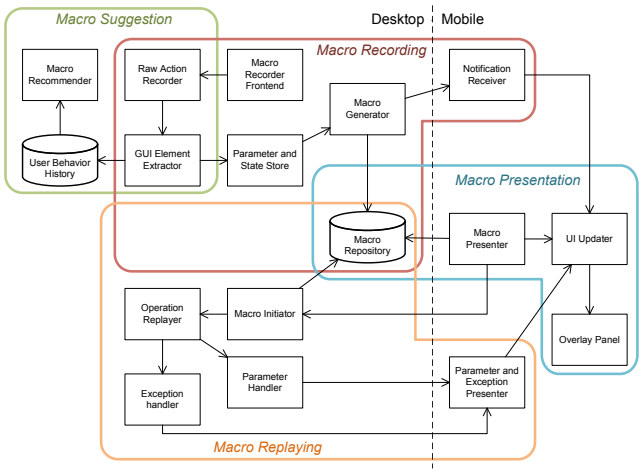


Figure 5: SmartVNC software architecture

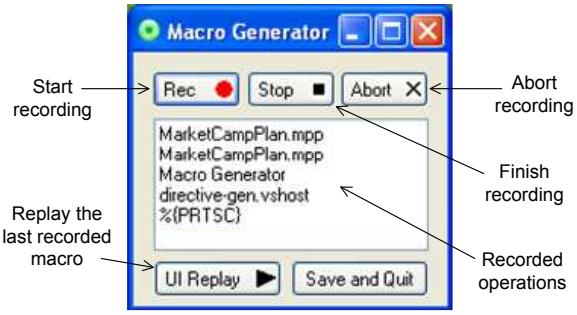


Figure 6: SmartVNC desktop UI screenshot

Table 1: Operation classification in parameterization

Parameter	check box, combo box, edit, list item, and radio button
Non-parameter	button, document, list, menu, menu item, pane, scroll bar, etc.

Macro generator: When the user clicks the stop button on the macro recording frontend, all the individual operations are aggregated into a single macro. The macro is a dynamic array of GUI element information recorded for each operation. Each entry in the array is self-sufficient to replay the necessary operation and the information stored contains the process on which an operation is performed, the unique identity of the GUI element so that it can be retrieved, the GUI element's state and the operation performed. The user can manually provide a name for the macro.

Operation Replayer: The *Operation replayer* is responsible for replaying an individual operation. Since SmartVNC records the full path of the GUI element handle in the GUI element tree, it simply walks through the tree from the root element of the tree to reach the required GUI element and thus retrieve its handle. In our implementation we use the FindChild() method provided by the UIA library to traverse the GUI element tree. Next, the recorded state of the GUI element is restored before the operation is performed on the element. Lastly, the operation is performed on the GUI element. For a mouse click operation, a mouse click is sent to the GUI element retrieved. The sendInput() function (available in the user32.dll) is used to replay a mouse click. Similarly, focus is set to the GUI element that is supposed to receive the keyboard operation and the raw keys are sent to the GUI element. The sendKeys() function available in the MS Windows Forms library is used to send the keyboard input to the focused GUI element.

Exception Handler: This component is responsible for handling one of the following exceptions that occur while executing a macro: 1) process latency caused by the underlying

OS when it is heavily loaded presenting the rendering of the next target GUI element in time; 2) missing prerequisite operations in the recorded macro (Ex: clicking on a hyperlink before the webpage if fully loaded); 3) notifications or alerts from the application that block the interaction in the user interface. All the above cases create a scenario in which the macro replayer is unable to find the next GUI element. The exception handler retries several times to avoid the latency issue and then hands control to the user via the exception presenter.

Macro Recommender: This component analyzes the user activity history on an on-demand basis and suggests macros for the user to create. The *Raw Operation Recorder* and the *GUI Element Extractor* are reused to keep track of the user activity on the PC. As introduced in the previous section, the macro recommender uses a suffix tree to process the user activity. A suffix tree is a well-known linear-time solution to the longest common substring problem. Figure 7 uses an example history of operations, say "ABCABCDABE" to show the steps in determining all longest-matching repetitive sequences using a suffix tree. (1) Each node in the suffix tree represents a sequence of operations and also the number of occurrences of the particular sequence (e.g. the leftmost "C_2" in (1) represents that "ABC" appears twice). By traversing the suffix tree once, the Macro Recommender is able to find all repeated sequences (shown in boldface). (2) The Macro Recommender removes redundant patterns by filtering the longest-matching repetitive sequences from the found sequences. It first removes redundant prefixes by leveraging the information stored in the suffix tree. "A" is a redundant prefix of "AB" since node "B_3" has a equal number of occurrence as its parent "A_3", which means "A" only appears as a sub-string of "B" but not separately. (3) The Macro Recommender removes redundant suffixes by establishing a reverse suffix tree and apply the same technique. The Macro Recommender keeps track of the discovered patterns and suggest the new patterns to the user for macro creation. The construction of the suffix tree can be achieved with an algorithm that has linear complexity in both computation and space [9]. As explained above, the removal of redundant patterns only require two traversal of the suffix tree. Thus, the proposed macro recommendation has computation and space requirements linear to the length of the user activity in terms of the number of operations.

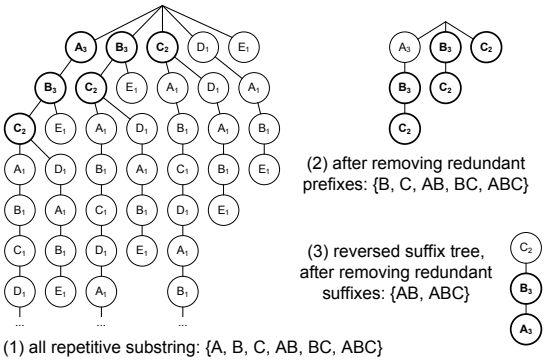


Figure 7: An example of macro suggestion with a suffix tree

Other Components: We now describe the other components that constitute the SmartVNC server: 1) *Raw Operation Recorder* is involved in capturing the raw user input like mouse clicks and keyboard entries. 2) *Macro Repository* provides an interface to store and retrieve the recorded macros in the persistent database. 3) The *Macro Initiator* handles the replaying of macros on the desktop. It interfaces with the SmartVNC client app and retrieves the macro from the repository when required and executes each operation involved in the macro using the *Operation Replayer*. 4) The *Parameter Handler* takes care of replaying parameter operations on the desktop. The handler provides the bounding box and a default value for the parameter, and it applies the user input to the GUI element.

5.2 SmartVNC Client on Smartphone

The SmartVNC client contains all the components of the Macro Presentation functional block and some components of the others. We prototype the solution by modifying AndroidVNC, an open-source VNC client. We modify only two files of the AndroidVNC source namely VncActivity, the main Activity object, and VncCanvas, the main GUI object of VNC interface. All SmartVNC communication from the smartphone to the desktop is asynchronous and bidirectional. We now explain the different components:

Overlay Panel: The *overlay panel* is a control panel that is embedded into the remote computing client and provides a set of buttons to the smartphone user. Figure 8 shows a screenshot of the overlay panel and the underlying VNC client. There are five sets of buttons shown in the control panel depending on the stage of macro execution: application menu, macro menu, macro execution menu, parameter menu, and exception handling menu. The overlay panel shows only one menu at a given time and the panel added to the remote computing client using a layout that allows overlapping. The switching between menus is realized by toggling the visibility of the previous and the current menus. The overlay panel is added to the content of the VncActivity object with a FrameLayout that overlaps the panel on top of the VNC client. Since Android only allows the UI thread to modify the UI, other components that want to update the overlay panel need to go through the *UI updater*, a component created by us in the UI thread.

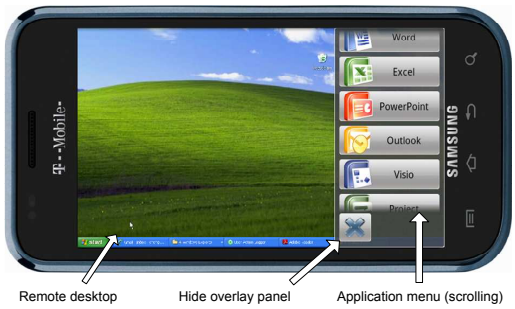


Figure 8: SmartVNC client screenshot

Macro Presenter: This component maintains the application menu and the macro menu, and it also responds to the selection of the user. It connects to the macro repository to retrieve the list of applications and the list of macros associated with a specific application. The applications and the macros are alphabetically ordered so that the users can easily find the macros. When an application is selected by

the user, the macro presenter sends a request to the macro initiator to bring the application to the foreground. When a macro is selected by the user, the macro presenter sends a request to the macro initiator to execute the selected macro. Both requests are sent with asynchronous communication so that the UI of the smartphone is not blocked. While a macro is being executed by the macro initiator, the macro presenter shows the macro execution menu that allows the user to control the timing of the execution. As we have discussed in Section 4.4, the timing control allows the user to extend the recorded macros with new operations in runtime.

Parameter/Exception Presenter: This component handles the presentation of parameters and exceptions to the user. While receiving a notification of a parameter from the parameter handler, this component first shows the parameter menu in the overlay panel via the UI updater. Then it zooms into the bounding box of the parameter. This is done by using the zooming and panning functions in the interface provided by the underlying remote computing client, which is VncCanvas in our prototype. The response from the user could be a new value or selecting a default value provided by the parameter handler. Lastly, the presenter accepts the response from the user and returns it to the parameter handler to proceed with the execution. Similarly, the parameter/exception presenter shows the exception menu upon an exception notification from the exception handler on the desktop side. The exception menu contains three options namely "Retry", "Ignore" and "Abort". These options are self-explanatory.

Notification Receiver: This component receives notifications of new macros from the macro generator and gives updates to the user in the overlay panel.

5.3 Future Implementation Considerations

Portability to other PC platforms: Since Mac OS and Linux also have their own accessibility frameworks [10, 11], the SmartVNC solution can be easily ported to other PC platforms as well.

Portability to other smartphone platforms: The SmartVNC solution can also be ported to other smartphone platforms such as iPhone, RIM, Windows Phone 7, Symbian, PalmOS as long as the source code of the target remote computing client is available. Note that the integration required with the client is very simple.

Extension to native mobile apps: The task effort on smartphones is high irrespective of whether an application is accessed through a remote computing or through a native app. While a native app might have an optimized GUI for the smartphone screen, operation aggregation will be nevertheless be useful in reducing task effort for repeated tasks. We plan to explore the use of SmartVNC for accessing networked, but native, applications on the smartphone as part of future work.

6. PERFORMANCE EVALUATION

In this section, we present the performance evaluation of our implementation of SmartVNC with experiments involving real users.

Prototyping: Our prototype testbed consists of 1) a Dell desktop running Windows XP SP3 with a Pentium-4 2.8 GHz CPU, 3GB RAM, and a 19-inch monitor (1280x1024) and 2) a Samsung Galaxy S smartphone running Android 2.1 with 1GHz CPU, 512MB RAM, and a 4-inch screen (800x480). We evaluate our solution with nine Windows applications and six tasks of varying complexities for each application. The full list of tasks and applications is shown in Table 2. While the tasks are pre-determined, they are derived from real-user activity dumps collected for the motivation results in Section 3.2. Since we observe that users not only use easy-to-find GUI elements but also those hidden inside layers of menus, we define the tasks with different levels of complexity to fully capture typical user activity. For comparison, we use AndroidVNC [4], the VNC client having the highest rating (4.5 stars) and most downloads (more than 250,000) in the Android Market. Both the AndroidVNC and SmartVNC apps are installed in the Android phone which connects to the PC via a local Wi-Fi network. In the rest of the section, we refer to AndroidVNC as simply VNC.

Metrics: We compare the user experience of SmartVNC with that of VNC and direct PC access using two objective performance metrics, time on task and task effort. Our definition of task effort only focuses on mouse/keyboard operations that can be identified and automated by a software system. Our definition of task effort can be considered as a variant of the Keystroke-level Model in GOMS family [12], which contains mental operations such as mental preparation and concentration shift that are not directly related to our solution. We also get subjective feedback from real users. We provide a CPU and memory profiling analysis of VNC and SmartVNC to show the overheads introduced by our solution. Finally, we provide statistics from the offline macro suggestion for the user dumps collected from ten users and the potential task effort reduction for these users using SmartVNC.

Experimental methodology: We invited twenty-two volunteers for our experimental evaluation, and all of them are students whose ages are between 20 and 30. While some of them are not smartphone users, all of them actively use their PCs for daily tasks. During the experiments, each user was randomly given two applications from the nine applications, and was asked to perform three tasks (one in each complexity category) for each application. While using SmartVNC, the smartphone had been loaded with the predefined 54 macros, and the user was asked to perform the six tasks with the corresponding macros. Since a user might not be familiar with the application or the smartphone, we let the user to practice until they feel comfortable to perform the tasks so that the learning effect is reduced in the experiments. We perform within-subject evaluation (the user performs the same tasks on PC, on AndroidVNC, and then on SmartVNC) so that the users can give us their subjective feedback with side-by-side comparison. It should be noted that the experimental results here are only applicable to the scenarios where an experienced user has created macros for her routine tasks, and she uses the macros to reduce her time and effort in doing these routine tasks from a smartphone. We use the traces from real users to evaluate the achievable time/effort reduction in accomplishing generic tasks using remote computing from a smartphone. We do not study the learning effort of using SmartVNC and defer it as part of our future work.

Measurement and analysis: During each experiment, we collect objective performance data with measurement tools built for both the PC and the Android phone. The tools measure both the task effort and the time on task for the user to execute a task. For the VNC and SmartVNC

Table 2: Task list and macros used for evaluation (E=Easy, M=Medium, and H=Hard)

App	Tasks		
Word	E	Open and print a file	Change format of a line
	M	Justify the entire text	Add border to document
	H	Insert picture & effects	Justify text & add a footer
Excel	E	Open and print a file	Sort data by a column
	M	Insert a bar chart	Insert a formula in a cell
	H	Import data from a csv file	Insert a scatter chart
PowerPoint	E	Add a picture in a slide	Change template for slides
	M	Print handouts of slides	Select an animation scheme
	H	Add a footer	Edit the master slide
Outlook	E	Print the current calendar	Print contact list
	M	Arrange mail	Arrange contacts
	H	Change email options	Change calendar options
Quicken	E	Print a summary report	Banking summary
	M	Export cash flow report	Add a transaction
	H	Compare spending by year	Find spending on clothing
IE	E	Print a webpage	Go to a specific webpage
	M	View an RSS feed	Check weather
	H	Change email options	Change tab options
SharePoint	E	Add a new announcement	Upload a new document
	M	Add a new event	Add a new task
	H	Edit permissions	Check in a document
Visio	E	Print a file	Show a category of shapes
	M	Change pattern of a shape	Add shadow to a shape
	H	Configure layout	Flip a figure
Project	E	Print a Gantt chart	View network diagram
	M	View a resource sheet	Sort event list by criterion
	H	Create a specified report	Print multiple views

apps, the measurement tool is integrated into the menu of the Android app. For the PC, the measurement tool is a stand-alone program that can capture raw user input to any application. The users provide their subjective opinion of using PC, VNC, or SmartVNC in executing each task. We use Google Docs to create an online survey form where they can give their subjective opinion in an anonymous manner. We use averages to provide overall performance evaluation, but we don't provide confidence intervals since the grouped tasks contain different tasks that may belong to different applications or complexity.

6.1 Overall Performance Improvement

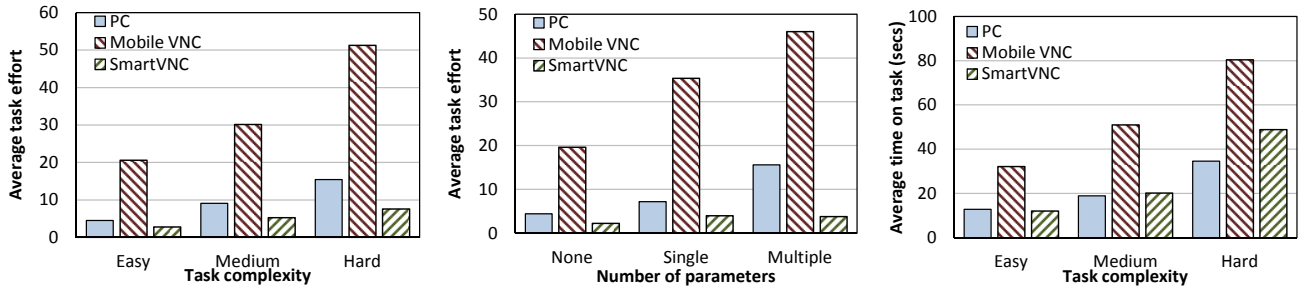
The performance results in Figures 9(a) show that our primary design goal of reducing task effort is achieved across different task categories. As discussed in Equation 1 in Section 3, task effort on VNC contains a component of task effort on the PC and a component of mobile inflation. SmartVNC reduces both the components as evidenced by a lower task effort than both the effort on VNC and the effort on PC. Figure 9(b) shows the task effort categorized in terms of number of parameters required for the task. We observe that with more parameters, the effort reduction ratio is higher since the users can significantly reduce their effort by using default values. Figure 9(c) shows a significantly lower time on task for SmartVNC when compared to those of VNC for all three task categories. Also, we observe that SmartVNC takes almost the same average time as working on a PC. While for some tasks, SmartVNC takes a slightly lesser time than a PC because of the task effort reduction with operation aggregation. The time on task in the VNC is significantly higher since the users not only have to perform all operations, each of them is inflated due to the input constraints in the smartphone.

6.2 Performance Improvement by Application

While we show average results in the previous subsection, Table 3 shows the reduction of time on task when using SmartVNC when compared to VNC for individual applications. We observe that the time on task for all applications by using SmartVNC. The applications have different types of GUI menus ranging from the traditional menus with a deeper structure for Quicken to the newest *Ribbon interface* for MS Office 2007 products. We observe that irrespective of the GUI menu type, VNC requires a lot of time for users to navigate the menu for performing tasks. The time reduction varies from 14 % for IE to 81 % for MS Outlook.

6.3 Subjective Opinion

After performing all experiments, we asked the users to provide their subjective opinion on using different platforms to accomplish tasks. Each user answered the question: "How would you rate your user experience in performing the task using certain platform?" using a likert scale value [13] ranging from 1 being the poorest to 5 being the best. Figure 10(a) shows first a significant decrease in the subjective evaluation for VNC when compared to PC, and our solution provides a significant increase back to the PC-level. Every-



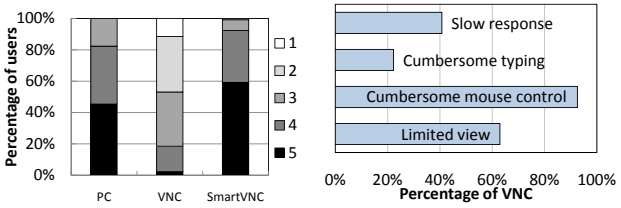
(a) Task effort by complexity (b) Task effort parameter type (c) Time on task

Figure 9: Overall performance enhancement with SmartVNC

Table 3: Time on task (secs) and reduction percentage

Application	VNC	SmartVNC	Reduction
Word	77.22	16.07	79%
Excel	59.10	25.63	57%
PowerPoint	50.53	21.04	58%
Outlook	41.75	8.04	81%
Quicken	105.43	24.81	76%
Internet Explorer	24.66	21.30	14%
Visio	38.96	13.33	66%
Project	41.93	10.19	76%
SharePoint	53.86	31.56	41%

one in the focus group rated SmartVNC greater or at least equal to VNC for every task performed. In fact, some users rated SmartVNC higher than a PC because our solution reduces the effort of performing operations on the smartphone to less than even that of performing on a PC. Figure 10(b) shows the main reasons of users' frustration on using VNC from the smartphone, and most of them are related to the interface constraints and increased task effort.

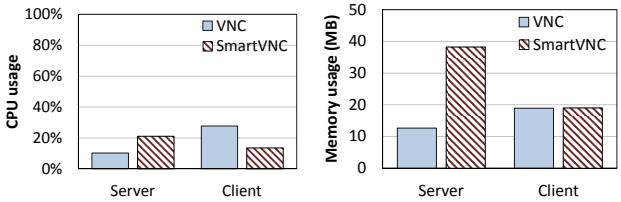


(a) Likert scale (b) Frustration on VNC

Figure 10: Evaluation of subjective opinion

6.4 Overhead Analysis

We now present a CPU and memory usage profiling of our solution. We collect the CPU and memory statistics while executing a hard task of MS Word on both the PC and the smartphone. The statistics for PC are collected using perfmon, a system monitor utility tool provided by Microsoft along with Windows. The statistics for the Android smartphone are collected using SystemPanel, one of the best-rated system monitor apps in the Android Market. Figure 11(a) shows the average CPU usage of VNC and SmartVNC on both the server side and the client side. The results of SmartVNC on the PC include the unmodified VNC server running on the PC. The SmartVNC server takes 7.19% atop the unmodified VNC server[3]. The integrated client takes less CPU than the unmodified VNC client, and it can be attributed to the reduced load on the smartphone side due to less user interaction with the app.



(a) CPU usage (b) Memory usage

Figure 11: Overhead analysis of SmartVNC

Figure 11(b) shows the average memory usage of VNC and SmartVNC on both the server side and the client side. The memory usage measurement is based on the unique set size in Android SystemPanel and the private bytes in Windows perfmon, since both indicate the memory size exclusively allocated to the process. The SmartVNC server takes about 26MB atop the unmodified VNC, and the overhead is acceptable since current PCs typically have a large memory. The SmartVNC client takes only 100KB more memory than the unmodified VNC client. Since we have designed the SmartVNC solution to have most of the processing on the server, the client is very lightweight and efficient.

6.5 Results of Offline Macro Suggestion

Table 4 shows the statistics from running the offline macro suggestion tool on the user activity dumps of the ten users we collected for Section 3.2. The table shows the number of tasks identified for aggregation on a per user basis. The table also shows the average number of operations for the tasks identified and the average frequency of repetition of the tasks. We observe that for most users, the tool is able to identify more than one hundred repetitive tasks. Further, we observe that these tasks are typically executed by the

[3]The CPU usage of the unmodified VNC server is also increased by 3.71% when running with SmartVNC since operations are executed at a faster rate.

user at least once or twice per day. The average length of the tasks is at least two for most users with User 10 having an average task length of 17.58.

Table 4: Statistics from offline macro suggestion

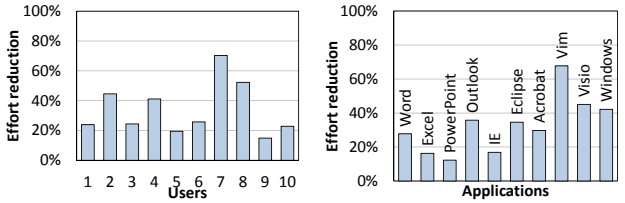
User ID	Number of tasks	Average length	Average daily frequency
1	143	2.34	2.96
2	652	2.77	0.47
3	1125	2.87	1.11
4	471	7.28	2.20
5	156	14.18	3.61
6	59	5.24	1.78
7	100	1.96	1.32
8	53	1.72	0.84
9	493	2.68	1.24
10	282	17.58	5.57

6.6 Trace-Based Evaluation of Task Effort Reduction

While we have shown task effort reduction for the specific pre-defined tasks, we now show the potential task effort reduction for realistic user activity. In a realistic scenario, it may not be feasible to optimize every operation that the user intends to perform from a smartphone. Thus, we base our analysis on the traces of user activity we collected from real users for evaluating the effort reduction achievable by SmartVNC [4]. We use the following equation to estimate the effort of executing a task with SmartVNC:

$$TaskEffort^{SmartVNC} = max(TaskEffort^{PC} \times Reduction, 1)$$

where *Reduction* is the effort reduction from *PC* that we observe in the experiments, and the value of which is 0.61. Note that the task effort of executing a macro is at least one. In the user activity history, we only match with operation sequences that have at least two operations. This fits with the minimum macro length in the task list and makes the analysis more realistic. Figure 12(a) shows the effort reduction achieved with SmartVNC for each of the users. The reduction ranges from 14.85% to 70.30%, and the average of 37.71% shows that SmartVNC can provide significant effort reduction in real user activity. Similarly, SmartVNC reduces task effort in each of the top applications used by the users, as shown in Figure 12(b).



(a) Effort reduction by user (b) Effort reduction by application

Figure 12: Trace-based evaluation of effort reduction

7. RELATED WORK

In this section we discuss work in literature and commercial products related to our solution.

Remote computing from the smartphone: As discussed in Section 2, there are several apps available for smartphones that allow remote computing to a PC using protocols such as RDP, VNC, or other proprietary protocols. In related literature, MobiDesk [14] proposes a thin client solution for mobile devices by optimizing the WAN traffic involved in performing remote computing. The solution is primarily meant for mobile laptops and is similar in principle to other remote computing approaches. PCoIP [15] is another product that optimizes remote computing traffic, especially video over IP networks. All these solutions do not address the problem of small real estate of the screen on smartphones.

Macros: As discussed in Section 3, the concept of macros is not new and is known in the context of application specific macros or raw macros. In [16], the authors present a raw macro-based solution for thin client computing. Mugshot [17], CoScriptor [18], SmartBookmarks [19], and Chickenfoot [20] are application macro solutions for web applications. WikiDo [21] and KarDo [22] are crowd sourcing based solutions for allowing non-technical users to help each other in setting up IT applications and executing computing tasks. The solutions record operations of one user on one computer and replays the recorded operations on another computer. While the concept of application agnostic macros proposed in WikiDo and KarDo is similar to our solution with respect to the accessibility framework used, our solution is specifically designed for accessing macros from a smartphone, and it provides features for macro extensibility, exception handling, and a macro suggestion tool. Apple automator [23] is an application for MAC OS that allows users to define workflows to access a user recorded set of functions automatically. The user is allowed to chose from a limited set of functions from each application to be used for the workflow. Tasker [24] is a similar automation application for the Android OS for automating a pre-defined set of operations on different apps running on the Android phone. ConfAid [25] is a task automation tool specifically designed for troubleshooting system misconfigurations.

UI customization for the smartphone: While we presented macros as a way to reduce task effort, there are other types of solutions possible. In particular, Merlion [26] is a solution for creating application mashups that allow users to define a smaller subset of GUI elements to be visible when using the application remotely from a smartphone. The solution works at a raw pixel level and cannot record the state of the elements. In [27], the authors propose a more reliable GUI mashup solution by performing image recognition of the GUI elements accessed. Again, the users can manually select a modified simple interface for mobile GUI for the application they want to access. However, this approach requires a complex image recognition component to identify the user intent and does not use the available information about the GUI elements. PageTailor [28] introduces reusable

[4]Note that the trace-based analysis is performed by assuming that the typical user activity on a smartphone will be the same as that on a PC. In reality, this might not be the case and the users might actually want to perform only a subset of tasks performed on a PC. SmartVNC will reduce task effort to much greater extent than our pessimistic estimate.

customization for mobile users, but the solution is specific to web pages instead of general applications.

8. CONCLUDING REMARKS

In this paper, we identify the practical issues of using state-of-the-art remote computing apps for smartphones. We introduce the notion of *smart-macros* that aggregate user actions required for a task. Using *smart-macros* as the key building block, we develop a remote computing solution for smartphones known as *SmartVNC*. Using our implementation of SmartVNC as a two ended solution, we show that the user experience can be brought back up to PC levels both in terms of objective and subjective metrics in accomplishing remote tasks from the smartphone. We also show that the overheads incurred by SmartVNC are minimal. Evaluating the SmartVNC system based on real usage is part of our future goals. We would like to deploy the system for real users to create *smart-macros* and access them from smartphones. In the real-world experiments, we will also study how our solution supports GUI applications written using third-party UI frameworks such as Qt, GTK+, and wxWidgets.

9. REFERENCES

[1] "Comparison of remote desktop software," http://en.wikipedia.org/wiki/Comparison_of_remote_desktop_software.

[2] "Remote Desktop Protocol," http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx.

[3] RealVNC Ltd, "The RFB Protocol." [Online]. Available: http://www.realvnc.com/docs/rfbproto.pdf

[4] "android-vnc-viewer," http://code.google.com/p/android-vnc-viewer/.

[5] "Microsoft Office," http://office.microsoft.com/.

[6] "iMacros," http://www.iopus.com/imacros/.

[7] "AutoHotkey," http://www.autohotkey.com/.

[8] "HyperSQL," http://hsqldb.org/.

[9] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, Sep. 1995.

[10] "Mac OS X Accessibility Protocol," http://developer.apple.com/library/mac/documentation/Accessibility/Conceptual/AccessibilityMacOSX/OSXAXModel/OSXAXmodel.html.

[11] "KDE Accessibility Project," http://accessibility.kde.org/.

[12] P. Holleis, F. Otto, H. Hussmann, and A. Schmidt, "Keystroke-level model for advanced mobile phone interaction," in *CHI*, 2007.

[13] R. Likert, "A technique for the measurement of attitudes." *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[14] R. A. Baratto, S. Potter, G. Su, and J. Nieh, "MobiDesk : Mobile Virtual Desktop Computing Categories and Subject Descriptors," in *MobiCom*, 2004.

[15] "PC-over-IP," http://www.teradici.com/.

[16] T.-Y. Chang, A. Velayutham, and R. Sivakumar, "Mimic: raw activity shipping for file synchronization in mobile file systems," in *MobiSys*, 2004.

[17] J. Mickens, J. Elson, and J. Howell, "Mugshot : Deterministic Capture and Replay for JavaScript Applications," in *NSDI*, 2010.

[18] G. Leshed, E. M. Haber, T. Matthews, T. Lau, C. Ave, H. Rd, and S. Jose, "CoScripter : Automating & Sharing How-To Knowledge in the Enterprise," in *CHI*, 2008.

[19] D. Hupp and R. C. Miller, "Smart Bookmarks : Automatic Retroactive Macro Recording on the Web," in *UIST*, 2007.

[20] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller, "Automation and customization of rendered web pages," in *UIST*, 2005.

[21] N. Kushman, M. Brodsky, S. R. K. B. Dina, K. Regina, and M. Rinard, "WikiDo," in *HotNets*, 2009.

[22] N. Kushman and D. Katabi, "Enabling configuration-independent automation by non-expert users," in *OSDI*, 2010.

[23] "Automator in Mac OS X," http://developer.apple.com/macosx/automator.html.

[24] "Tasker for Android," http://tasker.dinglisch.net/.

[25] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *OSDI*, 2010.

[26] I. Mohomed, "Enabling mobile application mashups with Merlion," in *HotMobile*, 2010.

[27] F. Lamberti and A. Sanna, "Extensible GUIs for remote application control on mobile devices." *IEEE computer graphics and applications*, vol. 28, no. 4, pp. 50–7, 2008.

[28] N. Bila, T. Ronda, I. Mohomed, K. N. Truong, and E. D. Lara, "PageTailor : Reusable End-User Customization for the Mobile Web," in *MobiSys*, 2007.