CAM: A Mobile Interaction Framework for Digitizing Paper Processes in the Developing World

Tapan S. Parikh, Paul Javid
Department of Computer Science
University of Washington
Seattle, WA 98195-2350 USA
tapan@cs.washington.edu

ABSTRACT
During our work with community microfinance groups in rural India, we found that paper plays a crucial role in many local information practices. However, paper-based record-keeping can be inefficient, so we need to link paper with the flexibility of modern information tools. A mobile phone can be the perfect bridging device. Here we present the CAM mobile document processing system, in which a camera phone is used as an image capture and data entry device. Our system is able to process paper forms containing *CamShell* programs - embedded instructions that are decoded from an electronic image. By combining 1) paper, 2) audio, 3) numeric data entry, 4) narrative scripted execution and 5) asynchronous connectivity, we have synthesized our experience into a system that we believe is well-suited for an important set of users and applications in the developing world.

INTRODUCTION
During our work with community microfinance groups in rural India, we found that paper plays a crucial role in many local information practices [2]. It is used ubiquitously as a method of data storage, exchange and establishment of trust between two transacting parties. It is a medium that the local people own and trust, and provides a greater sense of security than rented or borrowed appliances.

The mobile phone has been shown to be the most likely modern digital tool to support economic development in developing nations. The growth of mobile phone use in China, India and Africa resembles the growth of the Internet in the developed world in the 1990s. As shown in the example of Grameen Phone, if a mobile phone is shared by a group of people, it can be afforded by even the poorest communities.

In this paper we introduce mobile document processing as an accessible and locally harmonious way to provide information services to the developing world [1]. In our system a camera-equipped mobile phone is used for image capture and data entry. Document interaction is specified using CamShell - a narrative document-embedded programming language that allows developers to add interactive audio, data entry, validation, processing, and networking instructions to otherwise inert paper documents.

THE CAM USER INTERFACE
CAM unites a user interface, a programming language and a delivery mechanism for accessing networked information services. Like the web, our goal is to allow many different services to be developed and deployed on a common infrastructure.

CamBrowser
The CAM client application is called the CamBrowser, and has currently been implemented for Nokia Series 60 camera phones. CamBrowser is designed to process specially designed CamForm documents. CamForms contain visual codes - two-dimensional data glyphs containing up to 76-bits of data that can be decoded from a camera image [3]. Visual codes serve as references to interactive content embedded within CamForms.

CAM interaction consists of two primitives. The user can either *scan* codes from low-resolution images taken by the viewfinder in real time, or *click* codes by taking a high-resolution image using the joystick button. Our focus was on designing a simple and intuitive interaction model with well-defined affordances between actions.

CamShell
CamShell is the programming language that is used to define the interaction with visual codes embedded in the form. There is one visual code in every CamForm that serves as a *form identifier*. This code always has a 0 as the lowest-order bit. The next 4 bits identify the form to the CamBrowser application. The form identifier is used to load the *form description schema*, an XML file containing metadata about the form and specifying the underlying data elements (including any default or pre-existing values). The protocol that should be used to load the schema file (currently either HTTP, bluetooth or the local filesystem), and the required information to find the schema (a network address or a file name), is included in the remaining bits of the form identifier.

Visual codes with a lowest-order bit of 1 are *action codes*. Each action code invokes a separate callback function when it is *clicked* or *scanned*. These callbacks are mapped to a set of code entry points specified as XML elements in the form description schema. These XML elements contain

the executable code that should be invoked when the callback is activated. The executable code itself is written using a simple scripted programming language that includes support for function calls, control flow, arithmetic and basic datatypes [4].

CamShell provides an API for accessing the mobile phone's functionality - including the phone's user interface, networking and telephony features. Listed below are the main functions currently included in this API:

- **User Dialogs** - These instructions launch various user dialogs used to collect data, get confirmation or convey a message to the user. Each dialog can also be associated with audio and graphical prompts. Audio feedback has been found very useful by potential users.
- **HTTP Post, Get** - These instructions are used to make HTTP requests using the phone's built-in connectivity.
- **SMS, MMS, Email** - These instructions transmit asynchronous network messages. In the case of MMS and Email the current form image can also be attached.
- **Phone Call** - This instruction is used to make a phone call to a particular number.
- **Applications** - These instructions are used to launch other applications, such as the Web or WAP browser.

Each callback function can contain any number of sequential actions, some of which may be executed conditionally. Conditionals are useful when performing form validation. An example callback function is given below.

```
<function name="u_click" params="param1">
  seq = input_int("Please input Form ID");
  if (!seq) return false;
  uri = "http://abc.com/reload.php?"
       ."seq=".seq;
  return http_get(uri);
</function>
```

CamForms

Although there is nothing in the specification of visual codes or CamShell that dictates how CamForms should be organized, there is a convention that we have been developing to build a consistent and understandable metaphor for users. CamForms are conceptually organized into the following three sections:

- **Header** - The header contains the form identifier code and other codes containing static data (for example identifying the sequence number of a particular form instance). Header codes need to be clicked only once at the beginning of a form interaction. A default form action can be included that triggers all default data entry tasks.
- **Body** - The body contains input codes that are like HTML input tags. These are co-located with form fields, and are used to transcribe data from the document using the phone's keypad. To edit a form field the user must click on the input code, which brings up an editable dialog window. When a set of input codes is clicked together in one image, their respective dialogs are displayed in sequence. When the CamBrowser scans a visual code, the value returned by the associated *scan* callback function is shown on the screen. This is usually the value of the field.
- **Buttons** - Buttons are codes identified by text and/or icons that allow the user to perform various actions. Some example actions include submitting the form data to a web server, reloading the data from an online source or voiding the currently stored data for the form.

USABILITY EVALUATION

We conducted an initial evaluation of a prior version of this system [1] with microfinance staff and group members in rural Tamil Nadu. This helped us identify the basic usability issues, many of which have been rectified in the current design. The users' response to the system was very positive, particularly when compared to our earlier experience designing a PC interface for a similar user population [2].

We are now conducting a more detailed evaluation for simple data entry tasks using both CAM and a web-based interface on a PC. We are making this comparison because the alternative to CAM is that data is collected on paper forms in the field and entered using a PC at the branch office. A mobile interface is preferred because it would allow field staff to enter data as it is generated in the context of their regular documentation activities. If we can demonstrate that CAM has comparable efficiency, accuracy and learnability to a PC-based system, without a significant increase in cost, it could have important ramifications for the microfinance industry.

Current mobile data entry interfaces are known to be inefficient and difficult to learn. Much of this difficulty is due to the limited screen space of mobile devices, which makes navigation between different tasks and fields difficult. Our system addresses this issue by expanding navigation to the physical domain of paper documents.

CONCLUSION

In this paper we have described a mobile interaction framework for bringing information services to the developing world. By combining 1) paper, 2) audio, 3) numeric data entry, 4) narrative scripted execution and 5) asynchronous connectivity via the medium of an increasingly ubiquitous mobile device (the mobile phone), we have synthesized our observations into a usable and locally harmonious system that we believe is well-suited for an important set of users and applications in the developing world.

REFERENCES

1. T. S. Parikh. Using mobile phones for secure, distributed document processing in the developing world. *IEEE Pervasive Computing Magazine*, 4(2):74–81, April 2005.
2. T. S. Parikh, K. Ghosh, and A. Chavan. Design studies for a financial management system for micro-credit groups in rural india. In *CUU '03: Proceedings of the 2003 conference on Universal usability*, pages 15–22, New York, NY, USA, 2003. ACM Press.
3. M. Rohs and B. Gfeller. Using camera-equipped mobile phones for interacting with real-world objects. In A. Ferscha, H. Hoertner, and G. Kotsis, editors, *Advances in Pervasive Computing*, pages 265–271. Austrian Computing Sociery (OCG), Vienna, Austria, 2004.
4. Simkin language, May 2005. http://www.simkin.co.uk/.