Open Data Kit 2.0: Expanding and Refining Information Services for Developing Regions

Waylon Brunette, Mitchell Sundt, Nicola Dell, Rohit Chaudhri, Nathan Breit, Gaetano Borriello
Department of Computer Science and Engineering
University of Washington
Box 35350, Seattle WA, 98195, USA
{wrb, msundt, nixdell, rohitc, nbnate, gaetano}@cse.uw.edu

ABSTRACT
Open Data Kit (ODK) is an open-source, modular toolkit that enables organizations to build application-specific information services for use in resource-constrained environments. ODK is one of the leading data collection solutions available and has been deployed by a wide variety of organizations in dozens of countries around the world. This paper discusses how recent feedback from users and developers led us to redesign the ODK system architecture. Specifically, the design principles for ODK 2.0 focus on: 1) favoring runtime languages over compile time languages to make customizations easier for individuals with limited programming experience; 2) implementing basic data structures as single rows within a table of data; 3) storing that data in a database that is accessible across applications and client devices; and 4) increasing the diversity of input types by enabling new data input methods from sensors. We discuss how these principles have led to the refinement of the existing ODK tools, and the creation of several new tools that aim to improve the toolkit, expand its range of applications, and make it more customizable by users.

Categories and Subject Descriptors
H.4 Information Systems Applications

General Terms
Design

Keywords
Open Data Kit, mobile computing, smartphones, ICTD, sensing, mobile databases, spreadsheets, data tables, paper forms, vision.

1. INTRODUCTION
Smartphones are rapidly becoming the platform of choice for deploying data collection and information services in the developing world. They have quickly leap-frogged desktop and laptop computers due to their mobility, increased independence from the power infrastructure, ability to be connected to the internet via cellular networks, and relatively intuitive user interfaces enabling well-targeted applications for a variety of domains. In effect, developing countries are skipping the desktop and laptop phase of computing development, and are instead using smartphones and tablets for a range of tasks that have traditionally been performed on larger machines. In concert with this development, cloud services are providing many organizations with the ability to easily rent data storage space and scale hosting resources as needed, either locally or anywhere in the world.

We recognized two trends - 1) capable client devices with rich user interfaces and 2) cloud-based scalable data collection, computing, and visualization services - several years ago when we began the Open Data Kit (ODK) project at the University of Washington. Through ODK, we sought to create an evolvable, modular toolkit for organizations with limited financial and technical resources to use to create data collection and dissemination services. We chose Android as our development platform because its flexible inter-process communication methods allowed us to use existing apps for taking pictures, scanning barcodes, and determining location, rather than having to rewrite them ourselves, thus speeding development. ODK's development was guided by a few simple principles, namely:

- Modularity: create composable components that could be easily mixed and matched, and used separately, or together;
- Interoperability: encourage the use of standard file formats to support easy customization and connection to other tools;
- Community: foster the building of an open source community that would continue to contribute experiences and code to expand and refine the software;
- Realism: deal with the realities of infrastructure and connectivity in the developing world and always support asynchronous operation and multiple modes of data transfer;
- Rich user interfaces: focus on minimizing user training and supporting rich data types like GPS coordinates and photos;
- Follow technology trends: use consumer devices to take advantage of multiple suppliers, falling device costs, and a growing pool of software developers.

The name ODK refers to the entire suite of modular tools. Each tool in the suite has been assigned a name that describes its function. Previous work has discussed ODK 1.0 [6], which consisted of three primary tools: Build, Collect, and Aggregate. These provide the ability to design forms, collect data on mobile devices (e.g. phones, tablets), and organize data into a persistent store where it can be analyzed. Prior papers have also described the design of several new tools that are being incorporated into the ODK suite: Sensors [1], Scan [3] and Tables [7]. As we deployed the original tools, gathered feedback from users, and sought to incorporate new tools into the ODK suite, it became clear that there were some deficiencies in our design that needed

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM HotMobile'13, February 26-27, 2013, Jekyll Island, Georgia, USA.
Copyright 2013 ACM 978-1-4503-1421-3 ...$10.00.

to be addressed. This paper describes these deficiencies, and the rationale that drove a redesign of the inter-tool architecture of the tool suite, which will be released under the ODK 2.0 label.

ODK has quickly become one of the leading solutions for a wide variety of organizations, from small NGOs to large government ministries, and now has thousands of users in dozens of countries around the world. The projects for which it is being used range over an ever-increasing set of domains including public health (our original focus), environmental monitoring, and documenting human rights abuses. The ODK website has been visited by over 65,000 unique visitors from 202 different countries/territories and averages over 8500 hits a month. Additionally, over 11,000 distinct users have installed Collect from Google Play (a number that does not include organizations that install Collect directly when setting up their deployment). From our users' collective experience using ODK, we have seen many ways to improve the toolkit, expand its range of applications, and make it even more customizable. Recently, we conducted an extensive survey of the ODK user and developer community to better understand how people are using ODK and how organizations' data collection needs are evolving. 73 organizations completed our survey, providing information on 55 different deployments involving at least 5500 mobile devices in over 30 countries. This vast amount of feedback, in conjunction with the numerous deployment reports and feature requests submitted to our mailing list and website, led us to rethink the ODK system architecture. This paper reports on the changes that we are now implementing to our system architecture and applications, and the rationale behind each.

2. LIMITATIONS OF ODK 1.0

Our observations and survey responses can be grouped into four principal areas of refinement for ODK:

1. support data aggregation, cleansing, and analysis/visualization functions directly on the mobile device by allowing users to view and edit collected data;
2. increase the ability to change the presentation of the applications and data so that the app can be easily specialized to different situations without requiring recompilation;
3. expand the types of information that can be collected from sensing devices, while maintaining usability by non-IT professionals; and
4. incorporate cheaper technologies such as paper and SMS into the data collection pipeline.

The design of ODK 1.0 focused on collecting data in the form of surveys, and uploading completed surveys into the database for analysis and aggregation. It did not provide facilities for getting that data back out to clients to review and update. However, feedback shows that many users want to be able to store already collected data (either from past data collection or from a server database) on the device and use it to specify which data to display (e.g., a patient's past blood pressure readings) or to steer survey logic (e.g., select follow-up questions based upon a patient's medical history). One user told us, *[One limitation of ODK 1.0] is the lack of a local database on the device [that contains] previously collected information. For example, the last time I visited your household, there were 5 people living here. Are those 5 people still living here?* In addition, one of the largest requests that we received from users is to make it possible to view and edit data on the device. For example, one user told us, *We need a presentable way of viewing collected data on the device … like if you have a roster and need to make decisions based on some earlier responses, you need to be able to view this data.*

Rendering of the surveys in ODK 1.0 was accomplished using a variant of a W3C XForms standard defined by the OpenRosa Consortium. Although XForms can specify input constraints (to provide some immediate error checking abilities), form navigation logic (branching based on previous answers), and multiple languages (for local customization), It does not describe the visual presentation of the prompts and data types. This led to many specializations of ODK for different organizations, which was technically challenging for many users. One user told us, *We struggled to understand xml and the XForm. While the XForm is fairly simple, the xml structure is confusing. Some of the advanced features require core knowledge of xml coding.*

Originally, ODK assumed humans would enter data explicitly or, at most, gather data from sensors that were built-in to the device (such as GPS coordinates, barcodes, photos, audio, and video). It did not support the ability to interact with new sensors or process data captured from built-in sensors like the camera. However, gathering information from external sensors is an often-requested feature; such requests range from enhancing a health survey with data obtained from medical sensors, to automatically incorporating GPS and compass data with captured photos. One user told us, *Our [use case] requires us to measure the height of trees. We currently use a clinometer for this and enter the data manually. It would be great if we could access the clinometer [from the device] and use it as part of our data collection process*. Collecting data from sensors attached to the mobile device is attractive because applications can directly receive and process the data, obviating the need for manual data transfer by a human, which may be error-prone.

Finally, many organizations have extremely limited financial resources and still rely on paper forms or very cheap mobile phones to gather data, and there is a need to connect these media to the ODK ecosystem. Simple SMS is a very common form of communication on mobile devices, particularly in developing countries, where many people use basic mobile phones that have only text and voice features. In addition, many organizations are unable to afford the cost of purchasing and maintaining a mobile device for every field worker. Such organizations would prefer to use cheap and well-understood paper forms to collect data at the lowest level of the information hierarchy, and then digitize the data at a higher level to enable data transmission, statistical analysis, and aggregation. The limitations of the original ODK 1.0 tools are addressed by the design of ODK 2.0, a refined and expanded toolkit with a more flexible system architecture.

3. DESIGN OF ODK 2.0

The refinement and expansion of ODK is based on four core design principles that we are incorporating into all the tools (these stem directly from the four areas of refinement described at the beginning of Section 2 but do not correspond 1-to-1):

1. when possible, UI elements should be designed using a more widely understood runtime language instead of a compile time language, thereby making it easier for individuals with limited programming experience to make customizations;
2. the basic data structures should be easily expressible in a single row, and nested structures should be avoided when data is in display, transmission, or storage states;
3. data should be stored in a database that can be shared across devices and can be easily extractable to a variety of common data formats; and

4. new sensors, data input methods and data types should be easy to incorporate into the data collection pipeline by individuals with limited technical experience.

Our evolved system architecture is still governed by the over-arching concern that for computing tools to address the many information gaps in developing regions, information services must be composable by non-programmers and be deployable by resource-constrained organizations (in terms of both financial and technical resource constraints) using primarily consumer services and devices. To facilitate this, the new ODK 2.0 toolkit (individual tool names are italicized) provides a way to synchronize, store and manipulate data in *Tables* on a mobile device with a user interface that supports both the smaller smartphone screen and larger tablet form-factors, and allows viewing and manipulating data in a simple row format. In addition, the new design makes customization easier by using widely understood standard presentation languages, such as HTML and JavaScript, to facilitate a more easily tailored user experience on a per *Survey* basis. Furthermore, ODK 2.0 makes is possible to attach external *Sensors* to mobile devices over both wired and wireless communication channels, thereby reducing the amount of manual data transcription from sensors into survey forms, and also facilitates the automatic conversion of information recorded on paper forms to a digital format by using the camera of the mobile device to *Scan* documents. See Figure 1 for a block diagram of the ODK tool suite architecture.

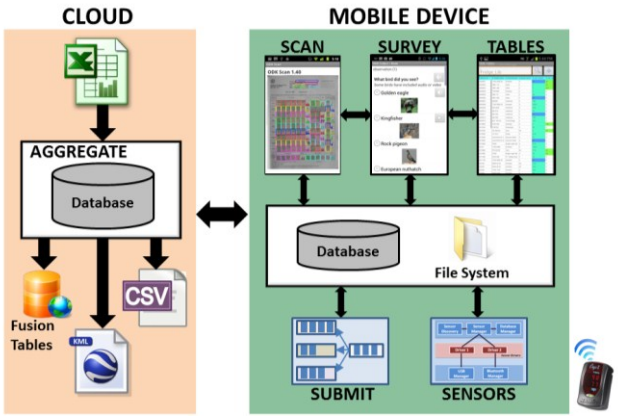

Figure 1: The new ODK 2.0 system architecture, showing cloud services (left) and mobile client services (right). In the cloud, Aggregate provides services to synchronize data across devices and export data in common file formats. On the device, a common database/file system is shared between the tools and across clients. Scan, Survey, and Tables (above the database) are tools for gathering, processing and visualizing input data; Submit and Sensors (below the database) are tools that augment Android to create additional services.

3.1 Data Management on Mobile Device

Many applications rely on previously collected data; for example, logistics management, public health, and environment monitoring often require workers to return and reference previously collected data to verify and possibly update conditions. In the previous ODK design, revising data from previously completed surveys was not supported. However, more and more of our users want to be able to use all or part of previous surveys to complete new ones (e.g., not re-entering patient demographics for a follow-up visit when that data was already collected in the original registration form). To enable data updating, aggregation, curation, cleansing and analysis functions on the mobile platform we created Tables. Tables allows a user to create new tables, add data, delete data, search data, scroll through data, add columns, configure data types, view graphs, apply conditional formatting, perform summary calculations, and synchronize the data with the cloud. Tables presents a user interface for editing and viewing data that is optimized for the smaller screens of mobile devices [7], and provides customization capabilities for users to easily configure the app for their use case. Tables has a number of built-in views, and allows users to explore their tabular data with customizable views defined by HTML/JavaScript files, thus making presentation much more flexible while avoiding recompilation. These views can pull data from, and link to, other tables, so that users can form an integrated app, rather than a set of loosely connected (or completely disconnected) tables. For example, a table of facilities can link to a table of specifics about individual resources at each facility. Alternatively, Tables can use Survey's strong data typing to add and edit entire rows and incorporate input constraint checking, or use Scan's image processing capabilities to add rows based upon filled-in paper forms. These tools (Tables, Survey, and Scan) use an inter-tool architecture based on a common SQLite database schema.

Another important refinement to Survey is making data collection and presentation more easily customizable. In Collect, changing the look-and-feel of a particular question type or extending the expression language (e.g., adding a count function) to express the user's business logic (e.g., visibility and value constraints) required changes to Java source code. This high barrier to change meant that we spent a significant amount of time refining our user interface because it needed to be generic enough to work for many use cases. It also created friction to the adoption of the technology because organizations lacked the skills or funding necessary to customize the tool. In contrast, Survey allows organizations to easily express their business logic and heavily customize the user interface for their specific use case through the use of JavaScript and HTML. We anticipate that Survey's JavaScript form interpreter, the use of open source toolkits (e.g., JQueryMobile, Handlebars, Backbone), and the greater worldwide prevalence of JavaScript and HTML coding skills will make it easier for individuals and organizations to make domain-specific customizations. Our design leverages the suite of standard ODK question widgets that encapsulate the rendering, event handling and business logic. These question widgets are then extended at runtime to incorporate rendering and business logic customizations (e.g., visibility and value constraints). Users can easily customize the user interface by specifying an alternative Handlebars template in the form definition, causing the widget to render using the alternative template.

User experiences from ODK 1.0 deployments show that although non-technical users are able to make small customizations to existing XForms, creating an entire XForm from scratch is often too challenging. Thus, to shield users from the complexity of writing XForms, we created Build, a tool that allows users to graphically compose surveys, and XLSForm (based on 'pyxforms' [14]) that gives users the option of writing their survey in an Excel spreadsheet that is automatically converted to an XForm. In ODK 2.0, we are building a revised converter to transform a spreadsheet to a JSON description that can be rendered using ODK's new interpreter that leverages web technologies. By allowing users to specify information in a spreadsheet, it enables non-technical users to remain shielded from the complexity of writing JSON and JavaScript. Users with minimal Javascript and

HTML skills will be able to copy and modify standard template files (e.g., use different HTML constructs or add CSS style classes) and reference these modified template files to customize the rendering of individual questions in the form or create new question types. In the same way, users can also revise the standard templates and CSS stylesheets to create an organization-specific look and feel. Users with more advanced Javascript and HTML skills can customize a question widget's event handling (e.g., add mouseover-like treatments) or define entirely new widgets.

3.2 Improved Input Methods

ODK 2.0 reduces the amount of manual data transcription from sensors into surveys by making it possible to attach external sensors to mobile devices. By hiding complexities such as the management of communication channels and sensor state as well as data buffering and threading, the Sensors framework [1] simplifies the code needed to access a sensor.

Sensors provides a common interface to access both built-in and external sensors connected over a variety of communication channels. Thus far, we have implemented channel managers for Bluetooth and USB, and plan to implement managers for WiFi and NFC in the near future. The USB Manager currently supports three USB protocols: Android's Accessory Development Kit (ADK) 2011, ADK 2012, and a USB Host serial channel. Sensors also provides a convenient built-in sensor discovery mechanism that allows users to discover sensors and associate the appropriate driver with a sensor. Users who want to integrate external sensors with their mobile devices download and install the Sensors app and sensor driver app from an app store such as Google Play. This facilitates the easy delivery of the application and driver updates to devices. Figure 2 (left) shows Sensors being used in a South African clinic to deactivate harmful contaminants (like the HIV virus) in breast milk. Sensors provides abstractions that delineate application code from code that implements drivers for sensor-specific data processing. The sensor driver abstraction allows device drivers to be implemented in user-space so that locked devices can be customized by end users. The framework handles the data buffers and connection state for each sensor, which simplifies the drivers. Separating application code from driver code also allows the code bases to evolve independently.

In addition to accepting and processing input from a variety of different sensors, the continued use of paper forms for data collection in resource-constrained environments made it important that we also facilitate efficient data entry from paper forms. Many of the paper forms used by organizations for data collection contain a mixture of data types, including handwritten text, numbers, checkboxes and multiple choice answers. While some of these data types, such as handwritten text, require a person to manually transcribe the data, others, like checkboxes or bubbles, can be analyzed and interpreted automatically. To take advantage of machine-readability, we designed ODK Scan [3], a piece of software that uses a lightweight JSON form description language to facilitate the processing of existing paper forms without the need to redesign or add coded marks to the forms. To add a form to the system, the user creates a JSON form description file that specifies the size, location and data type of each form field to be processed. The camera on the device is used to photograph the form, and computer vision algorithms use the JSON form description file to automatically segment and interpret the machine-readable data. The image processing components of the application are implemented using OpenCV, an open source computer vision library, while the user interface components are implemented using Android's Java framework. We use the Java Native Interface (JNI) to facilitate communication between the Java framework and OpenCV's native image processing algorithms. All of the image processing is performed on the device so as not to require an Internet or cellular connection. After the image processing is completed, Scan launches Collect so that users can manually complete the entry of data types that are not machine-readable. Scan makes this data entry process faster by exporting small image snippets of each form field to Collect, and the image snippets are displayed on the screen of the device alongside the corresponding data entry box, so that users can simply look at the image snippet and type in the value displayed. Figure 2 (center) shows an image of Scan being used to collect vaccine statistics in a rural health center in Mozambique.

Figure 2: Examples of ODK tools in action. Left: Using Sensors to monitor breast milk pasteurization that deactivates contaminants (e.g. HIV virus); Center: Using Scan to digitize paper based vaccine information in Mozambique; Right: Indigenous tribal member using Collect in the Amazon jungle.

Data can also be collected from and disseminated to users with cheap SMS-only phones. By acting as an SMS server, Tables enables anyone to send SMS messages to query an existing table or add rows to a table. We use the data table abstraction to implement basic access control measures based on the phone number from which the message was sent as well as locally-administered (on the receiving smartphone) usernames and passwords. For example, this allows a farmer with a cheap phone to post available produce to an agent at a remote market or to obtain the commodity prices in that market. This allows Tables to provide services that can be accessed from the cheapest and most common phones without introducing the complexity of an SMS gateway or other cloud-based server.

3.3 Data Management in the Cloud

Less technically capable users encounter significant barriers to leveraging the power of the cloud. To simplify the distribution of forms to mobile devices, the retrieval of data from devices, and storing and managing data, we designed Aggregate, an auto-configuring, ready-to-deploy server. Aggregate manages collected data, provides interfaces to export the aggregated data into standard formats (e.g. CSV, KML, JSON) and allows users to publish data to online services (e.g., Google Spreadsheet or Fusion Tables). Aggregate is a configurable generic data storage service that runs on a user's choice of computing platform (cloud-based or private server). Aggregate can be deployed to the Google AppEngine hosting service to enable a highly-available and

scalable service that can be maintained by unskilled users and less-capable IT organizations. However, many of our users have data locality and security concerns, either because the data cannot legally leave the country of origin, or because the data may contain sensitive identifiable information, or be high-risk or high-value data. For these users, AppEngine may not be appropriate. Aggregate can therefore also run within a Java web container (e.g., Tomcat) using a MySQL or PostgreSQL datastore. Communications security generally relies on HTTPS connections between client devices and the server. However, because many organizations do not have the funds to purchase or the expertise to install SSL certificates on their own servers, we provide user authentication and data security over HTTP communications through DigestAuth and the asymmetric public key encryption of form data before transmission to the cloud. If asymmetric public key encryption is used, the form data is stored in encrypted form on the server, which enables some organizations to continue to leverage the AppEngine cloud hosting service despite stronger data security requirements. In this case, users download the encrypted data to a computer and use a locally-running tool called ODK Briefcase to decrypt it using a private key.

To provide datastore independence, and because Aggregate parses the submitted XForm instance into column values (to better support filtering and visualization) and incorporates a dynamic datastore abstraction layer rather than a layer set at compile-time. Since XForms can define arbitrarily deep nested groupings of repeated questions, Aggregate performs a complex mapping of the XForm to a set of database columns and tables. This greatly complicates the presentation of the data, and the wide variety of different use cases created by users prevents a generic processing of these nested repeating sections when visualizing, publishing or exporting the data. Since Aggregate parses the submitted XForm instance into column values, a more capable data analysis package could be configured to operate directly on the underlying database tables. However, the complexity of this configuration makes it impractical for many of our users.

In ODK 1.0, the communications flow is unidirectional; blank forms flow from the cloud service (Aggregate) to mobile devices, and data from the filled-in forms flows back to the cloud service and then out to remote services or into file exports. Collected data can be deleted, but is otherwise immutable and provides a store of record. Data is stored (aggregated) in the cloud, where simple curation and data visualization tools are provided. Aggregate bridges the gap between mobile data collection tools and the sophisticated data analysis software able to derive complex results by providing many forms of data export.

In version 2.0, a simple row is the basic storage element; repeating groups are explicitly represented as linked rows across two different forms. The new design eliminates the complex backend mapping that made it difficult for organizations to access the database structures directly. The communications flow has changed so it is now a cloud-mediated peer-to-peer store-and-forward network. Any authorized device running Tables can create new surveys and share data with any of its peers and the remote services can publish surveys and data back out to the mobile devices. Retaining a cloud service (Aggregate) as both a datastore and a store-and-forward communications nexus enables robust peer-to-peer operations in intermittent and low-connectivity environments. The cloud also provides a central point from which to manage and disseminate a security model that can be applied and enforced independently on each device.

Since data is no longer immutable, Tables relies upon the user to resolve conflicts that occur whenever two users concurrently update the same row in a table. Conflicts are detected and resolved at the individual row level (in keeping with our row-based information model) between a row on the user's mobile device and a row on the server. This maximizes the system's ability to disseminate new and uncorrelated change across devices. Manual, client-side conflict resolution was chosen because: 1) established recent-modification conflict resolution techniques are inappropriate or difficult to apply across devices that may not be time synchronized and which may be in disconnected operation for extended periods of time; 2) since ODK targets a diverse set of use cases and application domains, any assumptions built into an automatic resolution mechanism will likely be inappropriate for some domains; 3) accurately expressing the procedural rules to be applied during automatic conflict resolution is likely difficult for non-programmers and capturing and applying these domain-specific rules would increase the complexity of the server design; 4) client-side resolution benefits by keeping the user involved with reconciling conflicts since many times they understand the semantics of the conflict and can better resolve it at the moment it is detected rather than by a more remote administrator at a later date.

Data submission is currently initiated by the user because connectivity is often intermittent and organizations want to control data transfer costs. To better use available connectivity that may be sporadic, and to improve data timeliness (both on the mobile device and when publishing data to the peers), we are designing a tool called Submit that will manage data transmission. Submit enables organizations to specify parameters such as data priority, data importance, deadlines, and the cost of the transport mediums. Submit then factors in the device's connectivity history, and intelligently uses the connectivity available (e.g., SMS, GPRS/3G, Wi-Fi) to create a priority routing system that improves data timeliness in the intermittent and expensive connectivity of the developing world. Connectivity history is an important factor in routing decisions, since there may be certain times of day when the device is within range of a Wi-Fi base station. Alternatively, depending on the data priority and the costs of other connectivity options, it may make sense for the data to be stored locally until the user returns to Wi-Fi connectivity.

3.4 Use Case: Cold Chain Management

ODK 2.0 is an expanded and refined set of modular tools for collecting and managing data in low-resource environments. This section describes one concrete use case in which ODK 2.0 could be used to improve the delivery of health and information services. The cold chain is a complex sequence of refrigeration equipment used to ensure that vaccines retain the correct temperature during transport and storage. Collecting and disseminating accurate and timely data regarding a country's cold chain improves resource-allocation and planning, but cold chain inventories are currently mostly paper-based systems that contain large amounts of inaccurate or out-of-date information. Replacing the paper-based system with ODK 2.0 could improve the speed and reliability of the inventory update process. For example, remote field workers could use Tables to automatically download the most up-to-date subset of cold chain data for a site from Aggregate, and use Survey to enter any new refrigerator information. Sensors could be used to continuously monitor the temperatures of refrigerators at the site, and the worker could use Tables to visualize this data and check for anomalies. Finally, the

worker could use Scan to digitize paper-based records that track the number of vaccines administered at this site to improve stock monitoring and resupply. All of these tasks could be performed quickly on-site and the data made immediately available to decision-makers and stakeholders.

4. RELATED WORK

A variety of other solutions attempt to replace paper-based data collection with digital tools. CAM [11] used its own scripting language to augment paper forms by using bar codes to trigger audio prompts for manual data entry. MyExperience [5] collects survey responses triggered by sensor events but does not address the larger issues of organizational information flow. Commcare [4] is the most related to Collect in that it targets use by health workers and also uses XForms, but it is less flexible in how it can be composed with other tools and requires recompilation to customize presentation. Manipulating small databases on phones has received less attention. Tools like Excel are available in smartphone versions but have not been adapted to small screens and do not work directly from a database rather than a file. Uju [13] enables the creation of small databases that can be populated or queried over SMS but does not integrate with tools that obtain data from sensors or paper forms. Extracting data from paper forms via crowdsourcing is being commercialized by Captricity [2], while LocalGround [12] processes manually annotated paper maps and adds the data to existing digital maps. Neither of these tools work in completely disconnected operation. Recent activity focuses on connecting external sensors to phones using audio jacks (Hijack [9]) and Bluetooth (Amarino [8]). Google released a sensor development kit for Android [15], and researchers have focused on low-power operation of external sensors (Reflex [10]). However, what distinguishes ODK 2.0 from other solutions is the interoperability of these elements, and the ability to do all the computation, analysis and visualization on the device.

5. FUTURE WORK & CONCLUSION

The original design of ODK assumed that a system administrator would have access to a computer to initially set up and administer the system, including designing forms and setting up data storage facilities. However, in many rural locations, computers are rarely available, which limits the adoption of ODK in these settings. To reach these areas, it is desirable to create a system that could be entirely set up and administered on a mobile device. While ODK 2.0 provides users with some methods for building information systems on mobile devices (e.g., database design with Tables, customized question widgets) it does not entirely remove reliance on computers as users are not able to configure their cloud service or write a device driver. Additional work is necessary to build a system that can be set up and managed entirely on a mobile device. The new ODK 2.0 design focuses on a core set of tools that enable users to move beyond treating mobile devices as simple input devices, and instead leverage mobile computing platforms to build more dynamic collaborative information systems in the field. We expect that the changes in design and the new capabilities of the software will lead to a rich new set of research challenges and opportunities that we plan to explore.

Open Data Kit provides organizations with a modular toolkit to build application-specific information services for use in resource-constrained environments. Our own experiences combined with extensive feedback from organizations using the toolkit have led to a redesign of ODK that aims to better meet the needs of a wider range of organizations. Specifically, our design changes include 1) favoring runtime languages over compile time languages to make customizations easier for individuals with limited programming experience; 2) implementing basic data structures as single rows, 3) storing data in a database that is accessible across apps and client devices; and 4) increasing the diversity of input types by enabling new data input methods from sensors. We discussed how the new system design led to the addition of several tools to ODK 2.0 and how the new system architecture enables its adaptation to an even larger and varied set of applications. The ODK tools and their source code are freely available for download at http://opendatakit.org and are distributed under an Apache2 license.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the support of Google Research, NSF Grant No. IIS-1111433, and an NSF Graduate Research Fellowship under Grant No. DGE-0718124. We are grateful for all our collaborators at PATH Seattle and the ODK community.

7. REFERENCES

[1] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, G. Borriello, "Open Data Kit Sensors: a Sensor Integration Framework for Android at the Application-level," Proc. 10th Intl. Conf. on Mobile Systems, Applications, & Services (Mobisys), 2012.

[2] K. Chen, A. Kannan, Y. Yano, J. M. Hellerstein, and T. S. Parikh, Shreddr: pipelined paper digitization for low-resource organizations, Proc. 2nd ACM Symp on Computing for Development (DEV), 2012.

[3] N. Dell, N. Breit, T. Chaluco, J. Crawford, and G. Borriello, "Digitizing Paper Forms with Mobile Imaging Technologies," Proc. 2nd ACM Symp on Computing for Development (DEV), 2012.

[4] B. DeRenzi, G. Borriello, J. Jackson, V. S. Kumar, T. S. Parikh, P. Virk, and N. Lesh, "Mobile Phone Tools for Field-Based Health Care Workers in Low-Income Countries," Mount Sinai Journal of Medicine, vol 78, no 3, 2011.

[5] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones," Proc. 5th Intl .Conf. on Mobile Systems, Applications and Services (Mobisys), 2007.

[6] C. Hartung, A. Lerer, Y. Anokwa, C. Tseng, W. Brunette, and G. Borriello, "Open Data Kit: tools to build information services for developing regions," Proc. 4th ACM/IEEE Intl. Conf. on Information and Communication Technologies and Development (ICTD), 2010.

[7] Y. Hong, H. K. Worden, and G. Borriello, "ODK Tables: data organization and information services on a smartphone," Proc. 5th ACM Workshop on Networked Systems for Developing Regions, 2011.

[8] B. Kaufmann and L. Buechley, "Amarino: a Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing," Proc. 12th Intl. Conf. on Human Computer Interaction with Mobile Devices and Services, 2010.

[9] Y.S. Kuo, S. Verma, T. Schmid, and P. Dutta, "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface," Proc. 1st ACM Symp on Computing for Development (DEV), 2010.

[10] F. X. Lin, Z. Wang, R. LiKamWa, and L. Zhong, "Reflex: Using Low-power Processors in Smartphones without Knowing Them," Proc. 17th Intl .Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[11] T. S. Parikh and E. D. Lazowska, "Designing an Architecture for Delivering Mobile Information Services to the Rural Developing World," Proc. 15th Intl Conf on World Wide Web (WWW), 2006.

[12] S. Van Wart, K. J. Tsai, and T. Parikh, "LocalGround: a Paper-based Toolkit for Documenting Local Geo-spatial Knowledge," Proc. 1st ACM Symposium on Computing for Development (DEV), 2010.

[13] L. Wei-Chih, M. Tierney, J. Chen, F. Kazi, A. Hubard, J. G. Pasquel, L. Subramanian, and B. Rao, "UjU: SMS-based applications made easy," Proc 1st ACM Symp. on Computing for Development (DEV), 2010.

[14] formhub. http://formhub.org/. [Accessed: 11-Oct-2012].

[15] Android Accessory Development Kit. http://developer.android.com/tools/adk/index.html. [Accessed: 11-Oct-2012].