Open Data Kit 2.0: A Services-Based Application Framework for Disconnected Data Management

Waylon Brunette, Samuel Sudar, Mitchell Sundt, Clarice Larson, Jeffrey Beorse, Richard Anderson
Department of Computer Science and Engineering
University of Washington
Box 352350, Seattle, WA 98195
{wrb, sudars, msundt, clarice, jbeorse, anderson}@cse.uw.edu

ABSTRACT

In resource-constrained communities, organizations often use information and communication technologies to amplify their limited resources to improve education, health, and economic opportunity. Over two-thirds of the world's population have mobile phones, yet less than half are connected to the Internet [23]. Organizations helping disadvantaged populations often rely on mobile devices as their primary computing resource because of their availability in resource-constrained contexts. However, to reach under-served populations, mobile applications often operate in areas with no connectivity or challenged network environments. Unfortunately, many mobile application frameworks are generally not well-suited for long periods of disconnected data collection and management. Furthermore, mobile application frameworks are generally aimed at users with significant technical skills and resources. In this paper, we discuss our experiences building, deploying, and refining the Open Data Kit (ODK) 2.0 tool suite. ODK 2.0 is a modular application framework that facilitates organizations with limited technical capacity to build application-specific information services for use in disconnected environments. We discuss ODK 2.0's flexible abstractions that enable users of varying technical skill levels to create customizable mobile data management solutions. We present ODK 2.0 case studies involving multiple organizations and discuss lessons learned from building a service-based mobile application framework for disconnected data management.

Keywords

Open Data Kit; ICTD; mobile application framework; disconnected operation; mobile databases; mobile data management

1. INTRODUCTION

Organizations focused on improving education, health, and economic opportunity in under-served communities often operate in areas with limited resources, power, connectivity, and infrastructure. According to a 2016 World Bank report, over 90% of the world's population live within mobile network coverage [23] making mobile computing technologies one of the few technologies well suited for use in global development interventions. Therefore, organizations often rely on the mobile phone as their primary computing device. While over two-thirds of the world's population have mobile phones, less than half the world's population are connected to the Internet [23]. This scenario where billions of people have mobile phones but lack consistent Internet connectivity will likely continue for years. Thus, organizations helping disadvantaged populations need to rely on mobile applications that can operate in disconnected environments.

Various research projects focus on improving Internet connectivity by extending infrastructures (e.g., long distance WiFi, village base stations). However, a parallel approach is needed that focuses on creating abstractions and frameworks for mobile applications to operate in challenged network environments until affordable, universal connectivity is a reality. Unfortunately, many mobile application frameworks are generally designed for relatively short periods of disconnected operation as they often only include basic offline features with limited caching. Furthermore, existing mobile application frameworks generally require programming skills and focus on features for programmers, such as platform independence across multiple mobile operating systems (e.g., iOS, Android). However, in resource-limited settings there is often a shortage of qualified programmers and limited funds to hire consultants.

Open Data Kit (ODK) [9, 20] is a mobile tools suite that operates in disconnected environments with abstractions that lower technical barriers. ODK enables governments, organizations, and individuals to use technology to "magnify their human resources" and has experienced widespread adoption. ODK's website has received over 700,000 unique visitors from 232 different countries/territories and averages over 30,000 hits a month. ODK's 1.x toolkit [20] is a successful data collection platform enabling users to collect millions of data points for a diverse set of domains (e.g., health, election monitoring, disaster response). Over 210,000 users have installed ODK Collect according to Google Play (excludes ODK derivative apps distributed by other companies or installations directly from the ODK website). Analytics reports ODK Collect is used daily by thousands of users in over 130 distinct countries. However, ODK 1.x's uni-directional data flow, linear navigation, and limited data types of Java-Rosa's XForm standard constrained its applicability to certain categories of use cases. While ODK 1.x was designed to enhance and replace paper-based data collection, the focus on simplicity left certain requirements unmet. The 2.0 tool suite [9] expanded ODK features to include two-way data synchronization, a local database for disconnected operation, runtime modifiable user interfaces, and a platform for customizable user applications.

Based on feedback from ODK 2.0 pilot deployments in over 18 countries by a variety of organizations, the 2.0 tool suite went through a significant redesign from the original ODK 2.0 vision [9].

MobiSys'17 June 19-23, 2017, Niagara Falls, NY, USA

ACM ISBN 978-1-4503-4928-4/17/06.
DOI: http://dx.doi.org/10.1145/3081333.3081365

This paper summarizes the constraints and feedback that drove the revision. ODK 2.0 was redesigned as a modular service-oriented architecture, comprised of six tools, with framework abstractions that were flexible enough to empower users of varying technical skill levels to build mobile data management applications tailored to their specific use cases. ODK 2.0 aims to create a *data management platform* for resource-constrained environments that provides greater functionality than the ODK 1.x's *data collection platform*.

This paper discusses ODK 2.0's revised framework and presents the design considerations that led to its evolved modular service-based architecture for disconnected data management. We describe how system design decisions were influenced by the challenged network environments and resource-constrained contexts. We examine the implications of the revised architecture and show it had relatively minimal impact on performance. We also share feedback from multiple organizations that experienced limitations with 1.x and participated in ODK 2.0 pilots and discuss how the revised framework has better met their deployment requirements.

2. DESIGN CONSIDERATIONS

The ODK 2.0 tool suite provides an Android application framework that reduces the complexity organizations face when creating a mobile data management application. Organizations in resource-constrained environments often lack the technical personnel or resources to build and customize information systems. Instead, organizations commonly use productivity software (e.g., Excel, Word) to create solutions customized to their domain and context. Microsoft Office is an example of software that is adaptable to multiple domains by users with no programming expertise. Unfortunately, productivity software is often designed for conventional personal computers which are usually poorly suited to resource-constrained environments. While personal-computer-based productivity applications generally have a broad focus for domain utility, mobile apps (Android native executable applications that will be referred to as 'apps' for the scope of this paper) tend to reduce their scope to a single task (e.g., calendar, maps, email) and often lack customization features.

ODK 2.0 enables creation and customization of domain independent mobile applications that address organization's needs within constraints imposed by Android. ODK 2.0's synchronization protocols and structures are designed to be adaptable in extreme mobile networking conditions, such as long periods of disconnection or low bandwidth and high latency. ODK 2.0 replicates data to the mobile devices, enabling the framework to preserve full functionality in disconnected environments.

ODK takes a multi-perspective approach to the creation of customizable abstractions for varying contexts [10]. The *platform perspective* incorporates the device and operating system perspective on aspects such as connectivity, power, features, and mobility. The *application developer perspective* encompasses the features and functionality of an 'app' that executes on a specific platform. It also incorporates the perspective of the software developer who writes the 'app'. The developer's choices about what functionality and platform features to expose constrain how the 'app' can be used. Unfortunately, developers likely do not fully understand how future users will want to deploy the framework in different contexts with varying connectivity, budgets, laws, data policies, user education level, etc. Therefore, ODK focuses on creating frameworks that give flexibility to a *deployment architect* who has an *application deployment perspective* that encompasses issues relating to contextual deployment requirements that are often dynamic.

We identify the *deployment architect* as the non-programmer or programmer with limited technical skills who adapts an ensemble of off-the-shelf software (e.g., ODK 2.0) to create a data management application. Unlike standard reusable model-view-controllers, ODK attempts to create abstractions usable by *deployment architects* emulating productivity software that targets non-software developers. In addition to *deployment architect*, other ODK *users* are often site managers, supervisors, and data collectors that use the configured data management platform to complete their tasks. For an organization to successfully implement a mobile application in a resource-constrained context, *deployment architects* must be able to scale their applications. ODK tool suites leverage commercial cloud offerings and Android's common APIs to simplify an organization's ability to scale. Building on Android OS enables organizations to easily purchase a diverse set of commercially available smartphones and tablets with varying form factors, battery life, ruggedness, and prices. While there is a notion that the cloud "effortlessly scales," in reality there are complex technical issues handled by an army of software engineers at cloud companies (e.g., Amazon, Microsoft). ODK 2.0's cloud-based components leverage this army and work in concert with ODK's Android application framework with non-programmer abstractions to insulate the *deployment architect* from technical details so that organizations can "effortlessly scale" their mobile data management applications.

2.1 Limitations of ODK 1.x

The ODK 1.x tool suite is a successful data collection platform used by many organizations to digitize their data collection in the field. The system design focuses on collecting data via digital surveys that are then aggregated in the cloud or on a PC for analysis. However, ODK 1.x's purposeful simplicity to enable a *deployment architect* to replace paper data collection with mobile data collection meant the tool suite lacked features for certain use cases.

To illustrate, consider the common scenario in which organizations use previously collected data to influence their next action when revisiting a specific location. Examples of such a scenario include logistics management, patient follow-up in medical care, and environmental monitoring. In these scenarios, users often return to locations and reference previously collected data that they either verify or update. Each visit may require field workers to complete multiple forms, produce multiple data records, and/or require multiple review steps. Each phase may draw upon data collected during any other phase or data found in supporting information tables, this data needs to be available even when disconnected. Revisiting data from previous surveys is not supported in ODK 1.x.

The multi-visit scenario requires bidirectional synchronization of data, support for complex workflows, references or updates to previously collected data, and security and data isolation for specific users. Working with partner organizations, we constructed a list of common requirements that were not met by the 1.x tools. These were discovered both through initial design sessions as well as lessons learned from field deployments. Some key requirements for ODK 2.0 that were beyond the scope of ODK 1.x include:

- Complex / Non-Linear Workflows
- Linking Longitudinal Data to Collected Data
- Data Security and User Permissions
- Reuse of Data Fields across Forms
- Bidirectional Synchronization
- Customizable Form Presentation
- Custom Apps Built with a Runtime Language (JavaScript)
- Sensor Integration
- Paper Digitization
- Custom Data Types that Update Multiple Fields in a Single User Action

To meet these requirements, ODK 2.0 expands its scope from ODK 1.x's *data collection platform* to a *data management platform*. A mobile "data management application" (designated as an 'application' to delineate from the previously defined 'app') is the customized set of activities created by an organization to perform its data collection and management workflows to accomplish its business objective.

2.2 Design Principles & Features

The ODK 2.0 tool suite is a parallel effort (not a replacement) to the ODK 1.x tool suite; it provides the general-purpose tools for a virtually unlimited set of use cases and enables organizations in resource-constrained environments to build, deploy, maintain, and ultimately own mobile data management applications and business logic. While many of the details have evolved, the four design principles that were outlined in the ODK 2.0 vision paper [9] for refinement and expansion of ODK remain the same:

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;*
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.*

Expanding on these basic design principles, a larger list of features were generated based on specific use case requirements. For example, dynamic value checking based on previous data was required to improve data integrity which is seen as a key benefit of digital data collection. Consider agricultural longitudinal studies: agricultural extension workers visit crops multiple times during a growing season. They track the progress of the crop to compare growing conditions and try to improve crop yields. However, the validation logic in ODK 1.x uses static formulas so it has an absolute min and max that would differ early in the season from later in the season. Thus, to better support this type of longitudinal study, ODK 2.0 should allow its *data management applications* to access previous crop heights and use dynamic calculations to catch data anomalies. The following includes a more detailed view of some of the key design goals of the ODK 2.0 framework.

- Workflow navigation should use intuitive procedural constructs, function independent of data validation, and allow for user-directed navigation of the form.
- The presentation layer must be independent of the navigation and validation logic.
- The presentation layer must be customizable without recompiling the Android apps via HTML, JavaScript, and CSS.
- Partial validation of collected data should be possible and the validation logic should be able to be dynamic.
- Local storage should be robust and performant for data curation and for longitudinal survey workflows using a relational data model.
- Multiple data collection forms should be able to modify data within a single, shared, data table.
- Foreground and background sensors should be supported for data collection.
- Disconnected operation should be assumed as data must be able to be collected, queried, and stored without a reliable Internet connection. When the Internet becomes available, the framework and cloud components should efficiently replicate data across all devices.
- User and group permissions are needed to limit data access.
- Cloud components should be able to fully configure the data management application remotely as well preserve a change log of all collected data.

3. RELATED WORK

A variety of existing solutions and application frameworks attempt to solve varied issues; however, ODK focuses on a certain class of problems. What distinguishes ODK 2.0 from other solutions is its focus on providing a suite of inter-operable tools that work together to provide base functionality for a flexible information management system. For example, Apache Cordova [1] (the open-source version of Adobe's PhoneGap) is a popular open-source framework for building mobile applications with HTML and JavaScript. It provides a native plugin-framework to access hardware features of the mobile device. Many mobile application frameworks wrap and augment Cordova in other products, such as Ionic [3] and Intel XDK [2]. Similar to ODK, Cordova uses a custom native SQLite library; however, Cordova and ODK have different overall focuses. Cordova, designed for programmers, helps solve the problem of cross compatibility for different OSes (e.g., iOS, Android, Windows, BlackBerry, Ubuntu, FireOS) to let programmers write code once and deploy it on multiple platforms. ODK 2.0 instead focuses on making it simpler for non-programmers to create mobile applications specifically for the Android platform. ODK focuses solely on Android compatible devices because these devices come in a variety of form factors with different price points, making Android devices popular in economically constrained environments, unlike more expensive iOS devices (Android had more than 87% world smartphone market share in Q2 2016 [4]).

Computing for global development research has produced multiple mobile application framework research projects. CAM [21], one of the earliest mobile application frameworks for resource-constrained environments, used J2ME phones with a custom scripting language and barcodes to augment paper forms and trigger custom prompts for manual data entry. Unfortunately, CAM was tied to a specific phone model, and while hooks were exposed for customization, the custom scripting language was seen as a barrier to entry as it was a new skill someone had to acquire. In comparison, ODK 2.0 exposed JavaScript as its customization language to allow the tools to have a pre-existing base of trained programmers. Another early example was the E-IMCI project [15], which used PDAs as the mobile computing device in resource-constrained environments to encode complex medical workflows. While E-IMCI was not customizable, it provided an early example of how mobile computing platforms could improve an organization's workflow in remote locations. The Uju project [22] focused on making it easier to create database-centric applications for SMS, but, unlike ODK 2.0, it was not designed to integrate with multiple tools that obtain data from various inputs (e.g., sensors, paper forms). The CommCare framework with CaseXML [16] has design goals similar to ODK; it enables organizations to customize their field worker's digital workflows. CommCare operates on both J2ME and Android devices and uses *Collect* (an ODK 1.x tool) as the base for the CommCare Android app by adding custom features to work with CaseXML. CaseXML lets organizations specify an application workflow and data exchange, enabling health workers to share 'cases' through synchronization of atomic transaction in

an XForm. While CommCare has been a successful platform for organizations with field healthcare workers, other organizations reported problems adapting CaseXML to other domains. The limitations reported by organizations with CaseXML contributed to the decision to abandon XForms entirely in the ODK 2.0 design.

Several research projects have tried to simplify mobile development by providing abstracted table APIs that assist mobile app programmers with data management. Two examples of such projects that build database-table-like schematics are Izzy [19] and Mobius [12]. Like ODK, Izzy tries to make application development easier by providing a simple API to access database tables; however, these APIs are targeted at programmers. Mobius takes a different approach and presents logical table extractions but uses a unified messaging scheme for updates. This approach becomes problematic for long periods of disconnected operation, which are expected to be experienced by ODK 2.0 users.

4. MOBILE FRAMEWORK

The ODK 2.0 tool suite (individual tool names are italicized) provides a platform for organizations to create customized data management applications on Android devices. Based on feedback from multiple pilot deployments (some are described in section 5), the ODK 2.0 architecture went through a significant redesign to address multiple issues. In the first iteration of the architecture, the modular mobile tools were designed as completely independent applications that directly accessed a single shared Android database. The basic concept was that all applications would individually read and write data directly into a shared database, thereby enabling data sharing and consistency. Unfortunately, the simplistic design led to database contention problems, security issues, performance issues, and limited code reuse. This section presents how the Android applications in the ODK 2.0 tool suite have been re-architected to follow a services-based architecture running on the Android device. Key aspects of the architecture are: 1) Modular Design – individual Android tools designed for particular tasks that, used in concert, can achieve complex behaviors, 2) Data and Configuration Management – the storage, sharing and protection of configuration and data across mobile devices and cloud components, 3) Synchronization Protocol – designed to adaptable in extreme networking conditions, 4) Services Architecture – common functionality consolidated into a unifying services layer.

4.1 Modular Design

The ODK 2.0 framework seeks to satisfy an organization's specific usage scenario through the composition of narrowly focused tools rather than a single monolithic app. An app that tries to provide abstractions for every usage scenario can be overwhelming, so ODK tools focus on a singular task and are designed to smoothly transition between one another to give the feel of unified application. The six apps shown in Figure 1 each contribute abstractions to specific areas: *Scan* the paper digitization framework, *Tables* the data curation framework, *Survey* the question rendering and constraint verification framework, *Sensors* the device connection framework, *Services* the common data services framework, and *Submit* the communication framework. To fulfill complex requirements, the deployment architect uses the necessary subset of tools in concert to build a customized data management application.

4.1.1 Services

The Android *Services* app contains common services that are used by all ODK 2.0 apps running on the mobile device. One of the key components of *Services* is the database service that abstracts database access to a single shared interface that enables the enforcement of consistency semantics and data-access restrictions. *Services* also exposes interfaces to other shared functionality, including a lightweight web server, user authentication, framework preferences, and the cloud synchronization protocol. The ODK 2.0 the synchronization protocol is discussed in Section 4.3 and the service-oriented architecture is discussed in detail in Section 4.4.

4.1.2 Survey

Survey focuses on data collection through the use of questions. Users progress through question prompts as they capture data, similar to the flow of an ODK 1.x form. However, departing from the 1.x model, JavaScript and HTML are used to define a suite of prompt widgets and to encapsulate the rendering, event handling, and form navigation logic required for data collection. The presentation layer can be easily customized by revising *Survey's* templates and CSS style sheets to create an organization-specific look-and-feel. To simplify survey design, the XLSForm syntax from ODK 1.x was restructured to reduce the number of spreadsheet columns and move complex programming syntax into a single column. The aim is to make it easier for users to initially learn the simpler *Survey* definition structure and then ramp up their knowledge as they require more advanced features. To help users keep complex workflows organized, the concept of 'sections' was added to enable logical groupings of questions. The concept of 'repeats' is replaced by the concept of 'subforms' to give users the full power of an independent survey definition when dealing with repeated input sets.

Survey increases an organization's ability to customize an application by implementing the user interface with a run-time scripting language that allows for customizing without recompiling. The entire flow of *Survey*, from the opening screen to the outline/index view of a form, can be redefined or extended, thereby enabling more customization than XForm syntax of ODK 1.x's Collect tool. Organizations can extend standard widgets at runtime to: change the rendering or workflow logic itself (e.g., complex types, value constraints), customize event handling, or define new question widgets. *Survey* has several new features that improve ODK's malleability to various application domains. Since the question widgets are all modifiable at runtime, organizations can now define custom question types with multiple input values but with a single question prompt (e.g., blood pressure, pulse/oxygen) instead of being limited to the one-question/one-value model of ODK 1.x's JavaRosa XForm. Furthermore, users can use the new complex commands to express non-linear workflow logic using 'gotos' in the survey definition. Buttons can be specified in *Survey* to perform user-directed form navigation expressed, effectively, as computed 'goto' (switch-like) statements. Users are also no longer limited to using one form to edit a row of data. *Survey* allows multiple forms to edit a row of collected data, which can assist in a variety of situations including longitudinal studies.

4.1.3 Tables

While *Survey* focuses on collecting and editing data, *Tables* focuses on displaying collected data to the user [8]. *Survey* serves as the primary tool to manipulate an individual row, while *Tables* provides functionality to display and interact with the entire data set. *Tables* serves as a platform to host what is essentially a webpage, allowing customizations via HTML, CSS, and JavaScript. Views of the data can be both tabular or graphical depending on the context and what is appropriate for an end user's task. Views can also link to other views or launch other ODK apps.

The flexibility provided by *Tables* greatly increases the usability of the 2.0 tool suite. Deployment architects can capture complex workflows and decision logic in *Tables*. For example, consider a

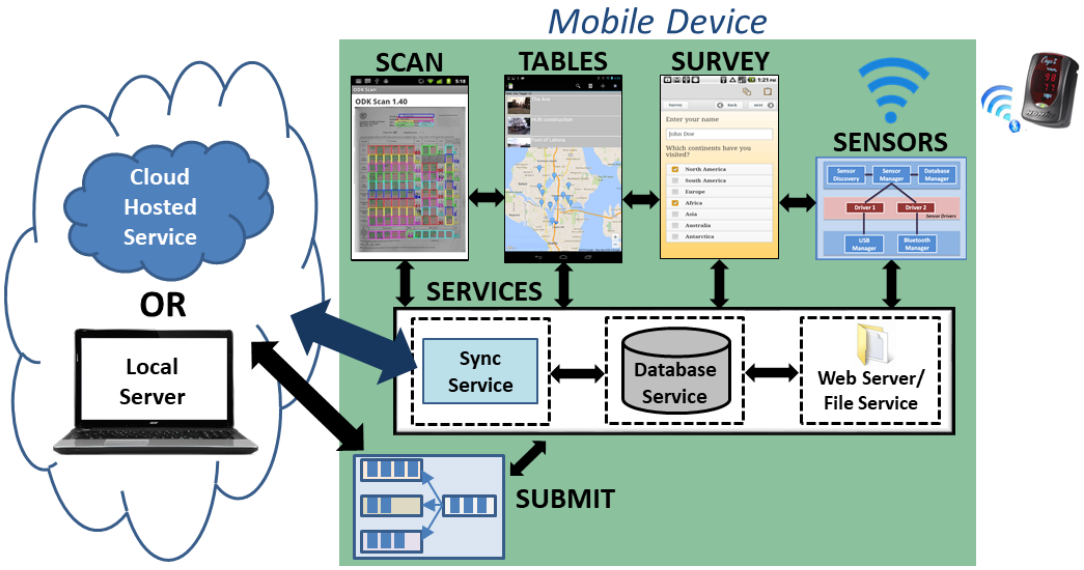

Figure 1: ODK 2.0 modular framework's five mobile tools are used in concert to create customizable *mobile data management* applications. The five ODK mobile 'apps' that comprise the framework are *Services*, *Survey*, *Tables*, *Scan*, *Sensors*, and *Submit*.

longitudinal study that needs to revisit subjects at multiple time points within a deployment. The data collector might be expected to follow different steps or complete a different set of forms when enrolling new subjects than when following up with existing ones. In an ODK 1.x deployment, the correct procedure must be imparted to a data collector through training. As a workflow becomes more complex, the training and support materials (e.g., flow charts) must increase as well. With *Tables*, however, the correct logic can be encoded in the app, both lowering the training burden and increasing compliance to the study protocol. Upon opening *Tables*, a collector would be presented with an HTML page with options to enroll or revisit a subject. When selecting the correct option, a new page would be opened that contains instructions and the set of forms required. Selecting one of these forms would then open *Survey* to permit data collection. While *Tables* apps will typically reference files (web content) stored on the device and synchronized via *Services* (in support of disconnected operations), deployment architects can also choose to issue network requests.

4.1.4 Scan

Scan bridges the gap between paper and digital data collection without forcing organizations to wholly adopt either scheme [13]. Paper remains an integral component in the workflows and data management processes of many development organizations, filling niches where digital systems are not currently appropriate [14]. *Scan* provides these organizations with a tool to quickly and easily digitize their paper forms in disconnected environments. Paper forms are adapted by adding *Scan*-compatible data input components, which include QR codes, multiple choice bubbles and checkboxes, structured hand-written number boxes, and free-form hand-written text boxes. Both digital and physical versions of this form are generated: a *Survey* form definition and a JPEG image file that can be used to print and distribute the *Scan*-compatible form. *Scan* can digitize completed forms in the field by taking a picture with the device's camera. Computer vision algorithms, running on the device, process and segment the form image into snippets corresponding to the individual data components [13]. These snippets (excluding free-form, hand-written text boxes that *Scan* cannot digitize programmatically) are then fed into classifiers that digitize the respective values. The digitized values can be reviewed and free-form, hand-written text values transcribed by viewing the form instance in *Survey* or *Tables*. All image processing is performed on the device without the need for an Internet connection. Form images can be processed in real time, typically requiring less than one minute per form, or saved and batch processed at a later time.

4.1.5 Sensors

Sensors simplifies the task of integrating external hardware sensors into an organization's data collection and management workflow [7]. For example, a medical application might use a pulse-oximeter during screening, or a vaccine refrigerator might need to report its temperature for regular audits. To address these requirements, *Sensors* provides a modular framework for organizations to implement drivers and user-level applications for third-party sensors that integrate into the ODK 2.0 ecosystem. The driver-level implementation and the user application are developed independently, and *Sensors* provides a common interface that abstracts communication channels between the two domains. This lets framework software programmers develop sensor drivers that can be reused by multiple user applications, while deployment architects can abstract sensor particulars and focus on their particular usage scenario needs. The *Sensors* app integrates with *Services* database directly, providing easy integration of sensors into complex data collection and management workflows.

4.1.6 Submit

The ODK 2.0 framework is built with performance in disconnected environments as a core tenant of its design philosophy. However, the synchronization service provided in *Services* requires Internet connectivity before data can be shared among devices. Despite growing access to cell coverage throughout the world, sufficient bandwidth to share the megabytes and gigabytes of data generated by data management applications is not always available, convenient, or cost efficient. *Submit* provides a framework for organizations to adapt their applications to share data using appropriate technologies for their various network conditions and data communication needs [10]. A variety of networking and data transfer technologies are exposed, ranging from peer-to-peer communications technologies (e.g., Bluetooth, WiFi Direct) to more traditional cellular and WiFi access. Deployment architects can prioritize which data should be shared using which technologies under what circumstances. For example, a clinic might share patient records over Bluetooth connections, schedule high-priority case files to be uploaded to the cloud over cell service as it becomes

available, and perform regularly scheduled full synchronizations at Internet cafes. *Submit* separates networking implementations from application logic by exposing a service API with which to schedule data transfers or receive notifications about changing conditions.

4.2 Data and Configuration Management

ODK 2.0 data management applications consist of survey definitions, web content, configuration files, data tables, and data rows. These are consumed by the framework to define the control flow, user interface, schema, data validation logic, and business logic of the application. Departing from the ODK 1.x framework, which uses an XForm-based hierarchical document model, the ODK 2.0 framework stores data in a relational database on the Android device. The presentation layer can execute arbitrarily complex SQL queries (e.g., joins, unions), vastly increasing the expressive capability of the 2.0 framework. This expressiveness is required for complex workflow decisions, such as the disaster response case study in Section 5.4 that requires queries of previously collected beneficiary data to determine future distributions. Relational databases also permit more complex relationships between data sets. Dependent data sets, similar to ODK 1.x's 'repeat groups' concept, that were previously store in the same document are now stored in a separate data table allowing the deployment architect to define business logic to maintain their linkage with other data sets.

The ODK 2.0 framework uses a cloud component to provide robust permanent storage of data and configuration information. The data management application interacts with the cloud component through RESTful interfaces. The ODK 2.0 cloud component manages the 1) user permissions (user authentication and group membership), 2) slowly changing configuration (configuration files, form definitions, data table definitions), and 3) frequently changing data (content of the data tables and associated row-level media attachments). To support disconnected operations *Services* uses the ODK 2.0 synchronization protocol to maintain a filtered snapshot of the user permissions and a full snapshot of its configuration and data onto each mobile device. Upon connecting to the Internet, the mobile device initiates the synchronization protocol to reconcile changes and update its local state to a new, reconciled snapshot of the cloud component's content.

On the device, a user's identity is established via a successful login to an ODK cloud component. Thereafter, the user's identity and permissions are cached until the user resets his or her credentials or until the device is next synchronized with the cloud component. During this disconnected period of operation, successfully unlocking the device is considered sufficient to re-confirm a user's identity. There are 4 classes of users: (1) anonymous and/or unauthenticated users, (2) authenticated *unprivileged users* with permissions to read and modify a subset of the data, (3) authenticated *superusers* with permissions to read and modify all data, (4) authenticated *administrators* can read and modify all data, update user permissions, and change the configuration files that specify the data management application. Administrators and superusers can manage which authenticated users can see or modify which rows. Unprivileged users may be given access and modification rights to individual rows that unauthenticated or anonymous users cannot access and/or modify.

4.3 Synchronization Protocol

The ODK 2.0 data synchronization protocol is designed to be adaptable in extreme networking environments with high latencies, low bandwidths, and long periods of disconnected operation. To accommodate these varying conditions, the synchronization protocol was designed to use a small granularity of change-tracking to enable smaller data transmission and simplify conflict resolution. A single database row was chosen as the base unit of the synchronization protocol to follow the design principles of ODK 2.0 outlined in Section 2.2. Tracking synchronization state at the data-row-level ensures the efficiency of the synchronization protocol by using a reasonably small unit of data. A common use case for ODK 2.0 entails multiple enumerators collecting longitudinal data over the span of many days with no network connectivity. Based on feedback from partners, off-the-shelf file-based solutions (e.g., Dropbox, OneDrive) are not sufficient as the probability of having multiple enumerators edit the same file during long periods of disconnected operation is very high. In these file-based solutions, resolving conflicts once connectivity is re-established is challenging. These issues are avoided with a row-based solution because the probability that multiple enumerators touch the same rows (e.g., visit the same sites) decreases; thus, less conflicts are generated.

Data rows use globally unique primary keys to track the data stored at the row level. If any data in the row changes, an new ETAG (a universally unique identifier) is generated to identify that changes have been made to the row. By tracking changes at the row level, the bulk of the database content on the device can stay in sync with the server without any communication overhead. Names of any files attached to these data rows are also globally unique, thus only a single instance needs to be transmitted for multiple rows to reference it. The files themselves are stored on the device's file system and treated as immutable once referenced by a data row. The server maintains a manifest of the application-level files, their hashes, and their size. It computes a unique ETAG from this manifest content. If the ETAG of the last application-level file manifest on the device does not match the corresponding ETAG on the server, then the application-level files have changed. In this case, the device then scans its folders to confirm that its local copies match those in the server manifest, resolving any conflicts. A similar interaction is repeated for table-level files.

The server assigns a schema ETAG and a version ETAG to each data table when it is created. This lets the device verify that its table schemas match those on the server and, if a version ETAG differs but the schema ETAGs match, the client knows that the device's copy of the rows are out of sync with the server. Every row is tracked with a version ETAG and a synchronization state: 'synced', 'new_row', 'changed', 'deleted', 'in_conflict', or 'synced_pending_files'. For each table, the rows in the 'new_row', 'changed', and 'deleted' states are processed. If these rows do not have conflicting changes on the server, the device's synchronization state for these data rows transitions into the 'synced_pending_files' state (or are removed, for those formerly in the 'deleted' state). Finally, for each data table, the data rows in the 'synced_pending_files' state are processed. Before beginning a synchronization, the user can choose whether to perform a possibly bandwidth intensive upload and/or download file attachments from the server. The choice enables users to optimize their use cases for the currently-available communication bandwidth and cost. If bandwidth is low or cost is high, they may elect to do nothing, leaving the rows in the 'synced _pending_files.' This recognizes that transmission of files over lower-bandwidth communications channels is generally both time- and cost- prohibitive. However, users in extremely remote locations may decided that the few times they reach Internet connectivity it is worth the cost to perform a full synchronization based on their business requirements. The change-tracking design enables the synchronization protocol to adapt to currently-available communication bandwidths and costs.

When a row change on the device conflicts with a row change on the server, the change on the server is pulled down to the mobile

device and the data row is marked as 'in_conflict'. The user must then resolve the conflict by either taking the server's change, the local change, or mix of the local change and the server change. If the user does not take the server's change, the synchronization state of the data row is updated to either 'changed' or 'deleted'. On the next synchronization, the local updates will then update the state on the server resolving the conflict.

4.4 Services Architecture

In the first iteration of ODK 2.0, tools were designed as completely independent Android apps that directly accessed a single shared Android database. The shared database presented users with a seamless data management application experience by smoothly transitioning between the tools (e.g., from *Tables* to *Survey* and back). The tools evolved to each include their own logic for accessing the database. From within the WebKit, the shared Android database was directly accessed via WebSQL. Stress tests of ODK app transitions exposed an instability between the WebKit and the Java layer's Android UI lifecycle[1]. Interleaving multiple Android UI lifecycle events during transactions resulted in deadlock if the outgoing activity did not release the transaction. The rate of deadlock occurrences varied by device model and OS version.

To resolve the deadlock issue the tools were re-architected to access the database through *Services*. The *Services* app provides a centralized database service that facilitates 1) an enhancement of initialization with automatic pre-population of tables with data rows from configuration files and 2) a reduction in overhead when opening an already-initialized database since *Services* maintains knowledge of the database state. After creating the database service we uncovered deficiencies within the Android database implementation including reference counting issues, database locking, and the database connection object being stored in thread local storage. These concerns led us to use a custom build of the SQLite database with a Java wrapper to eliminate the need to address behavioral differences across Android OS levels, 2) enable direct management of connections, 3) speed up opening of connections, 4) enable use of write-ahead-logging, and 5) enable changes to the native interface to minimize memory allocations.

While *Services* exposes content providers for a few basic static queries, the primary access to stored data is through the database service. The database service presents a restrictive API that does not expose database transactions or cursors. Instead, it exposes atomic-update primitives and queries that return snapshot-in-time result sets. This atomicity eliminates the possibility of a misbehaving client causing deadlock or leaking cursor resources. We chose a service over a content provider because 1) insert/update/delete operations would not capture the atomic update primitives used to simplify database management, 2) the full capabilities of a SQL query cannot be exposed without introducing our own expression language embedded within that URL as opposed to a service API, and 3) the ability to apply row-level access control cannot be expressed through a single content provider. To use a content provider would require a custom defined URL parser for SQL syntax. This would introduce custom non-standardized parsers causing a future pain point similar to the custom JavaRosa XForm definition and parser in ODK 1.x. Android Services have a 1 MB size limit on remote-procedure calls which is insufficient for large result sets. We implemented a primitive transport-level chunking interface using a client-side proxy to re-assemble chunks and only expose a higher-level abstraction to the tools. The client-side proxy provided additional benefits, such as caching and typed exception throwing across processes.

To create a unified user experience, all application-level settings were consolidated into the *Services* app. Additionally, the local web service and the synchronization service were centralized into *Services*. By placing the synchronization service within the same trusted layer as the database service, we could maintain a chain of custody for the authenticated user identity gained through the interactions with the cloud component. This authenticated user could then be used when enforcing row-level and table-level permissions inside the database service. More specifically, it enabled us to: 1) verify the active user's identity, 2) fetch that user's permissions from the cloud component, and 3) cache the user's identity and permissions within this trusted services layer where those permissions could be used within the database service to enforce the visibility and modification constraints on the data tables.

Services also simplifies tool updates. By effectively consolidating persistent state behind this service layer, individual tools can be upgraded in a rolling-update fashion without affecting persistent state. If the data representation (schema) needed to change, only a single update of the *Services* tool is required. Other benefits include defining our own service APIs makes it easier to port ODK software to other platforms and increased code reuse allows for improved testing and stability.

4.5 Experiments

To evaluate the revised architecture, six different variants of ODK 2.0 'apps' were created to compare performance timings. The builds differed by type of database, whether an Android service was used, and whether the logic was separated into two different APKs or combined into one. We performed the tests on three different Android devices, across a spectrum of performance capabilities: a ~$40 Vodaphone purchased in Kenya, a Nexus 7 tablet (a device commonly used by our partners), and a Nexus 6 phone. We used Android OS versions commonly found in our target regions.

The setup used to evaluate performance consisted of an ODK 2.0 data management application and automated Android UI tests. The application queried 200 rows of a 3000-row data set in succession until 100 iterations had been performed. A timestamp was recorded by the application before the query was issued, when the query results returned, and when the application had finished rendering the UI elements on the screen. Table 1 presents the average query turnaround time, the average WebKit rendering time, and the overall (Full RTT) times. It compares queries performed through the service API with the service and caller residing in separate APKs (the standard ODK 2.0 configuration), queries performed through the service API with the service and caller bundled into the same APK, and queries performed directly against the database without any service intermediary. These tests show that restructuring our tools to use a service-oriented architecture (SOA) did not incur prohibitive performance penalties. Comparing the *SOA Multi APKs* timings and the *Direct DB Access* timings showed a ~150 ms overhead with SOA; which is well below the 940 ms rendering time incurred by the WebKit and is below the limit of perceivable delay.

An alternative architecture combines all tools (e.g., *Tables*, *Services*) required for a particular usage scenario into a single large multi-dex APK. The timings results in Table 1 (comparing *SOA Multi APKs* timings and *SOA Single APK*) shows that there was a performance penalty, perhaps due to the increased proportion of available memory consumed by application code within a single APK. Combining only *Tables* and *Services* incurred a 25-50 ms penalty on a high-end device. If this penalty were driven by appli-

[1]The WebKit performs synchronous calls to the Java layer, but only asynchronous calls from the Java layer to the WebKit are provided. Android lifecycle event handlers are synchronous and provide no mechanism for asynchronous resumption of lifecycle transitions.

Table 1: Database Query Timing Comparison of Different Implementations on three different Android Devices. Service-Oriented Architecture in Multiple APKs, vs Service-Oriented Architecture in Single APK, vs Direct Database Access.

Device & Operating System	*Architecture*	*Database Type*	*Query Avg. (ms)*	*Query Std. Dev*	*Webkit Render Avg. (ms)*	*Full RTT Avg. (ms)*	*Full RTT Coeff. of Var.*
Vodafone v4.4.2	SOA Multi APKs	custom	2572	60	1474	4046	2.98
Vodafone v4.4.2	SOA Single APK	custom	2635	222	1475	4110	6.81
Vodafone v4.4.2	Direct DB Access	custom	2518	62	1446	3964	3.21
Vodafone v4.4.2	SOA Multi APKs	default	2685	56	1509	4194	2.75
Vodafone v4.4.2	SOA Single APK	default	2692	74	1539	4231	3.28
Vodafone v4.4.2	Direct DB Access	default	2631	76	1504	4135	3.17
Nexus 7 v4.4.4	SOA Multi APKs	custom	881	102	1049	1930	8.02
Nexus 7 v4.4.4	SOA Single APK	custom	821	80	1063	1884	7.86
Nexus 7 v4.4.4	Direct DB Access	custom	613	77	974	1587	8.58
Nexus 7 v4.4.4	SOA Multi APKs	default	993	92	1047	2040	6.78
Nexus 7 v4.4.4	SOA Single APK	default	1083	106	1067	2150	7.75
Nexus 7 v4.4.4	Direct DB Access	default	1358	209	1043	2400	9.89
Nexus 6 v5.1	SOA Multi APKs	custom	402	52	944	1347	17.63
Nexus 6 v5.1	SOA Single APK	custom	425	53	968	1393	16.61
Nexus 6 v5.1	Direct DB Access	custom	261	34	942	1202	17.87
Nexus 6 v5.1	SOA Multi APKs	default	615	84	956	1571	14.54
Nexus 6 v5.1	SOA Single APK	default	663	101	995	1658	13.36
Nexus 6 v5.1	Direct DB Access	default	386	41	942	1328	14.52

cation code size, then an SOA would be more performant than a monolithic APK should the six ODK 2.0 tools be combined. Tests to determine how the SOA scales given changing database query sizes were also performed. For this round of tests, the test setup again consisted of an ODK 2.0 data management application and automated Android UI tests. The application repeatedly queried a 3000-row data set with a fixed query size (one of 2, 20, 200 or 2000 rows) until 100 iterations had been performed via the Android UI tests. In this case, no UI elements were rendered after the query results were returned. The application recorded a timestamp before the query was issued and when query results returned. Table 2 shows the average round-trip time and standard deviation for the database queries. Table 2 shows the times for a Nexus 6 running the 5.1 version of the Android OS and a Vodafone running the 4.4.2 version of the Android OS. These results show that our query times scale as expected with a lower end device taking proportionately longer than a more powerful device.

Table 2: Avg Database Query RTT vs Number of Rows Queried

Rows	*Nexus 6 Avg. RTT (ms)*	*Nexus 6 Std. Dev (ms)*	*Vodafone Avg. RTT (ms)*	*Vodafone Std. Dev (ms)*
2	207	48	483	46
20	233	56	549	46
200	402	61	975	66
2000	2156	87	6008	139

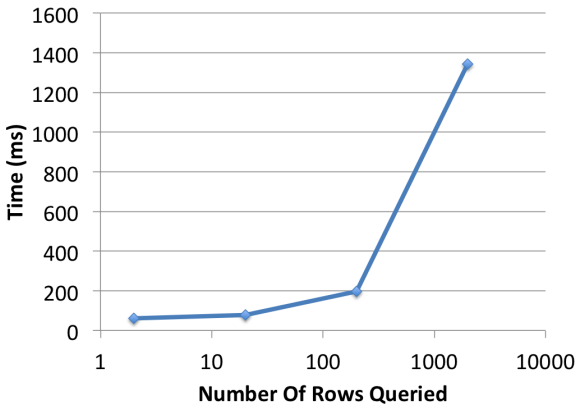

Figure 2: Nexus 6 5.1 Average Java To JavaScript Transfer Time vs. Number Of Rows Queried

Finally, the amount of time spent marshaling database query results between the Java layer and the JavaScript layer was measured. To determine the transfer time between these layers, an ODK 2.0 application and automated suite of Android UI tests were used with the 3000-row data set. A timestamp was recorded in a log file when the Java layer obtained the query result and when the JavaScript layer received the query result. Figure 2 shows the average time difference between when the database query result was available in the Java layer to when the database query result was returned to the JavaScript layer for 100 test iterations. This data shows that the transfer time from the Java to JavaScript layer did not have a major performance impact.

5. CASE STUDIES

A guiding principle of ODK has been to work with field partners during technology development to validate the suitability of the tool suite in resource-constrained environments. This section presents case studies from partners working in global health, disaster management, and other areas to demonstrate and verify the reuse, flexibility, and extensibility of ODK 2.0 application framework, as well as to report key lessons learned. We enumerate requirements for each case study in Table 3 that were identified as necessary in the ODK 2.0 framework but are not fulfilled by the ODK 1.x tools.

5.1 Childhood Pneumonia

The mPneumonia project is a partnership with PATH, a non-government global health organization, that supports health workers in low-resource environments. PATH sought to create a mobile application to improve health care providers' diagnosis and management of childhood pneumonia, a leading cause of disease deaths for children under 5 [17]. The resulting diagnosis and treatment application assisted lightly trained healthcare workers with screening

Table 3: Case Study ODK 2.0 Feature Requirement Summary

	Childhood Pneumonia	*Chimpanzee Behavior Tracking*	*HIV Clinical Trial*	*Disaster Response*	*Mosquito Infection Tracking*	*Tuberculosis Patient Records*
Complex / Non-Linear Workflows	X	X	X	X	X	
Link Longitudinal Data To Collected Data	X		X	X	X	X
Data Security and User Permissions	X		X	X	X	X
Reuse of Data Fields Across Forms			X	X		
Bidirectional Synchronization	X		X	X	X	X
Customizable Form Presentation	X		X	X		
Custom JavaScript Apps		X	X	X	X	X
Sensor Integration	X					
Paper Digitization						X
Custom Data Types Update Multiple Fields in a Single User Action	X	X		X	X	

tasks. *Survey* was used to digitize complex medical workflows and customize their form presentation, while *Sensors* was used to integrate a USB sensor to record a patient's oxygen saturation.

Previous projects found that digitizing the World Health Organization's Integrated Management of Childhood Illness (IMCI) protocol on PDAs led to increased adherence while maintaining examination time [15]. However, when other organizations tried to re-implement IMCI using ODK 1.x, they struggled to map IMCI's complex medical workflow to XForms. ODK 2.0 addressed the complexity with *Survey*, which allowed a non-programmer global health intern to digitize the IMCI protocol using an Excel workflow description format. Furthermore, the layout, styling, and presentation of each prompt in the form was customized to meet the specific needs of the mPneumonia project. These customizations were simplified by making changes to HTML and CSS, which enabled a rapid iterative design process.

The mPneumonia project added a pulse and a blood oxygen saturation reading from the patient to IMCI. The *Sensors* framework connected the pulse-ox sensor to the mobile device, allowing sensor readings to be written directly to the patient record. ODK 1.x only allows a single value to be recorded to the data model for an individual user action. A pulse-ox sensor reads two values at a time, requiring updates to two database columns. ODK 2.0 allows the specification of custom data types with multiple database columns enabling the pulse-ox data to be recorded in a single action.

To evaluate usability, feasibility and acceptability, PATH conducted studies in Ghana and India. They interviewed 8 health administrators, 30 health care providers, and 30 caregivers. PATH found that mPneumonia was feasible to integrate into rural healthcare workers workflow and had the potential to improve patient care. Interviewees reported mPneumonia was "easy to use" and lent confidence to their diagnosis and treatments [18].

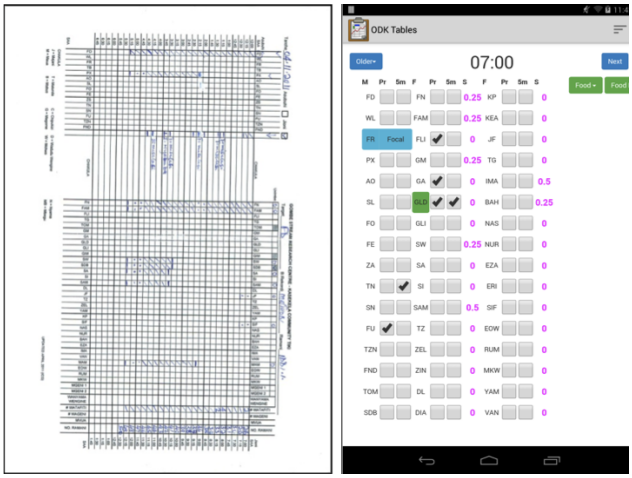

Figure 3: The paper version of JGI's chimpanzee behavior tracking form (left) compared to *Tables* digital version (right)

5.2 Chimpanzee Behavior Tracking

The Jane Goodall Institute (JGI), an early adopter of ODK 1.x, successfully replaced multiple existing paper data forms with ODK 1.x. However, after several attempts, they were unable to digitize one of their most important data collection efforts and needed an alternative to form-based data entry to be able to track chimpanzee behavior in real time. In this scenario, park rangers follow troupes of chimpanzees through the jungle with a clipboard and a paper form. The paper form is organized into a grid format, with names of chimpanzees and boxes indicating presence or absence, proximity to the group leader, and estrus state. A new form is completed every 15 minutes, and at the end of the day the information is transcribed and added to a database. There was no way to enforce internal consistency or data quality when using paper. However, the paper form's grid structure prevented the conversion to an ODK 1.x question based workflow. An example of the paper form is shown in Figure 3. Maintaining the format of the data collected is very significant to the organization, since it has been used for years and the rangers feel comfortable with it.

Conceptualizing the workflow as a series of webpages let the form be converted to an ODK 2.0 *Tables* application. The grid, created using an HTML table, represents a database row that associates the current time and the chimp ID. Edits to the data update the corresponding database row. Additional value is added by prepopulating information from the previous time point, lowering the burden on the ranger by reducing the duplicate effort of re-entering data every 15 minutes.

While more complex than other examples, this application was able to written with a much smaller codebase than would be required using native Android. The database is defined declaratively using CSV files. The application consists of a single page of HTML and roughly 1200 lines of JavaScript. While this approach requires some basic programming knowledge, to do the same task in Android programming constructs would require extensive knowledge of the Android persistence APIs, familiarity with the Android UI framework, an understanding of Android's activity lifecycle. A *Tables*-based approach allowed developers to leverage ODK 2.0 framework for most functionality and focus on solely on the UI to create an encapsulated and manageable mobile application.

5.3 HIV Clinical Trial

Adaptive Strategies for Preventing and Treating Lapses of Retention in Care (AdaPT-R) is a multi-year University of California San Francisco (UCSF) study that tracks HIV patient clinic visits. The AdaPT-R case study demonstrates ODK 2.0's applicability to medical trials that are longitudinal and require patient follow-ups at specific locations. The study requires daily synchronization of a field worker's mobile device so that the medical workflows and study parameters can be updated appropriately.

AdaPT-R is deployed in three clinics in Kisumu County, Kenya and two clinics in Migori County, Kenya that combined serve approximately 65,000 patients. It is a randomized control trial studying 1,745 enrolled patients of the approximately 17,000 HIV patients. The non-linear workflow needed to ensure protocol adherence along with historical data to track lapses in care required the flexibility of ODK 2.0. The project also required an ODK 2.0 application to integrate with their existing medical records systems. The UCSF team uses the ODK 2.0 tool suite to collect and synchronize patient data daily to an ODK Aggregate server. Once the data has been synchronized, it is combined with information from a separate medical records systems to determine patient status. The patient information is then synchronized to the field worker's phone, where it can be used during data collection the next day.

The AdaPT-R study has 18 employees using 17 Android devices to interact with patients. Research assistants were provided 1-2 hours of initial training and receive two ~30 min refresher trainings a year. Additional one-on-one training is provided by AdaPT-R's data manager in Kenya on an as-needed basis. Examples of one-on-one training include: 30-60 min trainings to familiarize site leads with ODK update process, and ~20-30 min to teach zipping files and logs from devices to send to manager for troubleshooting.

Early in the study, field workers experienced issues such as periodic application freezes because of database locking issues and sluggish navigation. The AdaPT-R workflow required users to move back and forth between *Tables* and *Survey*, thereby increasing the probability of database locking due to Android life cycle issues. The revised ODK 2.0's SOA, discussed in section 4.4, was built in response to some of the problems experienced by the AdaPT-R study. The AdaPT-R team deployed ODK's revised framework to one site initially and after experiencing immediate positive results decided to expedited the upgrade of the remaining sites.

Having all study participants available on the mobile device let the AdaPT-R team move devices between clinics without reconfiguring them (or physically moving paper files). This was helpful when devices broke or when the AdaPT-R team needed to reallocate a device to another clinic if one site had a higher volume of data entry compared to another. Additionally, AdaPT-R has research assistants who cover tasks for more than one clinic; having the data for multiple locations on a single device lets these employees use the same device for data entry at all locations.

When asked why they chose to use ODK 2.0, the co-primary investigator of the randomized control trial noted *"We needed a solution for capturing data from multiple forms and that would allow longitudinal follow-up of individual patients. We had experience with earlier versions of ODK, so the new features of 2.0 made it the only option for us if we wanted phone-based longitudinal form completion. Would definitely recommend ODK 2.0!"*

5.4 Disaster Response

The International Federation of Red Cross and Red Crescent (IFRC) is the world's largest humanitarian and development network that has millions of volunteers in 190 member National Societies. The diversity of use cases and business requirements of the

Figure 4: Red Cross enumerator using ODK to distribute goods in Belize after Hurricane Earl

National Societies demonstrate the need for a flexible, customizable system like ODK 2.0. A disaster response platform used by volunteers on the ground needs to be easily adaptable by a deployment architect since time is critical under these circumstances.

The IFRC created prototype disaster response software using ODK 1.x for beneficiary registration and custom software for distributions. The IFRC then contacted us to figure out how ODK could be used to create a single mobile application to handle both registrations and distributions. ODK 1.x was not suited to the task due to missing features (Section 2.1). Bidirectional synchronization is required to maintain a consistent database of beneficiaries and delivered distributions across all devices and distribution centers to prevent fraud, enable auditing, etc. User permissions are required to ensure that field workers see only the appropriate list of beneficiaries relevant to their work and cannot update data fields beyond the scope of their responsibilities. Their complex workflow that reuses certain data fields requires a more robust interface than basic form based navigation provided by ODK 1.x.

We worked with the IFRC to prototype a solution using a combination of *Survey* forms a *Tables* data navigation. The IFRC's goal to build a reference disaster response/recovery and crisis management application that leverages ODK 2.0 to handle common data collection and field management tasks. This reference application could then be adapted to the diverse use cases of its 190 member societies. The IFRC conducted an initial field feasibility study using ODK 2.0 to distribute disaster-aid cash at two locations in Jamaica involving 93 participants. The initial study had positive results, and the IFRC plans to conduct deployment pilots during actual disaster responses in the next few months to further evaluate ODK 2.0 usage for cash distributions during emergencies. ODK 2.0's customizable workflow, user permissions, rule and adherence enforcement, disconnected operations, and eventual data synchronization were critical features to successfully coordinate distribution of cash and supplies.

5.5 Mosquito Infection Tracking

Led by Monash University, 'Eliminate Dengue' is an international research collaboration focused on reducing the spread of Dengue, Chikungunya, and Zika. The program operates across a variety of countries, languages, terrains, and worker skill levels. Currently, Eliminate Dengue ODK 2.0 application is deployed in Brazil, Columbia, Indonesia, Australia, and Vietnam. Their operational workflow requires ODK 2.0's bidirectional sync protocol

to provide historical as well as timely data to their field workers during site visits to remote environments.

The Eliminate Dengue program is developing a method to reduce the spread of dengue by introducing the naturally occurring Wolbachia bacteria into local mosquito populations. This bacteria reduces the ability of mosquitoes to transmit dengue between people. Workers monitor mosquito reproduction at field sites until the bacteria is established in a critical mass of the local population and propagation is self sustaining. This requires repeated weekly visits to field collection sites. The ODK 2.0 framework is used to provide workers with an interactive map of site locations, historical data about each site, individualized instructions on what tasks to perform at which sites, and a data capture application.

The Eliminate Dengue team designed and developed their own custom ODK 2.0 *Tables* application using consultants specializing in web development. They decided to use *Tables* because *"developers don't like to be restricted,"* and *Tables* gave them the flexibility to customize their solution to their needs without being in *Survey's* question rendering framework. Furthermore, *Tables* let them *"focus on web dev"* rather than worry about editing Android code, as would have been necessary with an ODK 1.x solution. An example of a customization is their eschewing the provided Google Maps library for a runtime library called Leaflet.js[2] to serve their own map files from ODK 2.0's framework. This let them provide more accurate and customized maps to their workers in disconnected settings.

A key component of the program is regular visits to multiple sites over a period of weeks and months. These visits require historical data to be available to field workers, who may not be the only workers visiting the same site. This requirement was the driving factor in the Eliminate Dengue team's decision to use ODK 2.0 over ODK 1.x: when describing ODK 2.0's benefits, they said, *"we needed two-way synchronization"* and that all other benefits were discovered and leveraged later: *"we had a requirement to send data to the devices as well as receive from them."*

The greatest challenge to the team has been logistical: the velocity with which they are developing and deploying requires frequent changes to the data model. Neither ODK 1.x nor ODK 2.0 is well suited to this type of change. The Eliminate Dengue team is using JSON objects in the database to work around the issue, but in general changing data models remain an open problem in the ODK 2.0 framework. The trade-offs between a document based model storage vs row based storage is further discussed in section 6. However, the program manager reports *"the app is scaling quite well"* and that ODK 2.0 apps are *"quite easy to use and we haven't had any acceptance issues."*

5.6 Tuberculosis Patient Records

Mercy Corps' effort to combat tuberculosis in Pakistan provides a case study for the ODK 2.0 framework's ability to serve rural medical facilities that maintain paper workflows. This project explored how to incorporate less expensive paper collection methods into a hybrid workflow to reduce data collection and aggregation times in disconnected environments. The large data sets generated by these facilities, coupled with their unreliable Internet connectivity, provided scalability tests for the ODK 2.0 framework's service architecture and synchronization capabilities.

Mercy Corps maintains paper based patient records in local clinics, which are subsequently aggregated and used to generate reports for administrators. Paper records are valuable in this environment since they are cheap, trusted, and easily understood. However, shipping paper records to central facilities and transcribing data from paper to digital systems hinders reporting efforts with significant time delays and financial burdens [5]. We partnered with Mercy Corps to pilot ODK 2.0 effectiveness at streamlining the digitization and validation of their patient records and providing basic reporting to clinicians in the field. The pilot ran for three months with four field workers in two districts in Pakistan working with 122 patients. Mercy Corps' health register was converted to a *Scan*-compatible format. Their register had 39 data input fields, which expanded into 121 user data columns in the ODK database when factoring in metadata and image snippets for each field. Field workers used *Scan* to digitize patient records as they were filled in at the clinics, then validated the digitization with *Survey* at the point of capture. Additionally, a data reporting mechanism was developed on *Tables* for field workers to track overdue patient visits.

When Internet connectivity became available, usually at an Internet cafe, new data would be synchronized and used to generate administrative reports. The wide rows and extensive metadata for each patient caused the database size to grow, and with the spotty connectivity offered by Internet cafes, health workers struggled to upload the thousands of files generated each day. Our responses to these challenges are discussed in Sections 4.3 and 4.4.

The custom data management application built on *Tables* provided health workers with a summary of patient data and reminders of upcoming appointments. This required disconnected historical queries of patients under the care of the health care worker using the device. Additionally, workers would often delay their validation of transcribed records until after a visit was complete, sometimes validating an entire day's worth of visits in a single batch. In this case it was essential that they had disconnected access only to their own patients to prevent them from mistakenly altering records of other patients. To address this issue of filtering the database based on the context of the user, user and group permissions were developed, as discussed in Section 4.2.

At the conclusion of the pilot, Mercy Corps interviewed its field workers about their experience and performed timing comparisons between workflows. The interviews discussed by Ali [5] revealed that workers were particularly enthused about the reporting on the device, with one stating "*follow up visit report has revolutionized our work. Now, we will never miss out any patient*." Data collection was slowed by the need to validate transcribed data, but the transfer process was reduced from days via courier to minutes via connected synchronization [5]. The pilot showed that ODK 2.0 is robust enough to handle large datasets in disconnected environments and that organizations unable to fully adopt direct-to-digital workflows can benefit from field digitization.

6. DISCUSSION

For computing technology to solve global problems, researchers need to push not only the boundaries of technology, but also the boundaries of how the technology can be applied in various environments with differing constraints. Two-thirds of the world's population have access to mobile phones; however, technical limitations prevent billions of users from adopting some of the revolutionary capabilities of these mobile devices [6, 23]. The concept of "applied computer science research" is needed to extend the reach of technology to include the billions of people in the world that currently have limited access to it. To address this, research is needed to answer: "what are the proper abstractions and frameworks for mobile devices that are appropriate for a broader range of technical skills and familiarity?" In resource-constrained environments common assumptions used by designers and developers (e.g., connectivity and power availability) can vary significantly between contexts. For example, the availability of power can vary from an

[2]JavaScript mapping library: http://leafletjs.com/

always-on grid connection, to sporadic grid connections, to having to pay someone to use a generator or car battery to charge a mobile device. Depending on your location in the world, mobile devices move from disconnected environments to connected environments with dramatically different bandwidths and latencies.

The ODK research project seeks to identify appropriate technologies for resource-constrained environments and identify barriers that prevent existing solutions in high-resource contexts from being similarly adopted in these settings. ODK 2.0 was created to address the limitations uncovered through the widespread adoption of ODK 1.x. Field experience with real organizations is needed to understand the problems that prevent adoption of mobile data management solutions. Conceptually akin to testbeds (e.g., PlanetLab [11]), by working with organizations to build real systems that are deployed in regions with challenging constraints, we were able to test ODK 2.0 in a way that far extends beyond the lab environment. Deploying at scale reveals different classes of problems. ODK 2.0 has been piloted or is deployed in over 18 countries by more than 15 organizations. By listening to our partners and understanding their needs, we addressed limitations of ODK 1.x. To understand the contextual limitations, we developed and maintained a suite of tools and gave them to non-programmers to use in their mobile applications. While ODK is designed to operate in extremely diverse environments, the lessons learned can be applied generally to system design to make mobile system platforms more resilient to adverse and varying conditions.

A problem that has not been fully solved by the ODK 2.0 is how to handle frequent changes to the data model, as experienced by Eliminate Dengue (Section 5.5). Initial design discussions for ODK 2.0 pitted a document-based data model against a row-based model. The document-based model allows for easier updates to the data model, but extracting results and generating reports after multiple data models have been used becomes difficult and often requires hiring a programmer to write code to merge the data and perform data cleaning. The row-based model employed by ODK requires organizations to establish a schema before deploying, front-loading the costs, and is difficult to update if the schema must be altered, but it makes reporting simple. Ultimately, the row-based model was chosen because organizations have more existing expertise in understanding their business workflow and making a flexible schema requires less programming knowledge than dealing with data merging and cleaning issues. However, a scenario like Eliminate Dengue's, in which rapidly changing needs on the ground require frequent changes confirmed that if you have programming resources to handle the data merging and cleaning then a document-based model can make for easier iterative design.

ODK 2.0 focuses on scale and adaptability instead of a novel single purpose application useful only for limited cases. Through the generalizable application framework, ODK 2.0 allows users of varying technical skill levels to create custom solutions that support a wide range of scenarios. ODK 2.0 aims to create base APIs, standards, and free reusable libraries to minimize the start-up costs for organizations, researchers, and companies seeking to participate in the space. To create an ecosystem, ODK uses the permissive Apache 2 open-source to enable free use and encapsulation of the technology. This empowers companies to have business models based around both creating products with additional features or providing consulting assistance to global aid workers.

7. FUTURE WORK

Future expansion of the ODK 2.0 framework will focus on security and usability improvements. Currently, the local database contains a full copy of the data since the authenticated user may change and have different access privileges. However, for sensitive information it would be preferable to have a data wipe feature, where on a user authentication the previously synchronized data on the device would be deleted to increase security. This poses a trade off between bandwidth to continually re-download the subset of data versus data security. As ODK's 2.0 framework's abstractions mature from more successful deployments and the user base grows, we plan to start creating additional graphical tools to further reduce the skills needed to build and customize mobile data management applications. For example, we are working on a PC-based tool called "Application Designer" for creating, customizing and previewing forms. It will include a theme generator to help novice users graphically select basic styles and colors which will then automatically generated into a basic CSS style file. Similarly, the "Application Designer" will include a help wizard to generate basic *Table* views that will auto generate the corresponding HTML and JavaScript files.

8. CONCLUSION

ODK 2.0 provides development organizations with a new mobile framework to build data collection and management services for resource-constrained environments. This paper presents lessons learned from creating an application framework that makes mobile devices more useful in contexts where high-income countries' engineering assumptions do not necessarily match the infrastructure realities in resource-constrained communities.

Building a real system used by development organizations in differing domains shows the general applicability of the mobile framework. By piloting with these organizations and incorporating their needs and feedback into the development process, the ODK 2.0 framework was made robust to a variety of challenges met in the field including extreme mobile networking conditions such as low bandwidth, high latencies, and long periods of disconnected activity. ODK 2.0 provides a generalizable application framework for users with varying technical skill levels to create customizable mobile data management solutions that address global problems in resource-constrained environments. Implementing ODK 2.0's framework as an SOA yielded benefits in the areas of upgradability, security enforcement, management of global state, and future portability, without incurring a significant performance cost. ODK 2.0 provides a second, complementary, open-source toolkit to the ODK 1.x framework, that is targeted at *deployment architects* and is easy to use and easy to scale. By expanding the variety of use cases that ODK technology can be adapted to, the ODK 2.0 tool suite helps to achieve ODK's goal of "magnifying human resources through technology."

9. ACKNOWLEDGEMENTS

Both ODK 1.x and 2.0 were ideas of ODK visionary Gaetano Borriello. ODK has been a team effort that involved many people at the University of Washington. We would like to acknowledge Rohit Chaudhri, Nicola Dell, Joshua Fan, Ori Levari, Shahar Levari, Li Lin, Luyi Lu, Madhav Murthy, and Sang-Wha Sien for their work on ODK 2.0. We also thank Adam Rea, Sandy Kaplan, and Kurtis Heirmerl for their insightful feedback on this paper. Additionally, we would like to thank all the organizations (e.g., PATH, JGI, ADAPT, IFRC, Eliminate Dengue, Village Reach, Mercy Corps) who used ODK and provided feedback. Finally, we thank our anonymous reviewers and our shepherd Rajesh Balan, for their guidance and suggestions. The material in this paper is based upon work supported by Bill and Melinda Gates Foundation Grant No. OPP-1132099 and USAID Agreement AID-OAA-A-13-00002.

10. REFERENCES

[1] Apache Cordova. https://cordova.apache.org/, Dec 2016.

[2] Intel XDK. https://software.intel.com/intel-xdk, Dec 2016.

[3] Ionic Framework. http://ionicframework.com/, Dec 2016.

[4] Sartphone OS Market Share, 2016 Q2. https://www.idc.com/prodserv/smartphone-os-market-share.jsp, Dec 2016.

[5] S. Ali, R. Powers, J. Beorse, A. Noor, F. Naureen, N. Anjum, M. Ishaq, J. Aamir, and R. Anderson. Odk scan: Digitizing data collection and impacting data management processes in pakistan's tuberculosis control program. *Future Internet*, 8(4):51, 2016.

[6] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana, and K. Fall. The case for technology in developing regions. *Computer*, 38(6):25–38, 2005.

[7] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, and G. Borriello. Open data kit sensors: A sensor integration framework for android at the application-level. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 351–364, New York, NY, USA, 2012. ACM.

[8] W. Brunette, S. Sudar, N. Worden, D. Price, R. Anderson, and G. Borriello. Odk tables: Building easily customizable information applications on android devices. In *Proceedings of the 3rd ACM Symposium on Computing for Development*, ACM DEV '13, pages 12:1–12:10, New York, NY, USA, 2013. ACM.

[9] W. Brunette, M. Sundt, N. Dell, R. Chaudhri, N. Breit, and G. Borriello. Open Data Kit 2.0: Expanding and Refining Information Services for Developing Regions. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 1–6, 2013.

[10] W. Brunette, M. Vigil, F. Pervaiz, S. Levari, G. Borriello, and R. Anderson. Optimizing mobile application communication for challenged network environments. In *Proceedings of the 2015 Annual Symposium on Computing for Development*, 2830644, 2015. ACM.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[12] B.-G. Chun, C. Curino, R. Sears, A. Shraer, S. Madden, and R. Ramakrishnan. Mobius: unified messaging and data serving for mobile apps. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 141–154, 2307650, 2012. ACM.

[13] N. Dell, N. Breit, T. Chaluco, J. Crawford, and G. Borriello. Digitizing paper forms with mobile imaging technologies. In *Proceedings of the 2nd ACM Symposium on Computing for Development*, 2160604, 2012. ACM.

[14] N. Dell, T. Perrier, N. Kumar, M. Lee, R. Powers, and G. Borriello. Paper-digital workflows in global development organizations. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 1659–1669, 2675145, 2015. ACM.

[15] B. DeRenzi, N. Lesh, T. Parikh, C. Sims, W. Maokla, M. Chemba, Y. Hamisi, D. S. hellenberg, M. Mitchell, and G. Borriello. e-imci: improving pediatric health care in low-income countries. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 753–762, 1357174, 2008. ACM.

[16] B. DeRenzi, C. Sims, J. Jackson, G. Borriello, and N. Lesh. A framework for case-based community health information systems. In *2011 IEEE Global Humanitarian Technology Conference*, pages 377–382, Oct 2011.

[17] A. S. Ginsburg, J. Delarosa, W. Brunette, S. Levari, M. Sundt, C. Larson, C. Tawiah Agyemang, S. Newton, G. Borriello, and R. Anderson. mpneumonia: Development of an innovative mhealth application for diagnosing and treating childhood pneumonia and other childhood illnesses in low-resource settings. *PloS one*, 10(10), 2015.

[18] A. S. Ginsburg, C. Tawiah Agyemang, G. Ambler, J. Delarosa, W. Brunette, S. Levari, C. Larson, M. Sundt, S. Newton, G. Borriello, and R. Anderson. mpneumonia, an innovation for diagnosing and treating childhood pneumonia in low-resource settings: A feasibility, usability and acceptability study in ghana. *PLOS ONE*, 11(10), 2016.

[19] S. Hao, N. Agrawal, A. Aranya, and C. Ungureanu. Building a delay-tolerant cloud for mobile data. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pages 293–300, 2013.

[20] C. Hartung, Y. Anokwa, W. Brunette, A. Lerer, C. Tseng, and G. Borriello. Open Data Kit: Tools to Build Information Services for Developing Regions. In *Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development*, ICTD '10, pages 1–12, 2010.

[21] T. S. Parikh, P. Javid, S. K., K. Ghosh, and K. Toyama. Mobile phones and paper documents: evaluating a new approach for capturing microfinance data in rural india. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 551–560, 1124857, 2006. ACM.

[22] 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. In *Proceedings of the First ACM Symposium on Computing for Development*, pages 1–11, 1926200, 2010. ACM.

[23] World Bank Group. *World Development Report 2016: Digital Dividends*. International Bank for Reconstruction and Development (World Bank), 1818 H Street NW, Washington DC, 20433, 2016.