Integrating Mobile Collection Software with Health Applications

David Roberge, Bruce MacLeod, Brian Hartsock, and Ime Asangansi
Department of Computer Science
University of Southern Maine
Portland, ME United States
david.roberge@maine.edu

***Abstract*—This paper describes and analyzes three approaches for integrating mobile data collection software and health applications. The first approach uses a domain-specific language module that is installed in a health application that communicates directly with the mobile data collection software. The second approach uses Mirth Connect to integrate the mobile collection software with a health application. The third approach is writing custom code required to integrate mobile data collection software with a health application. In the final section of this paper, we provide an analysis of the three approaches, and make recommendations on which approach should be used based on the resources available to a health facility.**

OpenHDS; Mobile Devices; Health Software; System Integration;

I. INTRODUCTION

Mobile devices have provided an innovative way to collect information in low-income countries. Several open source projects such as the Open Data Kit[1] (ODK), and OpenXData[2] (OXD) have developed software applications that run on low and high-end mobile devices. The two major components in this type of software include a mobile client application and a server side application. Most mobile data collection software, including ODK and OXD, use Xforms[3] to represent form definitions that are rendered to the user, as well as for transferring data between the mobile phone and server side application. Typically, the form definitions are designed and uploaded to the mobile device before data collection has begun. After a round of data collection, the software has capabilities to upload the collected data to the server side application where it can be saved and reviewed by application users.

The health community has increasingly taken notice to the mobile data collection software. It provides a powerful platform to collect health related information in low resource settings. This information has historically been collected using paper-based solutions [1]. The generic nature and wide applicability of the mobile data collection software make it challenging to integrate it with health applications.

There have been initial attempts to integrate these two applications, but it has not been available to the general user base because they are implemented for specific use cases. We observe that providing a general solution to this problem would be advantageous to low resource facilities. These facilities would be able to collect health information using the mobile data collection software, and then transfer the information between one or several health applications. This has the potential of improving the health systems in these regions, although research in this area is still ongoing[4].

A. Integration Approaches

System integration involves bridging isolated applications so that they can communicate and share information with one another. Software is usually developed to solve a specific problem. For example, mobile data collection software solves the problem of using mobile devices to collect data. But many times, it is desirable to combine different systems with different capabilities. This can enhance or extend the capabilities of the integrated applications. Over the years, developers have tried to solve integration problems in four general ways [2]:

- File Transfer
- Shared Database
- Remote Procedure Invocation
- Messaging

In file transfer, one application writes a file and another application reads the file. With shared database, multiple applications share the same database schema. Remote procedure invocation can be used when an application exposes some of its functionality, and another application invokes that functionality externally. The invocation occurs in real time but synchronously. In messaging, an application publishes a message to a channel, and then other applications can read the

Portions of this project are supported by the generous contributions of the IDRC.

[1] Information on Open Data Kit can be found at http://www.opendatakit.org

[2] Information on OpenXData can be found at http://www.openxdata.org

[3] Xforms is a W3C Standard. More information about the standard can be found at http://www.w3.org/MarkUp/Forms/

[4] One example of this type of project is MoTech, an initiative launched by the Grameen Foundation to determine whether mobile phones can increase the quantity and quality of antenatal and neonatal care in rural Ghana.

978-0-7695-4595-0/11 $26.00 © 2011 IEEE
DOI 10.1109/GHTC.2011.91

message at a later time. A distinct difference between remote procedure invocation and messaging is that messaging is asynchronous, where as remote procedure invocation is synchronous.

The correct approach depends on aspects of the systems that will be integrated. File transfer and shared database require tight coupling between applications. This is something we were looking to avoid as we are trying to provide a more general solution. Remote procedure invocation was a strong consideration for our project. We were concerned that the synchronous nature of remote procedure invocation might be a problem; in addition to network reliability issues that may arise. Messaging is asynchronous and circumvents the problems with remote procedure invocation. Unfortunately messaging also introduces a few interesting problems. Since messaging is asynchronous, there needs to be a permanent communication channel between systems. For example, if a health application were to process the data collected by the mobile device and discovered that it had validation errors, it requires communicating those errors back to the mobile collection software.

B. Mapping the Xforms Data

After reviewing several working integration implementations, we observed many similarities between them. Most of the functionality was performing a manual mapping between the mobile data software Xform data representation and the internal data model of the health application. We focus on the problem of mapping the data between the mobile collection software and health applications. Although the mapping of data is at the core of the integration implementations, there are also cases of adding extended functionality. We recognize this is an important aspect to the integration problem, but we defer any consideration in this paper.

II. DOMAIN-SPECIFIC LANGUAGE

The first approach we propose to solve the integration problem is a custom implementation of a domain-specific language (DSL). A DSL is a programming language of limited capabilities focused on a particular problem domain [3]. The DSL is tailored to contain keywords and ideas that are applicable to the domain, and if done correctly, will be easy to read and understand for someone familiar with the domain. The focus of our DSL is mapping Xforms data representations to health applications internal data model. There are 2 components the DSL needs to accomplish: parsing the Xforms XML, mapping the XML elements to fields in the internal data model of health application, and invoking health software functionality. In both cases, we want to insulate the users from the low level constructs of performing these operations.

Fig. 1 depicts the architecture of our setup. We chose Open Data Kit (ODK) for the mobile collection software, and OpenHDS as the target health application. ODK provides a feature that allows users to invoke an external service upon a form submission, similar to a web service call. The invocation happens asynchronously. For our project, we've set up an ODK external service to point to a URL that is managed by the DSL module in OpenHDS. The external service invocation is like a RESTful-style request.

Fig. 2 illustrates a sample Xform instance. An Xform instance contains only the data for a given Xform definition. Fig. 2 represents a Visit event in OpenHDS. The DSL module uses a binding definition (explained next) to operate on an Xform instance. The Xform instance acts as input for the binding definition. The DSL module expects an Xform instance to be included upon invocation of a binding definition.

In Fig. 3, we demonstrate a binding definition (the numbers on the left are for explanation purposes only.) A binding definition is analogous to a procedure in a programming language. It defines a sequence of statements that are executed on the user's behalf. For our project, binding definitions are associated with a URL. The URL is a way for users to identify a binding definition, and also how they can invoke it. The DSL provides a small number of operations, including:

- *create* – Creating an entity within OpenHDS. This is restricted to only core types found in OpenHDS
- *lookup* – Retrieving a previously saved entity in OpenHDS. Again, the types that can be looked up are restricted.
- *set* – Setting a value on an entity in the context of the DSL. The set statement should be either a value or a reference to a variable in the DSL.
- *save* – Persists the entity to OpenHDS, provided there are not validation errors during the save operation.

As pictured in Fig. 3, the DSL provides a way to parse an XML document, and invoke application functionality. When navigating the Xform instance, the DSL expects users will supply valid XPath expressions (lines 1, 3, 4, 5, 6, 8, and 9.) Line 1 describes the context for the binding definition, and internally indicates the root element of the Xforms instance. Referring back to Fig. 2, this states that the visit element will be the root element in the Xform instance. In lines 3 and 4 of Fig. 3, lookup statements, the context value is combined with the value after the "with" keyword to create a full XPath expression. For example, the XPath expression in line 3 would evaluate to /visit/location, which has the value LOCMBI11.

The DSL also allows users to create temporary variables to store values. Variables are identified by a leading "$". Currently, only OpenHDS types can be assigned to a variable. Variables can be declared with either create or lookup statements. Users can reference properties of the OpenHDS types using dot notation. For example, in Fig. 3 on line 5, the *visitDate* property is being set on the Visit OpenHDS type. These properties correspond to fields on the Visit class in OpenHDS.

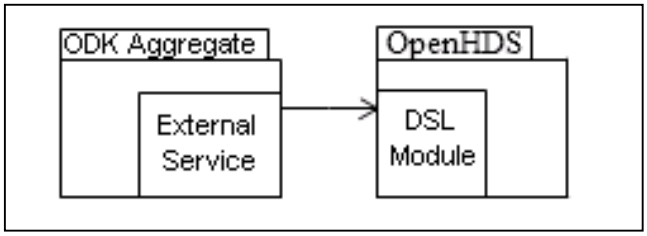

Figure 1. Architecture Overview for DSL

```
<?xml version="1.0" ?>
<visit>
 <theDate>12-12-10</theDate>
 <theRound>2</theRound>
 <extId>VISIT1</extId>
 <collectedBy>FWEK1D</collectedBy>
 <location>LOCMBI11</location>
</visit>
```

Figure 2. Example Xform Instance

```
1   context is '/visit'

2   create visit as $visit

3   lookup location as $loc with 'location'
4   lookup fieldworker as $fw with 'collectedBy'

5   set $visit.visitDate with 'theDate'
6   set $visit.roundNumber with 'theRound'
7   set $visit.collectedBy with $fw
8   set $visit.extId with 'extId'
9   set $visit.visitLocation with $loc

10  save $visit
```

Figure 3. Example Binding Definition

III. MIRTH CONNECT

A second approach to the integration problem combined Mirth Connect, ODK and the OpenHDS. ODK is configured to store data in a MySQL database. Using Mirth Connect, the ODK database is queried to look for any new data inserted by ODK. If new data is found, Mirth Connect will read the data and send it to the OpenHDS. Further details on this configuration follow.

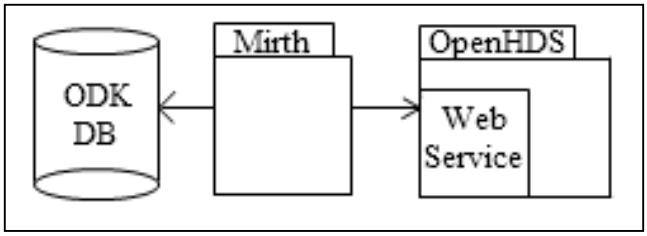

Figure 4. Architecture Overview for Mirth

Mirth Connect is a health care integration engine[5]. Mirth provides several components to read and write data from external applications. Mirth's intent is to act as a middleware component, helping to ease the burden of integrating health applications. In our case, we used Mirth to integrate mobile data collection software with a health application.

Mirth is distributed with an administrator interface where the entire configuration occurs. Users are required to perform the configuration for communicating with external applications. Most of this can be done using the GUI interface. Mirth also contains a scripting module which uses javascript as the scripting language. The scripting module allows users to perform more complicated computations.

Fig. 4 outlines the architecture for our Mirth deployment. In this setup, Mirth acts as a mediator, coordinating the interactions between the 2 systems, similar to the mediator object-oriented design pattern [3]. It is loosely coupled from both ODK and OpenHDS, and neither system is aware it is coordinating their actions.

Mirth is configured to read data from ODK using a database reader connector[6]. The database reader queries ODK to look for any new form submissions it has not already seen. If the database reader finds any new submissions, they are transformed into a message, and persisted in the Mirth data store. We keep track of which submissions have already been read using the primary key for forms in the ODK database.

After new submissions have been saved in the Mirth data store, they are relayed to OpenHDS using a web service sender. The web service sender is able to discover operations through a WSDL definition. The values from the form submission are inserted as parameters to the web service call. The web service sender invokes the web service, but delegates handling the response to another channel reader. If the response was successful, the message is marked as completed. If an error occurs, the message is flagged and should be reviewed at a later time by a data manager.

IV. MANUAL CODING

The third alternative we consider is manual or custom coding. In this case, we assume a team of developers have implemented the functionality based on a specific use case. There are dozens of projects that are currently doing this. Two examples are ODK Clinic, an extension to ODK mobile client

[5] More information on Mirth can be found at http://www.mirthcorp.com

[6] A connector is similar to a channel in a messaging system.

which allows users to insert data into OpenMRS, and MoTech, an ongoing project that uses OXD and OpenMRS for collecting data and sending messages for pregnant women in Ghana. We will focus on the MoTech implementation for this paper.

The architecture for the MoTech project includes a custom mobile client, based on the OXD mobile client, a server-side application, and an OpenMRS module. Users are able to collect the data with the OXD mobile client, and then upload the information to the server-side application. The server application communicates directly with the OpenMRS module, which performs validation and other computations for their use case.

The bulk of the data stored by the MoTech project is medical record information. Most of the information can be captured using the data model in OpenMRS, but there were also additional classes that needed to be created for things not covered by OpenMRS. We expect that most facilities will also want to collect additional information similar to MoTech.

V. ANALYSIS

In the last section of this paper, we describe our analysis of each approach and make recommendations on which approaches may work best. First, we judge each approach based on three requirements: general applicability, ease of configuration, and workflow. After we have analyzed the three approaches, we will make a recommendation for sites that may want to integrate mobile collection software with health applications.

A. General Applicability

For this requirement, we judge the approaches based on whether or not we believe the general user base could reuse the solution. We believe this is an important requirement because it would allow low resource facilities to use this software with relatively low cost.

In the first approach, the DSL has the potential to be a reusable component. For our project, we implemented the DSL in OpenHDS. But as you can imagine, there are dozens of applications that could be potentially used by health centers. This would require creating a DSL module for each health application in order to make them reusable. This is a considerable amount of work, and would probably need a lot of support from community developers. On the other hand, if it were to be implemented, we believe the DSL is an elegant solution that is easy to modify and change. The DSL has a clean interface for mapping the Xform instance data to the underlying health application data model.

Mirth Connect is also a candidate for a reusable solution. The software provides common general connections to external systems that makes it nearly trivial to setup. Compared to the DSL approach, there is likely not as much work that needs to be done to integrate with other health applications. Most contain web services, or other ways of inserting data into them. This only requires the implementers to discover how data can be inserted in the health software, and configure Mirth to use that mechanism.

Finally, the manual or custom coding is probably the hardest solution to reuse. In some cases, this is possible. For example, with ODK Clinic, it is not an issue for a site to reuse the ODK application with OpenMRS. But, if they wanted to use ODK with another health application, they would need to create the module to do that. In the MoTech project, it would be nearly impossible to reuse their solution, unless the use case was exactly, or nearly similar to theirs.

B. Ease of Configuration

For the second requirement, we consider ease of configuration. We argue that for a solution to be successful in a low resource health facility, it needs to cater to individuals with a limited or nontechnical background. We also assume that health facilities may not have the resources to hire developers or administrators to manage their applications. This means the solution has to be robust and be relatively easy to deploy and maintain.

The DSL module is bundled with the health application, and requires no explicit installation configuration. Out of the box, the DSL module contains binding definitions that correspond to only the core entities of the health application. This means that if the health facility does not need to add additional data, there is almost no configuration required for the health center. In most cases, health centers will want to add additional information, and would be forced to modify or add binding definitions. Since the DSL is focused on this problem domain, we believe it would not be difficult for the health facility to learn the language of the DSL. The only other configuration requirement for the DSL module is directing the mobile collection software to invoke the DSL module upon form submissions.

Mirth connect requires the most amount of configuration. Health centers would be required to maintain another software application, in addition to the data collection software and health applications. The Mirth administrator interface gives users a plethora of configuration options, and assumes previous knowledge of system integration. We feel this is a major weakness in the Mirth solution because not only would the implementers be forced to understand the foundation of system integration, but they also would be required to maintain and configure Mirth. An alternative that may work is to write a custom plugin for Mirth. Though this may solve some of the configuration complexities, we feel that maintaining Mirth would still be a difficult task.

The third approach will be the easiest to configure in many cases. Since these types of solutions are custom tailored to particular uses cases, most of the configuration is handled by the developers. This is an asset because the health centers can assume they will not be responsible for any configuration or need to maintain additional software.

C. Workflow

For the third and final requirement we consider workflow. We define workflow as the processes that take place during a normal data collection round, and the subsequent actions that occur to upload and validate the collected data and fixing any errors that may occur during data insertion in the health

software. We believe that an ideal solution communicates validation errors to the field workers that collected the data, and it is their responsibility to fix and resubmit the collected data.

In the current DSL solution, all binding definition invocations, and the corresponding input to the binding definition and its outcome are stored in the health application. It is up to the data managers to fix any bad data insertions. We would prefer that this information is propagated back to the mobile data collection software. This would require minor changes to the mobile client, and the changes would need to be introduced into the main source code repository. In the future, we hope to be implementing this functionality.

The workflow for Mirth will depend on the implementers. For our project, we decided to record and store any data errors in a separate table in the Mirth application. We were able to record whether the collected data was inserted properly based on the return value of the web service call. Though loose coupling is often preferred in software construction, we believe there needs to be a tighter coupling between the health software and mobile collection software to support a viable solution.

Finally, the manual code approach workflow is also dependent on the implementers. For the MoTech project, the developers decided that phone submissions would be synchronously sent to the server. Any forms that failed validation were flagged and left on the phone. All valid forms were deleted. The invalid forms also contained a message indicating why the form failed submission. This allowed the field workers to make the necessary corrections and then resubmit the form.

D. Recommendations

We feel that each approach has certain characteristics that make them valid solutions depending on the resources a health facility has available. Unfortunately, at this time, we do not feel the solutions are mature enough to work for the general use case. For example, the DSL module is not widely implemented, so support for this approach is limited. However, we believe that under certain circumstances, one approach may work better than the others.

A health center that has developer resources available to them would most likely succeed by implementing a custom module. The developers would be able customize the system until it met their specific needs. They would be able to avoid any pitfalls that they may incur if they were to try to configure a Mirth installation. Additionally, they would not need to be responsible for administrating the system. As mentioned previously, the current DSL implementation is not widely available, and cannot be considered a viable option until further adoption and implementation is reached.

In contrast, if a health facility has access to technical resources, but not developer resources, they would most likely succeed using the Mirth approach. There would be an additional upfront cost to learning Mirth, in particular what configuration needs to be done and how it will communicate with the external systems, but after this information is learned, Mirth provides the necessary components to integrate the mobile collection software with health applications. We found that there may also be added complexity because in order to do more computation, users need to use the scripting language Mirth provides. Again, this is not ideal for technical people, but will probably be the easiest to implement.

On the other hand, if a health facility lacks developer and technical resources, we feel that the DSL solution will work best. Our hope is to implement similar modules in other popular health applications, like OpenMRS and DHIS. Also, we hope to introduce changes to the most widely used mobile data collection software so they effectively communicate with the DSL modules.

VI. CONCLUSION

In this paper, we present 3 approaches to integrating mobile data collection software with health applications. We believe each approach is a solution for interested health facilities, depending on the conditions of the implementation.

We strongly support the first approach of using a DSL. The DSL has potential to be most generally applicable based on our experiences. The work towards completing a full implementation is ongoing, as well as introducing the required changes to the mobile data collection software.

We found Mirth is a viable option, but has many obstacles relating to the initial configuration, and ongoing maintenance. Some of the concepts and configuration options may be difficult for individuals new to system integration. Under the right conditions, and technical staff, Mirth is a candidate solution.

Finally, the manual or custom coded module is preferred if developer resources are available. We feel that it is unfortunate that in many cases health facilities cannot reuse these solutions; the lessons learned from these deployments will give good insight to completing a general solution that is available to a wider audience.

REFERENCES

[1] J. Phillips, B. MacLeod, and B. Pence, "The Household Registration System: Computer Software for the Rapid Dissemination of Demographic Surveillance Systems," Demographic Research, vol. 2, article 6, June 2000.

[2] G. Hohpe and B. Woolf, Enterprise Integration Patterns. Upper Saddle River, NJ: Addison-Wesley, 2003.

[3] M. Fowler, Domain-Specific Languages. Boston, MA: Pearson Education, Inc, 2011.

[4] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Upper Saddle River, NJ: Addison-Wesley, 1995.