MDB: A Metadata Tracking Microcontroller Micro-Database

Marsalis T. Gibson
EECS
UC Berkeley
Berkeley, CA
mtgibson@berkeley.edu

Javier Rosa
EECS
UC Berkeley
Berkeley, CA
javirosa@eecs.berkeley.edu

Eric A. Brewer
EECS
UC Berkeley
Berkeley, CA
brewer@berkeley.edu

ABSTRACT

This work in progress explores a database designed to enable data sharing on custom hardware data collection devices and prototypes. Projects and systems are frequently based on the Arduino framework, examples include ODK's FoneAstra [3], the Open Energy Monitor [7], and the Grove system of sensors [5]. The Arduino platform is targeted because of its ease of use, community support, and low cost as a data collecting device compared to other off-the-shelf sensors. However, there is a need for a framework suitable for microcontrollers that enable ease of integration into other data collection systems. This includes the ability to synchronize data with collection and aggregation devices designed to work offline as well as the ability to track sensors and describe data sources for other machines and users. To address the issue, we propose a solution based on an existing small database usable on the Arduino platform that would integrate into the Mezuri [6] data collection system. The database is designed to fit within the running memory constraints on a microcontroller to store sensor data with relatively few fields per reading on flash media. This framework, with explicit support for metadata, enables users in emerging regions to directly measure physical quantities as well as indirectly measure human behavior in future development projects involving direct sensing. The database can be used by a non-expert. In particular, we investigate the qualities that a technically inclined social scientist would look for when storing such data on microcontrollers. To enable Mezuri integration we will support metadata as a first class object accessible with additional utility functions and native synchronization support.

Keywords

Arduino, Microcontroller, Embedded Databases, Sensors, Data Collection, Emerging Regions, Metadata.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

ACM DEV '16, November 17-22, 2016, Nairobi, Kenya

ACM 978-1-4503-4649-8/16/11.

<http://dx.doi.org/10.1145/3001913.3006645>

1. INTRODUCTION

To improve the accuracy of behavioral data collection in emerging regions direct measurement of physical quantities is

increasingly being used. Additionally, the data may already be available as a side effect of a digital technology based intervention. If sensor configuration, deployment, maintenance, and synchronization workflows are not integrated into existing survey workflows, the amount of work is doubled. Surveyors may be familiar with electronic survey technology, however sensor configuration and data collection is a much rarer skill. Synchronization, metadata tracking, and standard storage interfaces are needed to simplify the task of retrieving data from sensors and configuring sensors alongside with survey processes.

Having an efficient and reliable DBMS for very simple and accessible microcontrollers will not only help engineers worldwide on their work, but it will also give other professionals and social scientists the ability to collect data efficiently and cheaply using readily available sensor platforms. This will also help to make the deployment, testing, and management of projects and devices in emerging regions easier.

2. BACKGROUND

An embedded DBMS designed specifically for sensor configuration management can enable researchers to conduct large scale randomized controlled trials (RCTs) by collecting and managing a large amount of sensor data from deployed devices. During a sensor backed RCT, separate survey data are managed and collected in order to supplement the sensor data. Having both survey and sensor data, researchers are able to gain a more accurate representation or grounding of the data that they are searching for [13]. Consequently, much time is wasted setting up two separate workflows, and thus, it is better if both were integrated in one infrastructure. The synchronization, metadata, and storage interface needed to integrate into these electronic survey workflows are not present in current sensor DBMS's.

In previous work, sensor data management for resource constrained systems gave little focus on metadata inclusion [1]. Furthermore, in most systems, the metadata that is collected is stored in a loosely associated table or database [8]. Because of the common use of database systems over the internet/wifi and on devices with memory, performance and connectivity, implementation methods for metadata inclusion either involves storing a specific database for metadata in the cloud or using verbose standards such as SensorML [9]. However, for the synchronization and integration of survey data with sensor data, it is critical that metadata and the sensor data be stored in the same location in the embedded system. Furthermore, many resource limitations must be taken into account when implementing a common storage space for sensor data and metadata. By

leveraging the same embedded DBMS for sensor data and metadata, we allow the user to integrate other kinds of data into the system and thus allow the integration of survey workflows.

3. DESIGN CONSIDERATIONS

Our target users for our initial system would be engineers supporting social scientists who have a need for independent automated data collection systems using low-cost custom or sensor systems. Another possible strategy would be to use a microcontroller with this system to act as a mediator for less customizable off-the-shelf devices. The database would be used to ease integration into a survey data collection workflow by keeping sensor metadata alongside the sensors themselves. This metadata would include the identifiers needed to map the sensors to their associated study participants and the sensor profiles assigned to them. Thus the sensor management can be integrated into the survey workflow, which will hopefully eliminate the need for field workers to be specially trained to work with sensor configuration, deployment, and data collection.

3.1 Example Scenario

A particular project motivating this work is that of the Stove Use Monitoring study in Darfur [13]. The study made use of the Thermocron iButton temperature loggers from Maxim Integrated to monitor cookstove usage and ODK 1.0 to administer electronic surveys. Recently, a second study in Bishoftu, Ethiopia was also conducted that made use of ODK 2.0 as well as the same sensors. In both studies, a separate group of fieldworkers managed the sensor related aspects of the project including programming, deployment, pulling of the data, and reconfiguration of the sensors for a second round of sensor data collection. In the Darfur study, the stoves were brought back by the participants to a central location where the sensors could be processed as surveys were also taken. Separate spreadsheet based tracking separate from the ODK survey workflow were used to record the sensor IDs and associate them with households. The programming and configuration were done using a Windows based laptop already tasked for storing the survey data. However, there was no integration between the two data sets. All joins of the data had to be done using the hand recorded spreadsheets of sensor/household mappings.

In the larger Bishoftu, Ethiopia study, the prohibitive cost of training two equivalent sized teams and the geographic and size of the project meant that the sensor team would often follow-up separately and slowly after the survey team had visited each site. This did not scale well. Context switching between surveys and sensors created difficulties in maintaining consistent quality of work. Surprisingly, a variety of time representations are used in Ethiopia and depending on the operator of the sensor, programming software sensors would have widely different time stamps. In order to handle all of the programming at once the stove sensors were configured in batches, but the batches didn't completely match with the survey field team deployment schedules and so some sensors would finish their programs early resulting in less data being collected during the actual deployment of the stove.

In both these situations the sensor configuration and deployment schedules were based on the survey schedules. However, having a separate team and deployment system for each meant that additional error prone coordination needed to take place.

3.2 Arduino Platform

Because of its popularity amongst researchers and its frequent use in development projects, we chose to base our design around the Arduino system, specifically on the Arduino Uno board. The Arduino is cost-effective and readily available which makes it very practical to deploy in large quantities. In addition, the Arduino ecosystem makes the device one of the most convenient devices to use. Not only does it provide great support for sensors, but it also provides an easy programming environment and contains many compatible devices that will allow for more functionality in your projects. Systems like the Grove sensor collection make available nearly a hundred sensors that are easy to integrate with Arduino code [5].

Memory constraints and speed constraints of the Arduino have to be taken into account in the design. Because the Arduino Uno only has 32 KB of Flash memory and 1 KB of EEPROM, the DBMS will have to have a very limited representation of metadata. The system will have to have some chosen fixed width definitions of the metadata and will also have to handle generic data. In addition, we will have to utilize an SD Card to store all of the data and metadata information if we want to store more than a few hundred reads of data at most. It would possible to fit most common metadata in the EEPROM and leave the bulk data for the SD Card. However, very descriptive metadata which we want to encourage could easily hit the EEPROM memory limitations. Old metadata can be overwritten with new metadata; however, we expect deletions and updates to be infrequent.

4. FEATURES

Our design for our micro database implements a minimal interface needed to collect data on an SD Card from sensors connected to an Arduino compatible device. We reduce the burden on the user of configuring datastores and managing the reading and writing of data reliably. With metadata support built-in, future synchronization and configuration steps will be easier. The metadata will also enable the transfer of relevant configuration data alongside data synchronization. Append only table operations are the dominant mode of storage, but metadata can be updated.

4.1 Metadata Support

One format for the metadata implemented in our design is based off the ODK Collect data format used on Android devices. The underlying base database functions are implemented on top of the Arduino Extended Database Library [12]. The below example descriptor table would be for one table. Each table on the microcontroller would have its own descriptor table. The table associated with these descriptors would have 5 columns for 3 sensors as well as time and a row id.

MDB descriptor format

_partition	_aspect	_key	_type	_value
Table	default	colOrder	object	[id,time, temp1, temp2, power1]
Table	default	localID	string	Arduino1
Table	default	indexCol	string	id
Table	default	name	string	FreezerControl
Column	temp1	dataType	string	integer

Column	temp1	unit	string	Celsius
Column	temp1	calib_offset	float	5
Column	temp1	calib_scale	float	0.0001
Column	temp1	accuracy	float	.25
Column	temp1	model	string	DS18B20
Column	temp2	dataType	string	integer
Column	temp2	unit	string	Celsius
...	...	...	...	...

The *partition* explains the scope of the metadata row. For example, is it for the entire table, a column in the table, or in rare circumstances for a row. The *aspect* specifies which partition in particular is being described, the *key* is what the value describes, and the *type* is the type of value in the last column. The type does not necessarily reflect the implementation type as that may vary from platform to platform.

The information in this table can be used to know what the configuration parameters are for the sensors as well as what form their output is in. Temperature sensor 1 for instance is a DS18B20 sensor needing a small correction in scale and offset temperature readings. This information is very valuable if you are trying to model the behavior of the temperature sensor in the field to interpret results.

4.2 Micro Database Interface

These are the 6 core functions needed to use the system.

`addStore(schema, version, localID, domain, globalID)`

`addEntry(storeLocalID, data)-> recordNumber`

`getEntry(storeLocalID, recordNumber) -> data`

`addMetadataEntry(storeLocalID, partition, aspect, key, type, value) -> status`

`updateMetadataEntry(storeLocalID, partition, aspect, key, type, value) -> status`

`getMetadataEntry(partition, aspect, key) -> metadataEntry`

The database interface is very simple and is oriented around a store and not necessarily a table. The local data stores include a schema as a string in a form like that used in the Python *struct* package and is primarily used to note the underlying C struct used for implementation. The actual concrete schema is the C struct declaration type used to represent the binary data inserted into the tables. This redundancy is necessary because of the lack of reflexivity in C doesn't permit us to just reuse the C struct declaration. The data in this case is a C byte array interpreted by the code using MDB.

4.3 Future Work

Presently, only the data storage system is implemented on the microcontroller. The synchronization functionality needed to pair with and Android devices and finally ODK Collect are still to be implemented.

The schemas used to configure the data stores on the microcontroller are presently tied to the underlying concrete types available to the microcontroller target. However, the schema representation should be constructible out of a metadata description table and compiled for use with the Arduino framework much like a very light variant of Protobufs [11].

Preliminary benchmarks show that database reads and writes allow simple temperature sensor based read and storage times of 1-2 readings per second which is adequate for our cookstove example. However, judicious caching of the metadata needed to locate and add new records is expected to increase the possible data collection rate.

5. ACKNOWLEDGMENTS

We would like to thank the Superb-ITS REU program and its director Tiffany Reardon as well as TIER graduate student Jordan Freitas.

Funding for this research was provided by the SUPERB-ITS program under NSF award CNS-1359499 and under CyberSEES award number CCF-1539585. This work was also supported by the Development Impact Lab (USAID Cooperative Agreement AID-OAA-A-13-00002), part of the USAID Higher Education Solutions Network.

6. REFERENCES

- [1] Ali, A. S., Zanzinger, Z., Debose, D. Stephens, B., Open Source Building Science Sensors (OSBSS): A low-cost Arduino-based platform for long-term indoor environmental data collection, *Building and Environment*, 100. 114-126. <http://dx.doi.org/10.1016/j.buildenv.2016.02.010>.
- [2] Brunette, W., Sodt, R., Chaudhri, R., Goel, M., Falcone, M., Van Orden, J., & Borriello, G. in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. 2012, ACM, 351-364.
- [3] Chaudhri, R. Borriello, G. and Thies, W. FoneAstra: making mobile phones smarter. In *Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions (NSDR '10)*. ACM, New York, NY, USA, 2010. DOI=<http://dx.doi.org/10.1145/1836001.1836004>
- [4] Chaudhri, R., Vlachos, D., Borriello, G., Israel-Ballard, K., Coutoudis, A., Reimers, P., & Perin, N. Decentralized Human Milk Banking with ODK Sensors. in *Proceedings of the 3rd ACM Symposium on Computing for Development*. 2013, ACM, 4-14.
- [5] Grove System. Retrieved September 16, 2016 from Seedstudio. http://wiki.seedstudio.com/wiki/GROVE_System
- [6] Kipf, A., Brunette, W., Kellerstrass, J., Podolsky, M., Rosa, J., Sundt, M., Wilson, D., Borriello, G., Brewer, E. and Thomas, E., (2016). A Proposed Integrated Data Collection, Analysis and Sharing Platform for Impact Evaluation. *Development Engineering*, 1. 36-44.
- [7] Open Energy Monitor. Retrieved September 16, 2016. <https://openenergymonitor.org/emon/>
- [8] Sandha, S.S., Randhawa, S., and Srivastava, B. Blue Water: A Common Platform to Put Water Quality Data in India to Productive Use by Integrating Historical and Real-time Sensing Data. *IBM Research Report*. 2016.

- [9] SensorML. Open Geo Spatial Consortium. Retrieved October 9th, 2016.
<http://www.opengeospatial.org/standards/sensorml>.
- [10] struct. Python Software Foundation. Retrieved September 16, 2016. <https://docs.python.org/2/library/struct.html>
- [11] Sumaray, A. and Makki, S.K. A comparison of data serialization formats for optimal efficiency on a mobile platform. in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12)*, ACM, (48), 6 pages. DOI:
<http://dx.doi.org/10.1145/2184751.2184810>
- [12] Whiddon, J. Arduino Extended Database Library. Retrieved September 16, 2016. <https://github.com/jwhiddon/EDB>
- [13] Wilson, D., Adam, M., Abbas, O., Coyle, J., Kirk, A., Rosa, J., Gadgil, A. Comparing Cookstove Usage Measured with Sensors Versus Cell Phone - Based Surveys in Darfur, Sudan. in *Technologies for Development*. 2015. 211-221.