1-2013

Provenance Framework for Mhealth

Aarathi Prasad
Dartmouth College

Ronald Peterson
Dartmouth College

Shrirang Mare
Dartmouth College

Jacob Sorber
Clemson University

Kolin Paul
Indian Institute of Technology, Delhi

See next page for additional authors

Dartmouth Digital Commons Citation

Prasad, Aarathi; Peterson, Ronald; Mare, Shrirang; Sorber, Jacob; Paul, Kolin; and Kotz, David, "Provenance Framework for Mhealth" (2013). *Dartmouth Scholarship*. 3087.
https://digitalcommons.dartmouth.edu/facoa/3087

Authors
Aarathi Prasad, Ronald Peterson, Shrirang Mare, Jacob Sorber, Kolin Paul, and David Kotz

Provenance framework for mHealth

Aarathi Prasad[1], Ronald Peterson[1], Shrirang Mare[1], Jacob Sorber[2], Kolin Paul[3], and David Kotz[1]

[1]Department of Computer Science, Dartmouth College
[2]School of Computing, Clemson University
[3]Indian Institute of Technology, Delhi

***Abstract*—Mobile health technologies allow patients to collect their health information outside the hospital and share this information with others. But how can data consumers know whether to trust the sensor-collected and human-entered data they receive? Data consumers might be able to verify the accuracy and authenticity of the data if they have information about its origin and about changes made to it, i.e., the *provenance* of the data. We propose a provenance framework for mHealth devices, to collect and share provenance metadata and help the data consumer verify whether certain provenance properties are satisfied by the data they receive. This paper describes the programming model for this framework, which describes the rules to be implemented for providing provenance-collecting capabilities to an mHealth application.**

I. Provenance in mHealth

Consider Jane, who is using one or more mobile health (mHealth) devices, continuously or periodically collecting her health-related information into her mobile phone. The phone periodically uploads this information, along with other health-related information that Jane manually inputs to her phone, to her electronic health record (EHR). Jane can then share her health information with her health providers, family and friends, peers, employer, insurer, and researchers. But how can these data *consumers* know whether to trust the sensor-collected and human-entered data they receive? What confidence do they have that it is accurate and authentic? Suppose Jane falls sick, and her husband Jack (her *caregiver*) collects her health information on her behalf. Jack has no experience using the devices. Can the system help him understand when he is not collecting accurate information?

Since personal and home-use devices are not maintained regularly, like hospital devices, *users* like Jane and Jack might not realize when the devices are malfunctioning or uncalibrated. If other people in Jane's household use similar devices, they might accidentally confuse their own device for Jane's. mHealth devices, being mobile, can be stolen or misplaced – and then used by another person. Jane, or her caregivers, might deliberately fake or hide data collected using mHealth devices, when there is an incentive to do so. In all these scenarios, the data collected and shared by the devices might be inaccurate, or about the wrong person. If so, its use could prove damaging or even fatal, especially if the data is used by health providers for diagnosis or treatment.

Data consumers might be able to verify the accuracy and authenticity of the data if they have information about its origin and about changes made to it, i.e., the *provenance* of the data. Previous research has looked at provenance of electronic health records and our work complements this research [1], [2]. In the case of health data that is collected using mobile sensors, it is necessary to collect information about the data's origin, so that consumers can determine the accuracy and authenticity of the data; our framework provides these capabilities, unlike the provenance middleware that may exist on the EHR server. Provenance has been identified as one of the challenges for data collected using mobile health devices [3].

978-1-4673-5494-3/13/$31.00

In one previous work, Medially, middleware for remote health monitoring, collects contextual provenance and triggers collection of high-fidelity data when certain contextual constraints are satisfied [4]. The middleware allows reconstruction of the contextual trigger states to help the data consumer understand why the data was collected. Our goals are different. Our framework will share *provenance metadata* with the data consumer, i.e., any context information that can attest to the authenticity and accuracy of the data and can help in interpreting the data. We also share a *confidence value* that the system assigns to the data, with respect to the metadata, so that the data consumer can infer whether the provenance properties she desires, like data accuracy, are satisfied.

To realize this vision, we propose a provenance framework for mHealth. The primary function of the framework is to collect and share provenance metadata and help the data consumer verify whether certain provenance properties are satisfied by the data they receive. An mHealth app *developer* uses the provenance framework to add provenance-tracking capabilities to her mHealth application. The developer works with a *domain expert*, i.e., someone who is familiar with the medical requirements of the application, to define the provenance properties desired by each class of data consumer (clinical staff may desire different properties than, say, family members), and to identify the provenance metadata required to verify these properties. Using tools in our framework, the developer and expert collaboratively prepare a *provenance model* for each class of data consumer in their application.

In our approach, the provenance model defines *rules* that determine how and when contextual metadata should be collected and when the device user or data consumer should be alerted to possible problems with the data. One straightforward approach would collect everything about the context and store it with the data, later analyzing this metadata to verify whether the provenance properties are satisfied. Our rule-based approach ensures that only necessary metadata is collected, saving energy and storage. It is also possible for rules to trigger immediate responses, based on the the data and metadata collected.

When the application is installed on the phone, these rules are added to a *rules file* on the user's mobile phone. A *provenance service* running on the user's mobile phone will interpret the rules to collect metadata, process it, bind it with the data, and send it to the electronic health record, or to alert the caregiver or others. The data consumer can identify problems in the sensor readings based on the *confidence* the system assigns to the sensor data.

Security Assumptions:

1) The phone and sensors are not compromised, i.e., the adversary can not compromise the hardware or software in the phone or in the sensors.
2) There is a secure connection between the phone and the sensor and between the phone and the server. (The phone and the sensors are capable of performing crypto operations such as encryption and hashing, have a common wireless radio to communicate, and have shared secret key(s) that are not available to the adversary. With these assumptions, the phone and the sensors can establish a secure communication channel.)

The provenance framework has three components – a modeling tool that is used by the developer and the domain expert to build the provenance model and generate the rules, the provenance middleware that implements a rule engine that runs on the user's mobile phone, and the graphical user interface for the consumer on the server side. This paper describes the kinds of rules to be implemented for providing provenance-collecting capabilities to an mHealth application. First we describe some motivating scenarios, then introduce the rules grammar, and then demonstrate how the rules would be applied to the scenarios.

II. Scenarios

Consider a specific scenario. Devi, a health worker, visits pregnant women in a village in India every week with a sensor kit that has mHealth devices like a blood-pressure cuff, a portable anemia sensor, heart-rate monitor, fetal monitor, spirometer, and weight scale, along with an Android phone and a wireless printer. The pregnant women are enrolled as patients in the village health clinic and the data collected by Devi is uploaded to their patient records in the clinic's electronic health record system. A printout of the collected data (and health instructions) is given to the patient. The system alerts Dr. Ravi in the event of an abnormal reading. If no alerts occur, Dr. Ravi checks the patient records once a week.

Case 1: **Spirometer reading.** Devi takes patient Ritu's lung capacity reading using the spirometer. When the reading is taken, the atmospheric dust and pollen count in Ritu's town is collected by Devi's phone from the weather website. The dust and pollen count readings are sent to Ritu's electronic health record along with the other vital-sign readings collected by Devi using the mHealth app on her phone. Dr. Ravi is alerted by the electronic record system that Ritu, one of his patients, has had a gradual decrease in her lung capacity over a period of four weeks. His first suspicion is that Devi's spirometer was not working correctly, but then he notices that there was a high concentration of dust and pollen around Devi's house. Since the system also attests to the fact that all the devices Devi was using were working correctly, the readings from the website was accurate and the spirometer readings were indeed from Ritu herself, he trusts the readings and investigates the possibility that Ritu is suffering from problems related to dust and pollen inhalation.

Case 2: **Hemoglobin reading.** One day, Devi is unwell, but still decides to go on her visits. She visits Ritu. The mHealth app records Ritu's hemoglobin count after she slides her finger into an anemia sensor [5]. Devi realizes later that she is unwell. She feels sick, but she had to visit Priya's house as well. So instead of visiting Priya, Devi takes her own hemoglobin reading using the anemia sensor and saves it as Priya's. The system alerts Aman, the clinic manager that Devi took a reading for Priya but her fingerprint did not match Priya's. Devi is warned about her carelessness, and the data is marked invalid.

III. Rules for provenance

The developer, with the help of the domain expert, constructs a set of rules to collect, process, store, and send provenance metadata to the patient's health record. The rules file and the library containing developer-defined functions are also installed on the mobile phone, when the mHealth application is installed.

The grammar described in Figure 1 describes the rules that can be interpreted by the provenance service. We have used a modified version of the Extended Backus-Naur form to describe the rule structure. The syntax of the form is as follows:

- `x ::= y` is a definition, where x is a non-terminal.
- `a ::= x|y` means a is defined either as x or y.
- `x => y` means y is executed when x condition is true.
- {x} means that x occurs zero or more times.
- [x] means that x is optional.
- (x|y) means either x or y is chosen from the group containing x and y.
- x denotes that x is a terminal. All underlined words are keywords in the grammar.
- A `word` is a sequence of letters starting with a lower-case letter and a `Word` is a sequence of letters starting with an upper-case letter. `number`, `string` and `boolean` are constant values as in most languages. `time` is represented as sequence of two characters separated by '/' (that denotes the date as mm/dd/yy) followed by a whitespace, and then by a sequence of integers separated by ':' (that denotes the time as hours:minutes:seconds). `duration` represents a time period, which is denoted by an integer, followed by whitespace and then followed by one of the

```
       stmt  :=  (rule | decl);
       rule  :=  condition => {action,}action
  condition  :=  event | condfunction
              |   not(condition)
              |  (condition)logop(condition)
              |  (extfunction)relop(value)
     action  :=  Word([args])
   function  :=  condfunction | extfunction
              |  action
condfunction :=  isWord([args])
 extfunction :=  getWord([args])
      event  :=  wordEvent[(args)]
      logop  :=  and | or
      relop  :=  ≤ | ≥ | <= | >= | ==
      value  :=   NIL | number | string | time
              |  boolean | duration | location
       data  :=  reading | window
       args  :=  {arg,}arg | pairs
        arg  :=  data | word | value
      pairs  :=  {pair,}pair
       pair  :=  ≤word,arg≥
     source  :=  wordSource
       role  :=  wordRole
       decl  :=  Source {source,}source
              |  Role {role,}role
              |  Event {event,}event
              |  Function {function,}function
              |  Library {word,}word
              |  Error {word,}word
```

Figure 1: Rule-based grammar for mHealth provenance framework

following words: secs, mins, hours, days, weeks, months or years. location is a sequence of three integers separated by a ':', which represents a value collected by a GPS sensor (denoted by degrees:minutes:seconds).

- source, role, reading and window are objects that represent a type of sensor data and metadata, a class of consumers, a data reading, and a window of data respectively.
- Any line in the rules file may have a comment, from a '%' symbol to the end of the line.
- Newlines and other white space are otherwise ignored.
- decl statement declares a word to be a specific type, namely a source, role, event, function, library or an error.
- actions cannot alter data, sources or roles.

Each rule specifies an action (or sequence of actions) that is executed if a condition occurs. A rule can be executed under different circumstances. First, a rule can be executed when an event is posted by one of the applications on the phone. An event service (part of the framework) continuously listens for application events on the phone. When one of these events occurs, the event service notifies the provenance service and the corresponding rule(s) are executed. Note events can have parameters that carry information from the posting application to the actions.

A rule can also be executed when a *conditional function* returns true or when the value returned by an *extractor function* meets a given condition. These functions are defined by the developer. A *source* has the following attributes: sensor handle, data (reading or window) and confidence. A *role* has attributes like name, contact information, biometric information and confidence. reading and window represent data objects, with attributes like data, confidence and the time and location at the start and end of collection of the particular data reading or window. In the case of reading, the data attributes are word-value pairs. A window is a sequence of readings collected over a window of time; attributes are word-value pairs. The details of these objects are handled in functions and are out of the scope of this paper.

Extractor functions are used to extract the object attributes. The attributes, including the data that is collected from sensors, are of type NIL, number, string, boolean, time or location or a combination of two or more of these types (reading or window). Conditional functions can test whether the extracted information meets certain criteria.

One more type of function is permitted in the rule grammar; actions are essentially *action functions* that are called by the provenance service when a rule is executed. These actions could for example, request the sensor manager to collect sensor data. We assume there is a sensor manager service in the mobile phone, responsible for maintaining a list of sensors connected to the mobile phone, collecting data from these sensors when asked, and distributing it to the requested parties (i.e., an app or the provenance service). Funf [6] and Open Data Kit [7] are two examples of such a sensor manager service. As soon as it receives any data from the sensor manager, the provenance service writes the metadata to a binary log file on the phone, for later use in auditing. The developer can also define action functions that notify people, process health data and provenance metadata (for example, generate a summary) and send this processed data and metadata to the patient's electronic health record.

The rule evaluation works (in effect) as follows:

```
Forever:
  For each rule, in order:
    if condition true, execute action(s).
```

The parameters used to evaluate a condition are the same parameters that are used while executing the corresponding action; even if newer data is available, the same data that was used to evaluate a condition is used while executing the corresponding action.

IV. Programming model for provenance middleware

Here is a sketch of how our approach will work. We imagine a collaborative process in which a domain expert (who understands the medical context) works with the application developer (a programmer building the mHealth application) to identify and define the provenance needs for the app.

1) Domain expert will decide what metadata is required by consumers and when it should be collected.
2) Developer, working with domain expert, defines sources used by the mHealth application for both health data and provenance/context data. Declare the different roles and include all the classes of consumers.
3) For each type of metadata, the developer should define rules to specify when to collect, re-collect and send the metadata to the patient's health record, when to notify consumers, and handle error cases (like when the sensor or sensor data is not available or manual data is not entered).
4) Some metadata might be a combination of data obtained from two different sources. In this case, the developer could write code for a "virtual sensor" that can be installed in the sensor manager.
5) For each metadata, call the action `Reading`(source) or `Window(source)` to ask the sensor manager to contact the relevant sensor and collects data from it as a single reading or a window of readings. The developer should specify a rule to handle the case when the sensor manager cannot find such a sensor.
6) The developer can define extractor functions like `getSensor(source)` to get information or `getSummary(getData(source))` to summarize data.
7) For each source, define a rule that is executed when the data is available to be used by the provenance service. Also define rules in the case of timeouts or other failures to retrieve sensor data.
8) If no rules are specified for timeout and error events, the provenance service calls an inbuilt function `Log(cause, errorcode)` that logs the timeout or error and its cause to the error log file.
9) Define a rule that uses the `Send`(..) action for each group of health data that is collected by the application. This action sends the provenance metadata needed to verify the provenance properties of the health data, to the patient's electronic health record. The `Send` function takes a list of attribute-value pairs and the EHR identifier.

The functions are written in Java and the files are compiled into a library that is installed on the phone along with the mHealth application and rules file. The provenance service reads the rules file and links with the library, before data collection begins.

V. Demonstration of rules

Next, we demonstrate how the provenance framework can be used by describing the rules for the two cases in Section II.

Case 1: **Spirometer reading.** This case demonstrates how provenance metadata is collected as a window of continuous readings from an external sensor, that is not a part of the health kit. The provenance property for clinicians is the accuracy of lung capacity, and anything that can help the clinicians interpret the lung-capacity reading.

The domain expert recommends that obtaining dust and pollen count in the patient's town can help clinicians interpret lung-capacity readings. To verify the accuracy of the lung-capacity reading, the clinician will need to be convinced that the spirometer was not faulty (confidence in sensor), was applied correctly (confidence in health worker) and was used on the right patient (confidence in patient). Lung capacity, dust and pollen are declared as sources.

```
Library mHealthKitLibrary;
Source lungCapacitySource, dustSource;
Role patientRole, healthworkerRole,
  clinicianRole, attendantRole;
Event recordSavedEvent;
Error lowConfidenceError, timedOutError;
```

The default library, provenanceLibrary, includes events like `dataAvailableEvent(source)` and `timeOutEvent(source)` and functions like `Window(source, time)` and `getSensor(source)`. The functions are overridden if they are defined again in the custom library mHealthKitLibrary, which is written in Java.

The sensor manager posts an event `dataAvailableEvent(lungCapacitySource)` after it collects the lung capacity reading from the spirometer. The sensor manager posts the `dataAvailableEvent` after each window is collected and ready to be distributed. The following rule defines when to collect dust and pollen count – after the provenance service receives the lung capacity reading from the sensor manager. The function `Window(source, duration)` asks sensor manager to collect readings taken during the last n days/hours/minutes/seconds.

```
dataAvailableEvent(lungCapacitySource) =>
  Window(dustSource, 7 days),
  Window(pollenCountSource, 7 days);
```

The following rules handle error cases. The first rule considers whether the confidence in the readings is less than the threshold recommended by the domain expert. `getWindow(source)` gets the last window of readings that was collected from the sensor; this value is not updated while an action function is using it, even if newer readings are available. The second rule is executed when the connection to the dust sensor times out. The `timeOutEvent(source)` event is posted by the sensor manager when the connection to the sensor times out. When the provenance service executes the `Notify` function, it contacts the sensor attendant with the second argument as the message. We assume that the sensors, along with data, also provide a value that indicates how much confidence the sensor has in the accuracy of the collected data. The function `getConfidence(object)` returns the confidence for the object. For sensors and sensor data, it is the value provided by the sensor. Confidence in people is determined by the value provided by the biometric sensors and the algorithm verifying the biometrics.

```
(dataAvailableEvent(dustSource))and
((getConfidence(getWindow(dustSource)))<(0.8))
  => Log(getSensor(dustSource),
    lowConfidenceError);
  Notify(getAttendant(dustSource), “Low
    confidence for reading”);

timeOutEvent(dustSource) =>
  Log(getSensor(dustSource),
    lowConfidenceError);
  Notify(getAttendant(dustSource), “Low
    confidence for reading”);
```

In this case, no additional rules are added to handle other error cases. All error events invoke the `Log(cause, errorcode)` function, which logs the error to the error log file. Implicitly, there is a rule `errorEvent = > Log(cause, errorcode)`.

The `recordSavedEvent` is posted when the health worker clicks on the Save button on the mHealth kit application, uploading data to the patient's health record. The provenance service then sends the metadata and the confidence values to the patient's electronic health record.

```
recordSavedEvent = >
  Send(patientEHR, <data, lungCapacitySource>,
    <metadata, dustSource>,
    <sensor,getSensor(dustSource)>,
    <confidence,
      getConfidence(getSensor(dustSource))>,
    <reading,
      getSummary(getWindow(dustSource))>,
    <confidence,
      getConfidence(getWindow(dustSource))>,
    <metadata, pollenCountSource>,
    <sensor,
      getSensor(pollenCountSource)>,
    <confidence, getConfidence
      (getSensor(pollenCountSource))>,
    <reading,
      getSummary(getWindow(pollenCountSource))>,
    <confidence, getConfidence
      (getWindow(pollenCountSource))>,
    <confidence, getConfidence
      (getReading(lungCapacitySource))>,
    <confidence,
      getConfidence(patientRole)>,
    <confidence,
      getConfidence(healthworkerRole)>);
```

Case 2: **Hemoglobin reading.** This case demonstrates how provenance metadata is collected from a virtual sensor (code installed on the sensor manager that “senses” derived data). The provenance property for the class of clinicians is the accuracy of the hemoglobin reading. The provenance property for the class of clinic managers is the efficiency of the health worker.

The domain expert recommends that tracking the health workers and monitoring their activities can help understand their efficiency. To verify the accuracy of the hemoglobin reading, the clinician will need to be convinced that the anemia sensor was not faulty (confidence in sensor), was applied correctly (confidence in health worker) and was used on the right patient (confidence in patient). The patient can be identified using her fingerprint that is also collected using the anemia sensor. Hemoglobin, fingerprint and healthworker efficiency are declared as sources. Healthworker efficiency is collected by a virtual sensor. The virtual sensor reads the location of the phone from its internal GPS sensor every time the sensor manager is requested by the mHealth application to collect some data. It also reads the data from the accelerometer, and determines whether the health worker was mobile or stationary. Based on the GPS and accelerometer readings, the virtual sensor computes the activity of the health worker, whether she is walking, whether her last GPS reading matches with the address of the patient she is taking measurements of, or whether it is a new patient.

```
Library mHealthKitLibrary;
Source hemoglobinSource, fingerprintSource;
Role patientRole, healthworkerRole,
  clinicianRole, clinicManagerRole;
Event recordSavedEvent;
Function isfingerPrintMatch(fingerprintSource,
  patientRole);
Error noNewReadingTakenError,
  wrongPatientError;
```

The sensor manager posts an event `dataAvailableEvent(hemoglobinSource)` after it collects the hemoglobin reading from the anemia sensor. The following rule defines when to re-collect hemoglobin reading – after the provenance service receives the lung capacity reading from the sensor manager, but finds it to be of low confidence. The function `NewReading(source)` logs the old reading and calls `Reading(source)`, which requests the sensor manager to collect reading from `getSensor(source)`.

```
(dataAvailableEvent(hemoglobinSource))and
((getConfidence(getReading(hemoglobinSource)))
<(0.8)) = > NewReading(hemoglobinSource);
```

The following rules apply when the reading is not received after a certain period of time. If request times out and no error occurs, it implies that the reading was not taken. We assume, for now, that the formula for confidence computation is provided by the domain expert. The `timeOutEvent(source)` is fired by the sensor manager when the connection to the sensor collecting source times out. When the provenance service executes the `Notify` function, it sends an email to the clinic manager. The clinic manager is notified that no new readings were taken for hemoglobin. Another definition of the `Notify` function takes a third parameter, an argument list, that the clinic manager will need to understand who was at fault and assess what went wrong. The clinic manager is also notified if the fingerprint obtained does not match that of the patient. The function `isFingerPrintMatch(fingerprintSource, patientRole)` returns true, if there is at least an 80% match of `getReading(fingerprintSource)` and `getFingerPrint(patientRole)`.

```
(timeOutEvent(hemoglobinSource)) and (not
(errorEvent)) =>
  Log(healthworkerRole, noNewReadingError),
  Notify(clinicManagerRole, “No new
    readings taken for hemoglobin”,
      healthworkerRole);

(dataAvailableEvent(fingerprintSource))
and (not (isFingerPrintMatch
(fingerprintSource, patientRole))) = >
  Log(healthworkerRole, wrongPatientError),
  Notify(clinicmanagerRole, “Wrong
    patient”, healthworkerRole);
```

The recordSavedEvent is posted when the health worker clicks the Save button on the mHealthkit application. The provenance service then sends the metadata and the confidence values to the patient's electronic health record.

```
recordSavedEvent = >
  Send(patientEHR, <data,hemoglobinSource>,
    <metadata,fingerPrintSource>,
    <sensor,
      getSensor(fingerPrintSource)>,
    <confidence, getConfidence
      (getSensor(fingerPrintSource))>,
    <reading,getReading(fingerprintSource)>,
    <confidence, getConfidence
      (getReading(fingerprintSource))>,
    <metadata,healthworkerEfficiencySource>,
    <reading, getSummary(getWindow(
      healthworkerEfficiencySource)>,
    <confidence,getConfidence(getWindow(
      healthworkerEfficiencySource))>,
    <confidence,getConfidence
      (getReading(hemoglobinSource))>,
    <confidence,getConfidence(patientRole)>,
    <confidence, getConfidence
      (healthWorkerRole)>);
```

VI. Future work

We will implement the provenance middleware – provenance service, event service and sensor manager – on an Android platform. The middleware will be available for download and can be installed on an Android platform. We expect the middleware to complement existing mobile-health data collection frameworks like Sana [8] and Open Data Kit [7], in that it provides provenance capabilities to these frameworks. We have presented the rules grammar, but are yet to provide a formal proof of completeness of the grammar. We also plan to explore ways to combine confidences of disparate sources using existing techniques like Dempster-Shafer theory [9].

Once the middleware is completed, we will build the modeling framework to help developers and domain experts construct and validate the provenance model and rules and the graphical user interface where recipients can view the data and verify its provenance properties.

VII. Summary

In this paper, we propose a provenance framework for mHealth, which collects and shares provenance metadata to help the data consumer verify whether certain provenance properties are satisfied by the data they receive. We also describe the rules the developer and domain expert need to construct to provide provenance-collecting capabilities for an mHealth application. We present the rules for two scenarios to demonstrate how the provenance middleware will work, and explained our plans for implementing the provenance framework.

Acknowledgements

We thank Ashutosh Sabharwal, Sanjiva Prasad, and the Dartmouth TISH group for their continued feedback.

This research results from a research program at the Institute for Security, Technology, and Society at Dartmouth College, supported by NSF under award numbers 0910842 and CNS-1143548, and by the Department of HHS (SHARP program) under award number 90TR0003-01.

References

[1] T. Kifor, L. Z. Varga, J. V. Salceda, S. Alvarez, S. Willmott, S. Miles, and L. Moreau, “Provenance in agent-mediated healthcare systems,” *IEEE Intelligent Systems*, vol. 21, no. 6, pp. 38–46, Nov. 2006. DOI 10.1109/MIS.2006.119

[2] T. D. Wang, C. Plaisant, A. J. Quinn, R. Stanchak, S. Murphy, and B. Shneiderman, “Aligning temporal data by sentinel events: discovering patterns in electronic health records,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 457–466. DOI 10.1145/1357054.1357129

[3] N. Davies, C. Efstratiou, J. Finney, R. Hooper, G. Kortuem, and M. Lowton, “Sensing danger: Challenges in supporting health and safety compliance in the field,” in *Proceedings of the Workshop on Mobile Computing Systems and Applications (HotMobile)*, Mar. 2007, pp. 34–38. DOI 10.1109/HotMobile.2007.7

[4] A. R. Chowdhury, B. Falchuk, and A. Misra, “Medially: A provenance-aware remote health monitoring middleware,” in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Mar. 2010, pp. 125–134. DOI 10.1109/PERCOM.2010.5466985

[5] “Biosense.” Available online: http://www.biosense.in/touchb

[6] N. Aharony, W. Pan, C. Ip, I. Khayal, and A. Pentland, “Social fMRI: Investigating and shaping social mechanisms in the real world,” *Pervasive and Mobile Computing*, vol. 7, no. 6, 2011. DOI 10.1016/j.pmcj.2011.09.004

[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 International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, Jun. 2012, pp. 351–364. DOI 10.1145/2307636.2307669

[8] Sana. Available online: http://sana.mit.edu/

[9] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 1976. Available online: http://www.glennshafer.com/books/amte.html