WINDWare: Supporting Ubiquitous Computing with Passive Sensor Enabled RFID

Asanga Wickramasinghe, Damith C. Ranasinghe
Auto-ID Lab,
The University of Adelaide, Adelaide SA 5005, Australia
Email:{asanga.wickramasinghe, damith.ranasinghe}@adelaide.edu.au

Alanson P. Sample
Disney Research, 4720 Forbes Avenue,
Suite 110, Pittsburgh, PA 15213, USA
Email:alanson.sample@disneyresearch.com

***Abstract*—Recent emergence of passive sensor enabled RFID tags (sensor tags) is creating new possibilities for low cost and maintenance free paradigm for industrial and ubiquitous monitoring applications (such as structural health, elder care, cold chain management). Despite the clear advantages, the widespread adoption of sensor tag technology requires the development of middleware to support the management of data streams with integrated sensor and ID (identification) data. In this paper we propose a generic middleware architecture adhering to the standardized EPCglobal architecture for managing integrated ID and sensor data streams from sensor tags. Furthermore, we demonstrate its successful implementation, WINDWare, through laboratory experiments and an application demonstration.**

I. Introduction

Passive Radio Frequency Identification (RFID) is unique because both power and data is wirelessly sent to the tag by a reader (as opposed to battery powered sensor nodes). Recent emergence of passive low cost [1] sensor enabled RFID tags (here after *sensor tags*) such as the Wireless Identification and Sensing Platform (WISP) [2] developed by Intel Research provide new impetus for commercial [3]–[5] (*eg:* in cold chain management where temperature is constantly monitored) and medical applications [6] while enabling ubiquitous monitoring [7]–[9] (*eg:* in elderly care to identify human activities). However, widespread adoption requires developing not only low cost sensor tags but also applications and technology neutral middleware that consider the unique nature of sensor tag data streams and provide capabilities to manage sensor and ID data integrated at the hardware layer [10].

Sensor tag data streams are unique because sensor data is embedded in the tag ID, such as the EPC (Electronic Product Code) [11]. Sensor tags employ a *query-only approach* to send sensor and ID data where a *Query* command for a tag ID results in reading the ID from sensor tag memory, and acquiring sensor data and embedding in the ID (Figure 1(a)). The alternative is to *Write* sensor data to memory and subsequently acquire the data by a *Query* and a *Read* command [12](Figure 1(b)). For example, researchers in Igarashi et al. [13] accessed the structured memory within an RFID tag to acquire sensor information. However, this approach consumes more power (148 μW for writing vs. 10 μW for reading [14]) and degrades performance because: i) the nature of radio wave propagation (i.e. Friis Transmission Equation [15]) implies that tags that consume more power must be activated at shorter ranges; ii) higher power consumption results in slower sensor data update rates (i.e. duty cycling); and iii) the two round trip times required to acquire sensor and ID data results in sensor data acquisition delays. Therefore, new middleware is needed to process data from sensor tags using the lower power *query-only approach* to: i) maximize the read range of sensor tags; and ii) maximize the rate at which the tags can stream data back to readers [2].

Furthermore, implementing middleware for managing sensor tag data requires addressing several key issues. Firstly, an efficient mechanism is required for discovering a sensor tag's capability (*eg:* types of embedded sensors) so that: i) data can be extracted and transformed correctly from a tag's ID; and ii) appropriate filtering and aggregation operations can be performed on the sensor data (as determined by the sensor data type such as temperature or acceleration). Secondly, an application agnostic sensor data and ID data subscription specification and a data model for reporting high level sensor tag data (*eg:* average temperature) to client applications are needed.

In this paper we develop, WINDWare (**W**ireless **I**dentification a**n**d Sensor **D**ata Management Middle**ware**), a generic middleware framework that not only addresses the lack of a middleware for simultaneously managing both RFID tag and sensor tag data, but also addresses the above challenges to facilitate application development in ubiquitous computing based on sensor tags. Additionally, WINDWare will accelerate the adoption of sensor tags such as the WISP. Furthermore, our middleware conforms to existing standards because we extended the Application Level Event (ALE) interface of the EPCglobal architecture framework as proposed in [10], [12] to facilitate subscription and reporting of data. We summarize our contributions below:

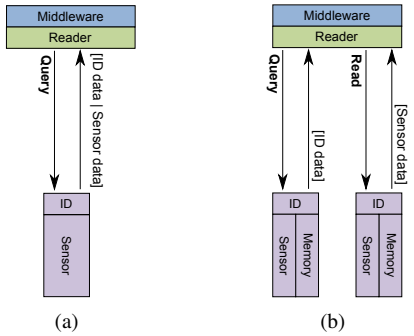


Fig. 1. Comparison of sensor data acquisition approaches: (a) embedding sensor data within ID; (b) reading sensor data from tag memory

978-1-4799-3587-1/14/$31.00 ©2014 IEEE

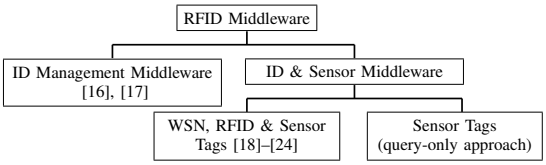


Fig. 2. Classification of existing RFID supported middleware

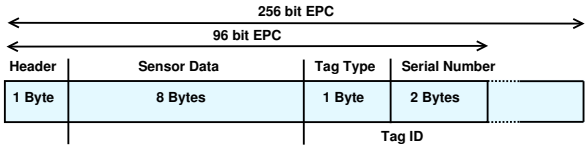


Fig. 3. Proposed tag format for 96-bit and 256-bit Electronic Product Codes

- We developed an extensible and generic framework which conforms to the standardized EPC global architecture for managing both ID data streams, and ID and sensor data streams that occur from *sensor tags*, for the first time (to the best of knowledge)
- We proposed a generic data format to embed sensor data in a tag ID, a new tag data model for integrated sensor and ID data representation, and an extensible data model to support subscription and reporting of ID and sensor data to client applications.
- We demonstrated a successful implementation of our middleware framework (WINDWare) and tested the validity of our framework by conducting experiments using both a simulated and a laboratory environment.

The following sections of the paper are organised as follows: Section II discusses related work; Section III presents our proposed architecture of WINDWare; Section IV details our implementation; Section V presents the experiments conducted and results; and Section VI presents an application of WINDWare and we conclude the paper in Section VII.

II. Related Work

Currently for RFID, among others, both commercial [18]–[20] and open-source [16], [17], [21]–[25] middleware are available. Figure 2 illustrates a classification of existing middleware, based on the middleware's ability to manage ID data and sensor data.

As shown in Figure 2, a number of middleware platforms [18]–[24] has the capability to gather data from wireless sensor networks where sensor and ID (unique object identifier) data streams are processed separately as they are generated from different sources (*eg:* RFID tags, wireless sensor networks). However, the applications that receive data (ID and sensor data) still have the challenge of associating sensor data (such as ambient sensors in the environment) with unique objects (such as sea food cases in a cold chain management application). For example,in the middleware proposed by Bade et al. [21], sensor information is processed to identify unacceptable environmental states and subsequent RFID reader scans identify the objects possibly in the undesirable state. Conversely, in Gama et al. [22], sensor data is polled from a sensor network upon receiving events from RFID readers and are presented as set of environmental information that corresponds to all the observed tags. In work done by Wang et al. [23], sensor and ID data are reported separately and data association implied is derived based on sensor location. These are significantly different low level data collection, high level data reporting, and sensor and ID data integration approaches to that required by sensor tags where sensor data and ID data integration is at the hardware layer (since the sensor is on the same platform as the ID carrier). Moreover the middleware developed is not generic but application specific.

Also, from the perspective of application development and interoperability it is also important to abstract from implementation, and provide a technology and implementation agnostic specification for subscription to and reporting of high level events. While researchers in Gama et al. [22] and Wang et al. [23] extend the standardized ALE (Application Level Event) reporting specification [11] for sending sensor events from sensor networks, they do not integrate ID and sensor data at the middleware level and are reported as separate high level events.

Although some existing middleware support sensor data acquisition from sensor tags [18]–[21], [24], they do not support the query-only approach but instead support the approach shown in Figure 1(b). However, these approaches are not agnostic to tag type (ID tag, sensor tag) since the tag must be first singulated and subsequently discovered to be a sensor tag (by way of a lookup as in Igarashi et al. [13]) prior to acquiring sensor data from a tag's memory. Furthermore, as highlighted in Section I, using the approach in Figure 1(b) is detrimental to the performance of a passive tag in terms of power consumption and time taken to obtain sensor data.

Therefore, existing middleware have no support for data acquisition from sensor tags that follows the query-only approach (see Section I). Most of the middleware fail to integrate ID and sensor data and report them as a single high level event. In addition, other middleware solutions are limited by their application specific nature. Consequently, we are motivated to design and implement a middleware that defines a simple, flexible and extensible tag data model for both RFID tags and sensor tags that support query-only approach and provide a generic framework for implementing, collection, filtering, aggregation and reporting of both ID and sensor data.

III. Middleware Architecture

In this section we present the architecture of WINDWare that manages ID and sensor data streams from passive sensor enabled RFID tags (sensor tags).

A. Proposed Tag Data Format

We propose a tag ID format (shown in the Figure 3) for sensor tags based on that used for WISPs [2]. Our proposed format uses the EPC (Electronic Product Code) defined in Tag Data Standard [11]. The **Header** field is set to `0x3D` (an EPC header currently unused and reserved for future use) to identify a sensor tag. This allows the middleware to process the sensor data in addition to the ID data as well as manage existing EPCs from ID tags. The content and the format in the **Sensor Data** section depend on the tag type. Proposed tag

TABLE I. DEFINITION OF RFID STANDARDS RELATED TERMS

Term	Description
EPC	The Electronic Product Code; a globally unique identifier
Event Cycle	Smallest unit of interaction with the ALE implementation during which the ALE interface implementation (middleware) interact with one or more Readers on behalf of an ALE client
ECSpec	Event Cycle Specification; data element defined in the ALE API which defines the parameters for **ECReport**, **ECReportSpec**
ECReport	Event Cycle Report; reports tag information for the current event cycle as specified in corresponding **ECReportSPec**
ECReportSpec	Event Cycle Report Specification; provides the content definition for the **ECReport**: readers, event cycle duration, and tag filtering
ECReportOutputSpec	EC Report Output Specification; specifies what information on the final set of EPCs in **ECReport** is reported
ROSpec	Reader Operation Specification; specifies reader operation parameters (e.g.: identifier, the boundary specification, priority) and configurations such as start and end triggers, and antenna configurations
ROReportSpec	Reader Operation Report Specification; appear as a sub element of **ROSpec** which describes the contents of the report sent by the Reader and defines report trigger events
ROBoundarySpec	Reader Operation Boundary Specification; defies the span of the operation (indicating the start trigger and stop trigger)
ROSpecStartTrigger	Appear as an element within **ROBoundarySpec**; defines the trigger event for the reader to initiate report generation

Source: EPCglobal Inc, Epcglobal ratified standards. [Online]. Available:http://www.gs1.org/gsmp/kc/epcglobal/ [11]

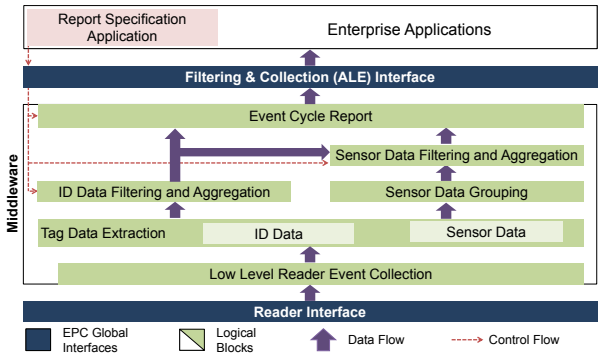


Fig. 4. Sensor tag data management middleware architecture

type definitions are based on WISP tag types[1]. A combination of the **Tag Type** and **Serial Number** are used to uniquely identify each sensor tag. Although the serial number is two bytes for a 96-bit EPC, using a 256 bit EPC can significantly increase the range of serial numbers, which is essential for large scale sensor deployments. Therefore, for a specific tag, only the content of the sensor data field is variable.

B. Architecture Overview

For seamless integration with existing applications, we base our architecture on the EPCglobal architecture [11]. Figure 4 illustrates the system's architecture of WINDWare. Below we discuss how our architecture meets a number of key requirements identified for RFID middleware in [26] as well as the specific processing requirements of sensor tags we identified in Section I.

Low level reader event collection. A standard reader interface (such as the Low Level Reader Protocol [11]) is used to collect tag reads from physical devices (*eg:* RFID readers).

Tag data extraction. In the event of an ID tag no additional operations are required, however in the event of a sensor tag, data embedded in the EPC (Figure 3) must be removed and the tag ID is reconstructed without the sensor data. The sensor data and ID data are then integrated into a single data structure defined by the Tag Data Model (see Section IV-A).

[1]see: https://wisp.wikispaces.com/Working+with+WISP+firmware

ID data filtering and aggregation. Filtering (*eg:* remove duplicate IDs) and aggregation operations (*eg:* count products of the same category as opposed to reporting tag IDs) specified by the user based on ID level data is performed by this module.

Sensor data grouping. Since it is possible to have multiple sensor readings from the same sensor tag, it is necessary to group the sensor readings by **Tag ID**. Here we do not filter to remove duplicates because this process can eliminate potentially crucial sensor values.

Sensor data filtering and aggregation. Most commonly, applications are interested in subsets of collected data (filtered data) [10], [26] such as temperature values above a threshold, or temperature of a certain product. Sensor data can also be aggregated (*eg:* in time or space domain) based on application requirements. For example, to provide the average temperature or to combine temperature data from different readers observing a physical location.

Event Cycle Report. An **EventCycle** is constructed according to a specification called **ECReportSpec** (see the ALE Specification [11]) generated by a client (see report specification application in Figure 4). While the **ECReportSpec** specifies several parameters, the two most important parameters are: i) the time interval during which tag reads are collected and processed; and ii) the readers from which tags are collected. Then Event Cycle Report module is responsible for managing the construction and transmission of the user specified report.

IV. MIDDLEWARE IMPLEMENTATION

We implemented our middleware by extending the Fosstrak open-source middleware [17] which only supports ID tag data management. Selection of Fosstrak is based on: i) its conformance to the EPCglobal architecture [17]; and ii) the support for LLRP (Low Level Reader Protocol) [11], a standard **Reader Interface**, to communicate with RFID readers.

The class diagram in Figure 5 illustrates our middleware design in Fosstrak. We have only outlined classes added to (Figure 5(a)) or modified (Figure 5(b)) in the existing Fosstrak implementation to reduce the complexity of the diagram. A complete specification of Fosstrak is available from the developer guide[2]. Modifications are made to support the specific functions outlined in Figure 4 for sensor tag data: i) tag data extraction; ii) sensor data grouping; iii) sensor data filtering and aggregation; and iv) event cycle report. Since ID management functionality is already a part of Fosstrak, other aspects of our proposed architecture utilized existing Fosstrak capabilities.

A. Proposed Tag Data Model

Figure 5(c) illustrates the proposed tag data model (as an extension of Fosstrak tag data model) for representing sensor data. New boolean attribute *sensor* is introduced to the `Tag` class to indicate the containment of sensor data. All implementations of a sensor tag specific data models are left to the discretion of the user/developer and must extend the abstract class `SensorData`. The `SensorData` class defines the attribute *type* to identify the type of sensor data reported by a sensor tag (Figure 5c).

[2]https://code.google.com/p/fosstrak/wiki/AleDevGuideOverview

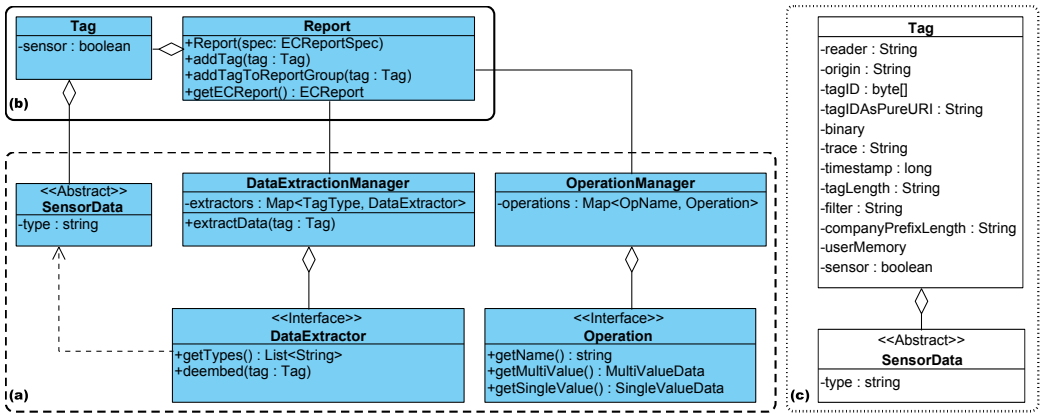


Fig. 5. Extensions: (a) new classes; (b) modified classes; (c) tag data model

B. Tag Data Extraction

`DataExtractionManager` and `DataExtractor` classes provide the framework for tag data extraction where sensor data is de-embedded from the tag identifier. `DataExtractionManager` act as a façade and delegate the processing of the tag to the appropriate `DataExtractor` (data extraction implementation relevant to the sensor data type, *eg:* acceleration data extractor for acceleration sensor data) which is identified by the tag type. The `DataExtractionManager` can obtain valid tag types (*eg:* `0D` and `0B` for acceleration tag) for an implementation of `DataExtractor` by calling `getTagTypes` method on the corresponding `DataExtractor` implementation. We have not specified data formats for various sensors and therefore it is possible to have multiple `DataExtractors` for the same sensor where the treatment of the sensor data field (Figure 3), is left to the discretion of the end-user and/or developer. Thus providing for a flexible and extensible design. Finally, `DataExtractionManager` converts a sensor tag EPC to a URI (Universal Resource Indicator) by only considering the tag ID portion of the EPC (see Fig. 3) to allow Fosstrak to continue to apply existing ID level filtering and aggregations operations to the ID data of sensor tags.

C. Sensor Data Grouping, Filtering and Aggregation

Unlike with ID data, a variety of filtering and aggregation operations can be performed on sensor data depending on application domain requirements and sensor type (*eg:* acceleration sensor). `Report` class was modified to implement sensor data grouping based on *logical readers* (behaviour of a reader from the perspective of a report subscriber) [11] and tag IDs. Sensor data filtering is managed by the `OperationManager` which is responsible for maintaining a list of implementations of `Operation` interface. Each implementation of `Operation` interface can encapsulate specific filtering or aggregation operation to be performed on the acquired sensor data. The operations specified in the **ECReportSpec** (see Fig. 7(b)) by subscribers are matched against the registered operation implementations using the operation name (see Fig. 5(a) `Operation` interface) to support sensor data filtering and aggregation requests.

D. Event Cycle Report

Sequence diagram in the Figure 6 visualizes the behaviour of an **EventCycle** (see ALE in [11] and Table I). During an **EventCycle** the `getECReports` method in the modified `Report` class is invoked to prepare a report according to the **ECReportSpec** specification (which is provided as a constructor argument to the `Report` class upon instantiation when subscribing to the report) and return an **ECReport** object.

Figure 7(a) shows extensions to the **ECReportSpec** data model to allow clients to subscribe to not only ID tag but also sensor tag data. The corresponding attribute descriptions are provided in Table II. Moreover, to enable users to specify sensor data filtering and aggregation operations with specific arguments, a new structure (**OperationType**) is introduced to the **ECReportSpec**. The unique *operationID* in **OperationType** allows the users to specify the same operation multiple times with different arguments and isolate each of the returned results without ambiguity (*eg:* first 10 FFT coefficients and last 10 FFT coefficients).

ALE standard [11] defines three kinds of report sets that client can subscribe to: i) CURRENT (tags in current **EventCycle**); ii) ADDITIONS (new tags in current **EventCycle**); and iii) DELETIONS (tags in previous **EventCycle** not in current **EventCycle**) for ID tag filtering. In light of sensor data, ADDITIONS and DELETIONS are meaningless and CURRENT is inadequate. Therefore new report set, SENSOR, is defined to specify filtering and aggregation operations on sensor data within the current **EventCycle** and these operations are evaluated only if they are specified along with the SENSOR

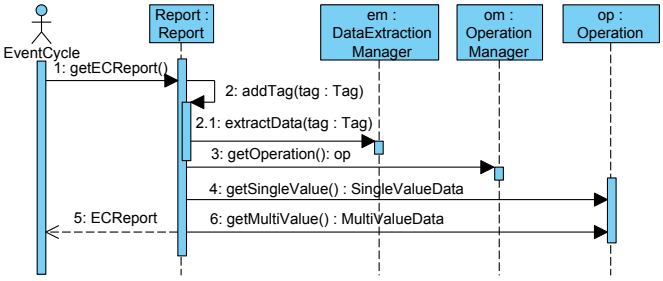
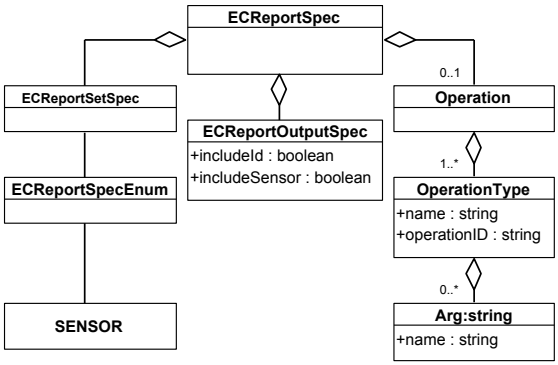
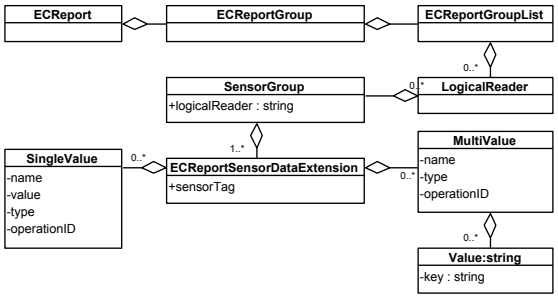


Fig. 6. Interaction with Fosstrak



(a) **ECReportSpec** data model extensions to specify sensor data operations



(b) **ECReport** data model extensions to report sensor data

Fig. 7. **ECReportSpec** and **ECReport** data model extensions (see Table II for attribute descriptions)

TABLE II. DESCRIPTIONS OF THE ATTRIBUTES IN EXTENDED **ECREPORTSPEC** AND **ECREPORT** DATA MODELS

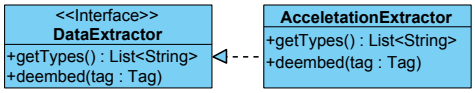
Element	Attribute	Description
ECReportOutputSpec	*includeId*	If *true* ID data of the ID tags are included
ECReportOutputSpec	*includeSensor*	If *true* ID data of the sensor tags are included
OperationType	*name*	Name of the operation which should corresponds to the implemented operation name
OperationType	*operationID*	Uniquely identifies the operation. Operation implementations return results with this ID
Arg	*name*	Name of the argument and is depend on the enclosing operation
SingleValue/MultiValue	*name*	The name distinguishes the multiple outputs from a single operation
SingleValue/MultiValue	*type*	Data type of the output (*eg:* float, double, int)
SingleValue/MultiValue	*operationID*	The *operationId* of the operation which produce the output
SingleValue	*value*	Value returned by the operation
Value	*key*	The key provides unique identification or meta-data for a specific value with in a MultiValue element.

report set.

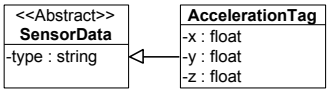
A new structure for reporting sensor data to subscribers (clients) is introduced to the **ECReport** while maintaining flexibility as well as extendibility to support future changes and multiple sensor types. Figure 7(b) shows the design of **ECReport** extensions where the attribute descriptions are provided in the Table II. Two data structures were defined to represent single-valued (**SingleValue**) and multi-valued (**MultiValue**) outputs from filtering and aggregation operations. Moreover, the optional attribute *key* specified in **Value** provides unique identification or meta-data related to the represented value. The value of the *key* is determined by the operation implementation (*eg:* for operations *FFT* and raw sensor data reporting the *key* can be the FFT component and timestamp of the reading respectively). Ability to represent the output of any operation is a significant advantage of this framework.

V. EXPERIMENTS AND RESULTS

Initially, the proposed design for WINDWare is implemented as specified in Section IV using Fosstrak-1.2.0 release. We implemented the `AccelerationExtractor` class which implements the `DataExtractor` interface and a tag data model for WISP tags with accelerometer sensors (Figure 10(a)). The sensor data, embedded in an EPC from a WISP, is stored in `AccelerationTag` which is assigned as sensor data to the corresponding `Tag` (see Figure 5(c)). Figure 10(b) shows the implementation.



(a) `AccelerationExtractor`



(b) Acceleration tag data model

Fig. 10. Acceleration tag implementation

We have implemented five concrete implementations of `Operation` interface to generate sum, average, raw sensor data representation, resultant acceleration and FFT (Fast Fourier Transform) for a sensor tag with an acceleration sensor that reports *x*, *y* and *z* acceleration components. We used **SingleValue** to report results from operations which return one value (such as sum and average operations). For operations which return multiple values, such as the FFT, **MultiValue** structure was used (Section IV-D).

We conducted two sets of experiments: i) with the real hardware; and ii) with an RFID emulator. Our experiments were designed to: i) evaluate ID tag filtering functionality (to confirm that existing Fosstrak functionality is not affected and that sensor tags are correctly processed by existing Fosstrak operations for ID tags); and ii) evaluate sensor data filtering and aggregation operations.

A. Experiments with a Hardware Setting

Our aim here is to test the overall functionality of WINDWare. We used an Impinj Speedway Revolution (R420-GX11M) reader with a circularly polarized antenna, two WISPs with acceleration sensors, and regular RFID tags. We specified two report sets (Figure 8(a)): i) CURRENT; and ii) SENSOR. In the CURRENT report set the WINDWare reported the ID data as expected in the original Fosstrak implementation and provided positive evidence that existing Fosstrak functionality is not affected by our extensions and additions (Figure 8(b)). Moreover, with the CURRENT report set, it was able to report the ID information of the sensor tags and confirm that the extensions to Fosstrak was successfully able to process the sensor tag data (see Figure 8(b)). When report set SENSOR is used, only the ID and sensor data related to sensor tags were reported (furthermore, the results of the sensor data filtering and aggregation operations were available, which is the expected behaviour) as shown in Figure 8(c). Results from the specification of both CURRENT and SENSOR report demonstrate the sensor tag data management capability of WINDWare.

```
<ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>... </logicalReaders>
  <boundarySpec>...</boundarySpec>
  <reportSpecs>
  <reportSpec reportName="CURRENT_Report">
    <reportSet set="CURRENT"/>
    <output includeRawHex="true" includeRawDecimal="true"
includeEPC="true" includeTag="true" includeSensor="true"
includeId="true"/>
   </reportSpec>
   <reportSpec reportName="SENSOR_Tags_Report">
    <reportSet set="SENSOR"/>
    <output includeRawHex="true" includeRawDecimal="true"
includeEPC="true" includeTag="true"/>
    <operations>
     <Operation name="Average" operationID="average">
      <arg name="total">8</arg>
     </Operation>
     <Operation name="Sum">
      <arg name="less than">0</arg>
     </Operation>
     <Operation name="FFT">
      <arg name="less than">50000</arg>
      <arg name="greater than">10000</arg>
     </Operation>
    </operations>
   </reportSpec>
   </reportSpecs>
</ns2:ECSpec>
```

Report set CURRENT

Report set SENSOR

(a) Report specification (**ECReportSpec**)

```
<report reportName="CURRENT_Report">
 <group>
  <groupList>
   <member>
    <epc>urn:epc:id:wisp:90044907186256000.11.63854</epc>
    <tag>urn:epc:tag:wisp-96:90044907186256000.11.63854</tag>
    <rawHex>urn:epc:raw:96.x3D013FE75DB2AB78800BF96E</rawHex>
    <rawDecimal>urn:epc:raw:96.188800963019596129613557251 66</rawDecimal>
   </member>
   <member>
    <epc>urn:epc:id:sgtin:38762147710.03.184649852029</epc>
    <tag>urn:epc:tag:sgtin-96:1.38762147710.03.184649852029</tag>
    <rawHex>urn:epc:raw:96.x302520CCEDEFC0EAFDFD247D</rawHex>
    <rawDecimal>urn:epc:raw:96.149001656227586818655878 60605</rawDecimal>
   </member>
   ...
  </groupList>
  ...
 </group>
</report>
```

ID data from sensor tag

ID data from ID tag

(b) **ECReport** for report set CURRENT in (a) with ID data from both ID tags and sensor tags

```
<report reportName="SENSOR_Tags_Report">
 <group>
  <groupList>
   <member>
    <epc>urn:epc:id:wisp:256866449905234665.11.35951</epc>
    <tag>urn:epc:tag:wisp-96:256866449905234665.11.35951</tag>
    <rawHex>urn:epc:raw:96.x3D039092ACDD5132E90B8C6F</rawHex>
    <rawDecimal>urn:epc:raw:96.188828951030152624937177 93903</rawDecimal>
   </member>
   <Sensor_Tag>
    <SensorGroup_LogicalReader="Reader1">
     <Sensor_TAG_Member_Sensor_Tag="1010000001011011">
      <Sensordata operationID="average" type="float" value="-246.98047" name="x"/>
      <Sensordata operationID="average" type="float" value="-6651.5625" name="y"/>
      <Sensordata operationID="average" type="float" value="1198.5156" name="z"/>
      <MultiValue operationID="dataStream" Type="float" name="x">
       <Value key="1374396036466">-246.98047</Value>
       <Value key="1374396036465">-246.98047</Value>
       ...
      </MultiValue>
     </Sensor_TAG_Member>
    </SensorGroup>
    </Sensor_Tag>
   </groupList>
  ...
  </group>
</report>
```

ID data from sensor tag

Sensor data from sensor tags

Single valued output

Multi-valued output

(c) **ECReport** for report set SENSOR in (a) with ID and sensor data from sensor tags

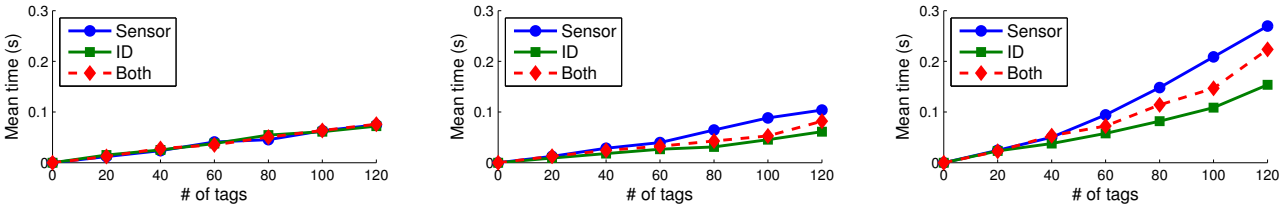
Fig. 8. Hardware experiment subscription and reporting

B. Experiments with a Reader Emulator

Rifidi Emulator[3] has the capability to emulate: i) LLRP protocol supported RFID readers; and ii) multiple EPC tags and tag types. Therefore, Rifidi Emulator is employed to emulate the functionality of one LLRP supported reader and multiple ID and sensor tags required for the experiment. Three report sets were specified: i) CURRENT and SENSOR; ii) CURRENT; and iii) SENSOR. In all report specifications, report event cycle is specified as two seconds based a near real-time monitoring application context (see section VI). All implemented sensor filtering operations were specified in subscription arguments for the SENSOR report set. Tests were carried out having: i) only ID tags; ii) only sensor tags; and iii) a mixture of ID and sensor tags (1:1 ratio for an unbiased assessment) to evaluate the middleware against each tag type. Having only the ID tags provide a baseline for comparing our middleware implementation as ID tags are processed by the existing ALE implementation in Fosstrak. We recorded the time taken to generate the reports (duration of `getECReportMethod` in **EventCycle**) specified in the subscribed **ECReportSpec**. The emulated reader was configured with the following LLRP settings: i) for **ROBoundarySpec**, **ROSpecStartTriggerType** (which defines the trigger event for the reader to initiate generation of reports according to the defined **ROSpec**) was specified as *periodic* with the period of *5000* ms; and ii) for reporting tags **ROReportSpec** (see Table I) was configured to trigger upon each tag detection or end of **ROSpec**.

Figure 9 depicts the results of the emulated test. Figure 9(a) shows the behaviour of the middleware for report set CUR-

[3]http://www.rifidi.org/



(a) Report set CURRENT (b) Report set SENSOR (c) Report sets CURRENT and SENSOR

Fig. 9. Report sets from the experimental results with Rifidi Emulator

RENT specified in **ECReprtSpec** which reported only the ID data for both ID tags and sensor tags. No difference in the mean time between sensor and ID tags are observed.

Graph in Figure 9(b) depicts the report generation time when report set SENSOR (only generate reports for sensor tags) is specified. It is observed that sensor operations consumed linearly increasing time with respect to sensor tags in the field of view. Where both ID and sensor tags are in the field, a report is only generated for 50% of the tags seen by the reader (as ID tags are filtered out from the SENSOR report set) and hence time measured reflects the time taken to process the sensor tags and filter ID tags.

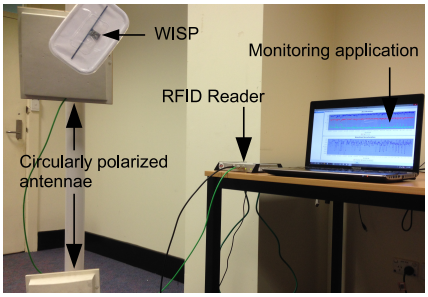
Figure 9(c) illustrates the mean report generation time when report set, CURRENT and SENSOR are requested. The SENSOR report set is requested with all the implemented filtering operations. The resulting report contained, two separate report sections (CURRENT report set and SENSOR report set). The results are similar to 9(b) but now time taken to process ID tags are also included.

VI. Example Application

To demonstrate our middleware, we developed a prototype monitoring application to identify potential damages to goods (eg. miss-handling of fragile items such as LCD screens) in a supply chain. The damage monitoring scenario is performed in a laboratory environment using Impinj Speedway Revolution (R420-GX11M) reader with two circularly polarized antennae as shown in Figure 11(a). The items being monitored are attached with WISP tags having a 3D acceleration sensor.

Identification of potential damages are two fold: i) an item dropping on to the ground is identified by detecting the free fall (where the resultant acceleration approaches zero); and ii) a damage due to a high impact or a shock is identified by high resultant acceleration. In order to receive notifications on potentially damaged items, **ECReportSpec** depicted in Figure 11(b) is used, where the *Resultant Acceleration* operation with `operationID` *threshold* is specified with two arguments to detect free fall (`less-than` 0.5) and high impact (`greater-than` 1.5). We have also subscribed to the raw sensor data streams (`operationID`: *dataStream*) and resultant acceleration (`operationID`: *ra*) to validate the potentially damage event reported by the *threshold* operation.

The Figure 11(c) shows a screenshot of the developed monitoring application. This application subscribes to WINDWare using the defined **ECReportSpec** (Figure 11(b)). Items are dropped on to the ground to simulate free fall and high impact (upon collision with the ground). In 11(c), we have



(a) Damage monitoring scenario equipment set-up

```
<operations>
  <Operation name="Data" operationID="dataStream"/>
  <Operation name="ResultantAcceleration" operationID="ra"/>
  <Operation name="ResultantAcceleration"
     operationID="threshold">
    <arg name="greater_than">1.5</arg>
    <arg name="less_than">0.5</arg>
  </Operation>
</operations>
```

Specification to evaluate resultant acceleration and filter to select potential damage events

(b) **ECReportSpec** for damage monitoring application

(c) Screenshot of the prototyped damage monitoring application

Fig. 11. An application of WINDWare for real-time monitoring

highlighted (plotted in red) the results reported by the filter operation to select potential damage events based on evaluating the resultant acceleration from the WISP attached to the item. By only subscribing to the operation with `operationID` *threshold*, events of interest (free fall and shock) are received. Therefore, use of sensor data management middleware such as WINDWare: i) reduces the complexity of client applications by allowing application developers to focus on business logic; ii) allows development of applications that are agnostic to underling sensing infrastructure; and iii) reduces network traffic from large volume sensor data streams.

VII. Conclusion and Future Work

In this paper we present the design, implementation[4] and evaluation of WINDWare (**W**ireless **I**dentification a**n**d Sensor **D**ata Management Middle**ware**), a middleware for passive sensor enabled RFID tags for the first time (to the best of knowledge). In particular, our middleware provides: i) extracting sensor and ID data; ii) operations on sensor data (filtering and aggregation); and iii) sensor data subscriptions and reporting. Our generic middleware architecture is implemented by extending Fosstrak. Furthermore, WINDWare adheres to the standardized EPCglobal architecture through implementing the ALE specification and using the extensions provisioned by standards. Finally, the real world applicability of WINDWare is demonstrated through a prototype application.

Passive sensor enabled RFID data streams are unique; each sensor event consists of both ID data and sensor data. The proposed middleware processes the ID data and sensor data in unison maintaining the important relationship between sensor data and its source. In contrast, other sensor data supported middleware lacks the capability to associate ID with sensor data, thus report sensor data and ID data separately [22], [23]. This association is paramount, particularly in real-time applications such as human activity recognition [7].

Currently reporting measurement units capability of sensor data is not provided. The ability to report measurements is seen as important in terms of interoperability and sharing of information. However, this limitation can be overcome by implementing unit conversion at the operation level and specifying the required unit of the output as an operation argument in **ECReportSpec**. Furthermore, the 1 byte tag type proposed, which only allows 256 tag types, may not be adequate to encode the tag capabilities in the future with the increasing numbers of embedded sensor types.

In future, we will include filtering and aggregation support for other sensor types and investigate the use of an efficient tag capability lookup mechanism to address the above mentioned limitations. Moreover, measurement units reporting ability will be incorporated to the data model to increase the interoperability and facilitate information sharing. Furthermore, creation of an open source project for the WINDWare, will be considered.

Acknowledgments

This research was supported by a grant from the Australian Research Council (DP130104614).

References

[1] M. Buettner, B. Greenstein, A. Sample, J. R. Smith, and D. Wetherall, "Revisiting smart dust with RFID sensor networks," in *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, 2008.

[2] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, "Design of an RFID-based battery-free programmable sensing platform," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 11, pp. 2608–2615, 2008.

[3] A. Ruhanen, M. Hanhikorpi, F. Bertuccelli, A. Colonna, W. Malik, D. Ranasinghe, T. S. Lopez, N. Yan, and M. Tavilampi, "Sensor-enabled RFID tag handbook," 2008.

[4] Smith, Joshua R, Ed., *Wirelessly Powered Sensor Networks and Computational RFID*. New York: Springer, 2013.

[4] WINDWare is available from:https://windware.googlecode.com

[5] K. Eom, C. W. Lee, N. T. Van, K. K. Jung, J. W. Kim, and W. S. Choi, "Food poisoning prevention monitoring system based on the smart RFID tag system," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, pp. 213–222, 2013.

[6] A. Dementyev and J. R. Smith, "A wearable UHF RFID-based EEG system," in *IEEE International Conference on RFID (RFID)*, 2013, pp. 1–7.

[7] D. C. Ranasinghe, R. L. Shinmoto Torres, K. Hill, and R. Visvanathan, "Low cost and batteryless sensor-enabled radio frequency identification tag based approaches to identify patient bed entry and exit posture transitions," *Gait & Posture*, vol. 39, pp. 118–123, 2014.

[8] R. L. Shinmoto Torres, D. C. Ranasinghe, Q. Shi, and A. P. Sample, "Sensor enabled wearable RFID technology for mitigating the risk of falls near beds," in *IEEE International Conference on RFID (RFID)*, 2013, pp. 191–198.

[9] V. Talla, M. Buettner, D. Wetherall, and J. Smith, "Hybrid analog-digital backscatter platform for high data rate, battery-free sensing," in *IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNet)*, 2013, pp. 1–3.

[10] T. S. López, D. C. Ranasinghe, B. Patkai, and D. McFarlane, "Taxonomy, technology and applications of smart objects," *Information Systems Frontiers*, vol. 13, no. 2, pp. 281–300, 2011.

[11] EPCglobal Inc, "Epcglobal ratified standards." [Online]. Available: http://www.gs1.org/gsmp/kc/epcglobal/

[12] D. C. Ranasinghe, K. S. Leong, M. Ng, D. W. Engels, and P. H. Cole, "A distributed architecture for a ubiquitous RFID sensing network," in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2005, pp. 7–12.

[13] Y. Igarashi, K. Miyazaki, Y. Sato, and J. Mitsugi, "A network architecture for fast retrieval of user memory data from sensor RF tags," in *2013 IEEE International Conference on RFID (RFID)*, pp. 184–190.

[14] Z. Cheng, X. Zhang, Y. Dai, and Y. Lu, "Design techniques of low-power embedded EEPROM for passive RFID tag," *Analog Integrated Circuits and Signal Processing*, vol. 74, no. 3, pp. 585–589, 2013.

[15] C. A. Balanis, *Antenna Theory: Analysis and Design*. John Wiley & Sons.

[16] X. Su, C.-C. Chu, B. S. Prabhu, and R. Gadh, "On the identification device management and data capture via WinRFID1 edge-server," *Systems Journal, IEEE*, vol. 1, no. 2, pp. 95–104, 2007.

[17] Fosstrak, "Fosstrak - open source RFID platform - google project hosting," 2013. [Online]. Available: https://code.google.com/p/fosstrak/

[18] Oracle, "Data sheet: Oracle warehouse management," 2014. [Online]. Available: http://www.oracle.com/us/products/applications/ebusiness/054354.pdf

[19] IBM, "IBM - WebSphere sensor events - united states," Feb. 2014. [Online]. Available: http://www-03.ibm.com/software/products/en/webssenseven/

[20] SAP, "SAP - SAP auto-ID infrastructure." [Online]. Available: http://www.sap.com/platform/netweaver/autoidinfrastructure.epx

[21] D. Bade and W. Lamersdorf, "An agent-based event processing middleware for sensor networks and RFID systems," *The Computer Journal*, vol. 54, no. 3, pp. 321–331, 2011.

[22] K. Gama, L. Touseau, and D. Donsez, "Combining heterogeneous service technologies for building an internet of things middleware," *Computer Communications*, vol. 35, no. 4, pp. 405–417, 2012.

[23] W. Wang, J. Sung, and D. Kim, "Complex event processing in EPC sensor network middleware for both RFID and WSN," in *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 165–169.

[24] I. Abad, C. Cerrada, J. A. Cerrada, R. Heradio, and E. Valero, "Managing RFID sensors networks with a general purpose RFID middleware," *Sensors*, vol. 12, no. 6, pp. 7719–7737, 2012.

[25] K. Dutta, K. Ramamritham, B. Karthik, and K. Laddhad, "Real-time event handling in an RFID middleware system," in *Databases in Networked Information Systems*. Springer, 2007, pp. 232–251.

[26] C. Floerkemeier, C. Roduner, and M. Lampe, "Rfid application development with the Accada middleware platform," *Systems Journal, IEEE*, vol. 1, no. 2, pp. 82–94, 2007.