DeftRFID: A Lightweight and Distributed RFID Middleware

Yingliang Lu [#1], Weifeng Zhang [*2], Zengchang Qin [*], Yao Meng [#], Hao Yu [#]

[#] *Fujitsu R&D Center CO., LTD.*
13/F, Tower A, Ocean International Center, No.56 Dong Si Huan Zhong Road, Chaoyang District, Beijing, 100025, P.R. China
[1] luyl@cn.fujitsu.com

[*] *School of Automation Science and Electrical Engineering, Beijing University of Aeronautics & Astronautics*
No. 37 Xueyuan Road, Haidian District, Beijing, 100191, P.R. China
[2] zwf@asee.buaa.edu.cn

Abstract **Radio Frequency Identification (RFID) has become a popular identification technology and the RFID application market is undergoing explosive development. A successful RFID application requires a dedicated middleware to maximize the benefits of RFID technology. *DeftRFID* proposed in this paper is a distributed, lightweight, and scalable RFID middleware. Compared with other existing RFID middleware, *DeftRFID* consists of three main layers which can be distributed across multiple machines, and provides friendly rules management interface to application developers. It also can be applied to other kind of sensor networks. Abundant functions of *DeftRFID* include: device management, data filtering, aggregation, transformation, dissemination, and functional Software Development Kit. Also, a Laundry Visual Management System (LVMS) was developed in this paper to show the utility of the middleware we proposed. It is proved that *DeftRFID* has high practical value.**

I. INTRODUCTION

Recent years Radio Frequency Identification (RFID) has developed successfully [1], especially in supply chain management (SCM) [2], and is considered to be dominant in the identification technology in the near future. It attracts lot of investments from governments and a number of enterprises for its vast application prospects and inviting profit. In traditional RFID applications such as access control, there was one-to-one relationship [3] between reader and application and therefore there was barely a need for RFID middleware. However in the novel RFID applications such as SCM, a number of readers need to be deployed to capture variety of data. Hence, RFID middleware has become the key component in developing RFID applications because of its plentiful functions including Reader and device management, Data management, Process management and Application development. The complete RFID system architecture with a RFID middleware is shown in Fig 1.

In this paper we present a novel lightweight RFID middleware named *DeftRFID* which has the ability to perform all the functions discussed above. *DeftRFID* is a modular and layered design which makes it flexible and expandable. The main three layers include: Application Interface Layer, Data Processing Layer, and Hardware Abstraction Layer. *DeftRFID* middleware bridges the gap between low-level sensor technology and high-level enterprise applications. It can translate the primitive information such as location and the time of sensing emanating from RFID sensors into meaningful, actionable information (e.g., out-of-stocks) which are needed for high-level applications. In order to evaluate the performance of *DeftRFID*, we build a Laundry Visual Management System (LVMS) based on Delphi.

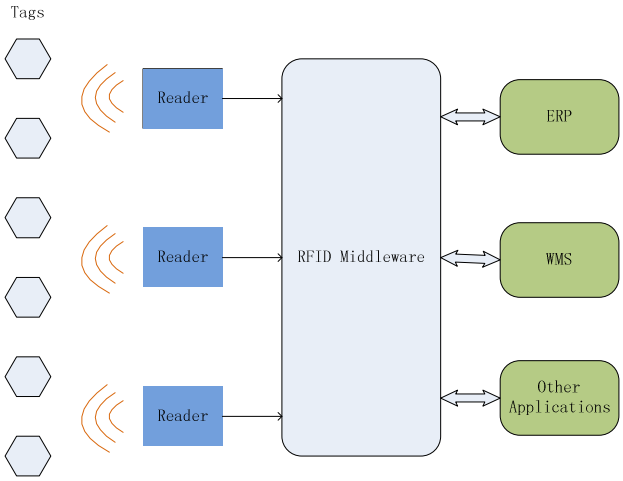

Fig 1. RFID system architecture

The rest of this paper is organized as follows: In section II the related work in RFID middleware development are discussed. The comprehensive architecture of *DeftRFID* is shown in section III and we give detailed introduction of all the components. After that the Laundry Visual Management System based on *DeftRFID* is built in section IV and several experiments are done. Section V gives our conclusion and prospect of the future work.

II. RELATED WORK

RFID middleware design research has attracted a few researchers and many important works have been proposed. The most related work to our research is *FlexRFID* [4] [5] proposed by Ajana. *FlexRFID* is a simple and smart RFID middleware which can manage and monitor RFID readers or other types of sensing devices, as well as process dynamically generated high volumes of noisy RFID data. *FlexRFID* is organized as a four-tier architecture consisting of application abstraction layer (AAL), business event & data processing layer (BEDPL), business rules layer (BRL),

and device abstraction layer (DAL). The Device Abstraction Layer of the *FlexRFID* middleware is responsible for interaction with various devices and data sources independent of their characteristics. The BEDPL acts as a mediator between the DAL and the AAL. Services provided by the BEDPL are data dissemination, data aggregation, data transformation, data filtering, duplicate removal, and data replacement. While *FlexRFID* provides all data processing capabilities along with the security and privacy features included in the data processing layer, it still need to be extended to support EPC standard. Also a complete system using various devices has not been developed.

Auto-ID Center has developed a suite of software named *Savant* [6] [7] for free. *Savant* has the ability to collect, accumulate and process Electronic Product Code (EPC) [8] data obtained from several RFID readers. *Savant* has a hierarchical architecture: Event Management System (EMS), Real-time In-memory Data Structure (RIED), and Task Management System (TMS). While *Savant* possesses many advantages such as processing massive flow on information but reduced network traffic, it is in lack of functionality for addressing business rules management, dealing with all types of sensor devices and providing data dissemination, filtering, and aggregation.

Another important RFID middleware is *MDI-SMURF* first proposed by Jeffery [9] [10] [11]. *MDI-SMURF* is an RFID middleware platform organized as a pipeline of processing stages with an associated uncertainty-tracking shadow pipeline. It aims to realize the Metaphysical Data Independence (MDI), a layer of independence that shields applications from the challenges that arise when interacting directly with sensor devices. In *MDI-SMURF*, data from readers flow into *Temporal-SMURF*, a smoothing filter that uses its statistical framework to correct for dropped readings common in RFID data streams. These cleaned readings are then streamed into *Spatial-SMURF*, a module that extends Temporal-SMURF's statistical framework to address errors and semantic issues that arise from multiple RFID readers deployed in close proximity. Finally a simple translation module converts the temporally and spatially cleaned readings to MDI readings by the vector representation. The principle contribution of *MDI-SMURF* is that it incorporates a novel statistical framework which enables it to continually and adaptively correct for the temporal and spatial errors associated with RFID data and produce data corresponding to the MDI interface. However, *MDI-SMURF* middleware provides no approach to defining rules for end-users.

The attractive market prospects of RFID applications naturally attract many "gold". A number of prestigious companies have developed their own RFID middleware products such as *Microsoft BizTalk RFID* [12], *Oracle Fusion* [13], and *Sun RFID Middleware* [14].

The above middleware respectively have their own fine features and defects. Literature [15] shows there are still many open issues after analysing the application requirements and RFID constraints. There is a conflict between the increase of readers and the real-time processing ability. Flexibility, reliability, and privacy protection also attract more and more attention. In addition, the information captured by a reader is usually of interest not only to a single application, but to a diverse set of applications across an organization and its business partners. Hence, different latencies need to be supported, since the desired notification latency depends upon the applications.

The main motivation of designing *DeftRFID* is to provide a lightweight and distributed middleware. Compared with the above middleware, the distinguishing characteristics of the proposed middleware are: (i) *DeftRFID* middleware adopts distributed architecture which makes it flexible, portable, lightweight, and free to expand. (ii) The rules used to do data aggregation and data transformation can be easily defined by user by the form of IF-THEN representation. This means the rule base in our RFID middleware could be flexibly made according to the specific application. (iii) *DeftRFID* provides a friendly and functional application program interface to facilitate the application developers. These superiorities make *DeftRFID* especially fit for developing small-scale and low-cost enterprise applications.

III. DEFTRFID MIDDLEWARE ARCHITECTURE

The *DeftRFID* middleware design provides the applications a device neutral, easy-to-use interface. It consists of three layers: Application Interface Layer (AIL), Data Processing Layer (DPL), and Hardware Abstraction Layer (HAL). By using distributed architecture design, the three layers of the *DeftRFID* middleware can separately run on different machines and communicate with each other through TCP sockets [16]. A diverse set of applications across an organization are interested in the captured information and different applications have different latency requirement. While other present RFID middleware such as *FlexRFID* mentioned in section II broadcast the captured data with different latencies to deal with this problem, *DeftRFID* adopts the strategy that it transmit data to applications only when the applications send a data query command. Fig 2 is a topological graph of typical application based on the *DeftRFID* middleware.

This distributed architecture provides possibility to build large scale sensor networks. On one hand, every layer of the middleware concentrates on its own responsibilities. On the other hand three layers complete the tasks by mutual cooperation. This architecture provides at least four advantages: Firstly, users can be geographically separate which is important for large corporations, where applications based on company-wide data are in different locations. Secondly, using multiple machines can improve performance and scalability. It significantly promotes the processing capability of the system. Thirdly, distributed architecture facilitates the modular system design. Hardware devices and software services can be added as modules with little efforts. Finally, this kind architecture can reduce maintenance costs. In distributed system, the layers communicate with each other through interface and do not need to know how the internal structure is implemented. The effect of this separation is that any changes to layer's implementation do not affect its interface. This allows unthinkable flexibility.

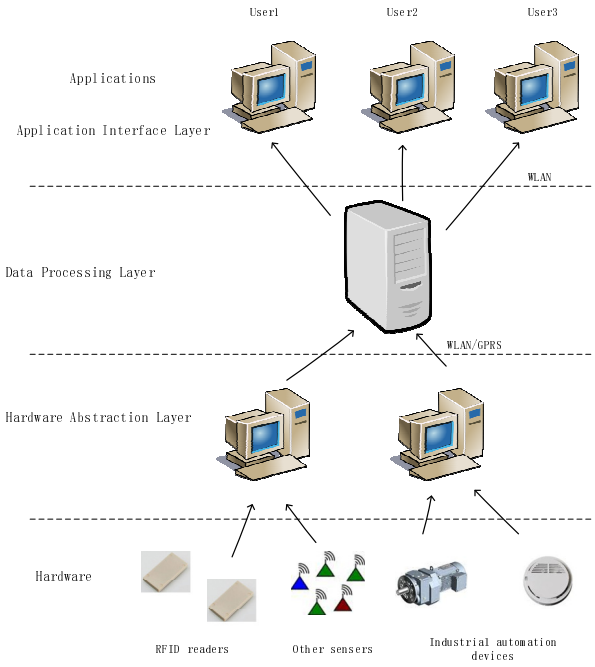

Fig 2. Typical topological graph of DeftRFID

In the following, we will focus on the implementation details of the three layers as shown in Fig 3.

A. *Hardware Abstraction Layer (HAL)*

The Hardware Abstraction Layer of *DeftRFID* middleware is responsible for concealing the complexity of diverse hardware devices. As shown in Fig 2, this layer can be distributed on multiple parallel machines. The main services provided by HAL are described as follows:

Diverse devices management: DeftRFID middleware support not only RFID sensors but also other sensors and industrial automation devices (e.g., motor, alarm). Devices can be added or deleted by user and the middleware provides one management thread for every device respectively. Hence, users can easily configure every device by sending device configuration command to these threads. *DeftRFID* also support variety of interface including USB, serial port, and Ethernet port.

Low-level functions: the HAL provides basic functions including activating / shutting down devices, reading tag data, writing data to tags, etc.

Duplicate removal: In traditional RFID middleware design, this function is usually put in the upper layer such as data processing layer. However, *DeftRFID* moves this module into the HAL to reduce the data stream flowed into upper layer so that the network load can be relieved. It is significantly valuable when the number of sensors increases briskly.

B. *Data Processing Layer (DPL)*

The Data Processing Layer (DPL), the core of *DeftRFID* middleware, acts as a mediator between the AIL and the HAL. The DPL provides a number of important services: data aggregation, data transformation, data filtering, data dissemination, data storage & query, and order transmission.

The procedure of data processing is described as follows: The Data storage & query module is responsible for storing and retrieving data. Here the data flowed from the HAL is first stored in the data cache and then flow into the data base. Also the data cache is inquired first when executing data query. The effectiveness of this measure will be tested in section IV. The Data filtering module extracts the most useful subsets of data. The filtered data has implicit meanings and associated relationships with other data, and need to be aggregated into summaries or proper inferences for applications. This service is provided by the Data aggregation module. Step further, the Data transformation module transforms the data into business events and deal with these events according to the rules stored in the rule base. The Data dissemination module takes charge of disseminating data to upper layer. And the Order Transmission module has tow functions. One is to tell the HAL how to deal with the events detected by the Data Transformation module. The other is sending the HAL the orders such as reader's basic information request and motor controlling commands from the AIL.

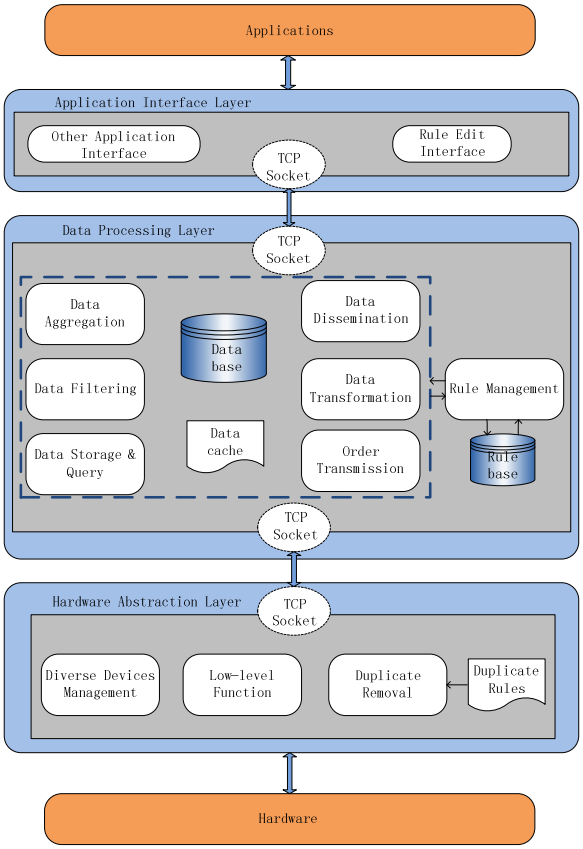

Fig 3. DeftRRFID middleware architecture

For the Rule management module makes our middleware distinguishing, we focus on it in the following.

Considering different kinds of applications using *DeftRFID* middleware may need to define rules to detect events and process them using the services provided by the middleware, we design the Rule Management module, an important component, by which users can add or delete rules easily. In *DeftRFID* all events and rules are required to be defined in a standard form. First we give the formal definition of event that is similar to the description in the literature [17] [18] [19].

Definition 1: *(Primitive event). PES={ PE_1, PE_2, … , PE_n} is a set of primitive events. Any primitive event has the following form:*

<ID , location, time, data >

where ID is the identification of a device, "time", "location", and "data" are respectively the time stamp, location parameters of the event, and the corresponding data .

For example, primitive event "*<T1, R1, 5, D>*" means tag T1 is observed by read R1 when time 5. Primitive events are the basic or core events turn from captured data. Based on primitive events, complex event can be defined recursively.

Definition 2: *(Complex event). Complex event (CES) is a sequence of events recursively defined based on primitive events using four underlying operations "and (&)", "or (|)", "not (~)", "followed ($\longrightarrow$)".*

(i) $E \in PES : E \in CES$

(ii) If θ, $\alpha \in CES$, then $\tilde{}\theta$, $\theta \& \alpha$, $\theta \mid \alpha$, $\theta \longrightarrow \alpha \in CES$

Using the above two definitions, users can define their own events flexibly according to need. For instance event "$(\theta \mid \alpha) \longrightarrow \omega$" means any of events θ and α occurs, followed by occurrence of event ω. In the rest of this paper both primitive events and complex events are abbreviated to events.

In our definition of rules, there are another two important components we call them *constraint* and *response*. "*constraint*" constrains the four elements of a event while "*response*" means how the middleware deal with the events after detection. A vivid example of rules combine these three components together can be seen in next section.

The rules can be defined according to use. For example we define *data transformation rules* to guide the Data transformation module. The Rule management module of DPL collects the events of the rules in rule base. These events are sent to the Data transformation module which is responsible for trying to extract such kinds of events from captured information. Finally once the events match, the corresponding responses are returned to the Order transmission module. Also, we can define filtering rules which will be used in the Data filtering module. These two kinds of rules are only examples. Actually, the kinds of rules are not confined to these examples. This mechanism endows the *DeftRFID* middleware with great flexibility and usability.

To prevent conflicts, once user add a new rule to the rule base, the Rule Management module will check whether conflicting rules already exist in the rule base and reject the new rule when conflict present.

C. Application Interface Layer (AIL)

The Application Interface Layer of *DeftRFID* provides friendly and functional interface to application developers by Dynamic Link Library (DLL). This layer provides the possibility to build lightweight and portable enterprise applications.

IV. LAUNDRY VISUAL MANAGEMENT SYSTEM

In order to illustrate the value and maturity of the *DeftRFID* middleware, we build a Laundry Visual Management System (LVMS) based on *DeftRFID*. The LVMS is designed using Delphi. The hardware used in testing consists of continuous current motors, Fujitsu RFID reader TFU-RW311, Fujitsu RFID tag WT-A511 [20] which is an enhanced UHF washable tag featuring downsized dimensions and heat-sealing capability, and other necessary automation devices.

Garment in the laundry are attached with tags so that they can be tracked. Combining the *DeftRFID* middleware and Fujitsu UHF tag technology, laundries will greatly improve receiving, shipping, and tracking garment while keep the cost low by improving workflow and efficiency. Fig 4 shows the diagram of the LVMS.

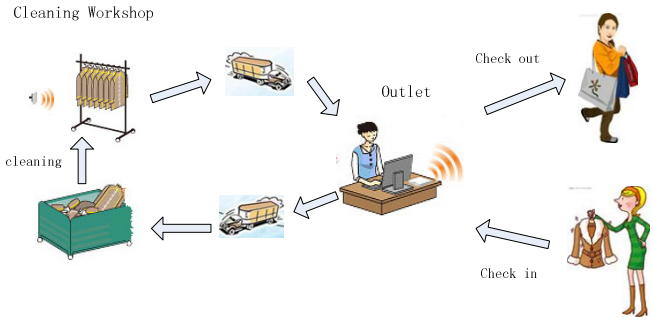

Fig 4. Overview of the LVMS

The core of the LVMS is *DeftRFID*. Based on this middleware abundant applications are developed, including Outlet Visual Management (OVM), Cleaning Workshop Visual Management (CWVM), Process Visualization (PV). Distribute architecture and independence of other run time library make LVMS lightweight and easy to be deployed. The OVM module is responsible for managing the garment in the outlet such as recording the information of customers when check in while deleting the information when check out, listing information of all the garment, etc. The service from CWVM is mainly tracking the garment in the process of cleaning.

Using the interface provided by *DeftRFID*, LVMS can define rules with little effort. As an example, one of the rules which is used in LVMS is that the motor which drives the conveyor belt should be stopped when a garment is detected twice within a certain period of time. Using the rule definition introduced in section III, this rule can be formally expressed as follows.

EVENT: E1 $\longrightarrow$ E2
CONSTRAINT: E1.ID=E2.ID
&
E1.location=E2.location=CONVEYOR
& 50sec<E2.time-E1.time<60sec
RESPONSE: STOP MOTOR
RULE: if EVENT, CONSTRAINT, then RESPONSE

This is only a tip of the iceberg. The *DeftRFID* middleware supports flexible rule definition.

Also we test the response time of *DeftRFID*. The response time is the time taken by *DeftRFID* to execute an API when requested by a client application. In our middleware, the data exchange though network between layers may become the obstacle to improve the middleware's response time. For this reason, we add a Data cache into the DPL and moved some data processing

modules such as duplicate removal to the Hardware Abstraction Layer (HAL). To prove the effectiveness of these measures, we compared the performance of *DeftRFID* and that with all data processing modules in DPL. The API used here is REFRESH_LIST, which returns all the tag IDs and the corresponding customer information of the current garment in a cleaning workshop.

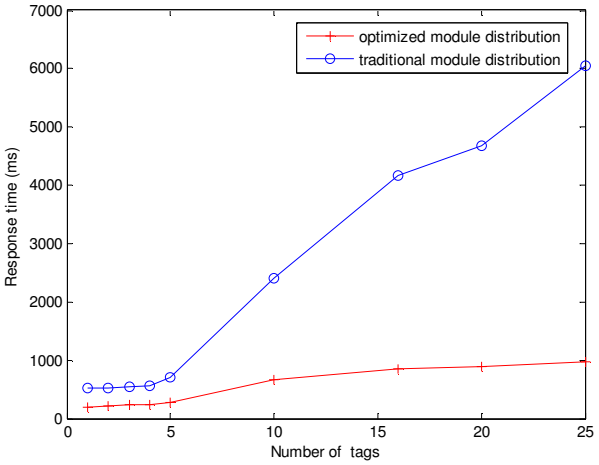

Fig 5. Response time

Fig 5 shows that when the number of tags in the networks increased, the response time of the middleware which optimized module distribution rose much more slowly than that with traditional module assigning. It is proved that distributing data processing modules in reason is an effective method to improve the performance of *DeftRFID* middleware.

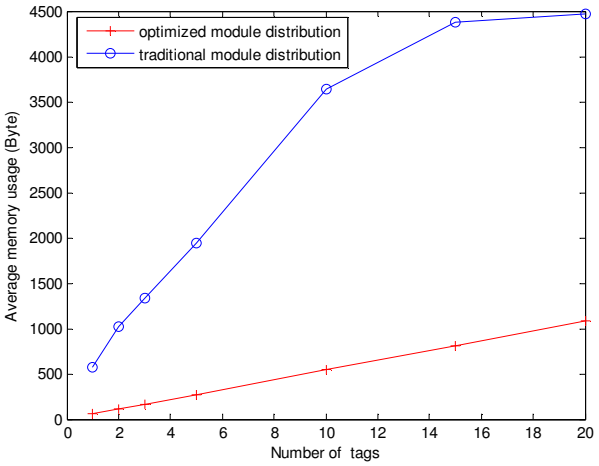

Fig 6. Average memory usage

Optimized distribution of data processing modules can also effectively reduce the memory usage of the central server in *DeftRFID* as shown in Fig 6. We tested average memory utilized by *DeftRFID* using the REFRESH_LIST. In the experiment, we called the REFRESH_LIST API ten times, calculated the average memory usage of the central server and gave the comparison between the memory usage when the module distribution was optimized and that with traditional module distribution.

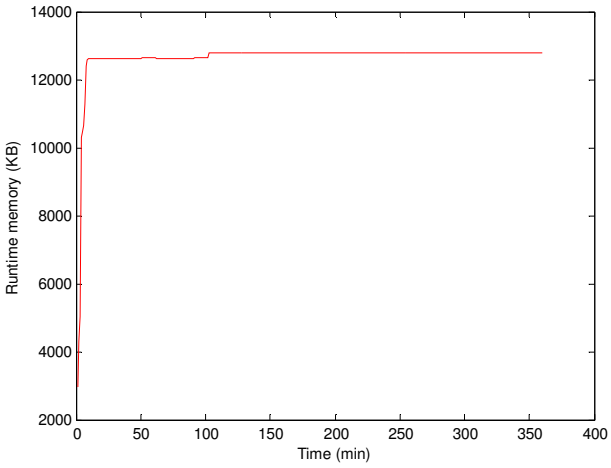

Fig 7. Runtime memory usage of HAL

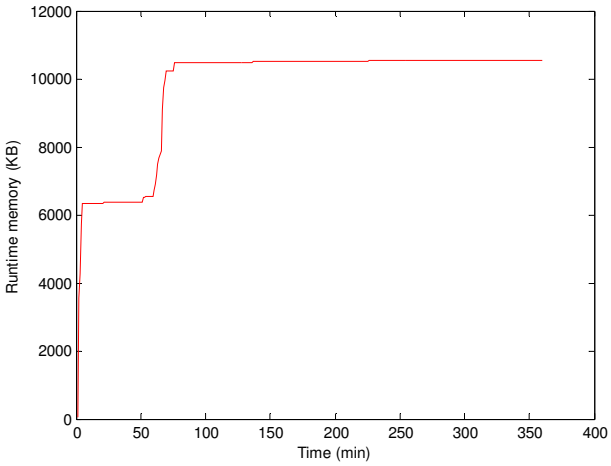

Fig 8. Runtime memory usage of DPL

We also did an experiment to examine the runtime memory usage of *DeftRFID* over a period of 6 hours. In this experiment, 2 different clients were connected to *DeftRFID* simultaneously and the number of tags increased with time from zero to ten. The above graphs Fig 7 and Fig 8 respectively represent the runtime memory usage of HAL and that of DPL which demonstrate that the *DeftRFID* middleware is lightweight.

V. Conclusion and Future Work

In this paper, we presented the design of *DeftRFID*, a lightweight and distributed RFID middleware. *DeftRFID* middleware has three important layers: the Hardware Abstraction Layer (HAL), the Data Processing Layer (DPL), and the Application Interface Layer (AIL). As one important characteristic of *DeftRFID*, a distributed architecture is adopted, so that these three layers can be deployed on multiple machines. It makes *DeftRFID* especially fit for the applications where both devices and end-users are separated geographically. *DeftRFID* provides ample functionalities including not only the common functions such as data filtering, transformation, aggregation, dissemination and device management, but also functional Software Development Kit (SDK) and friendly rule management interface. Finally we developed Laundry

Visual Management System (LVMS) to demonstrate the functionality of *DeftRFID* in real business system.

In the future work, the following can be done to improve the performance of *DeftRFID*: develop additional HAL device drivers to increase the range of supported devices; construct a highly compact RFID data by incorporating efficient data compression algorithms to reduce the load of central server and relieve the traffic of networks.

REFERENCES

[1] S. Polniak, *The RFID Case Study Book: RFID Application Stories from Around the Globe*, Abhisam Software, 2007.

[2] Q. Sheng, X. Li, and S. Zeadally, "Enabling Next-Generation RFID Applications: Solutions and Challenges," IEEE Computer, Vol. 41, No. 9, Sep. 2008.

[3] C. Floerkemeier, C. Roduner, and M. Lampe, "RFID Application Development with the Accada middleware Platform," IEEE Systems Journal, Vol. 1, No. 2, pp. 82-94, Dec. 2007.

[4] M. E. Ajana, H. Harroud, M. Boulmalf, and H. Hamam, "FlexRFID: A Flexible Middleware for RFID Applications Development," in: *Proc. WOCN'09*, pp. 1-5.

[5] M. E. Ajana, M. Boulmalf, H. Harroud, and H. Hamam, "A Policy Based Event Management Middleware for Implementing RFID Applications," in: *Proc. WIMOB'09*, pp. 406-410.

[6] T. Ishikawa, Y. Yumoto, M. Kurata, M. Endo, S. Kinoshita, F. Hoshino, S. Yagi, and M. Nomachi, "Applying Auto-ID to the Japanese Publication Business to Deliver Advanced Supply Chain Management, Innovative Retail Applications, and Convenient and Safe Reader Services," Auto-ID Center, Keio University, Oct. 2003.

[7] D. J. Glasser, K. W. Goodman, and N. G. Einspruch, "Chips, Tags and Scanners : Ethical Challenges for Radio Frequency Identification," Ethics and Information Technology, Vol. 9, No. 2, pp. 101-109, Jul. 2007.

[8] (2010) The EPC Global website. [Online]. Available: http://www.epcglobalinc.org/

[9] S.R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, "A Pipelined Framework for Online Cleaning of Sensor Data Streams," in: *Proc. ICDE'06*, pp. 140-145.

[10] S. R. Jeffery, G.Alonso, M. J. Franklin, W. Hong, and J. Widom, "Declarative support for sensor data cleaning," Lecture Notes in Computer Science In 4th International Conference on Pervasive Computing, Vol. 3968, 2006, pp. 83-100.

[11] S. R. Jeffery, M. Garofalakis, and M. J. Franklin, "Adaptive cleaning for RFID data streams," in: *Proc. VLDB'06*, pp. 163-174.

[12] (2010) The Microsoft BizTalk Server website. [Online]. Available: http://www.microsoft.com/biztalk/

[13] K. Frank, "Oracle fusion middleware and Microsoft interoperability: address enterprise-wide needs," Technical Report, Jan. 2005.

[14] (2010) The Sun RFID website. [Online]. Available: http://www.sun.com/software/products/rfid/

[15] C. Floerkemeier and M. Lampe, "RFID middleware design – addressing application requirements and RFID constraints," in: *Proc. SOC'05*, pp. 219-224.

[16] (2010) Orbited website. [Online]. Available: http://orbited.org/wiki/TCPSocket/

[17] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and Analyzing Massive RFID Data Sets" in *Proc. ICDE'06*, pp. 83-92.

[18] W. Wang, J. Sung, and D. Kim, "Complex Event Processing in EPC Sensor Network Middleware for Both RFID and WSN", in: *Proc. ISORC'08*, pp. 165-169.

[19] D. Gyllstrom, E. Wu, H. J. Chae, Y. Diao, P. Stahlberg, and G. Anderson, "SASE: Complex Event Processing over Streams", in: *Proc. CIDR'07*.

[20] (2010) Fujitsu fontech website. [Online]. Available: http://www.frontech.fujitsu.com/en/