Monitoring of RFID Failures Resulting from LLRP Misconfigurations

Rafik Kheddam, Oum-El-Kheir Aktouf, Ioannis Parissis
Univ. Grenoble Alpes, LCIS
F-26900, Valence, France
firstname.lastname@lcis.grenoble-inp.fr

Sanaa Boughazi
Toulouse III - Paul Sabatier University
31062 Toulouse, France
sanaa.boughazi@univ-tlse3.fr

***Abstract*—Radio frequency identification (RFID) technology is becoming a pervasive technology that is used in our daily lives and in a variety of domains, especially in critical domains such as medical field and transportation systems. Several standards are specified to support the use of this technology. LLRP protocol (Low Level Reader Protocol) is one of them. It is the communication protocol between RFID readers and a client (a middleware or an end user application). It uses several XML based specifications to set up a huge number of parameters in the reader's configuration in order to perform a correct tag inventory. Since this protocol is relatively new, there is no significant fault-tolerant mechanism that takes in consideration the protocol's behavior. In this paper we propose an approach that monitors RFID failures resulting from misconfiguration errors of LLRP protocol by using the logging system of this protocol.**

Keywords—LLRP; Model based diagnosis; RFID system dependability; RFID communication standard

I. INTRODUCTION

Radio Frequency IDentification (RFID) systems are heavily used in many domains such as supply chain management systems, medical applications and airport luggage handling systems. Despite the technological advances in this technology, RFID systems are still unreliable [1] [2]; their reliability is strongly related to many parameters such as the runtime conditions, and internal configuration of the whole system. Failures related to runtime conditions have been handled in a previous work using an online probabilistic diagnosis algorithm [3]. In this paper, we propose an off-line approach that monitors RFID systems according to their internal configurations. It is a mechanism that diagnoses the behavior of readers using the communication standard LLRP (Low Level Reader Protocol) [4]. LLRP message composing and management is implemented in several languages [5]. The Java-based version uses the Apache logging system Log4j [6]. Our proposed approach is based on this event logging system and can be summarized in these following steps. First, we analyze and identify all possible failures resulting from LLRP parameter misconfiguration (Fault Model). Next, a knowledge model is established; *i.e.*, for each failure, a set of possible causes are identified. The knowledge model is used to select all needed information from the log files to deal with the detected failure. Finally, when a given failure is detected, such as "*the reader does not identify all the tags located in its field-of-view*", a set of causes are selected to be checked in the LLRP log files according to the detected failure and the knowledge model.

The rest of the paper is organized as follows. First, Section 2 introduces RFID systems. Next, Section 3 summarizes related work on RFID system monitoring. Finally, Section 4 presents our proposed log file based diagnosis approach and the considered fault model.

II. RFID SYSTEM ARCHITECTURE

RFID systems are the next generation of object identification systems. They are replacing the well-known barcode systems [7]. They use radio waves to remotely identify objects without direct line of sight [8] [9]. A simple RFID system consists of three main components (Fig. 1).

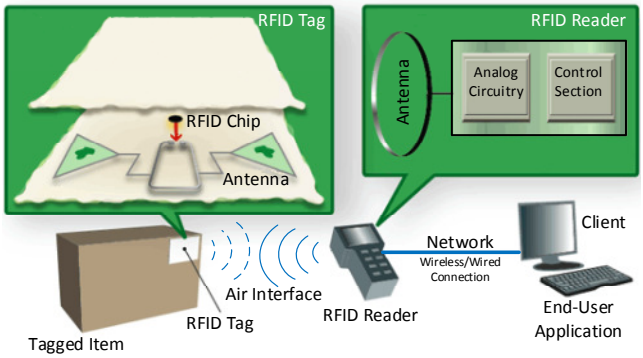

Fig. 1. RFID System Architecture

- RFID reader is a radio device that interrogates a tag using radio signal according to an air protocol (*e.g.*, UHF EPC Class 1 Gen 2 [10]) to perform two main tasks; tag inventory (the identification of all tags located in the reader's field of view) and tag memory access.
- RFID tag is an electronic label consisting of a micro-chip with a small memory and an antenna. It is used to uniquely identify an item by using an EPC code (Electronic Product Code) and carry some other information about the item.
- Client is the end user application or an RFID middleware in case of a complex and distributed system. Its role is to manage the RFID devices and to process their raw data in order to provide business

services. The communication standard used between RFID readers and clients is LLRP.

LLRP is a set of rules that covers the tag/reader communication. It uses two main XML based specifications: Reader Operations Specification (`ROSpec`) for tag inventory and Access Specification (`AccessSpec`) for tag memory access.

- `ROSpec` has three possible states {*Disabled*, *Inactive* and *Active*}.
- `AccessSpec` has two possible states {*Disabled* and *Active*}.
- An `AccessSpec` is always linked to one or more `ROSpecs` that will be responsible of triggering it. This means, when the identified tags by the running `ROSpec` match the tag conditions of the `AccessSpec`, the `AccessSpec` is selected for execution.
- The `AccessSpec` never runs before the corresponding `ROSpec`. It is never selected for execution while its current state is different from "Active" or if it does not appear in the list of `AccessSpecs` that might be selected by the running `ROSpec`.

As an example, Fig. 2 shows the system behaviour with one `ROSpec` (for a tag inventory).

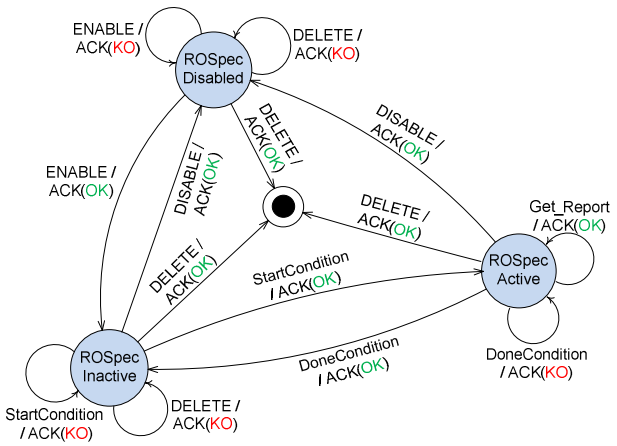

Fig. 2. Reader Operation Specification FSM

A new specification (`ROSpec` or `AccessSpec`) is always in a disabled state. In our example, the `ROSpec` moves to an inactive state when the reader receives correctly an "ENABLE" request from the middleware. The `ROSpec` is still inactive, waiting for a start condition such as "there is no `ROSpec` under execution" (*i.e.*, in active state), "the priority of the `ROSpec` is higher than those of remaining `ROSpecs`" or "there is an external triggering event" (*e.g.*, events from a motion sensor), *etc*. The reader can start returning the results of the `ROSpec` once this latter is active. A running `ROSpec` returns to inactive state when the inventory is done or if there is another `ROSpec` with a higher priority (*i.e.*, in this case, the `ROSpec` will be preempted). The `ROSpec` can also be disabled or deleted following a request of the middleware.

III. Related Work

In the last decade, RFID systems have been increasingly used in complex, distributed and critical scenarios. These new usages require monitoring each component of the RFID system in order to improve the reliability of the whole system. Many techniques have been designed to meet these dependability requirements. These techniques can be classified into two categories [11]; "reader status monitoring" and "reader performance monitoring".

- *Reader status monitoring* is an intrusive monitoring. It gathers all the techniques that monitor the status of the reader such as "is the reader powered on", "is the reader connected to the network", "are the reader's antennas operating", *etc*. The state of the reader can be checked for example by a simple SNMP (Simple Network Management Protocol) query such as:

  `ping "IP address of the reader"`

- *Reader performance monitoring* is a nonintrusive monitoring that uses the data available from normal operation of the system to process some reader performance values such as "Error frequency", "Read rate" (how accurate the reader is in identifying the tags located in its field of view), *etc*.

All these monitoring mechanisms are only interested in failure detection. It is primordial to have a fault-tolerant solution that is able of identifying the causes of a failure in addition of its detection. This will speed up the system recovery. In this paper we propose a complementary approach to existing failure detection mechanisms. This approach is intended to continue the diagnosis process once a failure is detected, to find out its causes. In this context, there is a technique called "Distinguishing Sequences" that is used to identify the exact state (the exact internal configuration) of the system. A distinguishing sequence in our case is a set of transitions that allow identifying the initial and current state of the reader or the middleware. For more details about this technique, see the work of Lee and Yannakakis [12]. To illustrate the principle of distinguishing sequences, let us take the example of Fig. 3. It represents a three-state FSM (Finite State Machine) such as the one of Fig. 2 with fewer transitions for sake of simplicity. This FSM has two inputs {A, B}, and two outputs {0, 1}.

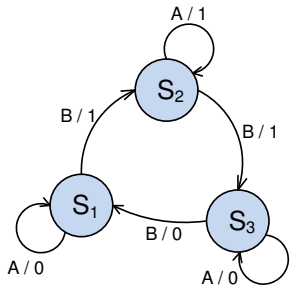

Fig. 3. Example of an arbitrary FSM

At the beginning, and after the onset of the failure, we assume that we don't know the internal state of the reader or of the middleware; *i.e.*, the FSM can be in any state. So, we construct the successor tree[1] of this FSM that will serve as a decision tree to identify the distinguishing sequences.

Fig. 4 shows the successor tree built up from the FSM of Fig. 3. We can see that the input A separates the state S_2 from S_1 and S_3. Then the input B separates S_1 from S_3. For example the sequence {A/1} is a distinguishing sequence for S_2 (*i.e.*, after applying the input A and observing an output 1, we are sure that the current and the initial state of the machine are S_2), and the sequence {A/0, B/0} is a distinguishing sequence for S_3; in the tree successor, this sequence leads to S_1. So, in the FSM of Fig. 3, when we go back from S_1 with the reverse sequence of previous distinguishing sequence (*i.e.*, {B/0, A/0}), we will end up at S_3 (that represents the initial state).

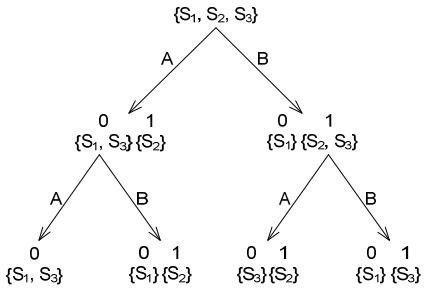

Fig. 4. The successor tree of the FSM

The distinguishing Sequence mechanism presents some drawbacks. By definition, a distinguishing sequence should always lead to a single final state. Thus, the sequence {A/0, A/0} is not a distinguishing sequence because it does not separate the state S_1 from S_3 and we can't go back to deduce the initial state. This is not the only condition to construct distinguishing sequences. The other one is that the reverse of the constructed sequence must also lead to a single state. The following FSM (see Fig. 5) shows this difficulty that may be encountered when building up distinguishing sequences. Thus, all sequences that start with the transition A/0 in Fig. 5 can never be distinguishing sequences, since their reverse sequences are non-deterministic (*i.e.*, we can never be able to deduce whether the FSM started in S_2 or S_3).

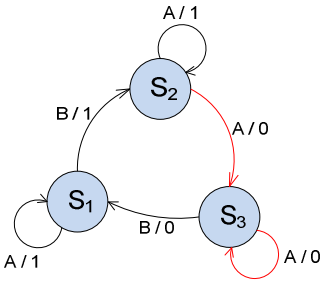

Fig. 5. Difficulty of constructing distinguishing sequences

[1] A successor tree shows the behaviour of an FSM starting from all possible states under all possible input sequences [12].

In addition to "not covering all states" of the monitored system,

- distinguishing Sequences are intrusive mechanisms where additional data are injected into the system to collect the needed information;
- performing a distinguishing sequence
  - is not always possible; *e.g.*, when the system has crashed or is broken down.
  - is unwanted; because it changes the state of the system. So, the system has to be stopped to perform this process.

IV. LLRP Log File Based Diagnosis

The log file based diagnosis approach that we propose, can be summarized in 4 main steps as shown in Fig. 6. First, when a failure is detected by a given monitoring mechanisms such as the one we propose in [3], it is classified according to the fault model. Then, a cause prediction is performed according to the type of the detected failure and the information contained in the knowledge model (see section IV.B). Next, a sub-content of the LLRP log file is extracted regarding the type and the occurrence date of the failure; this step is important because it allows the process that checks the LLRP causes to only focus on the useful parts of the LLRP logs. So, this will speed up the diagnosis. Finally, each possible cause will be checked by using the extracted information from the LLRP log file.

A. Fault Model

LLRP is a complex protocol with a huge number of variables that gives the user the ability of finely configuring an LLRP compliant reader. This is very useful to customize the configuration of the readers according to the user's needs and to have the control on everything in the system. However, some LLRP's variables are critical; if they are incorrectly set, the system will have an undesired behavior; *e.g.*, "the reader will not identify the expected tags". So, the failures that we are interested in are those resulting from LLRP misconfiguration (made by the user). We can summarize these failures as follows.

- *The reader is not identifying the tags located in its field of view.*
- *The reader identifies all tags but it does not transmit the result to the client.*
- *The reader misses to identify all tags or has a reduced read performance (e.g., the number of tag read attempts that succeed is decreasing).*
- *The reader identifies the tags but does not retrieve the requested data from the tag's memories; i.e., the reader does not perform tag memory access.*

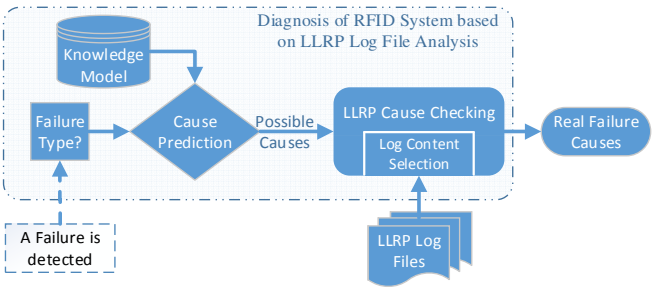

Fig. 6. LLRP Log File Analyzer

B. *Knowledge Model*

The knowledge model has two functions. (1) It associates to each failure a set of possible causes. This task is achieved by performing an FMEA (Failure Mode and Effect Analysis) on LLRP protocol. Table 1 presents a part of the FMEA that is related to the proposed approach. (2) The knowledge model defines the content of the LLRP log files. To achieve this task, we first need to modify the default LLRP logging format from simple text format (see Fig. 7) to an XML based format. This is done by the piece of code of Fig. 8. The XML based logging format is useful to automate the log file analysis.

```
5936 [AnonymousIoService-6] INFO org.llrp.ltk.net.LLRPIoHandlerAdapterImpl -
Message START_ROSPEC successfully transmitted
5940 [AnonymousIoService-6] DEBUG org.llrp.ltk.net.LLRPIoHandlerAdapterImpl -
<?xml version="1.0" encoding="UTF-8"?>
<llrp:START_ROSPEC
xmlns:llrp="http://www.llrp.org/ltk/schema/core/encoding/xml/1.0" Version="1"
MessageID="0">
        <llrp:ROSpecID>1</llrp:ROSpecID>
</llrp:START_ROSPEC>
```

Fig. 7. Default LLRP logging format

```
try {
        FileAppender fa1 = new FileAppender(
            new XMLLayout(),"LLRP_event_log.xml");
        fa1.setName("LLRP_event_log");
        BasicConfigurator.configure(fa1);
}catch (IOException ex){
        ex.getMessage();
}
```

Fig. 8. Piece of Java code for modifying LLRP logging format

An "easier to read" logging format is defined for the end-user in case he wants to manually monitor the system behavior. Fig. 9 shows the piece of code that configures the LLRP logging system to produce a more readable log format (see Fig. 10). The main differences between the default log format and the new log format are the presence of the date of the event and the content representation. The new log format is much easier to process in case of XML based format and easier to read in case of the user friendly format. The main content of the LLRP log file is the date, the source and the content of the event (the content of the received / sent message).

```
try {
        DailyRollingFileAppender  fa2 =
            new DailyRollingFileAppender (new PatternLayout(
            "Date: %d{dd/MM/yy HH:mm:ss.SSS}%n"
            +"Source: %t: %c%n"
            +"Msg [%p]: %m %n%n"), "LLRP_events.log","'.'yyyy-MM-dd");
        BasicConfigurator.configure(fa2);
} catch (IOException ex) {
        ex.getMessage();
}
```

Fig. 9. Piece of Java code for human-readable output

Date: 13/04/14 15:13:25.561
Source: AnonymousIoService-8: org.llrp.ltk.net.LLRPIoHandlerAdapterImpl
Msg [INFO]: Message START_ROSPEC successfully transmitted

Date: 13/04/14 15:13:25.561
Source: AnonymousIoService-8: org.llrp.ltk.net.LLRPIoHandlerAdapterImpl
Msg [DEBUG]:
<?xml version="1.0" encoding="UTF-8"?>
<llrp:START_ROSPEC xmlns:llrp="http://www.llrp.org/.../1.0" Version="1"
MessageID="0">
        <llrp:ROSpecID>1</llrp:ROSpecID>
</llrp:START_ROSPEC>

Fig. 10. User friendly output

TABLE 1. KNOWLEDGE MODEL

RFID Failures	Possible LLRP Causes
No tag identified by the reader	– The reader is not powered on. – There is no `ROSpec` to be executed by the reader. – The `ROSpec` is not active. – The start conditions of the `ROSpec` are not met. – The receive sensitivity of the used antenna is set to a low value.
The read result is not sent to the client	– The connection between the reader and the client was closed before sending the report. – The `ROReportTrigger` condition that defines when the reader has to send the report, is not satisfied.
The reader misses to identify all tags	– The receive sensitivity of the used antenna is set to a low value. – The `DurationTrigger`, the `NumberOfTags`, the `NumberOfAttempts` or the `T` value in `TagObservation-Trigger` is too small. – The `TagPopulation` value is incorrect.
No tag memory access	– There is no `AccessSpec` to execute. – The `AccessSpec` is not active. – The `AccessSpec` is not linked to the `ROSpec` under execution. – The identified tags do not match the `TagSpec` of the `AccessSpec`. – The specified `antennaID` in the `AccessSpec` is different from the one of the `ROSpec`. – `AccessReportSpec` is misconfigured.

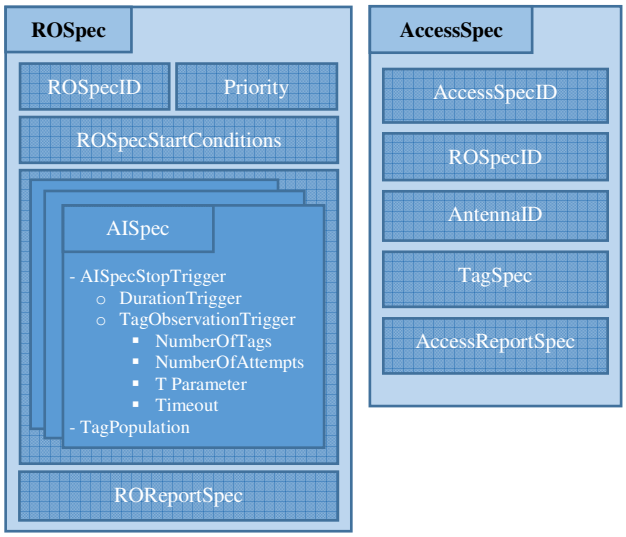

Fig. 11. ROSpec and AccessSpec Content

Fig. 11 shows the main parameters of a `ROSpec` and an `AccessSpec` that we are interested in. This will help understanding the failure causes reported in Table 1.

- `ROSpecStartCondition` can be a START request from the user, a "Periodic Time" (*e.g.*, execute the `ROSpec` each day at 8:00 *am*) or an external event (*e.g.*, an event from a motion sensor).
- `AISpec` is the main section of a `ROSpec`. It specifies all parameters required for tag inventory.
- `DurationTrigger` specifies the duration of the execution of the `AISpec`.
- `AISpecStopTrigger` can be specified instead of the `DurationTrigger`. It specifies the stop condition of the tag inventory. It can be a duration or tag observation parameter such as
  - `NumberOfTags`: this parameter is used to specify to the reader the number of tags to be identified in the current tag inventory. This means that the reader will identify only the specified number of tags, even if there are more tags in its field of view.
  - `NumberOfAttempts`: it is the maximum number of read attempts performed by the reader.
  - `T` parameter: it is an idle time between tag responses in milliseconds. If there are no more new unique tag observations for `T` milliseconds, the reader will end the tag inventory.
  - `Timeout`: it is always associated with one of the three other `AISpecStopTrigger` parameters described above. This parameter ends the tag inventory, whatever the value of the other parameters of `AISpecStopTrigger`.
- `TagPopulation`: this parameter indicates to the reader the number of expected tags in its field of view. The reader uses this parameter to manage tag signal collisions. The greater the value of this parameter is, the greater the duration of the tag inventory and the lower the number of tag collisions are; *i.e.*, when the value of `TagPopulation` is too small, there will be many tag signal collisions and the tag inventory will take much more time. When the value of `TagPopulation` is too big, the reader will expect more tags; so its anti-collision algorithm will run more slowly and the tag inventory will take much more time too.
- `ROReportSpec`: it defines through its `ROSpec-ReportTrigger` when the read report has to be sent to the client. The triggering condition can be either a minimum number of tags that have to be in the report or a timer.
- `ROSpecID` in an `AccessSpec` defines which `ROSpec` has to trigger the `AccessSpec`.
- `AntennaID` in a `ROSpec` or an `AccessSpec` defines the reader antenna that will be used to execute the `ROSpec` or the `AccessSpec`.
- `TagSpec` defines the tag filters. The tags that pass the filters are selected, and the `AccessSpec` will be executed on them. *e.g.*, "select only the tags having an EPC beginning with the pattern *aabc*".
- `AccessReportSpec` defines when to send the results of the `AccessSpec`, and also the contents and format of the report.

C. LLRP Failure Cause Checking

The failure cause checking process (called also the LLRP Log Analyzer) loads and extracts from the log files only the information that are useful for diagnosing the identified failure according to the knowledge model. This is a very important task, since the cause checking process is not able or will take much more time to process the huge volume of data contained in the log file (usually the log file exceeds 1000 Mo of data per day).

1) No tag identified by the reader: To diagnose such failure, the LLRP Log Analyzer will verify if the LLRP connection was successfully established between the reader and the client. Then, it will check if the `ROSpec` (that was responsible of reading the tags) was correctly set and enabled before the start conditions are met. This means the `ROSpec` was inactive (ready to be active) when the user sent the START command or when the motion sensor sent the triggering event. If this step does not reveal any misconfiguration error, the failure cause checking will verify if the power signal level (or the Receive Sensitivity[2]) defined in the `ROSpec` is not too low, so as the reader's field of view was reduced when the failure occurred.

[2] Receive Sensitivity indicates how faint a radio frequency signal can be successfully received by a given receiver. The lower the power level that the receiver can successfully process, the better the receive sensitivity [13].

2) The read results are not sent to the client: The LLRP log analyzer checks if the user does not send a STOP (resp., DELETE) request that has stopped (resp., deleted) the `ROSpec` under execution. LLRP log analyzer can also check if the minimum number of tags in the report (specified in `ROReportTrigger`) is set to a correct value; usually, it is set to "1" to allow real time result receiving. If this parameter is set to a great value, this means that the reader will not send the report until it identifies the specified number of tags, which may take a long time. The LLRP log analyzer can also check if the value of the timer in `ROReportTrigger` is set to a great value too; so, the report was not sent when it was needed.

3) The reader misses to identify all tags: The LLRP log analyzer checks if the `DurationTrigger` of `AISpec` is not too small or the `TagObservationTrigger` value was correctly set. This means that the number of expected tags, number of read attempts, etc. are correctly set.

Examples

- If the number of expected tags (`TagPopulation`) is lower than the real number of tags in the reader's field of view, there will be many tag signal collisions. So, the timeout will be reached before identifying all tags.
- If the `T` value that defines the idle time between tag responses is too small and if some tags have response time delays, the tag inventory will be ended and these tags will not be inventoried.

4) No tag memory access: In such failure, the LLRP log analyzer will check

- if the `AccessSpec` is active and the `ROSpecID` defined in the `AccessSpec` is the same as the one of the `ROSpec` under execution.
- if the identified tags pass the tag filters defined in `TagSpec`.
- if the specified antenna in the `AccessSpec` is the same as the one specified by the `ROSpec`.
- if the `AccessReportSpec` is correctly set to send the needed information at the right time.

Example

The monitored reader has two antennas. The `ROSpec` is under execution on the antenna #0 and the `AccessSpec` is ready to be executed on the antenna #1. The tag memory access operations may fail because the tags are in the antenna #0 field of view and out of range of the antenna #1.

If no cause is found for the detected failure, we assume that the causes are external to LLRP which are out of the scope of this paper. See our previous work [3] for more information about the causes of failures related to runtime environment, aging effect or hardware/software defects.

V. Conclusion and Perspectives

We have presented in this paper an ongoing work on RFID system dependability that aims to detect user misconfiguration errors. To do so, we have proposed to modify the default LLRP logging format in order to ease its processing. Then, we have identified all RFID failures to take into consideration in our approach. A set of LLRP misconfiguration causes are associated to each identified failure and recorded in a knowledge model. This knowledge model is used by a log file analyzer to deal with the detected failure.

The next step of our work is to implement the log file analyzer and to perform some test scenarios with the aforesaid user errors. This will allow us to see how reliable our proposed approach is. Then, we will improve the fault and the knowledge models. Finally, we will adapt our approach to online monitor RFID systems by directly analyzing the exchanged messages between the reader and the client instead of analyzing the log files.

References

[1] R. Derakhshan, M. E. Orlowska and X. Li, "RFID Data Management: Challenges and Opportunities," *IEEE International Conference on RFID,* pp. 175-182, 2007.

[2] S. R. Jeffery, M. Garofalakis and M. J. Franklin, "Adaptive cleaning for RFID data streams," in *Proceedings of the 32nd international conference on Very large data bases*, Seoul, Korea, 2006.

[3] R. Kheddam, O.-E.-K. Aktouf and I. Parissis, "SafeRFID-MW: a RFID Middleware with runtime fault diagnosis," *Journal of Communications Software and Systems,* vol. 9, no. 1, pp. 57-73, 2013.

[4] EPCglobal Inc., "Low Level Reader Protocol (Version 1.1)," October 2010. [Online]. Available: http://www.gs1.org/gsmp/kc/epcglobal/llrp.

[5] "LLRP Toolkit," [Online]. Available: http://www.llrp.org.

[6] "Logging Services - Log4j Guide," Apache Inc., [Online]. Available: http://logging.apache.org/log4j.

[7] H. Lehpamer, RFID Design Principles, Norwood, MA: ARTECH HOUSE, INC., 2008.

[8] P. Krishna and D. Husak, "RFID INFRASTRUCTURE," *IEEE Communications Magazine,* vol. 45, no. 9, pp. 4-10, September 2007.

[9] T. Hassan and S. Chatterjee, "A Taxonomy for RFID," IEEE 39th International Conference on System Sciences, Hawaii, 2006.

[10] EPCglobal Inc., "EPC Radio-Frequency Identification Protocols Class-1 Generation-2 UHF RFID (Version 1.2)," 23 October 2008. [Online]. Available: http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2.

[11] P. Sanghera, F. Thornton, B. Haines, F. Kung Man Fung, J. Kleinschmidt, A. M. Das, H. Bhargava and A. Campbell, How to Cheat at Deploying and Securing RFID, Massachusetts, USA: Syngress Publishing, Inc., Elsevier, Inc., 2007.

[12] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - A Survey," *Proceedings of the IEEE,* vol. 84, no. 8, pp. 1090-1123, 1996.

[13] D. A. Westott, D. D. Coleman, P. Mackenzie and B. Miller, CWAP - Certified Wireless Analysis Professional Official Study Guide: Exam PW0-270, Indianapolis, Indiana: Wiley publishing Inc., 2011