RFID-based Distributed Memory for Mobile Applications

Michel Simatic

Institut Télécom, Télécom & Management SudParis, 9 rue Charles Fourier, 91011 Evry Cedex, France,
Michel.Simatic@it-sudparis.eu,
WWW home page: http://www-public.it-sudparis.eu/~simatic/

Abstract. The goal of our work is to give a user equipped with an RFID-enabled mobile handset (mobile phone, PDA, laptop...) the ability to know the contents of distant passive RFID tags, without physically moving to them and without using a Wireless Area Network. The existing architectural patterns involving passive tags do not meet simultaneously all of these requirements. Our RFID-based distributed memory does. By associating vector clocks to tags, we replicate a view of this memory on each tag and each handset, and disseminate updates between all of the replicas. Thus a user can locally query the replica hold by their mobile handset without physically moving to a tag. We have developed a pervasive game as an application example. Using data collected during real game sessions, we evaluate the performance of our distributed memory. Then we discuss staleness and scalability issues. We conclude and give perspectives of our work.

Key words: Distributed memory, RFID, NFC, Vector clocks, Gossip protocols, Pervasive application, Game

1 Introduction

RFID tags should be increasingly used in (pervasive) mobile applications. They are easy to deploy and robust. In addition, the number of users equipped with an NFC-enabled mobile phone is sensitively increasing. For instance, according to Juniper Research, 700 million users will have such mobile phone by 2013 [5]. Most of the time, RFID tags take place in an architecture involving a Wireless Area Network (WAN) like Wi-Fi, UMTS, HSDPA... [16]. But using a WAN increases installation and operational costs. As an example, according to [10], the cost of data traffic is substantial in Germany, whereas negligible in Sweden. Our goal is to meet simultaneously the three following requirements: 1) users equipped with an RFID-enabled mobile handset (mobile phone, PDA, laptop...) can know the contents of any distant passive tag; 2) they do not need to be physically near the tag; 3) they do not use any WAN. This paper presents our answer: the RFID-based distributed memory.

To achieve this goal, we replicate the contents of the tags on each tag and each handset of the system. Thus a handset can get the value of the data stored

on a distant tag. It just has to query its own replica. Whenever a mobile handset meets a tag (or another handset), their respective replica are made consistent by comparing the vector clock values coupled with the replicas. Thus the users are the cornerstones of what can be considered as a gossip protocol. We can stimulate this communication network by having an application design which incites users to meet repeatedly during a session. This is especially true if the application is a multi-user pervasive mobile game. If its game design does not already integrate this feature, we can adapt it to promote players to collaborate actively.

The work presented in this paper does not tackle security issues. Indeed a comprehensive analysis of security with respect to RFID/NFC represents a dedicated research field [12].

The rest of this paper is structured as follows. Section 2 details why standard RFID architectures do not meet our requirements. Then, Section 3 presents how we use vector clocks to get an RFID-based distributed memory. Afterwards, Section 4 presents the pervasive game we implemented as an application example. It describes how the game design stimulates the communication network implicitly made by the players. Section 5 analyzes some of the figures observed during our experiments. Section 6 is a discussion about this architecture: it presents related work and analyzes staleness and scalability issues. We conclude and give perspectives of our work in Section 7.

2 Passive RFID Tags in Existing Architectures

Passive RFID tags are pieces of memory which can only be accessed when activated by a transponder. These tags always contain an identifier specific to the tag. Some of them hold also a chunk of memory which can be read or written by the transponder. Passive RFID tags are commonly used following three architectural patterns.

The first architecture is a centralized one (as defined by GS1 EPCglobal organization [1]): each product is equipped with an RFID tag. When this product comes close to an RFID reader (either because the product has been moved near a reader, or an operator equipped with a mobile reader has moved near the product), the RFID reader reads the RFID identifier stored on the tag. Then it gives this information to a dedicated application. By accessing a central database, the application is able to link this identifier to the identity of the product and the procedure to handle it. Notice that to have this architecture operational, a WAN is required all the time in order to link the RFID reader and the central database. But, using a WAN introduces installation and operational costs which may not be compatible with requirements of some applications.

Thus, a second architecture can be considered. It consists in making local copies of the central database on each mobile handset. Doing so, when a mobile handset makes an update related to a product, it updates its local copy of the database. Regularly, the device is connected to the database in order to upload its updates. Possibly specific procedures are applied in order to reconcile updates

coming from the different mobile handsets. It is this architecture which has been put in place by the city of Paris for the trees growing on the border of its avenues [13]. Each tree of Paris avenues is equipped with an RFID tag. During the night, in their storage room, tablet PCs are in contact with a central database in order to make a local copy of it. In the morning, each gardener takes one of these tablet PCs. Each time they does something to a tree, they reads its RFID tag with their tablet PC. The PC updates data on its local database. At the end of the day, the gardener puts back the tablet PC in its storage room. Finally, the tablet PC synchronizes its local database with the central database. The problem with this architecture is the following: user B cannot see the modifications done by user A on one piece of data unless user A has synchronized their device with the central database (after having modified the considered piece of data) and user B has also synchronized their device with the central database.

To solve this problem, a third architecture can be considered. It is based on RFID tags containing memory which can be written. In this case, the application reads/writes the updates concerning the product (*e.g.*, its history) on its associated RFID tag. Actually, the system made of all of such RFID tags corresponds to a distributed memory, each RFID tag holding piece of data of this distributed memory: there is no more need for central database storage. But this architecture experiences one severe limitation: a mobile handset has no way to have an idea of the value of the different tags of the system unless it is physically in contact with each tag. Thus the handset can never make queries on the whole contents of this distributed memory without physically moving to all of the tags.

None of the presented architectures meets simultaneously our three requirements. We propose a solution by introducing vector clocks inside the RFID tags.

3 Applying vector clocks to passive elements

In this section, we present how vector clocks are used to give to each handset the best up-to-date view of distributed memory (DM). This gives users the ability to make queries on a local view of DM.

Here are the main ideas of our solution. Each tag and mobile handset of the system holds a local view of DM. We associate a vector clock per local view. The vector clocks we use are plain vector clocks [15, 8]. Whenever a mobile handset comes into contact with a tag (or another handset), these two elements make their own view of DM consistent by comparing vector clock values. Thus, each of them takes advantage of the knowledge of the other element to get more recent information concerning DM evolutions. Doing so, they get a more up-to-date view of DM. The next paragraphs detail our solution.

The system we consider is made of two types of elements: RFID tags and mobile handsets (See Figure 1). Each element e holds a local view DM_e of the distributed memory. We note $DM_e[r]$ the view element e has of the contents of DM hold by RFID tag r. Each element e holds also a vector clock VC_e which is used to propagate operations done on DM. $VC_r[r]$ holds the timestamp of

the last update done on $DM_r[r]$ (which is the part of the distributed memory DM hold by RFID tag r). Meanwhile $VC_{e,e\neq r}[r]$ holds the timestamp of the last update of $DM_r[r]$ which element e is aware of.

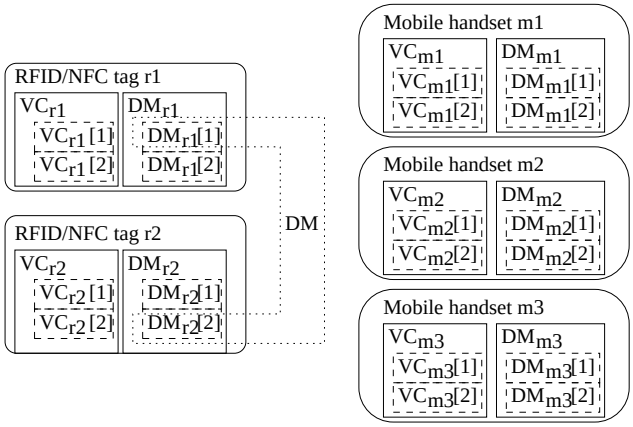

Fig. 1. Data present in a system made of 2 RFID tags and 3 mobile handsets

At application initialization time, for all elements e of the system, DM_e is initialized with the initial value of the distributed memory. This initial value depends upon the application which will be using the distributed memory. On the contrary, VC_e is initialized to a value independent from the application: $(0, \ldots, 0)$.

Then, during the rest of application lifetime, whenever a mobile handset m changes the value stored in $DM_r[r]$ of a tag r, it applies Algorithm 1.

Algorithm 1: Update of $DM_r[r]$ on tag r by mobile handset m

1 $DM_r[r] \leftarrow update\ of\ DM_r[r]$
2 $VC_r[r] \leftarrow update\ of\ VC_r[r]$
3 $DM_m[r] \leftarrow DM_r[r]$
4 $VC_m[r] \leftarrow VC_r[r]$

As in [17], usually $VC_r[r]$ is a logical clock: line 2 of Algorithm 1 is $VC_r[r] \leftarrow VC_r[r] + 1$. But, as in [11], we may take advantage of two conditions to save space on each tag (and thus improve the scalability). First condition is "The real-time clock of the last update of the tag r is stored among the data of $DM_r[r]$". Second condition is "Two subsequent updates always have subsequent and different real-time clocks". If both conditions are satisfied, $VC_r[r]$ can hold directly the real-time clock of the update: line 2 of Algorithm 1 becomes $VC_r[r] \leftarrow real\text{-}time\ clock\ of\ the\ update\ of\ DM_r[r]$. We save the space of one logical clock.

Another optimization can be coupled with the previous one. It can be considered if the real-time clock of the last query on $DM_r[r]$ is stored among the data of $DM_r[r]$. In the case there is no need to distinguish the real-time clock of the last update and the real-time clock of the last query, $VC_r[r]$ can hold the real-time clock of the last update or query: line 2 of Algorithm 1 becomes $VC_r[r] \leftarrow real-time\ clock\ of\ the\ update\ or\ query\ of\ DM_r[r]$. We save the space of one real-time clock.

In addition to Algorithm 1, by applying Algorithm 2, DM_e and VC_e may be updated whenever element e is able to exchange information with another element e'. In the context of an RFID/NFC-based application, this happens in two cases: a mobile handset is near an RFID tag (respectively another mobile handset) and is able to interact with it via RFID/NFC protocol (respectively NFC peer-to-peer protocol).

Algorithm 2: Making DM_e and $DM_{e'}$ consistent

```
1  foreach i, 1 ≤ i ≤ number_of_tags_in_the_system
2   if VC_e[i] < VC_e'[i] then
3     // Element e' holds a more up-to-date view of DM_i[i]
4     DM_e[i] ← DM_e'[i]
5     VC_e[i] ← VC_e'[i]
6   elseif VC_e[i] > VC_e'[i] then
7     // Element e holds a more up-to-date view of DM_i[i]
8     DM_e'[i] ← DM_e[i]
9     VC_e'[i] ← VC_e[i]
10  endif
11 endforeach
```

By applying Algorithms 1 and 2, each element e of the system has the best up-to-date view of the distributed memory it can have. But there is no guarantee that this *best* up-to-date view is the *most* up-to-date view of the distributed memory. As DM evolves during lifetime of the application, DM_e may contain stale data: there may be a tag $r, r \neq e$ for which $DM_e[r]$ does not correspond to $DM_r[r]$ currently hold by tag r. The only guarantee we have is that tag r did hold value $DM_e[r]$ at some point in time. Either it is its initial value. Or it is a subsequent value which induced the modification of $VC_r[r]$ (after application of Algorithm 1) and thus the propagation of this new value to $DM_e[r]$ (after application of Algorithm 2).

There are other staleness limitations. They are discussed in Section 6.

Despite them, an application running on a mobile handset m is able to make queries on DM_m: it has a view of the whole distributed memory without moving physically to the tags. Notice that the application can help reducing the staleness limitation. As in gossip protocols, if it stimulates information exchanges between elements of the system, propagation of updates will disseminate quicker in the system. Thus the gap between DM_e and DM will be thinner. Next sec-

tion presents an application example and shows how its design promotes the dissemination.

4 Pervasive Game as an Application Example

The RFID-based distributed memory presented previously has been implemented in the context of pervasive game "Plug: Secrets of the museum" (PSM). This section briefly presents PSM, makes the link between PSM and the data structures presented in Section 3, and describes how the game is designed to promote the dissemination of data.

PSM has been developed in the context of the PLUG project [3]. It is designed to let players discover, in a different way, meaningful objects of a museum [18]. In PSM, 8 teams of players are equipped with NFC-enabled mobile phones (Nokia 6131 NFC). Their main goal is to collect family of objects during a game session of at most 85 minutes. To do so, using a J2ME midlet we have developed, they exchange virtual cards (representing objects of the museum) either with one of the 16 RFID/NFC tags (ISO 14443, Mifare-NFC, 13.56 MHz, 1 Kbyte of RAM) located in different places of the museum, or with other teams (through NFC peer-to-peer communication). To reduce risks of deadlocks between players, every card has 3 instances in the game: there are 4 families and 4 cards by family times 3 instances, which makes 48 cards, in all. At the beginning of a game session, cards are shuffled and distributed between the 8 mobile phones (4 cards per handset) and the 16 RFID tags (1 card per tag).

PSM uses data structures presented in Section 3 as follows. $DM_e[r]$ (e being any element of the system, r being any tag) is a byte storing a value between 0 and 15 (each value corresponds to one of the 16 cards). $VC_e[r]$ is a short value (two bytes). It contains the real-time clock of the last update or query on $DM_r[r]$ as seen by e. Indeed we use the optimizations presented in Section 3. This is because our game has two interesting properties. The first one is that the clocks of the mobile phones are manually synchronized at the beginning of each day; we checked experimentally that the clocks do not drift away more than 1 second from each other during the day. The second property is that it takes at least 10 seconds for two players to exchange a virtual card with the same tag or to make a query on the same tag. As 10 seconds is much greater than 1 second: 1) we can apply the optimizations; 2) the real-time clock can be stored as the number of seconds since the beginning of the game (as a game session lasts at most 85 minutes —5100 seconds— there is no risk of overflow).

DM_e (respectively VC_e) is initialized to the 16 card values hold initially by the tags (respectively $(0, \ldots, 0)$).

At play time, whenever a team wants to exchange one of the cards hold by its mobile phone with the card hold by a tag, it must go physically near the tag. The application on the mobile handset applies Algorithm 1. Moreover, when a team wants to know what card is physically contained in a tag, it also has to go physically near the tag. We take advantage of this `read` operation to apply Algorithm 2 in order to make DM_{mobile} and DM_{tag} consistent. Thus, although

the team believes there is only a `read` operation (to display the card stored in the tag), there is actually also a `write` operation which possibly modifies DM_{tag} and VC_{tag}. Algorithm 2 is also applied whenever two teams exchange cards between their mobile phones (via NFC peer-to-peer protocol): this makes DM_{mobile_1} and DM_{mobile_2} consistent.

To help players in their search, game design introduces a hint function: a team can ask its mobile phone for an indication of an RFID tag which contains a virtual card convenient for its collection. The hint function is implemented by analyzing DM_{mobile} and VC_{mobile}. It considers the virtual cards stored in DM_{mobile} which correspond to the family collected by the team. Among these, it selects a card which the team does not have already and for which the information in VC_{mobile} is the most recent (Intuitively, the more recent it is, the more probable it is that the virtual card is still in the RFID tag). Actually the hint function provides two kinds of result. The first kind is the indication of a tag r and the card $DM_{mobile}[r]$ it was containing $currentClock - VC_{mobile}[r]$ seconds ago. The second kind of result is "For the moment, no tag contains a card which is interesting for you". This message is displayed when none of the cards of DM_{mobile} satisfies team's need.

The following paragraphs describe the game design specificities which promote the dissemination of data done by algorithms of Section 3.

As seen previously, whenever a mobile phone is in contact with another element of the system (should it be a tag or another mobile phone), we apply Algorithm 2 in order to make DM_{mobile} and $DM_{tag\_or\_mobile}$ consistent. The only way a team can know the card stored in a tag is to physically go to that tag: team is promoted to physically move to tags. Thus we stimulate data propagation via tags. Moreover, a team gains points whenever it exchanges a card with another team: team is promoted to do such exchanges. Not only does it foster human interactions inside the museum [9], but it stimulates data dissemination via mobiles.

Concerning exchanges between teams, a rule prevents two teams from spending their whole PSM session doing exchanges of cards (and thus getting corresponding points). To do so, in PSM, a team can exchange at most two cards with another team in a sliding window of ten minutes. This rule limits the number of exchanges made by a team during a session. For instance, in the case of a session lasting 85 minutes, a team can make at most $(85/10 + 1) \times 2 = 18$ exchanges. Even though it is limiting number of exchanges, this rule has indeed a positive effect on dissemination of data. If it was not there, at least one pair of mobile phones would not contribute any more to the propagation of data updates to other mobile phones and to tags. Said in other words, at least 2 of the 8 mobile phones would not be used any more for this dissemination task: we would lose at least $2/8 = 25\%$ of our "network" capacity.

This section has presented the game we have developed as an application example. Next section presents the experimental results we observed with this game.

5 Experimental Results

In June 2009, five public sessions took place in the *Musée des arts et métiers* (Paris). Each of them was played with 6 teams. In this section, we present our methodology to obtain data out of these sessions. Next, we analyze some of the data.

During each game session, each mobile phone logs all of the events triggered by the team using it. We log systematically the date of the event. The other logged information depends upon the type of the event. Here are some details about the events concerning the rest of this article:

- for `Application start` event, we log the game session identifier, the family collected by the team, the virtual cards hold by the mobile phone, DM_{mobile} and VC_{mobile};
- for `Consult a tag` event, we log the tag identifier, the virtual card contained in the card, DM_{mobile} and VC_{mobile} after DM_{mobile} and DM_{tag} have been made consistent;
- for `Exchange with a tag` event, we log the tag identifier, the virtual card given, the virtual card received;
- for `Exchange with another team` event, we log the identifier of the other team, the virtual card given, the virtual card received, DM_{mobile} and VC_{mobile} after DM_{mobile} and $DM_{mobile\ other\ team}$ have been made consistent;
- for `Hint request` event, we log the family collected by the team, the kind of hint result, and the information given to the team (in the case of a result of the first kind).

At the end of each day of tests, all of the log files are transferred from the mobile phones to a laptop. On this laptop, we run a Java application. It orders all of the events according to their `Date of event` field. Then it counts the different events. Finally it builds a history of the state of the system. The state concerning cards contained by each mobile is initialized by analyzing `Application start` events. It evolves whenever there is an `Exchange with a tag` or `Exchange with another team` event. The state concerning tags is initialized when analyzing first `Consult a tag` event. It evolves whenever there is an `Exchange with a tag` event.

As we are in front of a fully distributed application, it is quite sure that the history we get does not correspond to what did happen in the reality. Section 4 has presented the properties concerning time in PSM. Thus, in the history of events we get, we have the guarantee that the chain of events concerning each tag and each mobile phone is correct. It is the interleaving of these events we obtain which may not have existed in reality [4]. Thus, the figures we present hereafter are only an approximation of what really happened during application execution.

During the 5 game sessions, 30 mobile phones played sessions of 70 to 85 minutes. 3889 events were logged. There were 590 visits to a tag, 279 exchanges between a mobile phone and a tag and 142 exchanges between one mobile phone and another one.

14 hint requests were logged[1]. As our event analyzer builds a history of the state of the whole system, whenever there is a `Hint request` event, we can evaluate the correctness of this hint. 8 (respectively 3) out of the 9 (respectively 5) hint results of the first (respectively second) kind were correct. This means $(8+3)/(9+5) = 79\%$ of the hints were correct. But this percentage is computed with only 14 data. To have a bigger sample, we have modified the event analyzer: whenever analyzing a `Consult a tag` event, the analyzer generates an artificial `Hint request` event. To do so, it applyies the algorithm which the mobile would have applied if the player had triggered the hint function at that moment. With the generated hints, we observe that 490 (respectively 30) out of the 540 (respectively 50) hint results of the first (respectively second) kind were correct. This means $(490 + 30)/(540 + 50) = 88\%$ of the hints are correct. Of course, 88% is much lower than the 100% accuracy we would have obtained with a centralized architecture. But, we do reach 88% without installation and operational costs of a WAN. The acceptability of this rate of correct hints is application dependant. In the case of our game, the hint function is presented to players as a kind of gambling: a rate of 88% is acceptable.

One may think intuitively that a recent hint has more chances of being correct than an old hint. The beginning of Table 1 (until line "$[30, 34]$") confirms this conjecture. But we have no satisfying explanation for the high percentages observed in the last lines.

Another factor of the correctness of hints is how many tags are notified of a given change in a tag (see Figure 2) and how long it takes to disseminate such information (see Figure 3). If it is rather frequent that at least 3 tags are notified of the change of a tag (it happens with a frequency of 39%, see Figure 2), the frequency of notification of at least 12 tags is less than 9%.This is because the notification of a tag change takes time to spread itself among other tags.

Figure 3 shows for instance that, to have 3 tags notified of a change, it takes at most 5 minutes in 50% of the cases. So while the information of a tag change is spreading, the probability that the concerned tag receives a new virtual card is increasing. This new information will also spread itself, replacing the previous one. This explains why it is very rare (frequency of 3%, see Figure 2), that all of the 16 tags get notified of 1 tag change.

The dissemination of the new value of the tag takes time. It is possible that, at some point in time, the new value of a tag is spreading among elements of the system, while its old value is still spreading among other elements. Thus, it may happen that a tag is notified of stale information. To evaluate this last phenomenon, each time an element e is notified of a value change in tag r, we compare the date t_e of notification with the date t_r of next value change of tag r. Let's define the validity period as $T_{valid} = t_r - t_e$. If $T_{valid} \leq 0$, the value change of r which has been notified to e is stale. If $T_{valid} > 0$, the value change

[1] This number of logged hint requests is low probably because of the game design: asking for a hint is costing points to a team. Moreover, the hint was presented to players as not being 100% sure. We suspect this resulted in players not asking often for a hint.

Table 1. Relation between rate of correct hint result of the first kind and hint age

Age of the hint (in minutes)	Number of hint requests	Number of correct hints	% correct hints
[0, 4]	176	178	99%
[5, 9]	59	65	91%
[10, 14]	50	61	82%
[15, 19]	55	64	86%
[20, 24]	32	35	91%
[25, 29]	7	8	88%
[30, 34]	14	18	78%
[35, 39]	15	15	100%
[40, 44]	12	12	100%
[45, 49]	7	8	88%
[50, 54]	7	8	88%
[55, 59]	10	10	100%
[60, 64]	6	7	86%
[65, 69]	10	17	59%
[70, 74]	11	13	85%
[75, 79]	10	12	83%
[80, 84]	5	5	100%
[85, 85]	4	4	100%

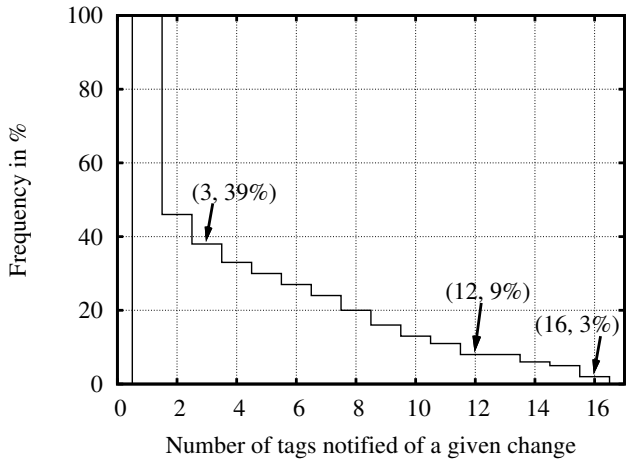

Fig. 2. Frequency of number of tags notified of a change (including the concerned tag)

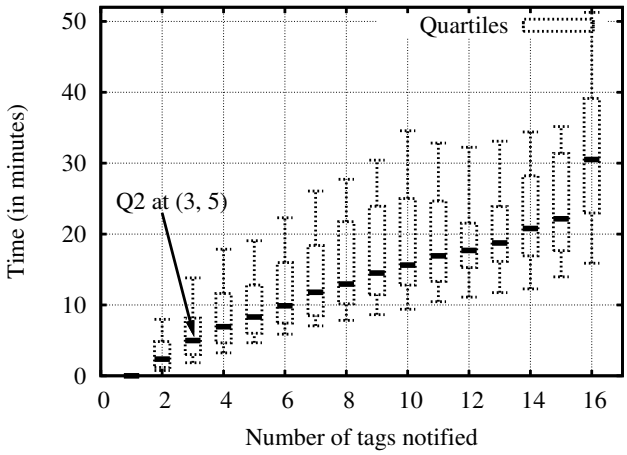

Fig. 3. Boxplot of times (in minutes) taken by a given number of tags to be notified of a change (decile D1, quartiles Q1, Q2, Q3, and decile D9)

of r is an up-to-date information. The results are summarized in figure 4[2]. In $FrequencyOfValidityPeriods\{T_{valid} \leq -1\} = 29\%$ of the cases, we disseminate stale information.

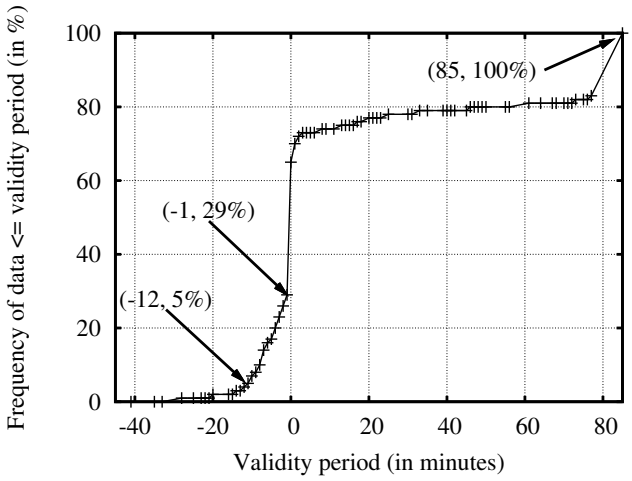

Fig. 4. Frequency of validity periods (T_{valid})

[2] Validity period +85 has a special meaning: the virtual card stored in tag r at moment t_r never changed again until the end of game session (which can last at most 85 minutes).

Figure 5 evaluates the impact of stale information dissemination on the perception of the global state by the mobile. To get it, for each `Consult a tag` event, we count the number of differences (*i.e.* stale tags) between DM_{mobile} and the real state. Figure 5 shows that DM_{mobile} contains no stale tag in $FrequencyOfAtMostNStaleTags\{n = 0\} = 6\%$ of the cases. In other words, DM_{mobile} contains at least one stale tag in $100 - 6 = 94\%$ of the cases. On the other hand, DM_{mobile} contains at least 5 (respectively 7) correct tags in 100% (respectively 99.8%) of the cases. In the context of our game, these percentages help understanding the difference of rate of correct hint results between the two kinds of results. Indeed first kind (respectively second kind) has a rate of $490/540 = 91\%$ (respectively $30/50 = 60\%$). This is because, with first kind of results, we consider only the value of one card among the 16 cards. As DM_{mobile} contains at least 12 correct tags in 71% of the cases, the probability that the selected card is correct is high. Whereas, with second type of results, we take a decision based on the value of all of the cards in DM_{mobile}. The probability of a correct answer can only be lower.

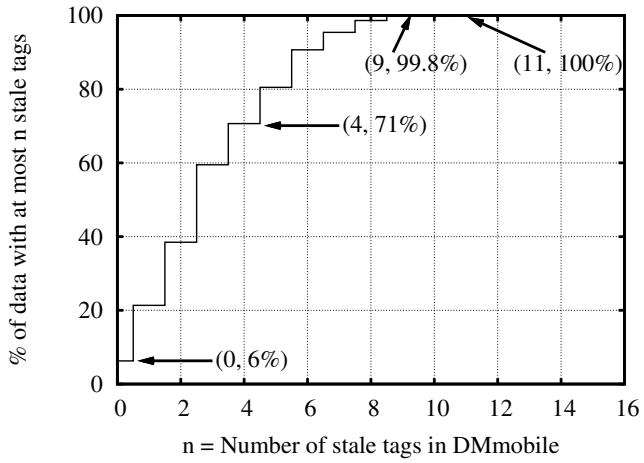

Fig. 5. Frequency (in %) of observation of at most n stale tags in DM_{mobile}

This section has presented the experimental results from five public game sessions. In summary, at least 3 tags were notified of a given tag change in 39% of the cases. The dissemination of this notification took at most 5 minutes in 50% of the cases. Moreover, in 29% of the cases, the notification received by a tag was stale. Despite this percentage, the view hold by each mobile had at most 4 stale tags in 71% of the cases. In addition to these "system" figures, we observed an "applicative" rate of hint success of 88%. This rate is acceptable by users in the context of our game (as the hint function is presented as a gambling function).

Next section analyzes our architecture with a more theoretical point of view.

6 Discussion

Section 2 presents how passive RFID tags are commonly used following 3 architectural patterns. In this section, we focus on scientific work related to our proposal. Then we analyze its limitations.

6.1 Related Work

As mentioned before, the work presented in this article relies on vector clocks theory [15, 8]. Moreover, our usage of vector clocks can be considered as a gossip protocol. Indeed, our algorithms satisfy most of the conditions defined by [2] to consider a protocol as a gossip protocol. For instance, when a handset interacts with a tag or another handset, the state of both changes in a way which reflects the other. The style of gossip protocol implemented in this paper is an anti-entropy protocol for repairing replicated data [7]. The data is the contents of the distributed memory. It is replicated on the handsets. Tags correspond to the network: when a handset interacts with a tag r, it leaves a message on the tag. The message contains information about what the handset has seen on the other tags and what information it has received from the other handsets. This message will be received by the next handset to interact with this tag.

This work can also be considered as an implementation of opportunistic data flooding. Such flooding is implemented in several sensor networks. For instance, *ZebraNet* is a mobile, sensor network designed to collect data about zebras [14]. To do so, a sensor is attached to each animal. It collects data locally and transmits them whenever another sensor comes within range. Periodic zoologists drive-bys can then collect logged data from many animals despite encountering relatively few within range. Our contribution is to integrate passive tags into such peer-to-peer architecture. Of course, a passive tag cannot be used to collect data locally. But it can hold information without any energy saving issue. This offers two interesting opportunities. First, the tag can be used as a data cache. A moving sensor can store data to be transmitted to another moving sensor which will come within the tag in the future. The second opportunity is that the tag can give a networking capability to any moving entity. Suppose zebras are equiped with passive tags. When moving from one static sensor to another one, these zebras may transmit information via their passive tag. They become "network" elements.

[6] presents an RFID-based distributed memory which does not require any WAN. This distributed memory is illustrated through two applications: *Ubi-Check* and *Roboswarm*. In *Ubi-Check*, an RFID tag is attached to each of the traveler's items. Each tag is initialized with a value specific to the traveler. All of these RFID tags are read after special points (*e.g.*, after an airport security control). Their values are transmitted to an application which checks that they are consistent. If it is not the case, it means that, at some point, the traveler exchanged one of their items with the item of another traveler. An alarm is thus triggered to warn the traveler that one of their items is missing. In *Roboswarn*, RFID tags are placed throughout a physical space to give direction information

to plain robots wandering in this space. These tags are initialized by dedicated robots before the standard robots are run. The major difference with our proposal is that, in *Ubi-Check* and *Roboswarm*, distributed memory data cannot be modified any more once the initialization process is over. In other words, [6] introduces an RFID-based distributed ROM which can be "flashed" (initialized), whereas we introduce an RFID-based distributed RAM.

6.2 Analysis of the Limitations

The architecture proposed in this paper has limitations concerning staleness and scalability. They are discussed in the following.

Staleness For any element e of the system, DM_e may contain stale data of three types.

Firstly, there may be a tag $r, r \neq e$ for which $DM_e[r]$ does not correspond to $DM_r[r]$ currently hold by tag r. This happened in 94% of the cases in our PSM experiments (see Figure 5).

Secondly, our architecture experiences the problem inherent to vector clocks in distributed systems [15, 8]: element e has no guarantee that DM_e corresponds to a value of DM which did exist at some point in time.

Thirdly, element e has no guarantee that it sees the whole history of changes of $DM_r[r]$ (for any tag r of the system). Suppose that, at time t_1, a mobile handset m_1 sets the value of $DM_r[r]$ of tag r to v_{t_1}. When applying Algorithm 1, $VC_{m_1} \leftarrow t_1$. Then, at time t_2, a mobile handset m_2 sets $DM_r[r]$ to v_{t_2}. Upon application of Algorithm 1, $VC_{m_2} \leftarrow t_2$. Afterwards, if m_2 comes to tag r' before m_1, when m_2 applies Algorithm 2, we have $DM_{r'}[r] \leftarrow v_{t_2}$ and $VC_{r'}[r] \leftarrow t_2$. When m_1 comes to tag r', because $VC_{m_1}[r] < VC_{r'}[r]$, $DM_{r'}[r]$ is never set to the value v_{t_1}.

In the following, we focus on the first type of stale data.

The rate of stale data and the period during which data remains stale is application dependant. How often do data evolve? How often are tags visited by application users? How long do users take to go from one tag to another?... In the context of PSM, the highest rate of stale data we observed is $11/16 = 69\%$ (see Figure 5). Concerning the period, 5% of the data remained stale more than 12 minutes (see Figure 4).

If the rate and/or period are too high for the application, three methods can be considered to reduce them. The two first ones are stimulating the "network". The third one introduces a WAN.

The first method consists in having a dedicated user who goes periodically through all of the tags just to read their contents and more important to update their DM_{tag} and VC_{tag}. By doing so, we reduce the risks of tags not being refreshed for a long time. In the case of PSM, this method would not be fruitful. Such a dedicated used would take 12 minutes to go through all of the tags. Thus this method would take off only $FrequencyOfValidityPeriods\{T_{valid} \leq -12\} = 5\%$ of negative dissemination (see Figure 4), reducing the total number

of negative dissemination by only $5\%/29\% = 17\%$, which is not worthwhile the effort.

Another method consists in periodically asking all of the users to meet all together to synchronize their DM_{mobile}. This method introduces a constraint on users who may not accept it. In the context of PSM, it would not be applicable: it would reduce too much the fun during the game.

The final method consists in a hybrid approach. Each mobile applies Algorithms 1 and 2. But, sometimes, under some circumstances which need to be defined and which may be application dependant, it connects to a server through a WAN. By applying Algorithm 2, it synchronizes DM_{mobile} and DM_{server}. This approach introduces a WAN (and the cost of data plans). But it guarantees a moderate use of the WAN (these costs would be restrained). Because it requires a WAN, this method may not be always applicable.

In this section, we have studied the staleness limitations of our proposal. But how do architectures presented in Section 2 behave concerning this limitation? First architecture (Centralized) and third one (Data stored only on tags) do not experience any staleness issue. But the former one requires a WAN. And, with the latter one, it is impossible to have an idea of the contents of a distant tag. The second architecture (Local copy of a central database) has worse staleness limitations than our architecture. Each time a user modifies the contents associated with a tag, the other users have an additional stale data in their local database. In particular, a user reading a tag already modified by another user will not see these modifications (whereas user does see them in the case of our proposal). A synchronization with the central database is mandatory to cleanup any stale data (whereas Algorithm 2 cleans up some of the stale data in the case of our proposal).

Scalability The number of data which can be managed by our architecture is limited by the hardware used.

First it takes time to read DM_r and VC_r from the tag to the handset, process Algorithm 2, and write back DM_r and VC_r to the tag. This time must be lower than an application dependant threshold TH. We suppose the processing time is negligible compared to the read and write time. Let TR be the transmission rate between the tag and the mobile, L the total length of one element of DM_r and one element of VC_r, and N the number of tag values which can be stored in DM_r. Then N is constrained by the following inequality: $N \leq \frac{TH \cdot TR}{2 \cdot L}$.

Moreover, the hardware has limited size. Let S be the size of the RAM in bytes. N is also constrained by: $N \leq \frac{S}{L}$.

We conclude:

$$N \leq \min(\frac{TH \cdot TR}{2 \cdot L}, \frac{S}{L}) \, . \tag{1}$$

Inequality 1 quantifies the influence of the optimizations presented in Section 3 on the upper bound of N.

In the context of PSM, we have $TH = 0.5$ seconds[3], $TR = 106$ kbps (we use plain ISO 14443 tags), $L = 3$ bytes, and $S = 1$ Kbyte. By applying Inequality 1, we get $N \leq 341$.

It is not possible to apply our architecture if there are more than a hundreds of tags. This is all right for some applications such as our game. But we believe it does not fit most applications, and in particular the ones classically addressed by the centralized architecture (which is able to handle thousands or millions of tags).

Addressing this scalability issue is one of our research perspectives.

7 Conclusion and Research Perspectives

The goal of our work is to give a user equipped with an RFID-enabled mobile handset (mobile phone, PDA, laptop...) the ability to know the contents of distant passive RFID tags, without physically moving to them and without using a WAN (which leads to installation and/or operational costs).

This paper studies how standard RFID architectures do not meet our requirements. Then it presents how we associate vector clocks to RFID tags to get an RFID-based distributed memory. This association may seem peculiar: we associate a vector clock (which is usually associated to active system elements like processors) to a piece of memory (which is intrinsically passive). This association is actually fruitful in two ways. On one hand, tags and mobile handsets can now be used to propagate information concerning the evolution of the RFID-based distributed memory; thus handsets are able to read the (cached) value of any piece of this distributed memory without being physically near the tag hosting that particular piece. On the other hand, this association turns users into network elements responsible to carry information between the tags; in other words, we create a Wireless Human-based Area Network (WHAN) playing the role of a WAN, but without installation and operational costs of a WAN. The tests we have made on an application example show that hints based on the analysis of the local view of this distributed memory are correct in 88% of the cases. The acceptability of this rate is application dependant. It is acceptable in the context of our game.

This architecture faces two issues. The first one concerns staleness. In 94% of the cases in our experiments, the view a mobile has of the distributed memory contains stale data. On the other hand, the highest rate of stale data we observed is 69%. The second issue concerns scalability. For instance, in our application, we could handle at most 341 tags. Some applications can cope with both of these limitations. It is indeed the case of our game. But the majority of applications cannot: these issues require further research.

[3] During our experiment, we observed that players were getting in a hurry at the end of a session. At that moment, they were spending their remaining time walking from tag to tag, putting their phone on the tag and immediately taking it off (in half of a second).

These are several research perspectives related to this work.

From a usage point of view, [9] analyzes some of the consequences of this WHAN on human relationships between application users. This study must be deepened.

From a technical point of view, the two first perspectives concern the issues identified previously: staleness and scalability.

A third perspective is related to the generalization of our work to a system containing any kind of processing devices and a distributed memory hosted not only by RFID tags, but also by the memory of some or all of the mobile handsets. There are three *sine qua non* conditions for this generalization. First, an element of this distributed memory can only be modified in one place (the element is the property of one specific RFID tag or processing device). Second, the set of RFID tags and devices contributing to the system is defined at the beginning of the system lifetime. Third, this set is ordered so that it is possible to associate one vector clock element to each system element. A perspective of our work is to see how it is possible to release one (or more) of these three conditions. Possible answers can be found within the Optimistic Replication community [17]. But the limited size of the RFID tags' memory requires at least adapting existing answers or bringing new ones.

Acknowledgments The author wishes to thank Eric Gressier-Soudan, Denis Conan, Annie Gentès, Aude Guyot-Mbodji, Camille Jutant and all of the reviewers, for all of their fruitful comments during the writing of this article.

References

1. F. Armenio, H. Barthel, L. Burstein, P. Dietrich, J. Duker, J. Garrett, B. Hogan, O. Ryaboy, S. Sarma, J. Schmidt, K. Suen, K. Traub, and J. Williams. The EPCglobal architecture framework. Technical Report Version 1.2, GS1 EPCglobal, September 2007.
2. K. Birman. The promise, and limitations, of gossip protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):8–13, 2007.
3. CÉDRIC, L3i, Musée des arts et métiers, NET Innovations, Orange, Institut Télécom—Télécom & Management SudParis, Institut Télécom—Télécom ParisTech, and Tetraedge. PLUG: PLay Ubiquitous Games and play more. http://cedric.cnam.fr/PLUG/, January 2009.
4. K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.
5. Christian D. 700 million of users of NFC mobiles in 5 years (in French). http://www.generation-nt.com/juniper-etude-technologie-nfc-mobile-utilisateurs-actualite-151831.html, September 2008.
6. P. Couderc and M. Banâtre. Beyond RFID: The Ubiquitous Near-Field Distributed Memory. *ERCIM news*, (76):35–36, January 2009.
7. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.

8. C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proc. of the 11th Australian Computer Science Conference (ACSC'88)*, pages 56–66. K. Raymond, February 1988.
9. A. Gentes, C. Jutant, A. Guyot, and M. Simatic. RFID technology: Fostering human interactions. In K. Blashki, editor, *Proceedings of IADIS International Conference Game and Entertainment Technologies 2009*, pages 67–74. International Association for Development of the Information Society (iadis), IADIS Press, June 2009.
10. S. Ghellal, J. Holopainen, M. Honkakorpi, and A. Waern. Deliverable D4.7: Final business guidelines. Technical Report D4.7, Integrated Project on Pervasive Gaming (IPerG), April 2008.
11. R. A. Golding. *Weak-consistency group communication and membership*. PhD thesis, University of California Santa Cruz, December 1992.
12. E. Haselsteiner and K. Breitfuß. Security in near field communication (NFC). In *Printed handout of Workshop on RFID Security RFIDSec 06*, July 2006.
13. ITR Manager.com. City of Paris is taking care of its trees with RFID tags (in French). http://www.itrmanager.com/articles/59758/59758.html, December 2006.
14. P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107, New York, NY, USA, 2002. ACM.
15. F. Mattern. Virtual time and global states of distributed systems. In *Proc. Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France*, pages 215–226. Elsevier, October 1988.
16. G. Roussos and V. Kostakos. RFID in pervasive computing: State-of-the-art and outlook. *Pervasive Mob. Comput.*, 5(1):110–131, February 2009.
17. Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.
18. M. Simatic, I. Astic, C. Aunis, A. Gentes, A. Guyot-Mbodji, C. Jutant, and E. Zaza. "Plug: Secrets of the Museum": A pervasive game taking place in a museum. In *Entertainment Computing - ICEC 2009, Eighth International Conference, Paris, France, September 3-5, 2009, Proceedings*, Lecture Notes in Computer Science. Springer, September 2009.