Choosing the Right RFID-Based Architectural Pattern

Michel Simatic

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/54069

1. Introduction

RFID provides a way to connect the real world to the virtual world. An RFID tag can link a physical entity like a location, an object, a plant, an animal, or a human being to its avatar which belongs to a global information system. For instance, let's consider the case of an RFID tag attached to a tree. The tree is the physical entity. Its avatar can contain the type of the tree, the size of its trunk, and the list of actions a gardener took on it.

When designing an RFID-based application, a system architect must choose between three locations to store the information: a centralized database, a database locally attached to the device hold by each user of the application, or the tag itself. Each location leads to an RFID-based architectural pattern[1]. But how to choose the right architectural pattern? What are the application attributes which must be taken into account in order to make the right choice?

The state of the art does not bring satisfactory answers. Indeed, when an article describes a RFID-based architectural pattern, it does not mention the application attributes which lead to choose this architectural pattern. On the other hand, some books or articles present the qualities of architectural patterns. But they do not take into account specificities of RFID. For instance, EPCglobal provides a standardized answer [2]: the centralized architectural pattern. A mobile device, NFC-enabled for example, reads an identifier on the RFID tag, then sends a message to a server which associates the identifier to the avatar stored in a central database. Thanks to its simplicity, this architectural pattern is used by several applications. But, it requires a global computer network: Such requirement increases operational costs. Moreover, it does not withstand an important number of simultaneous RFID read operations. Thus this pattern does not fit all RFID-based applications. [3] presents the stakes of introducing RFIDs inside an enterprise. But it does not contain any system architecture

1 . An. architectural. pattern. is. a. description. of. element. and. relation. types. together. with. a. set. of. constraints. on. how. they. may. be. used. [1].

thoughts. In a survey about RFID in pervasive computing, [4] presents several application examples. Depending on the application, avatars are stored either in a central database or in the tags themselves. But the authors do not give any clues on why an application has chosen to store its avatars in a given location. On the other hand, [1] lists the attributes which must be accommodated in a system architecture. Above all, there are the functionalities which are required from the system. Then, orthogonal to these functionality attributes, there are quality attributes. The authors distinguish system quality attributes (availability, modifiability, performance, scalability, security, testability, and usability), business qualities (time to market, cost and benefits, and projected lifetime), and qualities about the architecture itself (e.g. conceptual integrity). But the authors do not focus on RFID specific features.

So we have analyzed several existing industrial or experimental RFID-based applications. Moreover, we have developed RFID-based applications. From this experience, we identify the relevant attributes to compare RFID-based architectural patterns. We present them in section 2. With these identified attributes and their different aspects, we analyze four RFID-based architectural patterns, used by applications to access the avatar of a tagged entity. In the *centralized architectural pattern*, the mobile device reads an identifier on the RFID tag; then it contacts a server which associates this identifier to the avatar stored in a central database or in a database distributed between several companies [2]. With the *semi-distributed architectural pattern*, each mobile device holds a local copy of a central database associating RFID identifiers to avatars [5]. In the *distributed architectural pattern*, each RFID tag holds the avatar [6]. With the *RFID-based Distributed Shared Memory*, RFID tags hold the avatar and a replica of the avatar of other tags [7]. Sections 3 to 6 detail all of these architectural patterns: they present application examples and analyze the architectural pattern with the attributes identified in section 2. Thanks to this analysis, in section 7, we are able to provide guidelines to choose the convenient RFID-based architectural pattern. Finally, section 8 concludes this chapter and proposes perspectives for this work.

2. Architecture attributes and RFID technology

Relying on the experience gained by analyzing existing RFID-based applications and by developing RFID-based applications, we outline three architecture attributes among the attributes presented in [1]: (i) functionality, (ii) scalability, and (iii) cost. For each attribute, we present its different aspects which are influenced by the use of RFID technology.

2.1. Functionality attribute

Functionality is the ability of the system to do the work for which it was intended.

All architectural patterns give the ability to read/write the avatar of a read tag.

A first aspect of the functionality attribute is to check how the application behaves when it queries the avatar of a read tag. Is it guaranteed that the returned avatar has indeed the value which was last written? In other words, is there a staleness issue of avatar of a read tag?

The second aspect concerns the possibility of knowing the value (or having an order of idea of the value) of the avatar of a remote tag. By "remote", we mean that the user is not physically near the tag: The user is not able to put her reader on the tag. All she has is the identifier of the remote tag.

The third aspect is the staleness issue of the avatar of a remote tag. If the user is able to know the avatar of a remote tag, is it guaranteed that the returned avatar has indeed the last values associated to the tag?

2.2. Scalability attribute

The scalability criteria category evaluates how each architectural pattern behaves when there are numerous tags or numerous readers.

Its first aspect is the maximum number of tags which can be handled by the architectural pattern.

The second aspect characterizes the sensitivity of the architectural pattern to the number of simultaneous RFID tag read operations.

2.3. Cost attribute

The cost attribute groups all of the aspects which have an influence on the installation costs or the operational costs of the RFID-based application.

The first cost aspect concerns the requirement for a global network: do RFID readers have to be able to access at any time and any place to a specific computing machine (for instance, a server in the case of the centralized architectural pattern)? To fulfill this requirement, the readers may be equipped with a wired connection. In that case, the mobility of the readers is limited. The readers may also rely on Bluetooth® or Wi-Fi gateways. Both of these gateways may introduce installation costs. Moreover some readers may not be Wi-Fi enabled. For instance, the Nokia 6212 mobile phone is NFC-enabled, but has no Wi-Fi capabilities. Finally the reader may rely on a mobile data connection (e.g. UMTS, HSDPA, etc.). Such solution introduces operational costs because of data plans.

The second cost aspect concerns the RAM requirement on each tag. The more RAM there is on the tag, the more expensive the tag is. Notice that RAM may actually be prohibited on tags for technical reasons and not for cost reasons. For instance, application may require the use of low-frequency tags (e.g. 125 kHz), so that readers can interact with tags even though there is a liquid between tags and readers. In this case, the throughput is too low for a tag to host information other than its identifier.

The third cost aspect concerns the introduction of a new tag in the environment. For each architectural pattern, we determine the sequence of operations which is required in order to introduce a new tag in the environment. Knowing this sequence, we can determine how long this sequence lasts. Because this initialization procedure is executed by a human or a robot operator, its cost is proportional to the time spent.

The final cost aspect is related to the reinitialization of all of the tags. This criterion concerns only applications which, during their lifetime, need sometimes to have each tag given a new initial value. For instance, this is the case of Paris public transportation system. Users are equipped with a transportation pass containing an RFID tag. At the beginning of a month, each user has to reload her pass (to refresh her access rights): in other words, the tag has to be reinitialized. Some RFID-based games also require tag reinitialization. Indeed, in the case of non-permanent games, users play during successive game sessions. Thus at the beginning of each session, all of the tags must be reinitialized.

In this section, we have defined different aspects of three architecture attributes: (i) functionality, (ii) scalability, and (iii) cost. These aspects are influenced by the use of RFID technology. We use them to compare the behavior of four RFID-based architectural patterns. We start by analyzing centralized architectural pattern.

3. Centralized architectural pattern

This architectural pattern is often used by manufacturing applications. It has been standardized by EPCglobal [2]. When a reader is near a tag (for instance, the blue mobile in Figure 1), it reads the tag's identifier or an identifier stored in the tag's data zone (its Electronic Product Code in the case of EPCglobal). This identifier is represented by the hexagon in Figure 1. Then, the reader asks a server (ONS lookup service in the case of EPCglobal) which machine (EPC Manager in the case of EPCglobal) manages the avatar corresponding to the read identifier. When the server responds, the reader contacts this machine with the identifier of the tag. The machine queries its database and returns the avatar (for instance, the contents of the hexagon in the database in Figure 1).

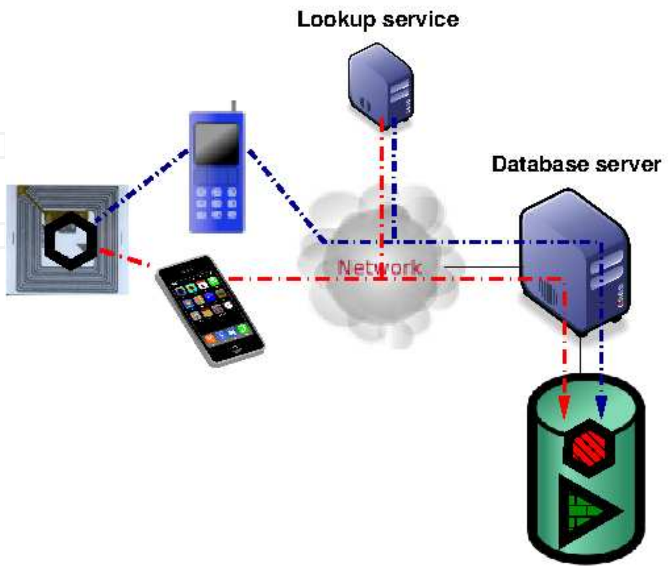

Figure 1. Centralized architectural pattern

Next section gives examples of this architectural pattern.

3.1. Examples

Aspire RFID is an Open Source middleware which is compliant to the specifications of EPC-global [8]. It proposes several examples of industrial applications for tracking goods.

Next paragraphs present products or prototypes developed according to centralized architectural pattern, but without being compliant to EPCglobal.

PAC-LAN is a game prototype in which players are equipped with NFC mobile phones without any GPS capabilities [9]. Players must interact with NFC tags which have been disseminated throughout a neighborhood. In a central database, the identifier of each tag is associated to geographical coordinates. When a player reads a tag, her mobile phone uses the UMTS network to contact the server with the tag's identifier. The server queries its database, finds the geographical coordinates, and broadcasts them to all of the players. An administrative application is provided to reset a game on the server. Such reset has an impact on all of the players' mobile phones.

[10] proposes an application so that visitors of an art exhibition can discover the paintings in another way. NFC tags are dispatched on the back of exposed paintings. Equipped with an NFC-enabled phone, the visitor puts her phone on spots of the paintings which intrigue her. Phone reads the identifier stored in the tag. Then, it contacts an uGASP server [11-12]. After consulting an internal database, this server indicates to the mobile phone what must be done: display a text, an image, or play an audio comment. Thus the author of the painting is able to communicate with the visitor.

Via Mineralia is a pervasive serious game which goal is to enrich the visit of a Freiberg museum [13]. In this game, the visitor uses a PDA equipped with an RFID reader. RFID tags (holding a unique identifier) are dispatched in the showcases which the museum wants to emphasize. When the PDA scans a tag, it sends an HTTP request (with tag's identifier) to a web server. To do so, the PDA uses a Wi-Fi network which covers the whole museum. The server answers to the PDA with multimedia information. The PDA displays them in a navigator.

Touchatag company (formerly Tikitag) sells NFC readers which can be connected to Windows or Mac-OS personal computers, and NFC tags dedicated to Touchatag [14]. A customer can then connect to http://www.touchatag.com web site, and define the reaction to be associated to the reading of one tag. When the NFC reader reads a tag, it contacts the Touchatag application which runs permanently on customer's personal computer. Then, via the Internet network to which the computer is connected, this application contacts a Touchatag service called *Application Correlation Service* (ACS). Touchatag application gives tag's identifier to ACS. Then, ACS queries Touchatag database to find reaction associated to the reading of this tag. It sends back this information to Touchatag application. The touchatag application reacts in the appropriate way. For instance, let's assume that the customer has specified the following action on Touchatag web site: when tag *r* with identifier *i* is put on the reader, customer wants her browser to access to Uniform Resource Locator (URL) of a web site *w*.

Then, when customer puts tag *r* on the reader, Touchatag application contacts ACS with identifier *i*. ACS replies with URL of *w*. Then, Touchatag application opens a browser with this URL *w*.

Skylanders is a video game developed by *Activision* company [15]. It requires the use of plastic figures. These figures contain an NFC tag. When a player puts her figure on top of a "Portal of Power" (actually, an NFC tag reader), the video game reads the identifier stored in the NFC tag. Then, the game contacts a server to get the information concerning the character which must be displayed: The figure becomes alive on the screen. Notice that, according to [16], information is also stored inside the tag: Thus the game can work without using a global network to contact a server. This means that Skylanders not only uses a centralized architectural pattern, but also a distributed one.

Based on all of these examples, next section analyzes centralized architectural pattern.

3.2. Analysis

Concerning the functional attribute, any transponder which wants to modify the avatar of a tag does so by sending a modification message to the server. Thus the server is always aware of the last update done on any avatar. As a reader always queries the central database to know the avatar of a tag, it is not possible that the read value is stale. Moreover, knowing a tag identifier, a reader is able to query the server to know the avatar associated to this identifier: the reader is able to know the avatar of a remote tag. As a mobile device queries the server to know the avatar of a remote tag, it is sure that the returned value is not stale.

Concerning the scalability attribute, the maximum number of tags which can be handled by this architectural pattern is limited by the number of avatars which can be stored in the central database. Let s be the average size in bytes of an avatar. Let $S_{central}$ be the maximum size in bytes of the database. We neglect the storage of the link between tag identifiers and avatars in the database. Moreover, we neglect the overhead due to the storage of data in the database. Then, the maximum number of tags is bounded by $S_{central}/s$. About sensitivity to the number of simultaneous reads, this architectural pattern is restrained by its centralized nature. The server holding the ONS lookup service may become a bottleneck. Moreover, the different servers of avatars may not return avatar values fast enough. Of course, it is possible to increase the number of servers. But that makes the hardware architecture more complex and more costly (from an installation and a management point of view). Thus this architectural pattern may not be applicable for some applications.

Concerning the cost attribute, the reader must always be in contact with the server holding the ONS lookup service and the servers of avatars: a global network is required. On the other hand, this architectural pattern only needs to read an identifier on the tag. And this identifier can be stored in ROM as it is never modified: no RAM is required on the tags. When a new tag is introduced in the system, three operations are required: (i) the tag is linked to the physical entity; (ii) the avatar of this entity is initialized in the central database; and (iii) a link between the tag identifier and this avatar is created into the central database. When all of the tags have to be reinitialized, a program is run on the server hosting the central database. It sets each avatar to its new value.

This section has analyzed centralized architectural pattern according to the attributes presented in section 2. This architectural pattern fulfills all aspects of functionality attribute. But this is achieved with the operational cost of a global network. Another disadvantage is a high sensitivity to the number of simultaneous read operations.

Next section analyzes semi-distributed architectural pattern which compensates the requirement for a global computer network and reduces sensitivity to the number of simultaneous reads.

4. Semi-distributed architectural pattern

In semi-distributed architectural pattern, mobile RFID-enabled devices (PDAs, mobile phones, etc.) are periodically synchronized with a central database holding all of the avatars (see Figure 2). Then, human operators carry the mobile devices near the entities to which the tags are associated. When a device comes close to an entity, the device reads the identifier of the entity's tag. By querying its local copy of the central database, the device is able to find the avatar of this entity. Any modification of an avatar is done on the local copy. It is propagated to the central database at the next synchronization.

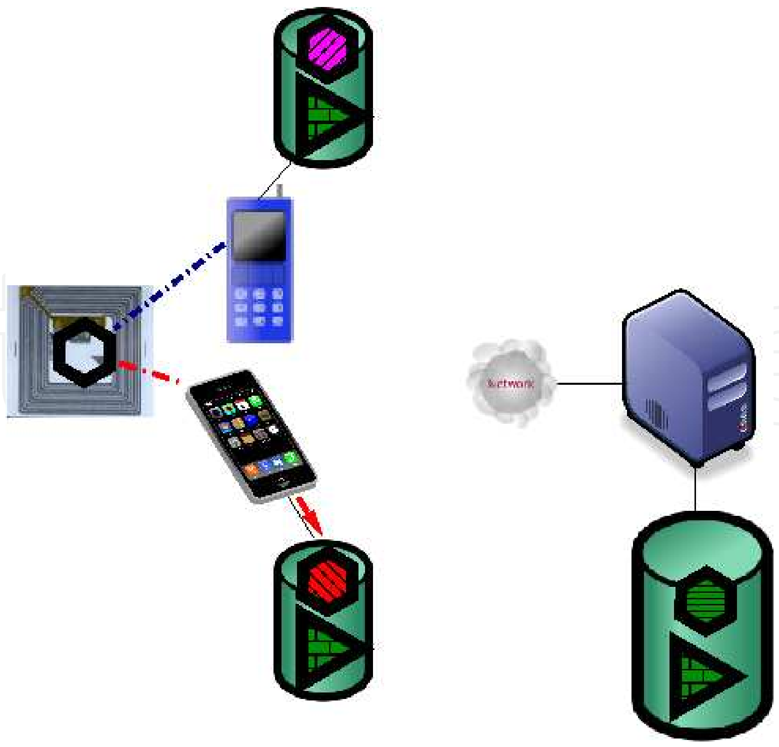

Figure 2. Semi-distributed architectural pattern

Next section presents an example of this architectural pattern.

4.1. Example

The unique example of use of such architectural pattern is Paris trees management application [5]. Each of the ninety-five thousand trees of Paris avenues is equipped with an RFID tag. Each gardener synchronizes her tablet PC with the central database before a new day of work. During her day of work, whenever a gardener does something to a tree, she identifies the tree thanks to its RFID tag: Her tablet PC modifies the avatar in the local database. Then, in the evening, she synchronizes her tablet PC with the central database. Thus she uploads her database updates and downloads updates from other gardeners.

Now, let's analyze the semi-distributed architectural pattern.

4.2. Analysis

Concerning the functional attribute, the avatar of a read tag may be stale. Suppose users U_1 and U_2 synchronize their mobile device with the central database. Then U_1 modifies avatar of tag *r*. Thus she modifies her local copy of the database. When U_2 comes to tag *r*, as her device reads its local copy of the database, the returned value of the avatar is the value before U_1's modification: the read value is stale. Notice we can limit this issue by assigning sets of entities to each mobile device. For instance, in the case of the Paris trees management application, a supervisor can assign a set of trees to be taken care of during the day, to each of the gardeners. If all of these sets are apart, this issue cannot be observed anymore. About remote tags, by querying its local database, the device is able to read the avatar of a remote tag, even though there is no global network. But the read value can be stale. It will be again correct only when all of the mobile devices have synchronized themselves with the central database.

Concerning the scalability attribute, the maximum number of tags which can be handled by this architectural pattern is limited by the number of avatars which can be stored in the central database and in the local copy of this database. Let S_{local} be the maximum size in bytes of the local database. The maximum number of tags is bounded by $min(S_{central}/s, S_{local}/s)$, which is likely to be S_{local}/s as mobile devices do not have as much memory as servers. Notice that this bound can be increased to $S_{central}/s$ by assigning to each mobile only a subset of the central database. For instance, in the case of Paris trees management application, the mobile device of a gardener could receive only the avatars of the trees she will take care of during the day. About sensitivity to the number of simultaneous reads, this architectural pattern is not as sensitive as centralized architectural pattern. It does not need to query a server upon each RFID tag read. Nevertheless all of the readers must periodically synchronize themselves with the central database. As the synchronization time is proportional to the number of readers, it may reach unbearable values. This issue can be tackled by limiting the number of avatars copied on the local devices, thus reducing the volume of data transferred between each device and the central database.

Concerning the cost attribute, the mobile RFID-enabled devices only need an access to the server hosting the central database during synchronization phase. At that moment, devices are probably near the central database: A Wi-Fi network may be used. Otherwise it is the local database which is queried. Thus no global communication network is required around the working area. Moreover, as in centralized architectural pattern, there is no need for RAM on the tags. When a new tag is inserted in the system, the procedure to be applied is the same as in the centralized architectural pattern. When all of the tags have to be reinitialized, a program is run on the server hosting the central database. It sets each avatar to its new value. However, the reinitialization of the tags will be effective only when all mobile devices will get synchronized with the central database.

This section has analyzed semi-distributed architectural pattern according to the attributes presented in section 2. This architectural pattern does not require any global network and has a medium sensitivity to the number of simultaneous tag reads. Nevertheless it faces a functional issue concerning the staleness of avatar read on a local (or remote) tag.

Next section analyzes the distributed architectural pattern which tackles the sensitivity and staleness issues.

5. Distributed architectural pattern

In distributed architectural pattern, the avatar of an RFID tag is stored inside the RAM of the tag (see Figure 3). Whenever a user is in contact with a tag, the reader works with the part of the RAM containing the avatar.

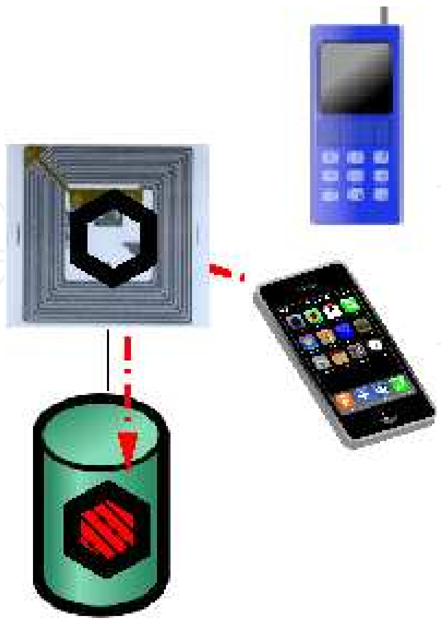

Figure 3. Distributed architectural pattern

Next section gives examples of this architectural pattern.

5.1. Examples

Nokia 6131 NFC phones are sold with three NFC tags. Each one triggers a different function on the telephone: One activates alarm function; another one plays a given music on the phone; the last one displays an NFC tutorial. To do so, the telephone reads the contents of the tag, this contents being coded as a Uniform Resource Identifier (URI) according to the NFC Forum's specifications of *Smarts Posters* [5,17-18]. When the phone is programmed to understand tags' contents formatted according to these specifications, these URIs can be used to tell the telephone to accomplish a given function like send an SMS, call a certain number, open a given web page, etc.

In fact, it is thanks to this Smart Posters specification that any NFC phone can exploit Touchatag tags mentioned in section 3.1. Indeed, these tags contain not only an identifier used by Touchatag application, but also an URI. This URI is the URL of a Touchatag web server with a parameter containing the identifier of the tag. Thus when a user touches a Touchatag tag with her mobile phone, the phone reads the URL and then opens a browser with this URL. Touchatag web server is then contacted, via 3G or Wi-Fi, with the identifier *i*. Then, web server contacts ACS (see section 3.1) with *i*. In the case where *i* is associated with a web site *w*, URL of *w* is sent back to Touchatag web server. This server returns an html page containing a redirection towards *w*. Finally, the browser displays *w*. Notice that, in the case of a Touchatag tag read by a mobile phone, phone uses distributed architectural pattern to determine the Touchatag web server to contact; but, the Touchatag web server uses centralized architectural pattern to translate the tag identifier into an action.

Once again, it is the Smart Posters specification which is used by *Connecthing* company to bring intelligence to mailboxes [19]. When a user scans a mailbox equipped with an NFC tag, her phone reads the URL stored on the tag (which contains an identifier corresponding to the physical location of the mailbox) and opens a browser to access this URL. This web page displays location of nearby mailboxes, the time at which postman takes the mail, etc.

Navigo, the Paris public transportation pass, is an example of an industrial application based on this architectural pattern, which does not use Smart Posters specification [20]. The 4.5 million Navigo pass users do not have an NFC reader. They are only given a pass which contains an NFC tag. With a vending machine, each user initializes her tag with the rights she buys to use the public transportation. Whenever she wants to use a public transportation, she presents her pass in front of an NFC reader. Locally, the reader checks the rights stored in the tag's RAM and opens the gate, if the access is granted.

Ubi-Check is an academic application example of distributed architectural pattern [21]. An RFID tag is attached to each of a traveler's items. At the beginning of their travel, 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). If an inconsistency is found among the read values, it means that, at some point, the traveler exchanged one of her items with the item of another traveler. An alarm is thus triggered to warn the traveler that one of her items is missing.

[22] proposes an academic system based on digital pheromones to find objects lost in a house. To do so, floor of the house is covered with RFID tags. An RFID reader is coupled with each house object. When user moves an object from point *A* to point *B*, the RFID reader associated with the object behaves like an ant which sets pheromones on the path it takes: The reader writes a digital pheromone (made of object identifier and timestamp of transit) in the RAM of each tag over which it goes. Notice that, like a natural pheromone which evaporates with time, whenever a reader finds no more room in the RAM of a tag (there are too many pheromones stored inside), the reader deletes the oldest pheromone from the tag. In case an object is lost, user takes a dedicated RFID reader and wanders around the house until she finds the digital pheromones of the object. Once she has located it, she follows the pheromone trace until the place where the object was left.

Roboswarn is an (academic) application to position robots (equipped with NFC readers) in a physical space to accomplish a certain task [23]. NFC tags are dispatched in dedicated places of a room (for instance, near a hospital bed which these robots will have to push so that a cleaning robot can accomplish its task). Each tag is initialized with location of other tags in the room and the timestamp of last cleaning. When robots enter the room, they look for an NFC tag. As soon as one robot finds one, it reads the position of other tags and transmits them to other robots. The other robots go to the other tags. If timestamp of last cleaning is too old, robots push the hospital bed and then write new timestamp of cleaning. Otherwise, robots do nothing.

SALTO Systems company is selling locks for electronic doors. The keys are NFC tags. To facilitate the management of all locks and tags, this company has developed SALTO Virtual Network (SVN) [24]. Thanks to this system, Heathrow airport operator is able to manage 1000 standard electronic locks (NFC-controlled) and 37 hot spots. These spots are special locks connected to a global computer network. They can: 1) unlock an entry access on the whole site, 2) initialize an NFC key with the right to open given locks during the day, 3) blacklist some NFC keys, 4) recover data collected by the key during the working day of its user. Indeed, each time a person unlocks an electronic lock with her NFC key, the lock reads data stored on tag to check user permissions and the list of blacklisted tags. But, the electronic lock also writes information like, for instance, the low charge of the battery powering the lock. Thus thanks to SVN, even though standard locks do not have access to a global computer network, they can receive information (e.g. list of blacklisted cards) and send information (e.g. low charge of battery): Standard locks communicate thanks to the network made of the users of the keys/tags.

Based on all of these examples, next section analyzes distributed architectural pattern.

5.2. Analysis

Concerning the functional attribute, as the avatar is written and read only in the RAM of the tag, there is no staleness issue of locally read tags. However, it is impossible to know the avatar of a remote tag.

Concerning the scalability attribute, there is no limit on the number of tags in the application environment. Moreover, such distributed architectural pattern is not sensitive at all to the number of simultaneous read operations (all of the operations are done locally).

Concerning the cost attribute, the reader does not need any global network to access to the avatar of the RFID tag. On the other hand, RAM is required on each tag. Its size must be at least the size of the avatar. This means that the avatar cannot contain too much information (e.g. MIFARE tags can offer up to 4 Kbytes of RAM, with 3440 bytes of net storage capacity). When a new tag is introduced in the system, only two operations are required: (i) the tag is linked to the physical entity; and (ii) the avatar of this entity is initialized in the RAM. About the reinitialization of the tags, it is application-dependant. Some applications require that a dedicated user goes through all of the tags to reinitialize them. In the case of Navigo pass, users are in charge of bringing their pass to a vending machine. This leads to long waiting lines at the beginning of a month, when users must initialize their rights for this month. This is why Navigo operator carries out experiments where users can initialize their tag using a dedicated NFC reader connected to their personal computer. To avoid reinitialization costs, some RFID-based distributed applications put in place special mechanisms. These mechanisms take into account elapsed time in order to automatically reset data. In Roboswarm application (see section 5.1), there is no need to reset the timestamp to trigger a new cleaning of a room. Each robot is aware of a deterioration level. Thus if the timestamp plus this deterioration level is greater than current time, it means that the room needs some cleaning again. With application for pheromone-based object tracking (see section 5.1), although tags have limited RAM capabilities, there is still no need to have a periodic session initialization which would clean up outdated pheromones. Each pheromone is written on a tag with a timestamp. Thus when the device attached to the roaming object meets a tag, it cleans up pheromones which have a too old timestamp, before writing the dedicated pheromone.

This section has analyzed distributed architectural pattern according to the attributes presented in section 2. This architectural pattern does not require any global computer network. And it is not sensitive to the number of simultaneous read operations. Nevertheless it faces a functional issue: it is not possible to get the avatar of a remote tag.

Next section analyzes RFID-based DSM which tackles this issue.

6. RFID-based distributed shared memory architectural pattern

RFID-based distributed shared memory (RFID-based DSM) mixes the qualities of semi-distributed architectural pattern and distributed architectural pattern [7]. The avatar of an RFID tag is stored in the RAM of the tag. In addition, each tag and each mobile device of the application environment holds a local copy of all of the avatars (see Figure 4). Moreover they hold a vector clock (see Figure 5). Each element of this vector clock is a number corresponding to the last version of the avatar which the tag or the device has learnt about (this is why [25] gives the name *version number* to this number). When a mobile device comes to a tag and modifies the avatar of the tag, this mobile increments the element of the vector clock (stored on the tag and inside its own memory) corresponding to the avatar of this tag. Whenever a mobile device meets a tag (respectively another device), the device and the tag (respectively the other device) compare their respective view of the avatars, by comparing their vector clocks values. Doing so, each of them learns from the other one the latest news (which they are aware of) about all of the avatars.

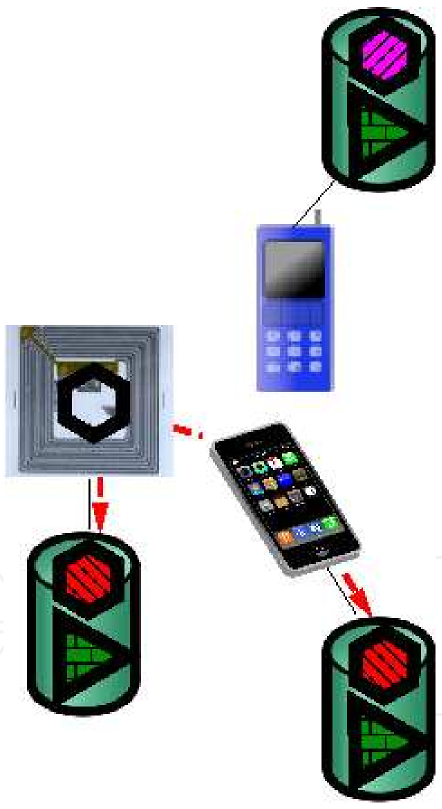

Figure 4. RFID-based distributed shared memory architectural pattern

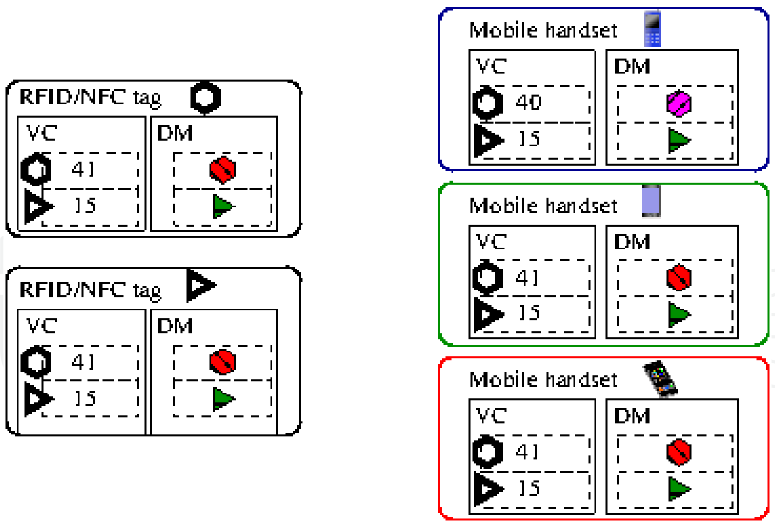

Figure 5. Data of RFID-based distributed shared memory

Next section presents an example of this architectural pattern.

6.1. Example

Plug: Secrets of the museum, an (academic) pervasive game [26] developed in the context of the *PLUG* research project [27], is the unique example of use of RFID-based DSM architectural pattern. In this game, 48 virtual playing cards represent objects of French Museum of Arts and Crafts (*Musée des arts et métiers*). These cards are dealt between 16 NFC tags (1 card per MIFARE tag, each of them being equipped with 1 KB of RAM) and 8 mobile phones (4 cards per Nokia 6131 NFC mobile). The players' goal is to collect cards of the same family on her mobile. To do so, players use their mobile to swap a card with a tag or another mobile.

Next section analyzes RFID-based DSM architectural pattern.

6.2. Analysis

Concerning the functional attribute, whenever a mobile device comes near a tag, there are two possibilities. Either the tag has been already initialized; in that case, as the avatar is stored on the tag, the value read on the tag is the most up-to-date. Or, the tag has not been already initialized; in that case, the first task of the mobile device is to initialize the tag; so that the value read on the tag after this initialization is also the most up-to-date value. Thus there is no staleness issue for avatar of locally read tag. Moreover, a mobile device holds a local copy of all avatars. Thus by querying this local copy, the device is able to answer to queries concerning a remote tag. However this local copy may not be up-to-date: There is a staleness issue for avatar of remotely read tag.

Concerning the scalability attribute, the maximum number of tags is limited by the size of the RAM of the tags. This architectural pattern stores copies of the avatar of all of tags and a vector clock. Let S_{tag} be the lowest size of the RAM of the tags present in the environment. Let L be the length of an element of the vector clock. Then the maximum number of tags is bounded by $S_{tag}/(s + L)$. Let's compare this bound to the bound of semi-distributed architectural pattern. For the latter pattern, the numerator is expressed in terms of Gigabytes. However it is expressed in terms of Kilobytes in the case of a tag's RAM. The maximum number of tags for RFID-based DSM architectural pattern is at least one million times lower than the maximum number for semi-distributed architectural pattern. About sensitivity to the number of simultaneous reads, RFID-based DSM requires that all mobile devices synchronize with a dedicated machine: RFID-based DSM is as sensitive as semi-distributed architectural pattern.

Concerning the cost attribute, in RFID-based DSM architectural pattern, the reader does not need any global network to access to the avatar of the RFID tag. Nevertheless RAM is required on each tag. Its size must be at least the size of the avatar times the number of tags in the environment (so that a tag can hold a local copy of all avatars). This means that the avatar can contain even less information than in the case of distributed architectural pattern.

When a new tag is introduced in the system, four operations are required: (i) the tag is linked to the physical entity; (ii) the avatar of this entity is created and initialized in DB_{init}, the database used to (re)initialize the local copy on each mobile (DB_{init} is stored on a dedicated machine which can be one of the mobiles); (iii) a link between the tag identifier and this avatar is created in DB_{init}; and (iv) all of the elements of the tag's vector clock are initialized to zero.

When tags must be reinitialized, a program P_{init} is executed on the machine hosting DB_{init}. This program computes the initial value VC_{init} of the vector clock for this session, so that each element of VC_{init} is greater, thus more recent, than all the vector clock elements in the mobiles. To do so, P_{init} can use two methods. The first method is twofold: (i) get the vector clocks of all of the mobiles; and (ii) compute the maximum value. This first method does not require additional memory on each tag, but requires additional communication between the mobiles and the dedicated machine. This method works because the vector clock of a tag evolves only when a tag is in contact of a mobile. Thus there is always at least one mobile device which is aware of the values stored in the vector clock of a tag: P_{init} does not need to be aware of the vector clock values stored on the tags. The second method supposes that each vector clock element is made of two fields: a "session identifier" field and a "tick in this session" field. Thus P_{init} has only to increase the session number and set all "tick in this session" fields to zero. This second method requires additional memory on each tag, but no additional communication between the mobiles and the machine running P_{init}. The choice of the method is application dependant. Once one of the two methods has been applied, the dedicated machine synchronizes each mobile device by sending the contents of DB_{init} and VC_{init} to the device. Afterwards, whenever a mobile device is in contact with an uninitialized RFID tag for this session, as the mobile device vector clock is greater than the tag vector clock, the mobile device initializes the tag. In other words, RFID-based DSM architectural pattern takes advantage of the fact that application users will go to tags, to initialize them: This pattern uses the communication network made by application users, instead of using a global computer network.

This section has analyzed RFID-based DSM architectural pattern according to the attributes presented in section 2. This architectural pattern does not require any global computer network. And it does not experience the issue of staleness of an avatar of a read tag. Moreover it is possible to query the avatar of a remote tag. Nevertheless this architectural pattern experiences staleness issues when accessing to avatar of a remote tag. And there is a scalability issue in terms of maximum number of tags which can be handled.

By synthesizing the conclusions observed for the different architectural patterns, the next section provides guidelines for choosing the most adequate pattern for a given application.

7. Guidelines for choosing the right RFID-based architectural pattern

Table 1 synthesizes the analysis of the different aspects of the architecture attributes made on all of the architectural patterns. In this table, values which are in italic correspond to aspects which are a limitation for this architectural pattern.

If the application requires the best level for all aspects of functionality attribute, then the centralized architectural pattern must be chosen. It is the only architectural pattern which experiences no issues within the functionality attribute. But this pattern has an operational cost due to the requirement for a global network. And this pattern is highly sensitive to the number of simultaneous reads.

	Central.	**Semi-distr.**	**Distributed**	**RDSM**
Staleness of locally read tag	No	*Yes*	No	No
Avatar of remote tag	Yes	Yes	*No*	Yes
Staleness of remote read tag	No	*Yes*	n.a.	*Yes*
Maximum number of tags	$S_{central}/s$	S_{local}/s	Infinite	*$S_{tag}/(s+L)$*
Sensitivity to number of simultaneous reads	*High*	Medium	None	Medium
Network required	*Yes*	No	No	No
RAM required on tag	No	No	*Yes*	*Yes*
Cost of introducing a tag (most costly operation)	Link tag to physical entity	Link tag to physical entity	Link tag to physical entity	Link tag to physical entity
Cost of reinit. Tags (most costly operation)	Reinit. Database	Sync. Database	*Go to all tags*	Sync. database

Table 1. Comparison of the RFID-based architectural patterns (italic values signal a limit for this architectural pattern)

If one of these last two issues is a problem, the system architect must consider the three other architectural patterns. Semi-distributed architectural pattern must be chosen if RFID tags cannot host RAM. This constraint may be due to cost motivations, but also technical constraints (use of low-frequency tags, see section 2).

If there can be RAM on tags, the maximum number of tags must be determined. If it is compatible with RFID-based DSM architectural pattern limitations, then this pattern should be chosen (as it is the least limited pattern for the functionality attribute). Otherwise the system architect should choose distributed architectural pattern (if there must be no staleness issue for read tags) or semi-distributed architectural pattern (if the cost of reinitializing tags is an important factor). Notice that the mixing of distributed architectural pattern and RFID-based DSM may be an interesting alternative. On each tag, we can store its avatar and the vector clock element corresponding to this avatar. Each mobile device holds a copy of all avatars and a full vector clock. By applying RFID-based DSM procedures, we get a solution for the limited maximum number of tags in RFID-based DSM. And in the same time, we solve distributed architectural pattern limitations (as we can query avatar of remote tags and we reduce the high cost of tag reinitialization).

To illustrate the use of these guidelines, let's consider the choice of the architectural pattern for the RFID-based game *Plug: Secrets of the museum* presented in section 6.1.

Each tag costs about 0.10 euro (respectively 1.50 euro) if it has 0 KB (respectively 1 KB with 752 bytes of net storage capacity, S_{tag}=752 bytes) of RAM. The avatar of a tag is the virtual card "contained" in the tag. There is a maximum of 16 cards in the game. Thus the avatar is coded as a byte value (s=1 byte). Concerning the vector clock, the project uses the synchronization method requiring a session identifier. To have an ever-increasing value, "session identifier" field stores the initialization time. This time is the difference, measured in milliseconds, between the session initialization time and midnight, January 1st, 1970 UTC. This storage requires 8 bytes per tag. Moreover, each tag holds the "tick in this session" field of each avatar stored in the tag.

This field is coded as a short (L=2 bytes). It represents a real-time clock, formatted as the number of seconds since the beginning of the game session (A session lasts less than 2 hours: there is no risk of overflow). This clock is the time known by the tag of the last update of the avatar of another tag. It takes about 20 minutes to attach each of the 16 tags to their correct location, so an average of 75 seconds per tag. Linking the tag and the avatar takes about 5 seconds per tag. Initializing the avatar is done in a few milliseconds by an initialization program. For reinitializing tags, synchronizing all of the 8 mobiles with a dedicated machine takes about 1 minute, that is an average of 4 seconds per tag. Notice that synchronization is based on NFC peer-to-peer communication. The project could have saved synchronization time if it has used Bluetooth®, but it would have used more battery.

If the project is going to use centralized or semi-centralized architectural pattern, it will use a dedicated machine for hosting the central database. This machine will be equipped with a 500 gigabytes disk ($S_{central}$=500 GB). In the case of the semi-distributed architectural pattern, the project will use half of the micro-SD memory of each mobile phone to host the local copy of the database (S_{local}=1 GB). If the project is going to use centralized architectural pattern, each mobile will have to be equipped with a SIM card giving access to a UMTS data plan. This will cost 15 euros per mobile per month. If the project is going to use distributed architectural pattern, it will take 13 minutes to go by all of the 16 tags to reinitialize them, so an average of 49 seconds per tag.

We apply these numeric values to table 1. Table 2 synthesizes the results. In this table, values which are in italic correspond to criteria which are a limitation for this architectural pattern.

Centralized architectural pattern requires a global network which costs 120 euros per month. The museum which hosts the game considers it is too expensive. We have to turn to one of the other architectural patterns. As the game must manage 16 tags and as the RFID-based DSM can handle a maximum of 248 tags (as *s*=1 byte), we can choose this architectural pattern. However, if s had been 250 bytes, this pattern could have handled only 4 tags: It would not have fitted. As there must be no issue about avatar of read tags (the game would not be fun), we would have chosen distributed architectural pattern (or combination of distributed and RFID-based DSM patterns, in order to reduce the costs of reinitializing tags).

8. Conclusion and future work

This chapter studies four RFID-based architectural patterns: centralized, semi-distributed, distributed and RFID-based DSM. It compares them according to nine aspects of three architecture attributes: functionality, scalability and cost. Despite their specific limitations, each architectural pattern fits the requirements of existing applications.

	Central.	**Semi-distr.**	**Distributed**	**RDSM**
Maximum number of tags if s=1 byte (s=250 bytes)	500 x 109 (2 x 109)	109 (4 x 106)	Infinite (Infinite)	248 (2)
Cost of computer network (per month)	*120 euros*	0 euro	0 euro	0 euro
Cost of tag (per tag)	0.10 euro	0.10 euro	*1.50 euro*	*1.50 euro*
Cost of introducing a tag (in seconds per tag)	80 s/tag	80 s/tag	80 s/tag	80 s/tag
Cost of reinitializing tags (in seconds per tag)	0 s/tag	4 s/tag	*49 s/tag*	4 s/tag

Table 2. Comparison of the RFID-based architectural patterns in the case of the game *Plug: Secrets of the Museum* (italic values signal a limit for this architectural pattern)

The chapter proposes guidelines for choosing the RFID-based architectural pattern which will best fit a given application requirements. These guidelines are tested in the context of an RFID-based pervasive game.

Future work concerns the analysis of these architectural patterns with respect to security architecture attribute. Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users [1]. This future work will determine the influence which the level of resistance to security attacks and the cost of implementing such resistance have on the guidelines provided in this paper. Another attribute we would like to study is the fault-tolerance of the different elements of the system.

Author details

Michel Simatic

Address all correspondence to: Michel.Simatic@telecom-sudparis.eu

INF Department, Télécom Sud Paris, Évry, France

References

[1] Bass L., Clements P., and Kazman R., Software Architecture in Practice, 2nd Edition. Addison-Wesley Professional, April 2003, ISBN-13: 978-0-321-15495-8.

[2] Armenio F., Barthel H., Burstein L., Dietrich P., Duker J., Garrett J., Hogan B., Ryaboy O., Sarma S., Schmidt J., Suen K., Traub K., and Williams J., “The EPCglobal architecture framework,” GS1 EPCglobal, Tech. Rep. Version 1.2, September 2007.

[3] Gonzalez L., RFID: Stakes for the enterprise! (in French). Afnor Editions, October 2008, ISBN-13 978-2-12-465153-5.

[4] Roussos G. and Kostakos V., “RFID in pervasive computing: State-of-the-art and outlook,” Pervasive Mob. Comput., vol. 5, no. 1, pp. 110–131, February 2009.

[5] 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 (last access in July 2012), December 2006.

[6] “Smart Poster Record Type Definition, Technical specification SPR 1.1,” NFC Forum, 2006.

[7] Simatic M., “RFID-based replicated distributed memory for mobile applications,” in Proceedings of the 1st International Conference on Mobile Computing, Applications, and Services (Mobicase 2009), San Diego, USA. ICST, October 2009.

[8] Aspire RFID: OW2 Aspire RFID: an RFID suite for SMEs. Available at http://wiki.aspire.ow2.org (last access in July 2012), 2011.

[9] Rashid O., Bamford W., Coulton P., Edwards R., and Scheible J., “PAC-LAN: mixed-reality gaming with RFID-enabled mobile phones,” Computers in Entertainment, vol. 4, no. 4, pp. 4–20, October–December 2006.

[10] Haberman O., Pellerin R., Gressier-Soudan E. et Haberman U. (2009). RFID painting demonstration. In Natkin S. and Dupire J., éditeurs : Entertainment Computing - ICEC 2009, volume 5709 de Lecture Notes in Computer Science, pages 286-287. Springer Berlin / Heidelberg.

[11] Pellerin R., Adgeg G., Delpiano F., Gressier-Soudan E., and Simatic M., “Gasp : a middleware for mobile multiplayer games”. http://gasp.ow2.org (last access in July 2012), July 2007.

[12] Pellerin R., Delpiano F., Duclos F., Gressier-Soudan E. and Simatic M. (2005) “Gasp : an open source gaming service middleware dedicated to multiplayer games for J2ME based mobile phones”. In 7th Int. Conference on Computer Games CGAMES'05 Proceedings, pages 75-82.

[13] Heumer G., Gommlich F., Jung B. and Müller A. (2007). Via Mineralia - a pervasive museum exploration game. In Proc. of Pergames 2007, Salzburg, AT.

[14] Touchatag: Using the Advanced HTTP Application. Available at http://www.touchatag.com/developer/docs/applications/advanced-HTTP (last access in July 2012), 2010

[15] Activision: Skylanders : Spyro's adventure. Available at http://www.skylanders.com/) (last access in July 2012), 2011.

[16] Planck S.: Kids going crazy for Activision Skylanders NFC game. Available at http://www.nfcrumors.com/01-17-2012/kids-crazy-activision-skylanders-nfc/) (last access in July 2012), January 2012.

[17] NFC Forum. NFC Data Exchange Format (NDEF) - Technical Specification 1.0. Rapport technique, NFC Forum.

[18] NFC Forum (2006c). URI Record Type De?nition - Technical Specification 1.0. Rapport technique, NFC Forum.

[19] Connecthings : 08/12/2011 - BAL intelligente @ mobulles Paris @ Lab Postal 2011 (Intelligent mailbox, in French). http://www.connecthings.com/node/123 (last access in July 2012), December 2011.

[20] Levallois-Barth C., “Navigo: simplification or absolute traceability”. FYP éditions, May 2009, ch. 5 – Security and data protection, in Evolution of digital cultures (in French), pp. 173–181, ISBN-13 978-2-91-657113-3.

[21] Couderc P. and Banâtre M., “Beyond RFID: The Ubiquitous Near-Field Distributed Memory,” ERCIM news, no. 76, pp. 35–36, January 2009.

[22] Mamei M. and Zambonelli F. (2007). Pervasive pheromone-based interaction with RFID tags. ACM Trans. Auton. Adapt. Syst., 2(2):4.

[23] Zecca G., Couderc, P., Banâtre M. et Beraldi R. (2009). « Swarm robot synchronization using RFID tags. In Pervasive Computing and Communications,” 2009. PerCom 2009. IEEE International Conference on, pages 1-4.

[24] SALTO Systems: SALTO Networked Locking System - SVN. Available at http://www.saltosystems.com/index.php?option=com_content&task=view&id=62&Itemid=57) (last access in July 2012), 2007.

[25] Murphy A. L. and Picco G., “Using LIME to Support Replication for Availability in Mobile Ad Hoc Networks,” in Proceedings of the 8th International Conference on

Coordination Models and Languages (COORD06), vol. 4038, Bologna, Italy. Springer Lecture Notes on Computer Science, June 2006, pp. 194–211.

[26] Simatic M., Astic I., Aunis C., Gentes A., Guyot-Mbodji A., Jutant C., and Zaza E., "'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, ser. Lecture Notes in Computer Science. Springer, September 2009, pp. 302–303.

[27] "PLUG: Play Ubiquitous Games and play more," http://cedric.cnam.fr/PLUG/ (last access in July 2012), August 2009.