SmartMart: IoT-based In-store Mapping for Mobile Devices

Dylan Hicks
Columbia University
dth2115@columbia.edu

Kevin Mannix
Boston University
kmannix@bu.edu

Hannah M. Bowles
Texas State University
hmb42@txstate.edu

Byron J. Gao
Texas State University
bgao@txstate.edu

***Abstract*—Quite often, when shopping in a supermarket (e.g. Walmart), shoppers are frustrated at locating the items on the shopping list and no assistance is available. On the other hand, retailers also lose a large volume of sales as a result. In this paper, we present a feasibility study that leverages the Internet of Things (IoT) technology to make store items "smart" so that they can automatically register and update their location information in an information retrieval system, allowing shoppers to search, locate, and map them on the store floor plan using mobile devices. A free-accessible Android-based mobile app `SmartMart` has been developed to demonstrate the promise of this preliminary work. Continuous development of this research could lead to a complete change in our day-to-day shopping experience.**

***Keywords*—Internet of things, in-store mapping, information retrieval, mobile devices.**

I. Introduction

Supermarkets are self-serving in nature, where customers use shopping carts in the store, search for the items they want to buy, place them into the carts and then proceed to the checkout counters. With little to none assistance, locating the shopping items in a big, sometimes mazelike, store can be very time-consuming, physically exhaustive and mentally frustrating. On the other hand, retailers lose about 20% of sales as a result [14]. Recent technological advances in mobile devices, indoor positioning and information retrieval have enabled various opportunities of turning this loss-loss situation to win-win. In this `SmartMart` project, we investigate the feasibility of seamlessly integrating these technologies to facilitate friendly and effective personal in-store shopping assistance. We have implemented and deployed a complete system including a free-accessible Android-based mobile app for demonstration purposes. Continuous development of this research could lead to a complete change in our day-to-day shopping experience.

Fig. 1 shows the main principled architecture for such mobile-based in-store mapping services. The Indoor Positioning System functions to "automatically" locate dynamic store items and send updated location information to the server for indexing. The Information Retrieval System maintains an inverted index and a database for store items containing catalog as well as location information. It functions as a server to receive query requests from clients (mobile devices) over the Internet, process them, and send ranked query results back to the clients. Customers interact with their smart mobile devices. With a mobile app (e.g., `SmartMart`) installed, such mobile devices function as clients, sending queries to the server and receiving query results through a friendly keyword search interface. The mobile app also provides a floor mapping interface that maps (selected) query results on the store floor plan. Additional useful features include in-store navigation, computing shortest paths for a list of items, incorporating advanced search capacities such as faceted search [13], [12] that provides progressive query refinement and personalized search [8], [10], [4] that provides search personalization. A third party mobile app can also possibly provide cross-store services if proven to be useful.

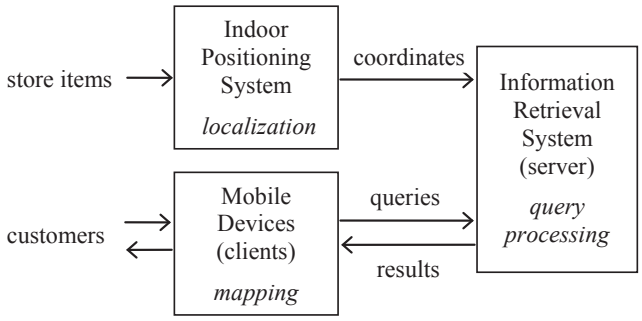


Fig. 1. Main architecture

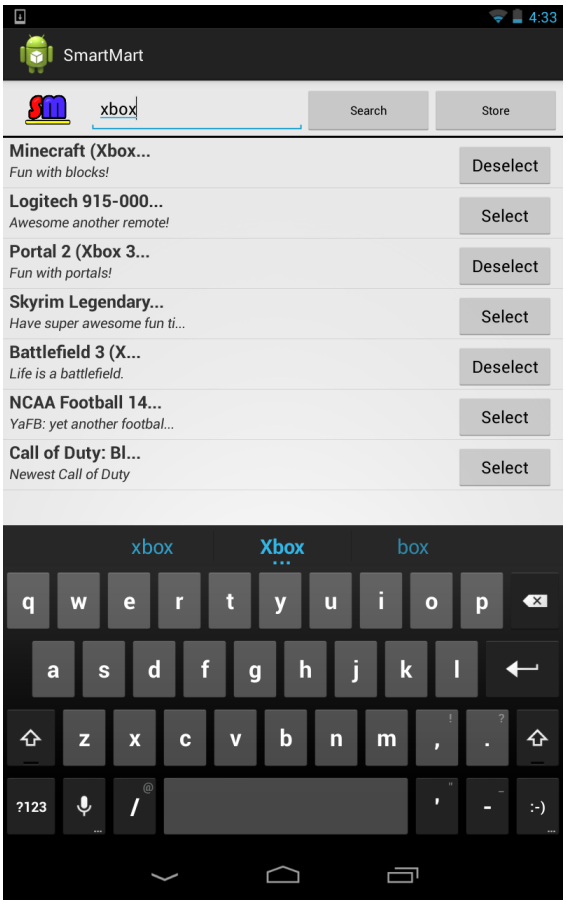
Store items are typically not statically located. They are placed on/off the shelves and moved around based on availability, popularity, and seasonal promotions. Manual registration of dynamic locations for hundreds of thousands of items can be very labor-intensive and error-prone. A carefully designed and deployed indoor positioning system (IPS) has the potential to greatly reduce this burden, enabling cost-effective in-store mapping services. An IPS is a network of devices used to wirelessly locate objects in real-time inside a building. It can be used alone, or in combination with manual registration as a hybrid approach for our application. Currently there is no de facto standard for IPS design, and any wireless technology can be used for locating based on cost, accuracy and reliability factors [5], [1]. Note that the popular global navigation satellite systems (GPS) are generally not suitable to establish indoor locations, since microwaves will be attenuated and scattered by roofs, walls and other objects.

In this preliminary study, we adapt existing RFID (radio-frequency identification) indoor positioning technology [11], [6], [3], [7] to the store setting to automatically read, compute,

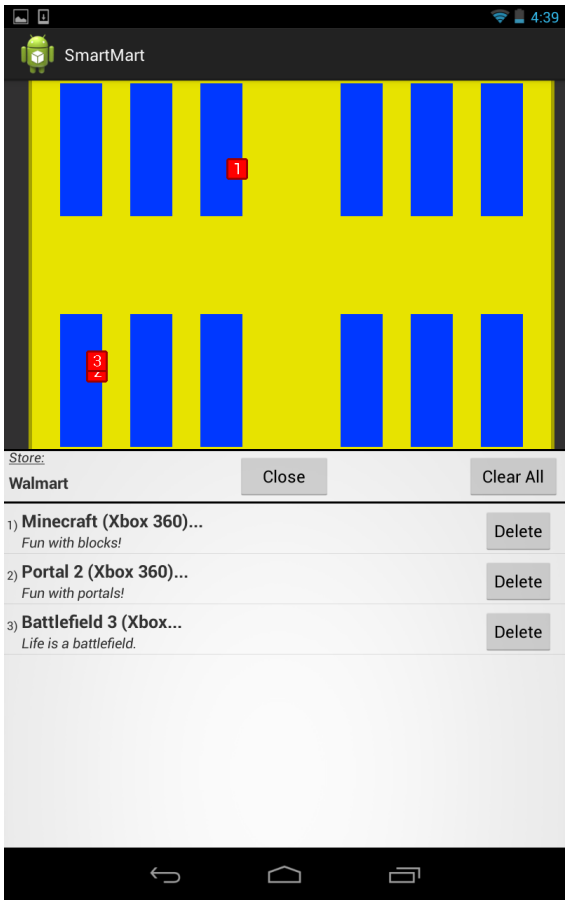
This research was supported by the Research Experiences for Undergraduates (REU) program from the National Science Foundation (NSF) under grant No. OCI-1062439. Dylan Hicks, Kevin Mannix, and Hannah M. Bowles participated in the 2013 REUIR summer program at Texas State University.

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254116



(a) Search screen



(b) Map screen

Fig. 2. Screenshots of `SmartMart`

and update item locations. RFID tags will be attached to items or price tags of items. RFID readers will be (optimally) configured to capture ID signals of RFID tags and locate them based on signal strength readings using localization algorithms.

RFID is a prerequisite and main enabler for the Internet of Things (IoT). IoT represents a fascinating vision for the next generation of Internet, where everyday physical objects are attached with sensors and seamlessly integrated with the Internet [2], [9]. IoT brings endless opportunities and will impact every corner of our planet. With IoT, we can build smart cities where parking space, urban noise, traffic congestion, street lighting, irrigation, and waste can be monitored in real time and managed more effectively. We can build smart homes that are safe, energy-efficient and convenient. We can build smart environments that automatically monitor air and water pollution and enable early detection of earthquake, forest fire and many other devastating disasters. Other applications include smart agriculture, smart animal farming, smart health and so on. This `SmartMart` project will be an interesting addition to the exciting family of smart IoT applications. To our best knowledge, `SmartMart` is the first system and mobile app of this kind. Existing shopping apps (e.g. Walmart Mobile App) do not provide location-based services. It is also quite different from Google Indoor Maps that only displays indoor floor plans and does not support searching and mapping of dynamic store items.

II. SmartMart Design and Implementation

In this section, we explain the design and implementation details for the three components of `SmartMart`: the client module, the server module and the indoor positioning module. In this preliminary study, we focus on building a complete functioning prototype that seamlessly integrates the three components. We leave sophisticated optimization of the individual components for future development.

A. The Client Module

Overview. The client side of `SmartMart` is in the form of an Android application, coded in Java, for any mobile device that runs the Android OS. The mobile app features two screens: a search screen and a map screen, as shown in Fig. 2. The search screen allows the user to type in keyword queries (e.g., xbox, grape), view ranked search results returned by the server, and identify/select the results of interest. The map screen maps the selected results on the corresponding store floor plan. To move between the two screens, simply swipe right or left on the mobile device.

Currently in `SmartMart`, a store floor plan is stored as an XML file containing geometric descriptions of physical features of the floor, also including detailers of fixtures such as entrances, hallways, and shelves. The XML file will be rendered smoothly at any desired display size on mobile

devices. An item location is specified as (x, y) coordinates, which can be easily mapped to the rendered floor plan image. Since a store can have multiple floors, each item is associated with the ID of a particular floor. `SmartMart` is designed to provide a universal store mapping service for arbitrary number of stores. The stores only need to upload the following information to the server: store item database, floor plan XML files in a pre-defined (straightforward) format, and item locations. The localization module and the server will work together to automatically update item locations. On the client side, the user can easily choose any registered store to begin the search.

Implementation. At the implementation level, the application is divided into two separate main Activities: the Search Activity and the Map Activity. One might imagine an Activity as a single screen in a mobile application. For example, a log-in screen may be one Activity while a settings page may be another within the same application. These Activities are essentially Java classes that extend the super class *Activity*, which has been created by Android developers. Android can also employ Java classes that are not Activities, such as object classes or helper classes. Each Activity must have an associated layout, which is defined through an Android XML Document and called on the creation of each Activity. Android developers have special XML tags that apply to Android Applications to more easily help create an Activity's layout, including tags such as *Relative Layout* (where each object in the layout is defined in relation to the others) and *Linear Layout* (where each object is defined chronologically and placed in the order in which they are defined). There are more tags to accommodate almost any type of layout desired, but only Relative and Linear Layouts are used in this project. The objects within the layouts are mostly children of the View class, created by Android developers. These include objects such as Text, Buttons, Images and Lists among many other more specific yet useful objects Android has defined for use in their applications.

Search Activity. The Search Activity provides a search interface allowing the user to issue queries. When the user enters text and submits the query by pressing the Search button, a query is sent to the server. An XML file is returned from the server with up to 15 results. Upon receiving this XML file, it undergoes parsing to separate each item into a separate custom object, with fields to contain its location (in x and y locations), description, title, and other numerous identification fields such as its UPC number. These objects are then stored in a List View (essentially an Android-defined layout object that creates a scrollable list) with a custom Adapter (Search Adapter). This Adapter is simply the way in which each row of the list will be presented. Android comes equipped with pre-made adapters, but more often than not it is appropriate to use a custom adaptor to easily adjust each row to the desired constraints. The Search Adapter requires every row to have two rows of text, each positioned to the left of the screen, with the bolded title of the item on top and a snippet of description below this title. To the right margin of the screen is the Select button. On default, this button will be labeled *Select*, but switches to *Deselect* when the button is pressed. This item will then be added to the *selected list*. The selected list is stored in a SQLite Database for easy access in each Activity.

Besides the Search button, there is a Store button that allows the user to choose one from the many available stores. The layout simply consists of a List View with a custom adapter. When this function is started, Android's asset manager is used to get an array of the paths to each XML file in the assets/map folder (each XML file is simply a map defined in XML). Each XML file found in that location is partially parsed until all the information is found. This information is its name and location (street, city, state, zip code) and each piece of information is defined by unique tags and usually placed near the top of the file. Each file becomes its own *Map Items* object and is placed in a global Array List. After all files in the specified map folder have been parsed, the custom adapter is given the Array List of Map Items and configures each row to present the proper information for each map (bolded store name on the left side and the address on the right side). On a Long Press (defined by Android as holding a finger down on a row for 2 or more seconds), the List View item pressed down on will be chosen as the map to be displayed, and a Toast (an Android-defined dialog-box of sorts) will be displayed to confirm that the map has been added correctly. When a map is selected, an *Extra* is added to the intent that will eventually start the Search Activity. This extra consists of a string with the file path to the selected map, and is transferred from the Search Activity to the Map Activity in the form of a string Extra as well.

Map Activity. The Map Activity contains the bulk of the workload done by the Android application. This Activity creates a bird's-eye-view map of a particular store through the Android Draw class. This map can be zoomed both in and out of and moved around with the touch of a finger. It also populates a List View with items the user selected in the Search Activity. For each item in the selected list, a red rectangular marker has been placed on the location of that item on the store map. Each item has a number associated with it, and this number is displayed on the map marker. One item can have multiple locations in the same store. The layout consists of the generated map with a row on the bottom of the map, consisting of the store name on the left of the row, an Open button in the middle of the row, and a Clear List on the right side on the row. On the click of the Open button, this row and the List View underneath slide about halfway up the screen of the Android device, displaying the items in the List View. Each item (row) in the List view can be deleted, resulting its marker being removed from the map. The List View is then repopulated and the numbering is redone as to not have any gaps in the number order. The Clear List button will remove all items and their respective markers from the list and map, respectively.

On the start of this Activity, Android's Asset Manager parses the XML file for a chosen map. This XML file can contain an arbitrary number of tags. Each map is broken into three distinct parts: the floor, the shelves, and the background. The XML files contain the tags for each floor and shelf rectangle (each specified by a unique tag), and within these tags exist the left, top, right, and bottom locations of each rectangle (also indicated by a unique tag for each location). Multiple rectangles of the same type can be drawn to be placed together or on top of one another to give the correct shape for a floor layout. The background is drawn first and has its dimensions calculated for the most extreme points of the floor,

with extra added on to give a buffer between the floor and the edges of the drawn map. The floor is then drawn on top of the background. The floor is divided into two parts, currently the background and the foreground. The foreground is defined by the coordinates given in the XML file, while the background arises from the foreground's dimensions, magnified by a small amount and with a slightly darker color than the background. This gives the floor a small border and overall a cleaner, more focused look to the viewer. The shelves are then drawn on top on the floor as defined by the coordinates parsed from the XML document. The Canvas (an Android class) that the shelves, floor, and background have been drawn on is then set as the image background for the Image View defined as the map in the layout XML file.

A Matrix (an Android class) is also set to be associated to this Image View for zooming and scrolling purposes. Using the Touch Listener provided by Android and placed on the map Image View, the motion of the user's finger is tracked and checked at each movement, no matter how slight. This allows the map to be adjusted, but not escape the bounds defined by the Android device's screen (for example, the map cannot go too far to the left or right off the screen). The bounds to which the map is confined is calculated through finding the layout parameters of the View objects within the layout and adjusting these for the screen density, as every Android device will have different screen size specifications. This makes the application dynamic and fitting for every device. The List View is then set with a custom adapter with the selected list chosen by the user in the Search Activity screen. Each row has the item's bolded name to the left with a short description in italics beneath, and a Delete button near the right margin. The header (the row with the store name, Open button, and Clear List button) as mentioned above and the List View below it can be considered as one large object. They move together and are essentially attached. On the click of the Open/Close button, an Object Animation occurs that slides the header and List View up to approximately the middle of the page. As described above, items can be deleted from the selected list. The item is deleted from the SQLite Database containing all the items in the current selected list and the adapter is reset. The header and List View object is similar to the Android-defined Sliding Drawer object, but as this type of object has been deprecated, it has been decided to recreate the Sliding Drawer object with more static View objects.

B. The Server Module

Overview. The server is the "engine" of `SmartMart`. It connects to the localization and client modules, updating location information of store items, processing queries sent from clients, and providing location-based information retrieval services. It maintains a database of all active store items, including their location information. To enhance portability, the database is in the form of an XML file. Each item in the database has a unique ID, along with as many additional fields as the user desires, such as product name, description, or Universal Product Code (UPC). Every item also has a unique RFID code for each product location and its last known position.

Implementation. The "server.php" file parses the XML data file and allows some commands to be performed on the database via TCP port 8000. The most important is invoked by the *lookup* command, which allows keyword queries on the databases. After a lookup command is issued, the client can ask for different pages of the results using the *page* command. The *get* command allows products to be found by ID (for instance if you want more info, or you want to check if it has been updated). Entries can be added to the database by using the keyword "add", then sending the product's XML description. The server will automatically assign the product an unique ID in the database and rebuild the index. Same with the *delete* command, simply specify the IDs to delete and they will be removed from the database and the index. The *update locations* command accepts an XML file of locations and RFID tag numbers from the client. It then searches the whole database and updates the correct entries.

The server constructs an inverted index for each searchable field. The terms in the inverted index are found by applying linguistic preprocessing (tokenization, stop-word removal, stemming, lemmatization) procedures. The server implements a straightforward query processing algorithm under the vector space model. This model takes the term frequency and inverse document frequency and creates a vector representing each document. Then, the query is represented as a vector as well and the cosine similarity between the query and each document is computed. Each field has its own weight so that more important fields, such as product name, score better than tertiary fields, like product description. Then the results are sorted by relevance score and sent to the client side mobile application.

The "index.php" file provides a management interface. The user may easily add/remove/update entries. This page uses all the lookup/add/delete keywords as described earlier, but they are hidden from the user.

C. The Indoor Positioning Module

Overview. Indoor positioning alone is a developing technology and currently there is no de facto standard. While any wireless technology can be used for localization, we feel that RFID (radio-frequency identification) technology is promising due to its decreasing cost and increasing precision. RFID indoor positioning research has received increasing attention in recent years [11], [6], [3], [7]. In this preliminary study, for a proof of concept, we adopt simple localization approaches. For the same reason, we use an active RFID solution in our system, which may not be cost-effective in practice.

RFID devices continuously scan the area for tags, and use the Received Signal Strength (RSS) to calculate the distance. Many algorithms have been explored using multiple readers to maximize accuracy in two dimensions. Methods include basic triangulation, dynamically updating calibration with reference tags, probability algorithms, and K-nearest-neighbor algorithms. The most successful systems must also take into account reader and reference tag placement. Fluctuating signal strengths and interference in a dynamic environment must be compensated for. Fig. 3 shows the raw data taken from our reader. The fluctuations are due to interference such as doorways catching the signal, metal in the building, and people walking by. In a store environment, we estimate that the margin of error that would allow for our app to retain

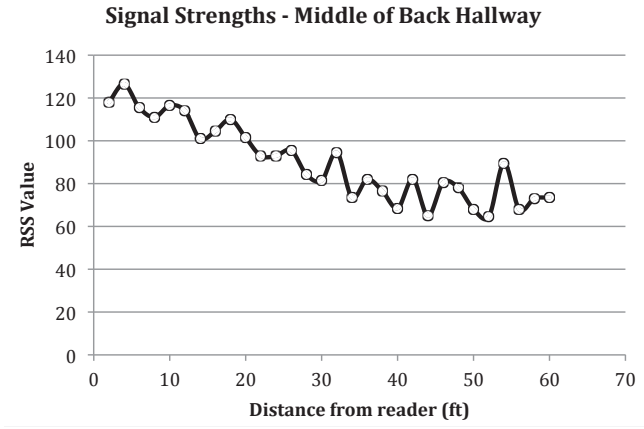


Fig. 3. Localization

functionality is 1-2 meters. We are confident that with multiple readers and careful choice of localization algorithms this can be accomplished.

Implementation. Our system uses a RFID Development Kit containing Wavetrend RX202 long range RFID readers along with active tags, as shown in Fig. 4. The readers work with the tags to capture the presence, identification and location of assets, people, vehicles and alarm triggered events. The readers communicate with a computer via a serial RS-232 or RS-485 protocol for custom software solutions. They are also able to operate on any new or existing Ethernet LAN/WAN network, with optional serial to ethernet converters. Range is adjustable through software. Antennas can be discretely hidden in walls, ceilings, and doorways to identify and track tag activity. A reader can simultaneously read multiple RFID tags at ranges up to 450 feet, and can also register and report the disappearance or unauthorized movement of individual RFID tags.

The readers send structured data packets to a computer via a serial port. They do not have the capability to request data from a specific tag, so they are set to automatically poll the area for tags and randomly sample all tags that are in the area. To request the information, the readers send out a data packet with the appropriate auto poll encoding, and receive back a response packet containing, among other data, the tag ID number and the RSS value. Each tag ID would be associated with a certain store item.

We take incoming RSS values from the reference tags along with a maximum RSS value, and find linear fits. Incoming tag data is then fit to one of those lines to determine its location in one dimension. We believe this piecewise approach maximizes the accuracy we can achieve with one reader. With multiple readers location can be determined in two or even three dimensions.

III. Demonstration

We have implemented and deployed a complete prototype system. Interested readers may test the system, mainly as a user (customer) on the client side, using any Android mobile device. The `SmartMart` mobile app is downloadable at the Google Play store [1]. The following is a simple manual for the

(a) RFID reader (b) Active tag

Fig. 4. RFID development kit

mobile app.

- On the opening screen one should see a text field, where keywords to be searched can be entered. Press the "search" button to submit a query. When the results display, press the "Select" button to add it to your selected list, or if it has already been selected, press the "Deselect" button to remove it.
- To choose the store you wish to locate click the "Store" button. Choose a store from the list that appears and click the "Ok" button to confirm this store as the map.
- To advance to the Map Screen, swipe your finger to the left to move to the right. On the map screen, you may use one finger to move around the map if it is bigger than your device's screen.
- On the Map Screen, hit the "Open" button to display your selected list. You may delete items off this list by hitting the "Delete" next to the item you wish to delete. To remove all the items from the list, simply hit "Clear All." To go back to the main screen, simply open up the selected list and swipe to the right to go to the left page.

For demo purposes, we have written a program that queries the upcdatabase.com website and populates the server database with sample data. It does this by using the random product function of the web site and parsing the resulting html file. However these entries are fairly low quality and only have the "name" and "upc" field. We have written another program that randomizes the product locations for testing purposes. We also provide a management interface [2] allowing the user to easily add/remove/update entries. To find entries the user enters a keyword query. Once they are located they may be updated or deleted.

IV. Conclusion

Many stores have an online presence indicating their type and quantity of goods, but do not indicate an in-store location. Currently the customer must rely on how the products are grouped and employee assistance to locate an item. Imagine if every customer was assigned an employee that knew exactly where everything was, even if it was recently moved. `SmartMart` leverages the Internet of Things ideology and information retrieval technology to turn the customer's Android

[1] https://play.google.com/store/apps/details?id=com.sm.smartmart

[2] http://dmlab.cs.txstate.edu/smartmart/

device into a free personal shopping assistant. Our preliminary study has demonstrated the feasibility and promise of the proposed approach.

There are many interesting directions for future work. In-store navigation and computing shortest paths for multiple items are immediately useful features. We may also consider facilitating advanced search capacities such as interactive exploratory search and personalized search. Various information retrieval techniques can be incorporated to improve search relevancy and result ranking. It is also necessary to design sophisticated RFID-based localization techniques for improved precision and robustness against interferences. We believe continuous development in this line of research will eventually lead to a revolutionary change in our day-to-day shopping experience.

REFERENCES

[1] K. Al Nuaimi and H. Kamel. A survey of indoor positioning systems and algorithms. In *Innovations in Information Technology (IIT), 2011 International Conference on*, pages 185–190, 2011.

[2] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010.

[3] M. Bouet and A. dos Santos. Rfid tags: Positioning principles and localization techniques. In *Wireless Days, 2008. WD '08. 1st IFIP*, pages 1 –5, 2008.

[4] Z. Dou, R. Song, and J.-R. Wen. A large-scale evaluation and analysis of personalized search strategies. In *Proceedings of the 16th international conference on World Wide Web*, 2007.

[5] Y. Gu, A. Lo, and I. Niemegeers. A survey of indoor positioning systems for wireless personal networks. *Commun. Surveys Tuts.*, 11(1):13–32, Jan. 2009.

[6] A. Lim and K. Zhang. A robust rfid-based method for precise indoor positioning. In *Proceedings of the 19th international conference on Advances in Applied Artificial Intelligence: industrial, Engineering and Other Applications of Applied Intelligent Systems*, IEA/AIE'06, pages 1189–1199, 2006.

[7] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *Trans. Sys. Man Cyber Part C*, 37(6):1067–1080, Nov. 2007.

[8] Z. Ma, G. Pant, and O. R. L. Sheng. Interest-based personalized search. *ACM Trans. Inf. Syst.*, 25(1), Feb. 2007.

[9] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Survey internet of things: Vision, applications and research challenges. *Ad Hoc Netw.*, 10(7):1497–1516, 2012.

[10] J. Pitkow, H. Schütze, T. Cass, R. Cooley, D. Turnbull, A. Edmonds, E. Adar, and T. Breuel. Personalized search. *Commun. ACM*, 45(9):50–55, Sept. 2002.

[11] S. S. Saad and Z. S. Nakad. A standalone rfid indoor positioning system using passive tags. *IEEE Transactions on Industrial Electronics*, 58(5):1961–1970, 2011.

[12] G. M. Sacco and Y. Tzitzikas. *Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[13] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.

[14] R. Yu. Retailers introduce indoor navigation in apps. *http://usatoday30.usatoday.com/tech/news/story/2012-08-27/big-retailer-mobile-apps/57381210/1*.