iNaaS: Indoors Navigation as a Service on the Cloud and Smartphone Application

Panagiotis Konstantopoulos
Technical University of Crete (TUC)
Chania, Crete, Greece
panagiotis.konstantop@gmail.com

Europides G.M. Petrakis
Technical University of Crete (TUC)
Chania, Crete, Greece
petrakis@intelligence.tuc.gr

Stelios Sotiriadis
Birkbeck, University of London
London, UK
stelios@cs.bbk.ac.uk

Abstract—We present iNaaS, an automated indoors navigation system as a cloud service for controlling users navigation and access in large venues (e.g., apartment buildings, shopping malls). The main idea is to install in venues, Bluetooth radio transmitters (beacons) that broadcast their identifier to nearby devices. A mobile application runs on user’s smart phone, receives beacon identifiers and is capable of estimating its distance from sensors in range. Encoded information is transmitted to the cloud where computation of the desired route takes place. iNaaS service responds with navigation instructions which are displayed on visitors’ mobile devices (e.g., smartphones). No personal information is sent to the cloud. Information management, including user profiles, access rights, permanent storage and data processing takes place on the cloud. iNaaS tracks users movements in buildings and offers prompt response on cases of disoriented users. We run an exhaustive set of experiments using simulated (but realistic) data aiming to evaluate both, iNaaS response time and scalability. The results demonstrate that the system can handle large number of users in real time.

Index Terms—navigation, tracking, cloud computing, IoT, Bluetooth, sensor, beacon

I. INTRODUCTION

The idea of the Internet of Things (IoT) combined with cloud computing [1], opens new horizons in the field of real time data collection and analysis. The use of wearable sensors and mobile devices and their capability for Internet connectivity provides significant benefits in applications areas that require fast and continuous monitoring of user data from anywhere (e.g. activity, health monitoring, smart cities etc.). In real-life applications, huge amounts of data are collected and analyzed (e.g. for scientific or business purposes). The solutions have to be scalable (to deal with the ever-increasing number of users and size of data), cost-effective, respond within reasonable time (e.g. taking into account the time constraints of the application) and, address concerns related to users privacy and data safety. Cloud is the ideal environment for IoT applications design and implementation due to reasons related to its affordability (no up-front investment, low operation costs), ease of deployment (taking advantage of IaaS and PaaS solutions already available in the market by many cloud vendors), low maintenance costs (easy software updates and enhancements), scalability (compute resources can be added on demand) and accessibility (IoT services can be accessed anytime from anywhere over the Web).

We focus on the problem of indoors navigation, access control and monitoring of people visiting large venues (e.g. corporate offices, department stores). In our use case scenario we assume that a visitor has expressed interest in visiting a particular site within the building. Access to visitors is granted, controlled or monitored by authorized personnel. There are lots of signs and inscriptions placed at selected points in the venue to assist visitors in finding their destination. Typically, a visitor would call employees for assistance. However, employees may not be available at all times or they may be spending significant amounts of their time in providing navigation instructions to visitors. Visitors would be struggling in finding their way around, spending long time wondering in the building and, finally, delaying to reach their destination. Indoors navigation as well as management of large infrastructures my benefit by using computerized services. These services might not be only capable of assisting users in navigating in the building but, at the same time, they might be also capable of, tracking users, dealing with critical events (e.g. health incidents) while ensuring users safety and improving users experience.

iNaaS is an indoors navigation system that runs on the cloud. In our scenario, each venue has a private cloud system [2] installed that runs iNaaS thus providing greater guarantees of protection of personal or business data as this remains in control of the infrastructure owner. To allow monitoring of users and to control their access to facilities, proximity sensors are installed in the venue. Each sensor has to be in the range of at least one other sensor. We use off-the-shelf Estimote beacon sensors¹ that use Bluetooth Low Energy (BLE) wireless protocol to communicate with other devices. BLE is a well established commercial standard for wireless connectivity of electronic devices, has large battery autonomy and is supported by most mobile device manufacturers. Estimote provides an application for mobile devices that makes possible to locate sensors (sorted by their distance) near a mobile device.

The venue manager registers the areas (with the beacon sensors installed) in the system. Each area is associated with one beacon sensor. Before visitors are granted access to a venue they must log in to iNaaS on the cloud using their mobile device (e.g. smartphone). An application that runs on

¹https://estimote.com
the mobile device is capable of estimating the distance of all sensors in range and transmit to iNaaS the area codes it receives. Encoded information is sent to the cloud where computation of the route takes place. iNaaS responds with navigation instructions which are displayed on the devices of the visitors. No person-specific information is sent to the cloud. iNaaS informs visitors in real-time whether they are still on the right track or, lost their way (in which case new navigation instructions are provided). If visitors keep failing in following navigation instructions or feel disoriented, warning (toast) messages are posted to their mobile devices and a message is sent to the system manager to take action.

Solutions to the problem if indoors navigation are known to exist with InfSoft\(^2\), IndoorAtlas\(^3\) and MapsIndoors\(^4\) being the most popular ones. IndoorAtlas uses the magnetic field inside the building as a map to accurately track a person. MapsIndoors by Google is built on Google Maps and can be interfaced with popular indoor positioning systems (e.g. WiFi positioning, BLE, positioning based on magnetic fields). The same as iNaaS, these systems rely on position sensors transmitting position information using a non-IP protocol (e.g. Bluetooth, BLE, RFID, ZigBee, Wi-Fi Direct, etc.). The map of a venue is usually integrated into the executable files of the application which, together with navigation instructions, are transmitted to the devices of the visitors. An important feature of all systems (and also of iNaaS) is user interaction with a mobile device over the network as some actions are performed locally on the device while, others are performed remotely on a server. All commercial applications promise high detection accuracy with minimal latency. iNaaS is not intended to compete commercial solutions in terms of accuracy or performance but rather, to show how a cost-effective solution is intended to compete commercial solutions in terms of accuracy.

II. iNaaS DESIGN AND ARCHITECTURE

iNaaS is a two-fold solution based on micro-services for the IoT (users’ smart devices) and the cloud side (that includes back-end services). Building upon principles of Service Oriented Architectures (SOA) design [3] and driven by the key requirements of today’s IoT systems for adaptability, low-cost and scalability, iNaaS architecture is modular and expandable. The implementation relies on modular services which are deployed on the cloud and implement fundamental functionality. Some of the service modules in iNaaS architecture are available as re-usable Generic Enablers (GEs) on FIWARE catalogue\(^5\).

Each service or groups of services are deployed on the same or different Virtual Machines (VMs) on the cloud. iNaaS is a composition of autonomous RESTful services [4] communicating with each other over HTTP even though these can be deployed on the same VM. Network delays are expected due to the nature of this design. However, the experimental results demonstrate that iNaaS is capable for responding in real time even under heavy workloads. Nonetheless, the advantage of SOA is, system modular design, ease of configuration (that best suits the need of an application), ease of maintenance and expandability. For example, more services can be added at-run time or, any service can be moved to a different VM (on the same or different cloud) with minimum overhead (i.e. only the IP of the service will change).

We followed a valid system design approach [5], [6] that identified (a) the functional components and their interaction, (b) the information that is managed and how this information is acquired, transmitted, stored and analysed, (c) the physical software entities that support the functional and information activities, (d) the requirements for assuring data, network and user security and privacy. iNaaS (reference) architecture [7] is described by a set of UML diagrams including (a) information (class) diagrams describing information that is handled by the system, (b) activity diagrams describing flowcharts for several types of user actions, the most important of them being, (c) an architecture diagram.

A. Use Case Scenario

There is an initial entry point and one or more destinations in each building. The visitor has expressed interest in visiting a particular site within the building. There can be predefined routes for each destination which are stored in a database. Shortest path routes can be computed as well (i.e. the routes between beacons in range form a graph). However, a route is not necessarily the shortest route, but rather the easiest or permissible route for the visitor. Permissible routes for each destination are defined by the system manager, can be computed in advance and stored in a database. iNaaS provides graphic tools for venue managers to define permissible routes for visitors. iNaaS computes navigation instructions in real time taking into account position information (i.e. distance of the visitor’s mobile device from the nearest beacon).

The iNaaS deployment diagram of Fig. 1 illustrates the physical entities and their interaction. There are four beacons and four visitors with mobile devices. Not all beacons are detected at the same time (e.g. beacon 2 is not detect by any device). A device can detect more than one beacons in range at the same time. These beacons are sorted by distance and the identifier of the nearest one together with user identifier and session information (i.e. session key) is transmitted to the cloud. This activity is continuously monitored by the system manager.

The iNaaS service that generates navigation instructions runs on the cloud. An authorized user has to log in to the cloud in order to receive the path on his/her mobile device.

\(^2\)https://www.infsoft.com/solutions/indoor-navigation
\(^3\)http://www.indooratlas.com
\(^4\)https://www.mapspeople.com/mapsindoors/
\(^5\)https://catalogue.fiware.org
iNaaS is also capable of tracking and handling the journeys of the visitor in a building, for interacting with the visitor and, for responding in cases of visitors diverging from their path. As long as the visitor moves on the path, he/she is prompted to advance to the next permissible location (on the path). If no permissible location is near (i.e. the visitor diverged from the path), a notification is send to the visitor together with a request to the administrator for assistance. Fig. 2 summarizes the navigation process.

Fig. 1. iNaaS deployment diagram.

Fig. 2. iNaaS navigation procedure.

As shown in Fig. 1, there are different types of users that interact with the system. Each type of user is associated with different services that handle different types (classes) of information the most important of them being:

**Users:** Describes users interacting with the system (i.e. visitors, system managers). Each user has an identifier, an email, and a name being displayed. System managers are authorized to import or delete visitors, to contact visitors and install beacons (i.e. associate beacons with locations in a venue).

**Subscriptions:** Describes interface information and functionality for registering users, beacons and events. The NGSI-9/10\(^{6}\) interface is used to describe interacting entities (visitors, beacons, areas), entities that publish events and entities that subscribe to these events (see Context broker / Publish and Subscribe service in Sec. II-C).

**Permissions:** Describes information and methods for handling the access rights that each user has per site. This information is stored in a table, where each record represents the identifier of a location, the value of which is the respective access right.

**Status:** Possible status situations are: The visitor has been lost, is at authorized or unauthorized location, or has reached his/her destination. The methods of this class are called whenever there is a change in the visitor’s location, so the method for determining the next permissible location in the path is executed.

**Locations:** Keeps information about venue locations and their management (i.e. name and location ID, sensor identifier, path), as well as methods for handling this information.

**Neighborhood:** Describes information about locations (and beacons associated with them) bordering a given area. It is represented by a table, where the key of each record represents the identifier of each location.

**Instructions:** Handles navigation and methods for accessing the database where this information is stored. Instructions have the form of triplets (i.e. location A → B, direction, where direction take values: left, right, forwards or backwards). Navigation instructions for permissible routes comprise of a sequence of such triplets. The next permissible location is the location of the nearest beacon on the same path. Navigation instructions and paths for each destination are defined by the system manager.

**Messages:** Describes information for messages exchanged between the system manager and the visitors. Messages can be generated automatically or be composed by the system manager.

**Logs:** Describes history data stored in a database and includes information for visitors navigation data, events and messages.

### B. iNaaS Front-End

The front-end runs on a mobile device and is the means for providing navigation instructions to the visitor. It connects to the back-end via internet (e.g. WiFI). The mobile application was developed using jQuery Mobile\(^7\) in conjunction with JavaScript, HTML5 and CSS. The mobile application can be

---


\(^{7}\)http://jquerymobile.com
ported to Android or iOS devices using Apache Cordova\(^8\). Fig.3 illustrates the set of services running on the mobile device and their interconnection.

**Application logic:** It orchestrates services running on the front-end, controls their interconnection and runs the following services in order: (a) retrieves user’s access rights from local storage, (b) establishes a connection with the back-end; If log in is confirmed, a session between the mobile application and the cloud is initiated, (c) initiates the BLE scanner and if beacons are found in range, their data (i.e. identifiers and distance from the mobile device) are sent to the back-end.

**BLE scanner:** It scans continuously the space for beacons in range; for any beacon found, it reads its MAC address, strength), any other measurement that the beacon may provide (e.g. temperature, acceleration) and sorts the beacons by increasing distance from the mobile device. This information is received by the application logic service. It is implemented using the beacon finder API by Estimote\(^9\).

**Local storage:** It is implemented using HTML5\(^10\) that provides a database that stores locally application information including user credentials and session keys for accessing the back-end (i.e. when a user enters a password, it is stored locally). If verification fails, the user is prompted to enter his/her log in credentials again. Local storage also supports temporary storage of application information in case of non-Internet connectivity.

**Connectivity service:** It is responsible for the communication between the mobile application and the cloud. The communication is asynchronous via AJAX calls \([8]\). The data exchanged is packed in JSON format and data (Base64) encryption (or decryption) is applied before transmission. The mobile application and the cloud need to communicate in the following cases:

- When logging in to the application to verify the identity and access rights of the visitor: It checks the local storage for user’s credentials (i.e. log in data or session key) and attempts to automatically connect the user with the back-end. Typically, only a session key is transmitted for an authorized user. The back-end verifies the visitor’s identity and responds with a permissible route (defined by the administrator or retrieved from the database). This information is displayed on the mobile device.
- As the visitor advances from one location to the next along a path and as the BLE scanner identifies the next nearest beacon, its identifier and distance (from the mobile device) are sent to the back-end. The back-end responds with an instruction on how to advance to the next permissible location on the path.
- For requesting navigation instructions, history data (i.e. previously visited locations in the path), change of destination (in which case the back-end responds with new instructions) and finally, when receiving notifications by iNaaS manager.

**Notification handler:** Allows messages to be sent to users and system administrators (e.g. for taking actions). This service enables the mobile application to receive (toast) push notifications even running in the background. These messages can be (a) alerts generated automatically in response to the current location of the visitor (e.g. the visitor is lost, or diverges from the defined path) or, (b) messages sent by the administrator (e.g. “assistance is on the way”). The service performs AJAX Long Polling requests \([8]\): sends requests at regular time intervals to the back-end to investigate if there are updates to download; after receiving and reading an alert, the back-end confirms that the update has been read.

![Fig. 3. iNaaS front-end architecture.](image)

**C. iNaaS Back-End**

The back-end is implemented on a FIWARE platform running on OpenStack. The services running on the cloud are either provided by FIWARE or are implemented by the authors. The services communicate asynchronously using Slim PHP\(^11\) (for handling REST requests and for communicating with the database), cURL\(^12\) for communicating data and requests between services and, finally, MongDB\(^13\) for the database. The asynchronous communication between services is implemented using AJAX. Fig.4 illustrates the set of services running on the cloud and their interconnection.

**Application logic:** In line with application logic on the front-end, its purpose is to orchestrate, control and execute services running on the cloud. When a request is received, it is dispatched to the appropriate service. For example, services regarding user accounts and access rights are dispatched (through application logic) to user management and identity management services. It is tightly related with the connections and the context broker services (below) which form the communication channel with the mobile application. Finally, application logic performs basic security controls (e.g.

\(^8\)https://cordova.apache.org
\(^9\)https://www.npmjs.com/package/cordova-plugin-estimote
\(^10\)https://www.w3.org/blog/news/archives/4167
\(^11\)https://www.slimframework.com
\(^12\)https://curl.haxx.se
\(^13\)https://www.mongodb.com
checking if a session between the mobile application and the cloud has been initiated).

**Keyrock identity management:** It provides user identification and authorization services (i.e. access control based on user roles and access policies). It is implemented using the Keyrock Identity management (IdM) service of FIWARE. Keyrock IdM provides a Single Sign On (SSO) service for users and, implements an OAuth2.0 mechanism. Applications accessing protected resources are checked whether they are authorized to do so using their OAuth2.0 credentials.

**Publish - Subscribe Context Broker:** This service acts as a mediator for the position measurements sent from the front-end to the back-end. Data entering the Context Broker refers to the proximity sensor identifier and the desired destination. The service notifies the application logic for changes in location information (i.e. the nearest beacon changes). It is implemented using the Publish-Subscribe Context Broker service of FIWARE.

**User management:** Implements functionality for user management, including creating, editing, deleting users and their profiles, their access rights and navigation history. This information is stored in the database from where it can be shared with other services. For example, application logic service checks the database whether the current location of the visitor is in the list of permissible locations of a path.

**Location/Beacon assignment:** The service provides functionality for creating, editing, deleting sites (i.e. areas or locations in buildings), their correlation with beacons, defining areas bordering them, and for defining permissible routes. This information is stored in the database.

**User activity monitoring:** The service tracks users in real time. This information is displayed on the mobile device of the visitor and also on the interface of the system manager. System managers and visitors are notified in the event of incorrect route or access to unauthorized locations. Permissible route information, user profiles and access rights are retrieved from the database where history data (i.e. visited locations) are stored as well.

**Data storage:** This is shared database where data generated from all services are permanently stored. The database maintains all necessary correlations between data (e.g. users and permissible routes), checks their validity when importing, editing and deleting data and triggers actions if incompatibilities are detected (i.e. violation of access rights if a visitor is attempting to access a non permissible location or diverges from the defined path). It is implemented using MySQL. The rationale of this choice is to take advantage of referential integrity for all types of data and their correlations as well as, of triggering mechanisms provided by the relational model. Compared to a NoSQL database (e.g. Apache Cassandra) which is not designed to support this functionality in the first place), the complexity of this solution may create a significant performance drain especially for a large set up.

**Navigation Instruction Utilities:** It computes the navigation instructions based on information received by the mobile device of the visitor including, user identifier, session information, location information (represented by the identifier of the nearest beacon) and route information retrieved from the database. It executes the algorithm of Fig. 2. The service communicates with the Publish - Subscribe Context Broker which notifies the service on change of the nearest beacon. If the new beacon belongs to the route (i.e. the visitor is still on track) new navigation instructions are computed. If not (i.e. the visitor diverges from the path) but a beacon can still be detected (even not one belonging to the route), instructions are generated automatically to bring the visitor back on track (i.e. based on information of previously detected beacons). If no new instructions can be generated (based on the location of the current beacon) then the visitor is probably lost and a notification is sent to the visitor (e.g. not to leave the current position) and to the iNaaS manager to take action (i.e. notify personnel to provide assistance).

**Connectivity service:** In line with connectivity service running on the front-end, it implements secure asynchronous communication between the front and back-end. Encoded data are formed in JSON and are exchanged using AJAX calls.

III. PERFORMANCE EVALUATION

We run an exhaustive set of experiments and we analyze the performance limits of the services running on the cloud. All services run on the same core VM, with the exception of the Publish - Subscribe Orion Context Broker that runs on a separate (second) VM, the database service that runs on a third VM and, the Identification – Authorization service (Keyrock Identity Management) that runs on a fourth VM. Identity Management service is not causing any performance bottlenecks as it is addressed only at log in or for authorizing access to resources. Similarly, MySQL is known to scale-up very well (in all our experiments, referential integrity checks

---

14https://catalogue.fiware.org/enablers/identity-management-keyrock
15https://oauth.net/2/
16https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
17https://cassandra.apache.org
are deactivated). Publish/Subscribe Orion Context Broker VM runs on a remote FIWARE node (FIWARE is a federation of distributed cloud infrastructures). We can only measure the response time of this VM but we can’t change this VM (i.e. this is a shared VM servicing many applications on the cloud).

In the scenario of Sec. II-A the frond-end (mobile device) discovers a new beacon on a path and sends a request to the back-end to receive new navigation instructions (i.e. the next permissible location on a path). In the following, we study the scalability of this service running on the core VM.

The core VM has one processor (x86_64 processor architecture, 2,800MHz, first level cache size 32KB, second layer 4,096KB cache size), 2,048MB RAM, 20GB hard drive capacity, runs Ubuntu 14.04 and an Apache HTTP server. The resource usage metrics are taken using the Linux htop command. We use ApacheBench\(^{18}\) to issue multiple simultaneous requests to iNaaS. In ApacheBench we are opted to define the total number of requests and how many of them will be executed simultaneously. All measurements of time below account also for the time spent for the communication between VMs or between services within the same VM.

In the first experiment we issue 2,000 requests in a sequence (i.e. concurrency = 1, the requests are executed sequentially). Table I shows this result. The average response time is 437ms. Almost 75% of the response time is consumed on Publish-Subscribe Context Broker VM. Speeding-up the response time of this VM would require installation on the same FIWARE node together with all other VMs running iNaaS.

In the next experiment we issue 2,000 requests, with increasing concurrency values (i.e. concurrency > 1, more than 1 requests are executed concurrently). Table II summarizes the results of this experiment. Response times improve drastically with the simultaneous execution of requests (i.e. the Apache HTTP server switches to multitasking) reaching their lowest values for concurrency between 50 and 100. Even with concurrency = 300 and although the server load reaches a peak, the average execution time per request is close to real-time.

Processing capacities may increase exponentially or raise restrictions for space or bandwidth (especially for concurrency > 1). iNaaS may produce big amounts of data and requests, requiring large processing capabilities, which can surpass the capacities that our experimental system set-up is able to provide. In our set-up, most iNaaS core services are implemented in a single VM thus overloading the VM when the number of concurrent service requests exceeds a limit. An obvious solution to dealing with requests would be to employ additional VMs each running a single service (or a small group of services). Alongside, we can allocate additional VMs implementing the same service (or groups of services) thus having more than on VM sharing the load.

In all measurements above, the performance of the mobile device is not considered. The iNaaS mobile application runs on an ordinary Android smartphone (Android 5.1 ARM Quad-Core CPU 2.2GHz, 2GM RAM, 32GB Flash storage) consuming less than 20% of the CPU time and 100MB RAM. Scanning for beacons in range (using the API by Estimote) consumes significant amounts of resources takes up 5,000ms to discover 9 beacons. Presumably, the performance of this operation will be improved by using a faster Android device and more advanced beacons.

### IV. Conclusions

iNaaS monitors visitors’ activity and supports navigation in large indoors venues. The response times of iNaaS can be improved significantly by applying certain optimizations on services on the cloud. As a future extension, we plan to experiment with more advanced beacons (i.e. with faster scanning times and better location accuracy). Incorporating elaborate navigation algorithms and data analytics functionality on the cloud, are also interesting directions for future research.

### REFERENCES


---

\(^{18}\)https://httpd.apache.org/docs/2.4/programs/ab.html