A comparison of GIS architectures for implementing indoor location-based services

Valerio Vianello*, Cristiano di Flora, Christian Prehofer
Nokia Corporation, Finland
Email: valerio.vianello@uniparthenope.it ,{cristiano.di-flora, christian.prehofer}@nokia.com

Abstract— **This paper describes our experiences and lessons learnt in building a Geographic Information System (GIS) specifically designed for indoor location-based services. The proposed system was built by mashing-up commodity Open Source software and novel research prototypes realized within our labs. The overall approach relies on intense usage of standard web technologies and REpresentational State Transfer (REST) APIs as a way of enabling easy mashup of off-the-shelf and proprietary components. The key design and implementation aspects of our solution are described in detail, including a discussion on how we represented and augmented the concept of indoor location within our services, as well as on how we integrated them with commodity GIS services originally designed for outdoor scenarios. Further, we show different approaches for implementing client-side applications capable to interact with our indoor location-based services, and we report experimental results of test campaigns for evaluating the impact of those approaches on resource constrained devices.**

Index Terms— **Indoor, GIS, Smart Spaces**

I. Introduction

This paper describes our experiences and lessons learnt in building a Geographic Information System (GIS) specifically designed for indoor location-based services within our GIS smart places infrastructure. While there has been considerable research and commercial success on outdoor (GPS-enabled) Location Based Services (LBSs), we focus here on indoor LBSs. Our focus here is to understand how commodity GIS technologies can be used for our mobile indoor solution. In this way, we aim both to extend the scope of GIS systems and to use them as standards for indoor applications. The proposed system was built by mashing-up commodity Open Source software and novel research prototypes realized within our labs. The overall approach relies on intense usage of standard web technologies and REpresentational State Transfer (REST) APIs as a way of enabling easy mashup of off-the-shelf and proprietary components. Consumers' satisfaction has an important role in our system design and we focus on evaluation of GIS services for mobile devices in this paper.

This paper is based on "Leveraging GIS technologies for the creation and provisioning of web-based smart places services" by C. di Flora, and C. Prehofer, which appeared in the Proceedings of 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS 2008), Capri Island, Italy, October 1-3, 2008, Lecture Notes in Computer Science 5287 Springer 2008

* Was a Visiting Research Scholar at Nokia Research Center, on leave from Dipartimento per le Tecnologie, University of Naples Parthenope, Italy

As far as system architecture is concerned, the paper describes two alternative solutions for map rendering. A first one renders the map on the client-side using commodity JavaScript libraries, and the other one renders the map completely on the server-side. Both server and client-side approaches have their known advantages, especially when users perform repeated operations on maps like displaying points of interest or panning. For the server-side rendering, no computation on the mobile device is needed, but maps are transferred as they are, which may result in repeated transfer of complex map material. Furthermore, one may expect that client-side rendering is more responsive and does not need to re-download map material. The paper gives detailed insights for mobile GIS systems on existing mobile devices and shows in which cases these general expectations hold or not.

The paper is organized as follows. Section II discusses the rationale behind our work and related research. The key design and implementation aspects of our solution are described in detail in Sections III and IV, respectively, including a discussion on how we represented and augmented the concept of indoor location within our services, as well as on how we integrated them with commodity GIS services originally designed for outdoor scenarios. Section V outlines an evaluation strategy for indoor GIS, and reports the results of our experiments executed according to the outlined strategy. Section VI concludes the paper by outlining the main lessons learnt and future research directions.

II. Motivation and Background

Outdoor location based services have been widely successful and have created an enormous ecosystem of applications around them. Such applications have fueled the integration of applications by using the mashup approach. For instance, it is now very common for web-based services to show geo-tagged items on a map application. Our focus on indoor and mobile location-based services is motivated by research on pervasive and ubiquitous computing, which often build also on indoor positioning and services on indoor maps. In this context, we developed services for indoors based on our system and called these "smart space" (application) platform. There is extensive literature on middleware for such ubiquitous applications, for instance [1] surveys 29 approaches up to 2004. This survey covers many systems for distributed

doi:10.4304/jsw.4.7.664-674

mobile computing and most of them consider location information, even though positioning technology was not widely spread at that time.

We think that there are a number of essential research issues towards a wide spread ecosystem of indoor positioning services. In our view, the key issues for the widespread usage of indoor positioning services are as follows:

- Establishing common standards and practices for indoor positioning and map services which are interoperable and easily available
- Integrating indoor positioning services into existing web-based services
- Enabling mobile, context aware applications which are easy to deploy and use
- Making the services usable by resource constrained devices
- Ensure privacy and security regarding personal information such as location information

Regarding the first item, we see several missing pieces. First, while GPS is nowadays widely available outdoors, there is no such established and deployed indoor positioning technology. Moreover, most of them need dedicated device-side or infrastructure-side hardware. Indoor positioning has been using very different technologies such as bluetooth, RFID or WLAN based. Even further, there are no established standards for handling maps and geo-spatial data, whereas such standards are available for outdoor LBS [2]. Our approach here is to build on WLAN based positioning, which we see as a widely available technology, and to use such open standards for geospatial information systems for indoor settings.

Regarding the second issue above, integration with existing services is needed because such services not only provide considerable technology but also toolkits and substantial amount of already available geospatial data as, for example, in outdoor maps applications. This may also include personal data, such as private contacts or pictures. We also observe here that most of these application mashups use technologies which are simple and integrate easily into web applications, such as REST style APIs and RSS feeds.

Another problem is that deploying mobile services is difficult, as there is a large variety of devices and different operating systems (in different version) on the market. Furthermore, these devices can be quite different in terms of resources like memory and connectivity. This problem has hampered the deployment of mobile services in general, and is more severe in our case as we also want to integrate positioning information as well as other context data. We are focusing on web based applications as they are easy to access, to deploy, and to manage. However, they do not have access to local context data of the user. Different options can be chosen in this context, based on downloadable client software for context collection as well also using upcoming standards for browser-based access to context data such as the W3C Delivery Context Client Interface (DCCI) initiative [3].

With regard to services usability, it is clear that users using devices with different resource constrains could get different experience from the same service. For this reason, our aim is to provide services where the quality is assured even though the device used to access the service has low capabilities.

Another challenge, which is closely tied with the above is to connect with (other) applications and enable application mashup, while preserving privacy. The issue is that ubiquitous context information is typically privacy sensitive and existing application mashup techniques do not support this sufficiently.

In this paper we discuss how we tackled the described issues, and we show how to use GIS solutions for indoor settings and show how the web can be used as a platform for these services. This covers connecting to services in other web based applications. We also discuss how to integrate context information into our service. We do not cover in full detail more secure ways for application mashup. Here, we see a few key ingredients emerging, such as OpenID and OpenAuth [4], which rely on novel and more flexible decentralized approaches to authentication and authorization and thus can provide a mashup-friendly security infrastructure. A similar architecture for mobile devices is presented in [5], where the location context data is sent separately to a server, while the service is hosted from a web server which obtains the context data from this server. This paper focuses on visibility models and does not cover indoor positioning aspects as done here. Other works on applying mobile GIS solutions are for instance [6], which focuses on tour guide applications. [7] focuses on indoor GIS for mobile devices, but does not address application mashup and platform aspects nor interactive JavaScript front-ends as we do here.

III. Web-based Indoor GIS solutions

A. The web as a platform for indoor, mobile location-based services

Before discussing how we designed and implemented our indoor location sensing API, it is worth shedding some light on the architecture of the overall web based service platform that the API is part of. A detailed discussion of this architecture goes beyond the scope of this article. The interested reader may refer to [8] for further details about it. In this sub-section we will focus on the key design decisions underlying this architecture, and on their effects and implications on the proposed GIS solution. Compared to related work in this area, the decisions that characterize our approach are as follows:

- HTTP-based communication: HTTP and web services are used as the primary means for integrating and mashing-up software across devices.

doi:10.4304/jsw.4.7.664-674

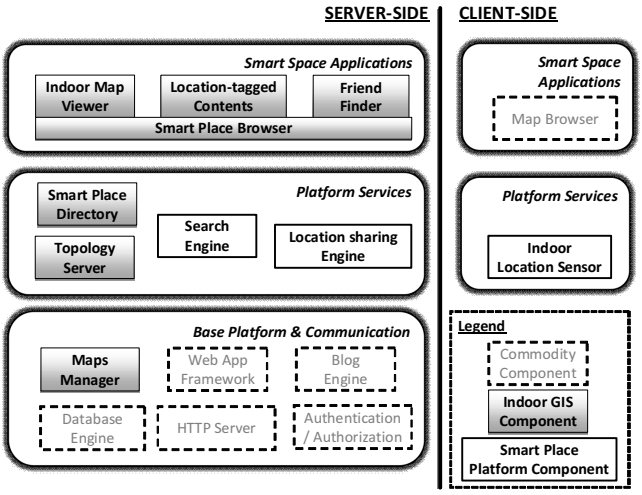


Figure 1. High level architecture of our Smart Places platform, including server-side (left side) and client-side (right side) components. Grayed boxes represent Indoor GIS components. White boxes represent either commodity open source components (dashed border boxes) or smart places platform specific components (solid border boxes).

- Reuse of existing web technology: a key problem with existing solutions in the ubiquitous and pervasive computing research community is that they are rarely reusable. By relying on existing web technology, the infrastructure interoperates with a large number of devices already available in the current market.
- Multiple runtimes for application execution: High end mobile devices are now offering a much wider range of runtimes for implementing and running applications and services, including traditional Java and C++ runtimes as well as support for Python and other scripting languages. In this way, platform developers can use several runtimes. This means that many existing components used on the web can be used in our infrastructure as well. Moreover, having the basic location enablers available through HTTP based communication, makes it possible and easy to mashup local (both situated and on device) and remote location-based web services all together.

The system is organized in multiple layers, which are depicted in Figure 1 and briefly described in the following. Please refer to Section IV for further details about how we designed the GIS components and what commodity components we included in them.

The **Base Platform & Communication** layer contains several commodity commercial and/or open source components that we see as necessary to realize a full web platform in the smart space. Using commodity web components allows us to bring many features to the smart space such as, for example, easy creation and deployment of services using well known authoring/distribution tools and framework for web applications, user management and security solutions, database and content management systems. As far as location-awareness support is concerned, the platform currently includes a **Maps Manager** component, which is in charge of providing Create Read Update Delete (CRUD) primitives for indoor maps data, i.e., raster layers representing building and floor plans, as well as several features (vector) layers representing floor topology elements (rooms, corridors, halls) and static Points-of-Interests (e.g., in an airport smart place, they could include restaurants, shops, check-in desks, terminals, gates, and other categories of interest to people visiting that space).

Platform Services can be created by using the technologies in the **Base Platform & Communication** layer. A few typical platform services are shown in Figure 1. They include two actual indoor GIS components, namely **Topology Server** and **Smart Place Directory**. The **Topology Server** is in charge of providing low-level information about the physical structure of the available smart places, such as for example details about how a given building is structured in floors, wings, and rooms. Places are modeled by using an hybrid hierarchical location model [9], and each physical location is assigned a URI that other services and applications can use as tags to associate items (e.g. media contents, blog posts, user location) to a certain physical location (further details about the model are reported in [10]). The **Smart Place Directory** acts as a mediator between **Smart Space Applications** and the other indoor GIS components of the platform (i.e., components belonging to either the **Platform Services** or **Base Platform & Communication** layers). It provides a set of REST APIs that allow **Smart Space Applications** to access other **Platform Services** and **Base Platform & Communication** functionality through a consistent and common API. The API allows CRUD access to most of the location-dependent data available in the smart place, such as people location, location-specific contents, and POIs, in a web-application friendly way. In fact, it supports several widely adopted data-interchange formats, including XML-based formats like ATOM and RSS feeds, as well as more lightweight formats such as the JavaScript Object Notation (JSON). All the mentioned components so far are intended to be

doi:10.4304/jsw.4.7.664-674

deployed on the server-side.

Smart Space Applications re-use and combine the **Platform Services** functionality with other on-device features in order to provide meaningful and helpful functionality to end-users. These may be web applications, which can be accessed using a browser and which can be hosted on the mobile web application server. Alternatively they can be implemented as stand alone applications written using any of the existing device specific development kits and that access the deployed **Platform Services** through http-based communication. In Figure 1 we show a few key examples of such applications, which are described in the following. The **Smart Place Browser** represents a very generic entry point to available applications, providing an AJAX API for **Smart Space Applications** developers to create new end-user applications. The API exposes and leverages the key abstractions implemented by **Platform Services** to a developer-friendly interface. Some of the services made available by the API are: search and indexing of places in a smart space, indoor positioning that allows user devices to determine where they are in the smart space, functions to retrieve geographic information about maps and POIs. New applications, such as the **Indoor Map Viewer**, **Location-tagged Contents**, or **Friend Finder**, have been implemented on top of this API, as we will show in Section V.

It is worth noting that, as Figure 1 clearly shows, our approach is based on a very thin-client model, in which no particular **Base Platform & Communication** or **Smart Space Applications** components are assumed to be deployed and pre-installed on the client-side. In fact, in order to use the services, the client-side just needs to have the Map Browser component, which, as we will describe in Section IV-A, could be simply implemented as a web browser capable of rendering rich web applications. Additional components, such as for example the **Indoor Location Sensor** in Figure 1, might be required in order to enable usage of some applications (like the Friend Finder) or to improve user experience with other applications (e.g., to automatically adapt or initialize the UI of the **Indoor Map Viewer** and **Location-tagged Contents** applications based on the actual indoor location of the end-user).

B. The adopted indoor-positioning technique

The proposed solution relies on an experimental WLAN indoor positioning technique under research and development at Nokia Research Center. The technique is based on WLAN scanning and on further processing of the scanning results, including measurement of the Received Signal Strength from all reachable access points, from which the current location of the mobile device is calculated. All steps are performed on the terminal side, e.g. on end-users smart phone or PDA. In the rest of this sub-section we will just shed some light on the key aspects of this technique that are required in order to understand the herewith described indoor GIS solution. The interested reader can refer to [10] for further details on its design and implementation.

One interesting aspect of the adopted indoor positioning technique lies in its quick and easy deployment in out-of-the-lab real-world settings. In fact, the technique only requires a-priori knowledge of a list of known WLAN APs along with information about their physical location in the target building (which is typically a well-known piece of information). In this way, it does not require any off-line measurements of the received signal strength nevertheless. In other words, no radio maps of the target environment and related calibration of the algorithm are required, which in turn makes the proposed smart places infrastructure more easy to set-up than other state-of-the-art solutions [11].

The outcome of the proposed algorithm is a symbolic location information structured according to the location model mentioned in Section III-A. It is worth noting that the concepts of building, floor, and section could be eventually replaced by other concepts and semantics if needed. In other words, different symbolic location models with different granularities could be adopted as far as the technique is accurate enough to support the required granularity. For example, in an environment with very large sections and rooms, such as a shopping mall or an airport, the model could also take into account the concept of rooms within a single section.

IV. Implementation and Prototyping

A. Implementing the indoor GIS prototype

In Section III-A we introduced the main indoor GIS components at a very high-level of detail. Since we wanted to create a practical GIS solution for indoor smart spaces that could work also from mobile devices, we needed to combine the web-based smart space platform and the indoor positioning technique, described in Section III-A and III-B, respectively, with additional components providing traditional GIS functionality, such as maps, navigation, and geo-spatial queries support to commercial mobile devices.

In compliance with our overall smart spaces approach described in Section III, we decided to rely on open APIs and protocols suitable for integration with web-based applications and services. In the following we discuss a few key implementation decisions that we needed to take when prototyping the indoor GIS part of our smart places platform. In addition, we also provide further details about how indoor GIS components interact one with each other in order to fulfill their responsibilities. More specifically, we describe how we implemented and prototyped our first example of an indoor GIS system based on the design guidelines and concepts described in Section III. The overall architecture of the implemented prototype is depicted in Figure 2. When implementing the first prototype we had to satisfy the key requirement of providing simple REST APIs for other services to create composite functionality (mashups) out of the basic building boxes provided by our platform. To this aim, we

doi:10.4304/jsw.4.7.664-674

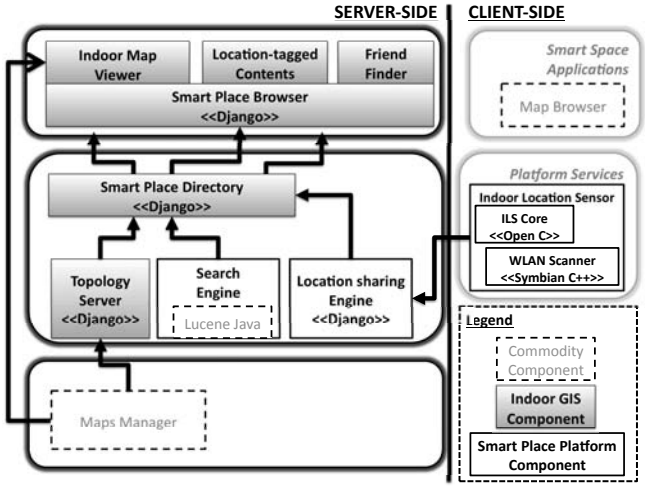


Figure 2. Implementation of our indoor GIS prototype. Grayed boxes represent Indoor GIS components. White boxes represent either commodity open source components (dashed border boxes) or smart places platform specific components (solid border boxes). Lines represent interaction related to Indoor Map Viewer, Location-tagged Contents, and Friend Finder case study applications.

implemented the **Smart Place Directory** in the Python programming language by using the Django web application framework. Using the Django framework enabled us to quickly implement different facades for the same back-end data, in order to support CRUD interfaces to manage location-dependent smart place resources based on all the data interchange formats (JSON, ATOM, RSS) mentioned in Section III-A. Similarly, most of the realized web services were implemented as Django applications. This gave us a lot of flexibility in designing the actual web service interfaces we wanted to use to expose the indoor location-dependent data stored in our back-end.

As far as the **Maps Manager** component is concerned, we wanted to rely as much as possible on established standards for representing indoor GIS data, such as maps, points-of-interests, and related meta-data: to this aim, we decided to adopt the Open Geospatial Consortium (OGC) Web Map Service (WMS) and Web Feature Service (WFS) [2] to represent maps and POIs in the **Maps Manager** component. This decision was motivated not only by technical requirements, but also by our higher level goal of evaluating how easy and feasible was the idea of using commodity GIS solutions to support indoor LBS. Several commodity implementations of WMS/WFS servers were available in both the commercial and open source community. After evaluating the different available options, we decided to adopt two different open source products, namely Geoserver [12] and Mapserver [13], which provided quite a complete implementation of WMS and WFS specifications.

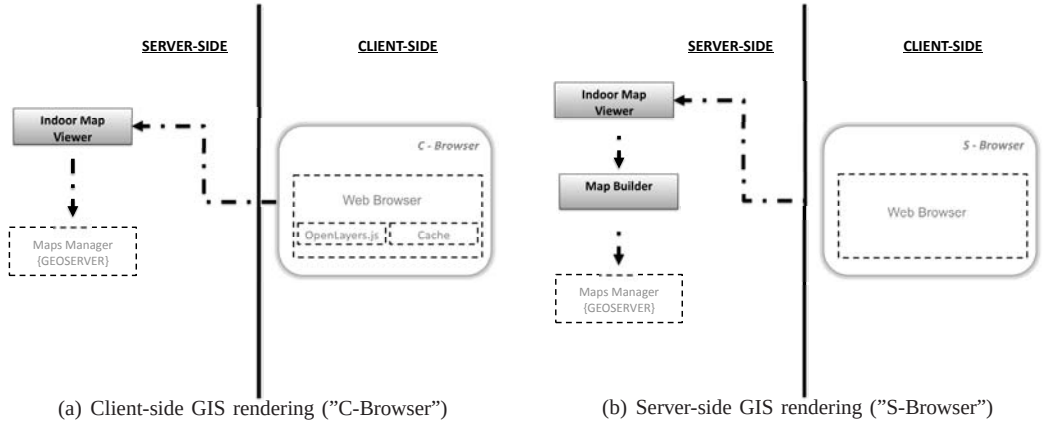
The search functionality was implemented in the Java programming language as a Java servlet component relying on the Java Lucene indexing engine. The on-device **Indoor Location Sensor** needed deeper integration with the native OS services in order to access the wireless lan management APIs and retrieve data from them. Hence, it was implemented in the C++ programming language on top of Nokia S60 C++ SDK. In this phase, we tried to enhance code-portability as much as possible. For this reason, we wrote most of the code in standard POSIX C language by using the Open C libraries for Symbian OS. This code, including the algorithm to calculate location from wireless LAN scanning results, was encapsulated in the so-called **Indoor Location Sensor (ILS)Core**. The only non-portable part of the **Indoor Location Sensor** implementation was the **WLAN Scanner** component, which required low-level access to the wireless LAN management API of Symbian OS and thus needed to be coded by using Symbian C++ specific classes and programming idioms.

When designing our solution, we wanted to provide very open APIs for accessing the available Indoor GIS data from both mobile devices and fixed/desktop devices. To this aim we had decided to create all the data related to the map in a vector format. Since the screen size of a mobile device, like a mobile phone, and the screen size of a computer desktop are quite different, using, for example, a raster layer for an indoor map is not the best solution. The image quality will be very different according with how much the image is scaled, furthermore is not reasonable create different images for all possible screen size. Instead, using data in vector format, the Maps Manager can produce images scalable to any size and the quality of the image is only determined by the resolution of the display.

B. Two architecture solutions for map rendering

The described Indoor Map Viewer component is in charge of drawing the indoor maps on the client device. For the proposed Web based Indoor GIS, we implemented two versions of this component, namely Client-side GIS Rendering (C-Browser) and Server-side GIS Rendering (S-Browser). These implementations are depicted in Figure 3. The C-Browser is based on the open source Open Layers JavaScript API [14]. It relies on a client-side geospatial-data cache to support off-line viewing of indoor maps. Open Layers allows dealing with most common GIS functionality (e.g., dynamic creation of

doi:10.4304/jsw.4.7.664-674



(a) Client-side GIS rendering ("C-Browser") (b) Server-side GIS rendering ("S-Browser")

Figure 3. Two versions of Indoor Map Viewer component. (a) shows the 'C-Browser' version where the final map is rendered at the client-side and (b) shows the 'S-Browser' version where all the operation to render the map are done at the server-side

maps by mashing up map data coming from different sources, DOM interface to several GIS data representation formats) while solving already a lot of common cross-browser technical issues (due to presence of different Javascript rendering engines). The geospatial data cache can be used to store the data, which Open Layers uses to draw the maps, on the client device. An indoor map typically consists of multiple data layers, including at least one layer for the base map and additional layers representing different map decorators. A typical map decoration layer is the Points of Interest (POIs) layer. Each POI of this layer consists of geospatial coordinates as well as metadata like POI id, POI name, POI category. In the so-called C-Browser solution it is possible to store map decorators and the Open Layers code itself in the cache, thus minimizing the amount of data to download in order to render a certain map. In this solution, the client device must be capable of executing advanced JavaScript code, and to manage and maintain the geospatial data cache. Map rendering is fully executed on the client-side, which in turn requires a not negligible amount of resources (CPU, RAM, Power) to be consumed.

The second solution, namely S-Browser, has been implemented in order to overcome the issues related to client-side rendering on resource-constrained devices. In fact, it allows any device equipped with a web browser capable of minimal JavaScript capabilities, to render our indoor maps. In this implementation, indoor maps are rendered on the server-side, and presented to the client-side as dynamically generated raster maps, rather than vector data. Such a solution requires an additional component to be deployed on the server-side, namely Map Builder, which provides a web-based interface to generate raster maps and present them to the client-side within an HTML page. It is worth noting that in the S-Browser solution, the map image created at the server-side is not a simple static image. In fact, while Openlayers is capable to add some basic map navigation functions like map-zoom and map-pan, in the S-Browser solution these functionalities are added by the Map Builder on the server-side and presented to the client-side through dedicated panning/zooming controls within the generated dynamic HTML page.

V. Evaluating the Indoor GIS Prototype

The decision about which of the presented architectural solutions should be adopted to implement a mobile GIS is not straightforward. Both server and client side approaches have advantages, especially when users perform repeated operations on maps like displaying points of interest or panning. While on the one hand server-side rendering does not require intensive computation on the mobile device, on the other hand it requires maps to be transferred as they are in full size. In this section we provide several insights for mobile GIS systems on existing mobile devices.

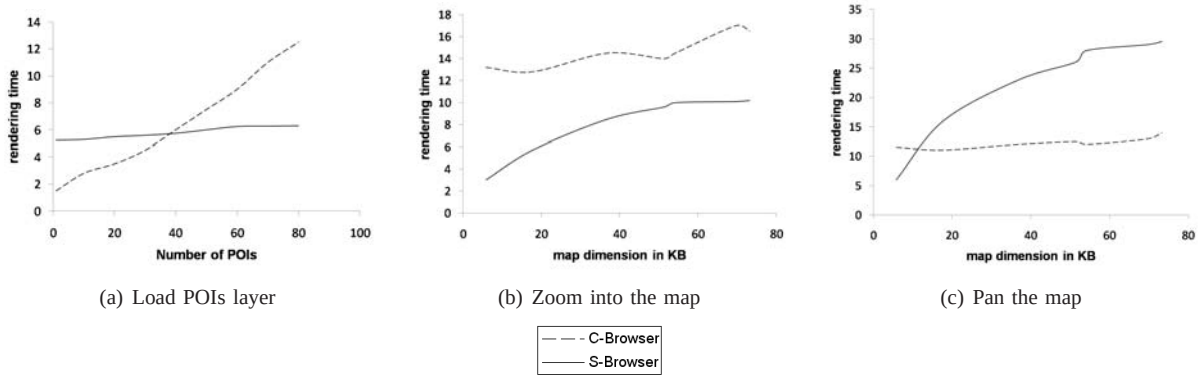
A. Comparing server- and client-side GIS rendering

For GIS applications, the type of used maps and the capabilities of the device used for browsing the maps affect the performance of the map rendering process. The complexity of the maps to render can vary from very simple maps (depicting only room borders and corridors) to more detailed and complex maps. While client-side rendering of maps can be efficiently executed on commodity personal computers, this is different on a resource constrained device, such as a mobile phone or a PDA. These are the typical cases for the indoor maps, where the users want to see the maps while they are looking for the right gate of an airport or for the food area of a shopping mall. In general, there are two extreme cases for GIS applications:

- Users with powerful devices and fast network connection requesting detailed complex maps.
- Users with very resource-constrained devices requesting very simple maps.

For the first case, client-side rendering is clearly desirable, while server-side rendering fits in the second case. Mobile GIS users typically need to render maps and geo-spatial data on their own personal mobile devices. For current mobile devices with fast 3G connectivity, it is not obvious

doi:10.4304/jsw.4.7.664-674



(a) Load POIs layer (b) Zoom into the map (c) Pan the map

Figure 4. A comparison of C-Browser and S-Browser on map rendering time, in three use cases

which approach to take. It is important to understand how the map complexity and the device capabilities can influence the decision of rendering the map at the client-side or at the server-side.

Our decision strategy is based on the evaluation of the following key properties of the system to implement:

- Performance of the client and server-side systems. We focus here on the client-side, where limited mobile devices are the main issue
- Size and complexity of the map material considering also display size, which has a direct impact on the generated data-traffic.
- Operations to modify/navigate through maps (panning, zooming, decorating)

As regards performance, a meaningful parameter to compare different solutions is the map rendering time. We define the map rendering time as the time spent by the application to satisfy a user request. In fact, it includes the time to create the map, at the server-side or at the client-side, and the time to render the map on the client device display. Furthermore, the analysis should consider one user operation or a sequence of typical operations within a given set. The rendering time is influenced by several aspects of the maps to render, which are shortly described in the following.

Tiling vs Untiling: Tiling allows to reduce the complexity of large maps by splitting them in smaller and simpler to render blocks. Tile size is an important parameter that affects the performance of the system. Tiling typically results in increased overhead on the client-side in terms of network connections (the web browsers download each single tile as a separate image) and CPU (due to tile caching algorithms typically implemented on client-side to enhance the end-user experience). Setting the optimal value for tile size is not straightforward since it depends on screen resolution and size, which are very device-specific parameters.

Number / type of POIs: POIs can be either static (their location do not change over time) or dynamic (represent objects or people dynamically moving across a certain space). Some examples of static points of interest for indoor maps are toilets, restaurants, elevators and escalators. Dynamic points of interest in our GIS prototype could be for example the locations of friends. In addition, the number of POIs and the amount of metadata (POI's attributes) coupled with them is a factor that affects the map rendering time. Considering dynamic points for example, it is more advantageous to render the map at the client-side because the client could keep the previously map and only update the old point location with the new one. If the map is rendered at the server-side however, application needs to request new maps continuously only to update the location of the dynamic point of interest. On the other hand, when the number of POIs available for a map increases, the application at the client-side has to manage more data for drawing the points on the map, which in turn can negatively affect map rendering performance (this is especially true for resource constrained devices).

Type / size of geospatial data: geospatial data is delivered as either vector or raster data. Vector data formats allow enhancing the look-and-feel of delivered maps and scale it to different types of devices, even though they are heavier to render on client-side than raster data. Furthermore, most of GIS-related vector data format may not match the licensing and data-protection requirements of map data owners. For example, delivering vector data through the OGC Web Feature Service implies this very detailed information about cartography/topology is delivered as a text document (XML-based) to the client side via a web interface. Hence many services prefer to send a raster map built at the server-side using the vector data. While the former solution his harder to design and requires specific algorithms for client-side rendering, the latter is easier and does not require any knowledge for the map rendering application either when the final map (base map plus decorations) is built at the server-side or at the client-side.

B. Evaluation Results

Following the above, we have first evaluated client-side vs server-side rendering for several parameters in an isolated setup. In this setup, a simple browser script draws maps and performance is measured. For a more realistic setting, we also implemented a realistic application and evaluated several use cases for these. In this setup, we

doi:10.4304/jsw.4.7.664-674

Indoor map	One floor of a shopping mall. The size of this map was about 20 KB
Number of layers	The map was composed by two layers. One layer with the base map (rooms, corridor) and one layer with POIs (elevators, escalators, shops, toilets...)
Type of Points of Interest	Static. The position of POIs were fixed.
Number of Point of Interest	81 POIs. POI data included geospatial coordinates, unique id, name, country, city, building, floor, section, and category
Map operations	Load base layer, Load POIs layer, zoom in and zoom out, pan, and query a POI.
Client Device	Nokia N95 with 64 MB of RAM connected with our Indoor GIS by a wireless LAN.
Sampling interval	250 ms. Each 250 ms we took a sample of memory, cpu and power status of the client device

TABLE I.
SIMULATION SETUP PARAMETERS

perform a set of operations as bundle and compare the overall results. While this does not give such detailed comparisons, it shows a more holistic view of the end user impact. Due to the different nature of these setups, we could compare power consumption and memory usage only for the second evaluation method. We compared client-side and server-side rendering in three main use cases:

- Load POIs layer
- Zoom into the map
- Pan the map

The results of this comparison are depicted in Figure 4. In the first use case, the user starts from a base map and wants to add a decoration layer with static points of interest. The chart related to the use case shows how the map rendering time changes based on the number of static points of interest used for decorating the map. While the number of POIs does not affect the S-Browser solution, it is evident that it affects quite much the C-Browser one. Since in the C-Browser the map is rendered at the client-side, more points there are on the map and more time the application takes to draw them in the right location.

In the other two use cases, we observed the dependence of the map rendering time from the size of the map. While in ”zoom into the map” use case the user, starting from a map with a certain level of zoom, wants to make bigger one section of that map, in the ”pan the map” use case, the user, starting form the middle of a map, browses it to see the most far left place. It is worth noting that we used the map size to simulate complex and simple maps because a complex detailed map has a bigger size than a simple map. In both use cases, the map size affects much more the performance of the S-Browser solution since it downloads a new image after any browsing actions. This is more visible in the ”pan the map” use case because the application need to download a new map each time the user moves towards left side of the map.

In order to validate and refine our indoor GIS prototype further, we implemented a case study application, namely **Indoor Map Viewer**, which combined our indoor GIS back-end services and accessed them through C-Browser and S-Browser. Similarly, in order to test the indoor location sensing feature, we implemented a **Friend Finder** application which allowed end-users to check their own friends' location, to see it on a map, and to evaluate the distance between themselves and their friends in the smart place. In order to show indoor maps, the **Friend Finder** application re-used most of the **Indoor Map Viewer** functionality. Similarly, in order to validate and refine the location-based search functionality, we implemented a **Location-tagged Contents** case study application, which allowed end-users to generate, search, and retrieve location-based contents, such as pictures, videos, text documents, and blog posts and comments. The implemented applications allowed us to evaluate and refine the web-based approach to indoor LBS service provisioning to commercial mobile devices. The implemented applications confirmed the feasibility of using web technologies for fast and easy deployment of smart places services. Most of the time was spent to implement and refine the application logic of our indoor GIS components, and we were able to easily deploy and run the services on a heterogeneous device base. As for server-side components, we were able to deploy them on Linux, Windows, and Apple Mac Os X devices. The implemented case study examples, realized as Ajax applications, could be accessed from mobile devices, including Nokia S60 devices as well as Linux Internet Tablets (Nokia N800 and N810), and in general from any device running a web browser that included either the Mozilla or Safari web engines (including desktop / laptop clients). Table I shows the main parameters of the evaluation setup. The aim of this case study was to measure the performance of our client solutions on resource constrained devices. To this aim, we used commodity Nokia S60 smart-phones. Data was collected by using the Nokia Carbide Profiler tool, using a sampling interval of 250 ms. Network traffic was monitored and analyzed through a TCP/IP network sniffer. The data collected through the Carbide Profiler included CPU load, memory usage, and power consumption. Tables II, III and IV show the results of our experiments with respect to the C- and S-Browser solutions. In tables II and III, the *CPU time* column represents the time frame (in seconds) during which more than 90% of the CPU was

doi:10.4304/jsw.4.7.664-674

	load base map layer	load POIs layer	Zoom in	POIs query
RENDERING TIME	11s	13s	11s	1.5s
CPU TIME	8s	11.5s	9s	1s

TABLE II.
C-BROWSER

	load base map layer	load POIs layer	Zoom in	POIs query
RENDERING TIME	5s	3.5s	3.5s	2s
CPU TIME	2s	1s	1s	$<$1s

TABLE III.
S-BROWSER

	MEMORY USAGE	POWER CONSUMPTION	NETWORK DATA
C-Browser	12632 KB	1350 mW	409,229 KB
S-Browser	1500 KB	1550 mW	528,883 KB

TABLE IV.
C-BROWSER VS S-BROWSER

allocated to the browser application. The *memory usage* column shows the amount of RAM memory allocated to the browser application. Finally, the power consumption column indicates how long the device can keep on with the current workload based on the average power consumed by the phone during the test case. Furthermore, we measured the rendering time and the amount of data exchanged on the network during a single map request consisting of the following main steps:

- Loading the base map.
- Loading a POI layer.
- Zooming in.
- Querying the map in order to retrieve additional POI metadata.

We structured the experiments according to those four operations and for each of them there is a corresponding value in the table. As far as performance is concerned, the S-Browser solution is quite better then the C-Browser solution. In fact, during the first three phases, the C-Browser rendering time is 2 times higher than that of the S-Browser. Only in the last phase, where the user queries the map, the C-Browser is just a bit faster than the S-Browser, because in the C-Browser the data are kept in memory since the previous elaboration for drawing the POIs layer and in S-Browser these data have to be downloaded from the server. As regards to CPU load and memory usage (first column table IV), the S-Browser solution shows a better behaviour than the other one. In fact, the data collected for the S-Browser are one magnitude order better then those of the C-Browser. Such results are justified by the different workload on the client in the two approaches. When we use OpenLayers, the map is built at the client side, using the data retrieved from the GIS server or cached at the client side. This operation requires both CPU time and RAM memory space. This is especially true for the POI layer loading and rendering phase. In fact, drawing the POI layer on the map requires a lot of RAM memory, because all the data about the available POIs must be loaded and stored in RAM memory. When the size of the POI layer exceeds the maximum amount of RAM available on the device, the browser application hangs. Our experiments showed that dynamic memory allocation in OpenLayers components has not been optimized for memory constrained devices. A lot of RAM resources, allocated by OpenLayers code, were not released, thus leading to frequent and not negligible memory leaks, which in turn caused also more recent and powerful devices to return memory full errors when trying to visualize some of the implemented applications. While these dynamic memory issues do not cause any problems in desktop or Internet Tablet devices, on certain low-end or less recent devices (e.g. Nokia N80 or E70) it was impossible to run the implemented applications, due to the limited amount of RAM memory available on those devices. Instead, using the S-Browser solution, the overhead on the client side is very low. The only tasks for the client is to make a request for the map, specifying all the layers he wants to use, and to download the raster image dynamically generated by the Map Builder component. Finally, as regards power consumption and network data traffic, we emulated a typical user behaviour with the map viewer prototypes. In the simulation, the client first sees the whole map, then zooms into one section of the map and pans among closed section, then zooms out to see the whole map again and to query some POIs, and finally he zooms in again in one section to look again the area in which he is interested in. Each emulation round lasted for one minute. Second and third column in table IV show the values of the average power consumption, in milliWatt, and of the amount of data, in KiloByte, seen during the simulation. With regard to Power Consumption, we have about 1550mW for S-Browser against about 1350mW for C-Browser. The battery of the used Nokia S60 smartphone has a capacity of 950mAh with a voltage of 3.7V (about 3515 mW). Therefore, the user can use the S-Browser application for about 2 hours and 15 minutes, while C-Browser could be continuously used for about 2 hours and 40 minutes.

doi:10.4304/jsw.4.7.664-674

Such a result can be related to the different usage of the network connection in the two different solutions. In the mobile device, the network service is one of the services that requires more power, and the solution based on S-Browser uses the network much more than the other solution because it gets a new map, from the GIS server, after each operation. In fact, the network analyzer reported that with C-Browser, almost 410 KB were exchanged between the client and the server against almost 530 KB of the S-Browser solution. It is worth noting that in all the simulation with C-Browser, we used the cache to keep the OpenLayers JavaScript code and the data for building the POIs layer. Otherwise, the C-Browser needs to download much more data. For the same one minute long simulation, if for the client side we use C-Browser without the cache, the client and the server exchange almost 1500 KB. It is worth noting that the C-Browser solution presented also a JavaScript/DOM related issues: OpenLayers is strongly dependent on DOM 2.0 APIs. OpenLayers assumes a complete DOM 2.0 or 3.0 model to be supported by the browser engine. Unfortunately, browser engines even on very recent devices (like the Nokia N95) do not fully support these specifications. This created problems when parsing some of the XML documents returned by our smart places API through the DOM API. The problem was solved by re-coding the applications in such a way that they were relying on JSON-formatted interface rather than on ATOM/RSS formatted data. In this way we were able to guarantee that all applications could still work on all the mentioned types of devices. Such a modification had also the positive side-effect of improving the performance of the provided applications due to the more light-weight logic required for parsing and creating the data.

VI. Summary

This paper discussed our experiences with building an indoor GIS based on commodity GIS standards and protocols and Web 2.0 application development principles. We showed that the combination of scripting languages with web application frameworks, such as Python and Django, gave us a lot of flexibility in designing the actual web service interfaces we wanted to use to expose indoor GIS data. We implemented a few case study applications on top of the proposed GIS solution, which confirmed the feasibility of using web technologies for fast and easy deployment of smart places services on a heterogeneous set of commercial off-the-shelf devices. As far as GIS clients are concerned, we pointed out that using commodity libraries, such as OpenLayers, can create a few performance issues for resource constrained devices. Moreover, commodity libraries might have not been designed by taking into account the limited RAM capacity of mobile devices. Frequent and not negligible memory leaks created problems also on more recent and powerful devices. By adopting a server-side map rendering strategy we delivered our indoor location-based services to low end devices. Our experiments point it out that the main drawback of client-side map rendering approaches is the amount of information needed to build the map. Similarly, a critical aspect for server-side map rendering is the amount of data downloaded from the server. If the internet connection available for the client device is slow and the size of the map is big, operation like panning the map can be very slow and unusable for the client. Overall, we believe there is a clear need in research and industry to agree on standards for indoor LBS, with respect to both geospatial data representations and related APIs to re-use them in a Web 2.0 environment. Existing outdoor-related standards might lay the groundwork for such activities, even though they should be extended in order to support not only the concept of location as physical position but also symbolic location concepts.

Acknowledgements The authors would like to thank all their project co-workers, in particular Jilles van Gurp, Heikki Mattila, Jaakko Kyro, and Pasi Liimatainen.

References

[1] C. Endres, A. Butz, and A. MacWilliams, "A survey of software infrastructures and frameworks for ubiquitous computing," *Mob. Inf. Syst.*, vol. 1, no. 1, pp. 41–80, 2005.
[2] "Opengis standards specifications." [Online]. Available: http://www.opengeospatial.org/standards
[3] "Delivery context client interfaces (dcci), w3c candidate recommendation," Dec. 2007. [Online]. Available: http://www.w3.org/TR/DPF/
[4] "Oauth core 1.0 protocol," Dec. 2007. [Online]. Available: http://oauth.net/core/1.0/
[5] R. Simon and P. Frohlich, "A mobile application framework for the geospatial web," *WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM*, pp. 381–390, 2007.
[6] J. W. Kim, C. S. Kim, A. Gautam, and Y. Lee, "Location-based tour guide system using mobile gis and web crawling," in *W2GIS*, ser. Lecture Notes in Computer Science, Y. J. Kwon, A. Bouju, and C. Claramunt, Eds., vol. 3428. Springer, 2004, pp. 51–63.
[7] J. Candy, "A mobile indoor location-based gis application," *5th International Symposium on Mobile Mapping Technologies (MMT07), Padua, Italy*, 2007.
[8] J. van Gurp, C. Prehofer, and C. di Flora, "Experiences with realizing smart space web service applications," *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 1171–1175, Jan. 2008.
[9] C. Becker and F. Durr, "On location models for ubiquitous computing," *Personal Ubiquitous Comput.*, vol. 9, no. 1, pp. 20–31, 2005.
[10] C. di Flora and M. Hermersdorf, "A practical implementation of indoor location-based services using simple wifi positioning," *Journal of Location Based Services*, vol. 2, no. 2, pp. 87–111, June 2008.
[11] A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki, "Practical robust localization over large-scale 802.11 wireless networks," pp. 70–84, 2004.
[12] "Geoserver web site." [Online]. Available: http://geoserver.org
[13] "Mapserver web site." [Online]. Available: http://mapserver.gis.umn.edu
[14] "Openlayers web site," http://www.openlayers.org.

doi:10.4304/jsw.4.7.664-674

Valerio Vianello received the degree cum laude in computer engineering from the University of Naples Federico II, Italy, in 2007. He is currently working toward the Ph.D. degree at the University of Naples Parthenope.

Cristiano di Flora is a Software Architect in Nokia Devices RD, Maemo Software, Tampere (Finland). He holds a Ph.D. Degree in Computer Engineering from the ”Federico II” University of Napoli (Italy).

Christian Prehofer is Distinguished Research Leader in Nokia Research. In the last ten years, he held different management and research positions in the mobile communication industry. He obtained his Ph.D. and his habilitation in computer science from the TU Munich in 1995 and 2000.

doi:10.4304/jsw.4.7.664-674