Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 32 (2014) 445 – 452

Procedia Computer Science

5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014)

Building-Linked Location-Based Instantaneous Services System

Julian Ohrt*, Volker Turau

Hamburg University of Technology, Institute of Telematics, Schwarzenbergstr. 95, 21073 Hamburg, Germany

Abstract

For services which are meant to be used mainly inside buildings, a distribution using global app stores is inappropriate. Instead this work presents a provisioning system (*BLISS*) for building-linked location-based services. The multi-purpose application *MultiApp* implements the client-side of the *BLISS* specification. When a user enters a building, *MultiApp* instantaneously detects available services and allows to install and execute them. The client part of services is executed within the *MultiApp* environment and can provide functionalities similar to those offered by native applications. *BLISS* facilitates developing indoor location-based services by providing bi-directional communication between service providers and clients.

Selection and Peer-review under responsibility of the Program Chairs.

Keywords: indoor services, smart buildings, LBS framework

1. Introduction

The number of mobile applications in the Google Play Store as well as in Apple's App Store is constantly increasing. With each store offering well more than 800,000 apps [1], selecting those of interest is troublesome for users. As most stores only provide text based searching, the search results depend on the quality of the app description and the entered search keywords. For the subset of apps offering services which intended usage is bound to a limited geographical area, it is much more user-friendly if those services were presented proactively to the user at the appropriate location. Especially inside buildings this could help users find relevant services quickly, e.g., to control the lights in the current room or to navigate within the building. Even though location-based services (LBS) in general are becoming more popular, those restricted to indoor environments, have not received much attention by research and industry. Most probably because of a missing cheap and reliable localization technology which allows room-based or even more precise localization of mobile devices within buildings. However, the authors believe that such a technology will become generally available in the foreseeable future.

Assuming that indoor localization is available, this work presents building-linked services useful to house owners, office employees, or visitors in public buildings. Building-linked in this context means that the area of intended usage of the service is inside or close to the building. Further, building-linked services are usually provided by devices inside the building, e.g., a control unit of an elevator or a smartphone. Based on the presented services, requirements for a system for providing building-linked, location-based services are deduced. The main contribution of this work is a specification, design and

* Corresponding author. Tel.: +49-40-42878-3704 ; fax: +49-40-42878-2581.
E-mail address: julian.ohrt@tuhh.de

Selection and Peer-review under responsibility of the Program Chairs.
doi:10.1016/j.procs.2014.05.446

implementation of such a system called *BLISS*: Building-Linked Location-Based Instantaneous Services System. The client part of *BLISS* is implemented by a smartphone app called *MultiApp*. It is an application which is installed only once on a device and can autonomously detect and install available *BLISS* services. Both reactive and proactive usage patterns of services are supported. For example, an elevator could thus be called manually or automatically when approaching it. The *BLISS* specification together with *MultiApp* is intended to facilitate the development as well as the usage of indoor LBS.

2. Related work

The development of LBS is at least since 2005 an objective for many researchers. First, emphasis was on position acquisition without GPS [2] and position management between devices [3]. In 2008, Priggouris et al. proposed an XML-based format for LBS which can be injected and installed at run-time into a multi-purpose framework application [4]. One year later, Rechert presented MobIS; an LBS framework for developing standalone mobile applications [5]. To the best of our knowledge, MobIS is the only developing framework which was actually used to create commercial applications. With the introduction of iOS and Android, however, all mentioned technologies became obsolete as they were not compatible with these new operating systems. In 2010, Bareth et al. proposed a concept similar to the one by Priggouris. It is extended by a provider server which allows to search apps by geographic region. Further, installed apps can be invoked proactively, by entering, crossing, or leaving pre-defined regions (geofences). However, the actions the service can execute on the smartphone are only defined vaguely [6].

All reviewed literature on universal concepts for creating LBS makes strong simplifications or only considers special types of LBS. In Table 1 we present LBS use-cases mainly for usage indoors. To the best of our knowledge, there is no framework or tool available which allows modeling all of the below mentioned use-cases.

Table 1: Use-cases of indoor location-based services

(1) `Navigation:` Similar to GPS-based navigation systems, this service is able to show a map of the current building, it shows the user's position, provides a list of points-of-interest, and allows to calculate paths between multiple positions.	(2) `Door Bell:` Instead of using a conventional door bell, this service allows to do the same using a smartphone. A proactive version can – after being installed – automatically ring the bell when approaching the door.	(3) `Counting People:` This service counts the number of persons that are inside a given geofence. It allows proactively informing the client user when persons enter or leave the geofence. For detection the server requires appropriate sensors.
(4) `Locate Me:` This service is installed as service provider on a mobile device for a specific building. Other smartphones when being inside the same building can detect and install it. Afterwards they are able to query the service provider for its current position and heading (server requires compass).	(5) `Switch Service:` This service is used to control the lights when being inside a room but with the light switch being out of reach. It can also be used for remotely switching off a group of lights. As enhanced use-case this service makes the light follow the user as he is moving.	(6) `Voucher Service:` After installing this service, a user may receive vouchers in the vicinity of participating establishments, like restaurants or shops. The delivery of vouchers may depend on factors like opening hours, duration of staying close, time of last voucher, weather conditions, etc.
(7) `Information Request:` Telling this service where you are, will make it provide you with location-linked information regarding a specific question. Services of this type could answer questions, like "On which days of the week is the current room cleaned by the cleaning personal?" or "Who's office is this?".	(8) `User Configuration:` When approaching a predefined geofence, this service activates a setting or a function. Examples include: switching on the light when entering a room or calling the elevator when approaching the office building. In case of shared devices access priorities need to be managed.	(9) `Home Guard:` The home guard service checks for and warns about potentially dangerous or undesirable situations. For example, when the last resident leaves the house the service checks whether all doors and windows are locked. For detecting according situations the server requires accessible sensors.
(10) `Bulletin Board:` Users of this service can post and view multimedia messages at virtual, room-linked bulletin boards. When entering a room with new messages, the user is proactively informed.	(11) `Trace Users:` Mobile users signed up for this service, regularly or when changing position, send their own position. This way movement profiles of registered users are created.	(12) `Tour Guide:` In a museum the tour guide service shows the visitors proactively information about the closest exhibits. The compass of the smartphone is used to order the list by relevance.
(13) `Reception Service:` This service welcomes invited guests in large buildings. When a guest enters a building for a meeting, the navigation service is started and leads the guest to the location of the appointment. Optionally, the host is informed about the arrival of the guest.	(14) `Smartphone Reminder:` When leaving the office – i.e. when the door is being closed – this service checks whether the owner's smartphone is near by. If it is close but not moving, it was probably left inside the office, and informs the owner audibly about this potential oversight.	(15) `Intruder Alert:` Having a sensor attached to a door which detects the opening of the door, this service determines whether any of the owners (of the office, house, etc.) is close. If not, an intruder might have opened the door and all owners are being alarmed via their smartphones.

3. Use-Cases and Requirements

This section summarizes requirements for a system able to offer building-linked location-based services. Table 1 summarizes 15 use-cases of such services. The last three have not been discussed by literature or research. Even though the list contains only a fraction of conceivable services, the authors believe that a system, which is able to model these presented use-cases, can also be used to create many other, useful services. Based on Table 1 a specification for the system is defined.

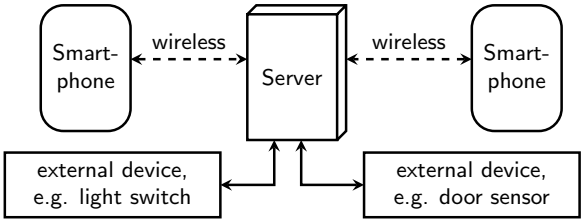


Fig. 1: Involved entities in location-based services

Following hardware entities are essential: Smartphones, stationary servers acting as service providers, and optionally external devices which are attached to the servers. The smartphones communicate wirelessly with the servers as shown in Fig. 1. Also smartphones can act as servers and provide services.

3.1. General Requirements

Communication between server and client should not rely on an Internet connection. All service clients (i.e., smartphones) must be aware of their own location. Since location information is often shared (e.g., `Locate Me` (4) or `Trace Users` (11) services) and services might be abused or manipulated (e.g., `Voucher Service` (6) or `Intruder Alert` (15)), authentication and authorization of all involved users and devices is required. However, in this work access restrictions are not considered.

To circumvent a single point of failure, the system must not depend on a server for coordinating buildings or services. This way there is no single point of knowledge of position data of users (trusted-third-party), which drastically increases user privacy. Further, users can choose which services they want to install. Their position is never revealed to any other service.

3.2. Detection

The main idea of this work is that any service in any building can be detected autonomously by the client. Thus, the first, most important requirement is the detection of available services. Most services should be detectable in the whole building, e.g., the `Home Guard` (9) or the `Tour Guide` (12) services. However, services with a limited operating range should only be found within their range, e.g., a `User Configuration` service (8) for a specific device, or the `Smartphone Reminder` service (14) which is linked to a room. Thus, a service must be able to limit its detection range.

Further, there are services which do not require to be detectable at all times. E.g., the `Locate Me` service (4) is only detectable for a short time during which a partner device installs the service. Afterwards the service is hidden so no third party can detect it. Thus, a service must be able to control its visibility.

3.3. Activation and Installation

After detection, services of interest need to be activated. Activation may either be a single time execution of the service (e.g., useful for the `Navigation` (1), `Door Bell` (2), and `Switch` (5) services) or it may involve an installation. In the later case the service can be invoked later, optionally even if the server is not in range anymore. An installed service can be running in the background without permanent connection to the server, e.g., the `Voucher Service` (6) or the `Bulletin Board` service (10) to inform the user about new vouchers and posted messages, respectively, when the server becomes available.

The system must also support services which should not be detected autonomously. The `Reception Service` (13), e.g., is only meant for invited visitors of a building. They should be proactively welcomed by the service when approaching or

entering the building, consequently the service must be running and thus be installed a priori. Hence, manual installation of services is required, e.g., by entering a link or scanning a QR code (provided by the host when inviting guests).

No service may become active before executed by the user.

3.4. Service functionalities

Each service is a distributed application. It consists of a server part (SP) and a client part (CP). Whereas the SP is only stored on and executed by the service provider, the CP needs to be transferred to and executed by client smartphones. Communication capabilities between SP and CP are discussed in Section 3.5.

3.4.1. Server part (SP)

Two cases need to be supported: SP can either be executed on a stationary server or on a smartphone (e.g., for the `Locate Me` service (4)). For smartphones, SPs may be implemented as native apps which are platform dependent. It must be possible to support common mobile operating systems. For stationary servers the same SP needs to run on different operating systems.

For both cases, SPs are permanent background processes which expose their offered services. They must be able to perform server-side logic, store and retrieve data, manage user sessions, provide a GUI for administration purposes (e.g., to show movement profiles collected by the `Trace Users` service (11)), and be able to access hardware components. When running on a smartphone only integrated hardware is supported, e.g., the compass for the `Locate Me` service (4). For stationary servers the SP of services also needs to be able to access and control third-party hardware, e.g., motion detection sensors for the `Counting People` service (3).

3.4.2. Client part (CP)

The client part of a service is executed on smartphones within a single multi-purpose app. While the implemenation of the app depends on the mobile platform, the CP must be operating system independent.

The CP is able to interact with the user using graphical interfaces, show multimedia content, and obtain the current position. It can run quietly in the background and become active on server, timing, or positioning events (e.g., the `User Configuration` (8) or `Home Guard` (9) services). It also needs to access system functionalities and smartphone hardware, e.g., the motion sensor for the `Smartphone Reminder` service (14) or the camera for the `Bulletin Board` service (10) to share a photo. Furthermore, the CP needs to be able to execute complex algorithms, e.g., homomorphic encryption to protect positioning information. Summarizing, its range of required functionalities is comparable to that of native smartphone applications. However, its code is transferred dynamically to the client and must be operating system independent.

3.5. Communication

The first communication between client and service provider is about publishing and discovering available services. Once the user chooses to install a service, the system defines means for the client to download the CP from the server.

After installation, communication takes place between a matching pair of SP and CP. Both entities need to be able to initiate communication. CP-to-CP communication does not need to be supported. Further, there are conceivable use-cases where a smartphone assumes the role of the server, e.g., the `Locate Me` service (4), thus SP-to-CP communication where both parts are executed by smartphones needs to be supported.

Additionally, services may create their own communication sockets – independent of *BLISS* – to enable usage of third-party protocols like WebDAV or RMI and to contact third-party servers.

4. Design and Implementation of *BLISS* and *MultiApp*

This section describes the design of *BLISS*. The system should support all above introduced use-cases as well as all requirements described in the previous section. The goal of this design is to be able to implement a multi-purpose application for building-linked location-based services (*MultiApp*) which is installed only once on a smartphone. It is able to detect and access any number of services in multiple buildings which comply to the *BLISS* specification. SPs are implemented as standalone applications. Interfaces are specified as much as required in order to guarantee smooth interaction between *MultiApp*, CP, and SP.

BLISS is intended for usage within buildings offering a reliable local area network with full WiFi coverage. Mobile users as well as servers are connected to this network. The network configuration must allow servers to listen for and accept incoming socket communications. Mobile ad-hoc networks, e.g., using Bluetooth, are not supported.

4.1. Existing software components and technologies

For implementation of *MultiApp*, CPs, and SPs, open standards will be used. An overview of current technologies for fundamental functionalities required by *BLISS* is provided in the following.

4.1.1. Detecting services

Being connected to a building-wide wireless local area network *BLISS* can rely on multicast approaches for detecting services. Discovery in peer-to-peer networks as described by Srirama et al.[7] is not required for building-linked services and is thus not considered.

Available protocols for LAN discovery include: the DNS-Based Service Discovery, the Simple Service Discovery Protocol defined and used by Universal Plug and Play (UPnP), and the Service Location Protocol.

UPnP does not only provide a discovery mechanism but also supports remote procedure calls and event notifications. Its open and detailed specification allows implementing UPnP for current operating systems. Further, several open source implementations of the UPnP protocol stack are available. Due to these reasons *BLISS* will rely on UPnP. However, it is expected that modifications and/or enhancements to the UPnP standard are required.

4.1.2. Service structure

As *BLISS* services are distributed applications offered by a server, the CP needs to be transferred from the server to the client. It must be able to perform all tasks described in Section 3.4. It thus contains program code which has to be executed on the client. The CP can either be native or interpreted. For native program code individual CPs for each supported target operating system are required. This increases development efforts and costs. This approach is thus not further considered.

In case of interpreted CPs, this paper proposes two technologies which could be adapted by *BLISS*.

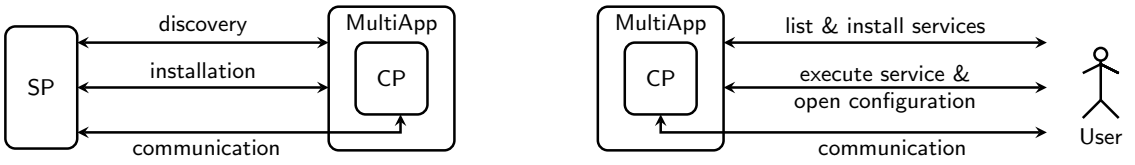
OSGi The OSGi service platform allows dynamically loading Java software components (Bundles). Each Bundle offers any number of OSGi services. The CP of a *BLISS* service can be a Bundle. *MultiApp* needs to provide a Bundle container which allows loading and unloading Bundles at runtime. However, by default Bundles cannot provide a graphical user interface on server-side. To circumvent this issue Berndt evaluates technologies for generating interactive OSGi components using meta models for defining GUI elements[8]. Alternatively, for the Android platform Bundles could be adapted in such a way that they contain native Android GUIs. This is possible because Android applications are also Java programs. For other operating systems this approach is not directly applicable.

Apache Cordova Better portability to different operating systems provide CPs which are written in JavaScript and executed inside the Apache Cordova framework. Cordova – formerly known as PhoneGap – allows JavaScript code to issue native system calls, e.g., to access address book or camera. At the same time, with CP being a web application running inside a browser frame, the GUI is provided using HTML components. Due to the high portability of Cordova web applications and simplicity of creating GUIs, this later approach is chosen for developing *BLISS*.

4.1.3. Communication

BLISS requires both client and server – which may be smartphone or stationary server – to initiate communication connections. Contacting stationary servers with fixed IP addresses is possible because those servers can be configured to listen on predefined TCP ports for incoming connections. Contacting smartphones is more complicated since IP addresses of mobile clients can change at any time.

Modern mobile operating systems provide a notification services to circumvent this issue (e.g., Apple Push Notification Services or Google Cloud Messaging for Android). They make use of a persistent connection between smartphone and a server of the mobile operating system provider. The server forwards messages on behalf of third-party services to the mobile devices. A drawback of this approach is the requirement of permanent Internet connectivity. Furthermore, messages sent by the notification services are limited, both in content size and number.



(a) Communication between server and client

(b) Interaction between client and user

Fig. 2: Interfaces of *MultiApp*

Since this project assumes that clients are not permanently connected to the Internet, *BLISS* cannot solely rely on an Internet-based notification service for pushing messages to clients. However, since constant connectivity to the local network is assumed, the UPnP eventing system is applicable. UPnP is also suited to invoke methods offered by the SP, even if its IP address changes. UPnP is thus the technology of choice for implementing *BLISS*.

4.1.4. Localization

More than one decade indoor localization technologies have been proposed and researched. A survey of existing techniques for indoor positioning and navigation is provided by Mautz[9]. Even though this paper does not put emphasis on the localization technology, at least an approximate user position is required for testing and evaluating *BLISS* and its services. *BLISS* will rely on a simple, room-based fingerprinting approach presented by Zhou[10]. It works offline, relies on WiFi measurements only, and does thus not require additional hardware. It is published as part of the SvgNaviMap project introduced by Ohrt and Turau[11].

4.2. BLISS system structure

The SP is a standalone Java application which implements the UPnP protocol stack. Relying on Java, operating system independency is ensured. At the same time native device drivers can be accessed using *Java Native Access*. For smartphones, SP is implemented as native application.

Each SP implements a UPnP device which has a unique name and belongs to a category. For specifying a *working range* which is the physical area of its intended usage, the UPnP standard needs to be extended. Each UPnP device offers exactly one UPnP service which provides any number of UPnP actions. The SP needs to supply its CP which is transferred to and installed by clients. Any CP only interacts with its originating SP, e.g., the client part of a `Switch Service` (5) of one building will not switch lights in any other building. The communication protocol is described in Table 2.

The CP of a service is a compressed web application which is inflated by *MultiApp* and afterwards executed within the Cordova framework. *MultiApp* must thus provide a Cordova container which displays and executes CPs. The interface for communication between CP and *MultiApp* is specified in Section 4.3. The required user interface for *MultiApp* and its CPs is described in Table 3.

A CP which resides within SP is said to be *Uninitialized*. Once it is transferred to a client it enters *Running* state. It can be stopped either by the user, the CP itself, or *MultiApp*. Afterwards it can be started again. For the SP, *BLISS* distinguishes following lifecycle states: *Running*, *Hidden*, and *Stopped*. Only in state *Running* and *Hidden* a service can offer services. A hidden service does not reply to discovery requests. The discovery and installation interfaces are specified in Table 2.

4.3. MultiApp interfaces

This subsection specifies the *MultiApp* interfaces for interaction with SP, user, and CP.

Inter-device communication In order to publish and detect as well as install and use services, *BLISS* defines an interface for communication between server and client as shown in Table 2. While the first two interfaces are used by *MultiApp*, the communication interface is used by the CP only, as shown in Figure 2a.

User-Client interaction The interfaces required for interaction between the user and the client application are explained in Table 3 and graphically summarized in Figure 2b.

Table 2: *BLISS* interfaces for communication between server and client

`Discovery`	`Installation`	`Communication`
A server instantiates for each *BLISS* service one UPnP device with one UPnP service. For finding UPnP services, clients can query for name, type, and *working range* using the extended UPnP discovery protocol. Only matching services in *Running* state will respond. Hidden cannot be discovered. However, they may provide a URL to its UPnP description file which allows manual discovery and installation. For distributing URLs, optionally as QR-code, the SP requires a graphical user interface or communication capabilities, e.g., for sending an e-mail.	For running a service on the client, internally the service is installed and after termination de-installed. To install a service, *MultiApp* downloads the CP from the according UPnP service by calling the corresponding UPnP action *getCpUrl* which each service must implement. The return value is a URL from which *MultiApp* can download the CP archive. The SP must thus integrate, next to the UPnP device, a web server. After downloading the CP, *MultiApp* deflates and injects it into the Cordova container so it enters *Running* state.	When both CP and SP are running, the programming interface described in Section 4.3 can be used for mutual communication. CP can invoke all UPnP actions offered by its SP and may register for UPnP events. After successful registration the SP can send events to its clients by modifying UPnP state variables. As changes of state variables are propagated to all clients, the event handlers must implement an appropriate filter. Further, CPs can open HTML5 web sockets to contact third party servers. *MultiApp* deactivates the same origin policy for its CPs.

Table 3: *BLISS* user interfaces for interaction with *MultiApp* and CP

`Installation`	`Execution`	`User Interface`
MultiApp offers a list of detected services from which the user can choose to install. For hidden services which are not detected by *BLISS*, an interface is provided for manually entering the installation URL, which is provided by the SP of the service. *MultiApp* features a QR-code scanner for entering the URL.	*MultiApp* offers a list with all installed as well as all available services which do not need installation. For installed services it is also possible to access a configuration interface. When opening a service or its configuration interface, *MultiApp* displays a web container and loads according HTML files.	Once started, each service is responsible for its own GUI. The HTML files are displayed in a full screen web container. *MultiApp* provides a menu to close and switch services. When switching, the service continues running and can thus still receive notification events.

***MultiApp* programming interface** A CP is an enhanced web application. Next to HTML5 technologies it can also rely on JavaScript functions offered by *MultiApp*. To avoid repeated implementations, *MultiApp* provides a programming interface for its CPs which at least allows a) Retrieving the last known position, and b) Initiating a new localization measurement.

Further, *MultiApp* provides event notifications for the following events: a) Single and periodic time events, b) Entered, crossed, or left a geofence, c) Stayed inside or outside a geofence for a duration, d) Moved a distance (in any direction), and e) Notify service if new position data is available. If an event occurs and the service is not running, it is started by *MultiApp*.

Since *MultiApp* executes CPs inside a Cordova enabled web container, they can also access JavaScript functions which are offered by Cordova, e.g., to access the local file system or the native notification system. Finally, the UPnP eventing system and all UPnP actions exposed by the SP are made accessible by *MultiApp* to the corresponding CP.

5. Conclusions and Future Research

This work presents a collection of building-linked services useful to house owners, office employees, and visitors of public buildings. As these services are meant to be used mainly inside the according building, a distribution using global application stores (like Google's Play Store or Apple's App Store) is inappropriate. Instead this work presents the concept of a system (*BLISS*) which is able to detect offered services in a building as soon as a user enters it.

The multi-purpose application *MultiApp* implements the client-side of the *BLISS* specification. It allows to install and execute any number of services. The client-part of a service is independent of the mobile operating system. It can provide functionalities similar to those offered by native applications. *BLISS* further facilitates developing indoor location-based services by providing bi-directional communication between service providers and clients. It also provides the current location and allows clients to register for position events.

We believe that by using MultiApp most presented services can be implemented. Our experimental implementation of *MultiApp* proves that the concept is feasible for the Android platform. We implemented the `Home Guard` service (9) (without sensors) and the `Switch Service` (5). Both services can be run as standalone Java programs as well as Android applications.

Currently we are extending and enhancing *BLISS* and its *MultiApp*. The following list outlines limitations of the current design and provides ideas and concepts for improvements.

Mode of connection between server and client As a first step, this work assumes that service provider and smartphone are connected to same local network. For visitors of public buildings this is a severe limitation as they usually cannot connect to the local network. Solution approaches include a) relying on a mobile Internet connection or b) introducing a well known WiFi SSID to which *MultiApp* automatically connects.

Client-to-client communication At this point no client-to-client communication is supported. However, e.g., a paging service would benefit from it both in terms of response time as well as privacy. For enhancing *BLISS* we will evaluate the possibility of using push notifications between clients.

Communication outside buildings The current specification does not support client-server communication once the client left the local network. However, for some services global communication would be useful. E.g., the `Intruder Alert` service (15) should also be capable of warning owners that are outside the building about a potential intruder.

Session handling UPnP was not designed to handle multiple users in different contexts. Rather, there is one global service state which is accessible by all clients reading or listening to changes of the UPnP state variables. For privacy reasons and in order to facilitate session handling, alternatives to UPnP will be evaluated.

Container for SP We plan to introduce a service container for the execution of the SP. This way operating system dependencies are reduced. For services provided by smartphones the same Cordova environment as for the CP can be used. For desktop PCs it should be evaluated whether the OSGI technology is applicable.

Installing hidden services For installing a hidden service, clients only need the URL to UPnP description file. *BLISS* must support an authorization mechanism which allows SPs to make sure that it only offers its services to authorized clients.

Invoking external services *BLISS* does not allow services to activate other services at this point. However, e.g., the `Reception` service (13) requires to start the `Navigation` service (1).

References

1. Number of apps available in leading app stores as of june 2013. 2013. URL: `http://tiny.cc/number-of-apps`.
2. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., et al. Place lab: Device positioning using radio beacons in the wild. In: *Pervasive Computing*. Springer; 2005, p. 116–133.
3. Cupper, A., Treu, G., Linnhoff-Popien, C.. Trax: A device-centric middleware framework for location-based services. *Communications Magazine, IEEE* 2006;**44**(9):114–120.
4. Priggouris, I., Spanoudakis, D., Spanoudakis, M., Hadjiefthymiades, S.. Open middleware architecture for the lbs domain. *Journal of Electronic Commerce in Organizations (JECO)* 2008;**6**(1):19–37.
5. Rechert, K.. Mobis: A pragmatic framework for location based services. In: *Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on*. IEEE; 2009, p. 141–144.
6. Bareth, U., Küpper, A., Ruppel, P.. Geoxmart-a marketplace for geofence-based mobile services. In: *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*. IEEE; 2010, p. 101–106.
7. Srirama, S.N., Jarke, M., Zhu, H., Prinz, W.. Scalable mobile web service discovery in peer to peer networks. In: *Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on*. IEEE; 2008, p. 668–674.
8. Berndt, G.. *Generierung von interaktiven OSGi-Komponenten für Android*. Diploma thesis; Technical University Dresden; 2010.
9. Mautz, R.. *Indoor Positioning Technologies*. Ph.D. thesis; Institute of Geodesy and Photogrammetry, ETH Zurich; 2012. Habilitation thesis.
10. Zhou, Z.. *Accuracy of Machine Learning Based Wi-Fi Localization*. Project work; Hamburg University of Technology; 2013.
11. Ohrt, J., Turau, V.. Simple indoor routing on svg maps. *Proceedings of IEEE Workshop on Location-based Services for Indoor Smart Environments (LISE)* 2013;.