Spaceify - a Client-Edge-Server Ecosystem for Mobile Computing in Smart Spaces

Petri Savolainen†, Sumi Helal‡, Jukka Reitmaa†, Kai Kuikkaniemi†, Giulio Jacucci†, Mikko Rinne ¬, Marko Turpeinen†, Sasu Tarkoma†
†HIIT/University of Helsinki/Aalto University, firstname.lastname@hiit.fi
‡Aalto University and University of Florida, firstname.lastname@gmail.com
¬ Aalto University, firstname.lastname@aalto.fi

ABSTRACT

Spaceify is a novel edge architecture and an ecosystem for smart spaces — a technology that extends the mobile user view of today's common space services (e.g., WiFi) to a richer portfolio of space-centric, localized services and space-interactive applications.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design

Keywords

client-edge-server, architecture, smart space, ecosystem

1. INTRODUCTION

Resource-poor mobile devices can benefit from utilizing external resources that are available both in the cloud and in the proximity of the device. For example, offloading computation to the cloud can both make the mobile applications faster and make the battery of the mobile device last longer [5]. Additionally, using a server found in the environment for the offloading can help to keep latencies at bay [6]. Besides the computational, storage and networking resources utilized in mobile offloading, Internet of Things (IoT) proposals add sensors and actuators in the list of utilizable resources in the environment of the mobile device.

The web programming model is becoming increasingly popular on the mobile devices with HTML 5 and mobile operating systems such as Firefox OS [1] and Tizen [3] adding support for installable web applications. However, the traditional client-server model of the web, combined with the sandboxed JavaScript execution environments on the devices offers little support for utilizing the resources available in the cloud and in the proximity of the devices.

In this poster we present the Spaceify ecosystem that is based on client-edge-server architecture. In the client-edge-server model, a web application is executed not only on a client and a server, but also on an *edge node* which is typically a smart access point — a resource-rich device through which the mobile devices are connected to the internet. In addition to offering its own computing and storage resources to be used, the edge node also acts as a central hub through which the resources in the space become available to the three-tier web applications.

Spaceify supports running both transient (i.e. JavaScript programs embedded in HTTP responses) and persistent code (installed native and JavaScript applications) on the edge nodes. From here on we will refer to the embedded JavaScript code executed on the Spaceify edge nodes as *Spacelets* and the installed code as *native applications* and *JavaScript applications*. This naming convention is analogous to the well-known convention of calling on-page pieces of Java code "Java applets" and installed Java code "Java applications".

The main two goals of the Spaceify ecosystem can be summarized as follows:

(1) To contribute to the digital transformation of physical spaces such as malls, schools, university campuses, downtown districts, etc. Currently, the web is accessible from many spaces via WiFi or cellular, but then the specificity of the space is washed out in the vast morasses of the web content and apps. What Spaceify aims at is to transform the space into a web of its own — a specialized, local web of things strictly inside the space, including services, people, and sensors.

(2) To provide significantly enhanced user experience by enabling intuitive and effective interactions of mobile users with the space. Spaceify aims to achieve this goal over an evolving mobile / ubiquitous device technology.

To achieve these goals we designed an architecture based on a friction-free ecosystem, in which we do not require new standards or any restrictive software layering other than existing web technology.

2. RELATED WORK

Satya et al. [6] highlighted the impact of latency on the user experience of mobile offloading. In order to achieve low latency, they proposed deploying *cloudlets*, the counterpart of Spaceify edge nodes, resource-rich computers one hop away from the mobile devices. Their solution is, however, concentrated only on making the computational and storage resources of the edge server available to mobile applications, and they do not address utilizing other resources that may be available in the environment. In contrast to lightweight web technologies used in Spaceify, their solution is virtual-machine-based, which can impose high overheads. Indeed, according to measurements of Satya et al. [6], launching a virtual machine to serve the mobile clients can take tens of

MobiCom'13, September 30–October 4, Miami, FL, USA.
ACM 978-1-4503-1999-7/13/09.
http://dx.doi.org/10.1145/2500423.2504578.

seconds even on a dedicated cloudlet. Finally, a major difference to cloudlets is that unlike cloudlets, Spaceify is based on an ecosystem with several well-defined roles that encourage, support and monetize adoption of its architecture. For instance, it gives return-on-investment incentives to space owners for installing an edge node in their premises.

Another work that relates to Spaceify is the Open Pluggable Edge Services (OPES) architectural framework [4]. Like Spaceify, OPES essentially utilizes a client-edge-server architecture for performing content adaptation to provide enhanced user experience. This is done by modifying the network traffic between the providers and the consumer of data through a protocol suite for invocation and tracking of OPES services. Unlike OPES, Spaceify addresses the problem of content adaptation by allowing external service providers to run software on-demand on the edge node in a restricted execution environment. This way external service providers can under pre-defined security policies access space-specific information such as user presence information or indoor map data on the edge node.

3. SPACEIFY ARCHITECTURE

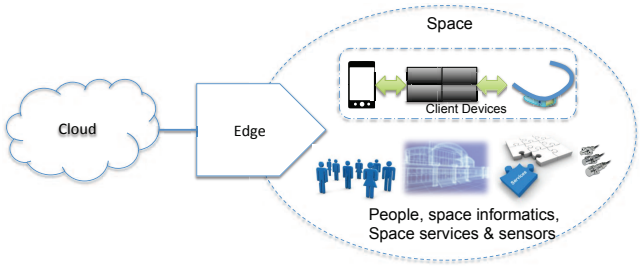

Figure 1: A Spaceify System Instance

A Spaceify system instance in Figure 1 consists of:

(a) A physical space containing mobile users, space specific services (e.g., shops and restaurants), space informatics (maps, models), sensors (such as WiFi and Bluetooth as location sensors, crowd sensors, capacity counters, noise sensors, and virtual sensors such as heat maps of hot activities and people). Mobile, wearable and other shared devices and interfaces are part of the space and represent client devices through which the mobile user may interact with the space services.

(b) An edge node that represents a specialized local computing and integration platform. It acts primarily as an efficient gateway to accessing and localizing the execution of web applications as will be shown later. It also acts as a point of integration of web applications, local space-specific applications, space sensors and mobile and other client devices. Such integration is designed to turn the smart space into a web of its own accessible seamlessly through multiple client devices (e.g., smart phone, public multi-touch displays, etc.) The seamless client device integration (maintaining sessions while switching among devices) is implemented in Spaceify but is not the focus on this poster.

(c) The web at large and its applications and services that may take part in localizing services in the space.

3.1 Edge Node Architecture

The edge node architecture shown in Figure 2 consists of three layers composed of open source components. The

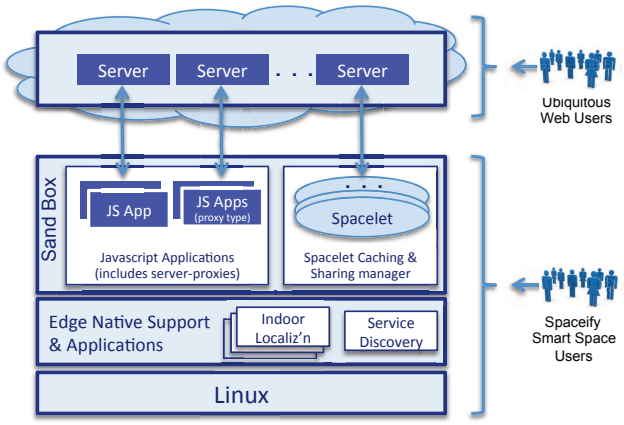

Figure 2: A Spaceify Edge Node Architecture

lowest layer is the GNU/Linux operating system on top of which the native Spaceify applications are installed using the default package management system of the underlying Linux variant. Spaceify system services such as Service discovery are also implemented as native Spaceify applications which makes the system highly modular. On the highest layer is the JavaScript sandbox in which JavaScript applications and Spacelets — dynamically downloaded applications provided by "Spaceified" web services — are run.

Native applications are intended for implementing services that require access to kernel services or specialized device drivers. Native applications are packaged in the package format of the underlying Linux variant (such as *.deb*) and sold in the Spaceify app store.

JavaScript applications are bought from the app store and permanently installed on the edge node by the Space owner. They are used for providing higher-level services such as storage or navigation that do not need direct access to the underlying operating system. They are distributed as *.zip* archives containing the JavaScript and other resource files along with the application manifest file. The packaging, installation and usage of Spaceify JavaScript applications is very similar to that of JavaScript applications on the Tizen mobile operating system [3].

The third Spaceify application type are Spacelets, which in contrast to JavaScript apps are not explicitly installed, but are transferred along HTTP responses from web services. The structure of the Spacelets is similar to that of JavaScript applications, consisting of JavaScript and resource files along with the manifest file.

On the edge node, Native applications, JavaScript applications and Spacelets (from here on referred collectively to as applications) can all provide services to each other and to the connected mobile devices. The services that are provided by native applications are often interfaces to hardware, such as indoor location devices, sensors and actuators. Building on low-level services provided by native applications, the JavaScript applications and Spacelets can then provide higher-level services such as indoor localization or persistent storage to each other and to code running on the connected mobile devices. Spaceify applications can be installed at will from the app store and web service providers can add Spacelets to their web content, which makes the Spaceify software stack extensible and ever-evolving.

This extensibility is achieved by allowing each application to run one local HTTP and one local WebSocket server.

One of the core components of the Spaceify stack is the Service discovery agent. It is a special native application that keeps track of which application is running on which port and what service it offers. Service discovery also implements access control by allowing and disallowing access to specific ports from specific applications using the Linux *iptables* mechanism.

Whenever a Spaceify application starts, it announces its presence and the name of the service it provides to the Service discovery agent. Whenever another application, running on the edge node or on a mobile device, needs some service from the edge node, it requests the service by its name from the Service discovery agent running on a well-known port in the address *edge.local*. The Service discovery agent goes through its list of registered services, checks whether the requester has rights to access this service, and in the positive case, returns the port number at which the requested service is running. While doing all this, the service discovery agent also creates an *iptables* rule that allows the requestor to access the server port of the requested service.

Resources and specific services available in a given Spaceify system may not be known beforehand to applications and application developers. They are typically chosen and deployed at the discretion of the space owner dynamically during the system operation.

4. SPACEIFY ECOSYSTEM

The spaceify ecosystem is based on well-defined roles that partly resemble those of well-established mobile ecosystems such as iOS or Android.

- **Spaceify inc.** develops and distributes the Spaceify software stack and runs an app store in which native and JavaScript applications are sold. Spaceify inc. also grants licenses to partner developers for developing native applications.
- **Partner developer** is a licensed developer of native Spaceify applications.
- **Application developer** develops JavaScript applications to be sold on the app store and installed on the edge nodes. JavaScript application development does not require a license.
- **Web service provider** hosts a web service that utilizes Spacelets. Hosting a "Spaceified" web service does not require a license.
- **Space owner** owns the physical space, the edge node and decides on the installation of software to the node. Space owner often employs a space system administrator to do the actual system administration work.

5. EXAMPLE: INDOOR NAVIGATION

This example demonstrates what we articulated in the first goal of Spaceify. It shows how map applications embedded in web pages that would normally return a google map utilizing GPS to the mobile user changes behavior inside a shopping mall, and returns an indoor map utilizing indoor localization technology instead. For example, a map returned when searching for *Pizza Hut* from home is therefore very different from the map returned for the same in a shopping mall, all be it using the same search. Let us now present our use case to explain the edge node architecture.

A shopping mall decides to acquire an accurate indoor navigation system for its customers. Quuppa [2] is chosen as the positioning hardware provider, and the hardware is installed in the mall and connected to the Spaceify edge node of the mall. As a partner developer, Quuppa has published the driver for their hardware as a Spaceify native application in the Spaceify app store. The system admin (works for the Space Owner) installs the Quuppa application on the edge node and configures the edge node so that all Spacelets from `maps.google.com` have the right to use the Quuppa application. This is done because Google has recently added support for accurate indoor navigation into Google maps, and routinely adds the *edge-manifest* HTTP header into all HTTP replies for the URL `maps.google.com`.

Next time a customer connected to the Internet through the edge node of the mall browses to `maps.google.com`, the edge node intercepts the *edge-manifest* HTTP header from the HTTP reply, and downloads the manifest file the header points to from Google. The edge node inspects the manifest file, checks the permissions required, downloads the required JavaScript files and maps generic service references in the manifest to actual available service interfaces found through the Service discovery agent running on the edge. For instance, "*bluetooth_indoor_positioning*" could be mapped to the Quuppa native application. Finally, the edge node starts the Spacelet according to the lifecycle policy stated in the manifest. After Google's indoor positioning Spacelet starts running it advertises its own presence using the Spaceify service discovery mechanism. Spacelet caching and sharing is not covered in this poster even though they are utilized by the edge node architecture.

As the customer's device receives the web page from `maps.google.com`, the JavaScript contained in the page checks for the existence of an indoor positioning Spacelet by querying for the service "*qoogle_indoor_positioning*" at the well-known port of Spaceify service discovery agent running on the edge node. The on-page JavaScript then opens a WebSocket connection to the Spacelet at the port that was returned by the Spacelet service discovery, and starts receiving real-time data from the Google indoor positioning Spacelet based on which it displays a detailed indoor map to the user ready for inspection or navigation.

6. REFERENCES

[1] Firefox OS – Mozilla | MDN. `https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS`. Retrieved July 4, 2013.

[2] Solution | Quuppa. `http://quuppa.com/solution/`. Retrieved July 4, 2013.

[3] Tizen | An open source, standards-based software platform for multiple device categories. `http://www.tizen.org`. Retrieved July 4, 2013.

[4] R. Chen A. Barbir, R. Penno, M. Hofmann, and H. Orman. An Architecture for Open Pluggable Edge Services (OPES). RFC 3835 (Informational), August 2004.

[5] A. Saarinen, M. Siekkinen, Y. Xiao, J. Nurminen, M. Kemppainen, and P. Hui. Can offloading save energy for popular apps? In *MobiArch'12*, pages 3–10.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.