Tampereen teknillinen yliopisto. Tietoliikennetekniikan laitos.
Tutkimusraportti 2010:2
Tampere University of Technology. Department of Communications Engineering.
Research Report 2010:2

Bilhanan Silverajan & Alexander Pyattaev

Future Services and Overlay Architectures: State of the Art Report 1

UBISERVE Project Deliverable D4.1

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Tampereen teknillinen yliopisto. Tietoliikennetekniikan laitos.
Tutkimusraportti 2010:2
Tampere University of Technology. Department of Communications Engineering.
Research Report 2010:2

Bilhanan Silverajan & Alexander Pyattaev

Future Services and Overlay Architectures: State of the Art Report 1

UBISERVE Project Deliverable D4.1

Tampereen teknillinen yliopisto. Tietoliikennetekniikan laitos
Tampere 2010

ISBN 978-952-15-2476-9 (PDF)
ISSN 1459-4617

Ubiserve Project WP4: Service Overlay Design, Architecture and Testing Deliverable D4.1

Future Services and Overlay Architectures: State of the Art Report 1

Bilhanan Silverajan and Alexander Pyattaev
Tampere University of Technology

October 2010

Table of Contents

1. Introduction

The UBISERVE-project (Research on Future Ubiquitous Services and Applications) is a joint research effort dedicated to advance research in the field of ubiquitous services. The project focuses on future services building on mobile communication systems and services.

Within this Project, our aim in Work Package 4 is to develop an overall overlay architecture providing a rich set of service facilitators for mobile users of wireless convergence devices that contain hardware capabilities to accurately pinpoint their locations, possess multiple wireless interfaces and have large storage capacities. The architecture will also support contextually aware self-organising devices creating an on-demand, dynamic overlay, capable of supporting multipath, transport-independent communication among users and devices. Thus useful information needs to be gleaned from service providers, network operators, infrastructure, sensors and devices in the immediate operating environment.

We are already witnessing an era, where cheap powerful hardware and competitive pricing and availability of wireless broadband networks have propelled their adoption by the mass market. This presents tremendous business opportunities for device manufacturers and network operators. However, the greatest single factor today defining the popularity or dissatisfaction of specific combinations of devices and technologies appears to be the availability, deployment and ease of use of a multitude of services.

This State of the Art document is the first project document which analyses the key enabling technologies for service overlays, platforms, frameworks and communication mechanisms for creating such architectures that would prove indispensable in the creation of future services. In short, the following questions are examined:

- What kinds of service overlays exist today?
- How are service overlays relevant towards services in wireless devices?
- What communication technologies can we expect future services to use?
- What are the dominant development platforms for smartphone applications?

2. Service Overlay Networks

As the number of users, services and combined processing power of networked computers multiply, increasingly new ways of efficient and optimized interconnections amongst these nodes are being developed to better serve user and application needs. Service Overlay Networks have become a popular way to facilitate the deployment of various kinds of applications and services. A Service Overlay Network can be perceived as a virtual network built atop existing physical networks, whose nodes form logical point-to-point links with each other. These overlay nodes can be customized and their logical links re-arranged to change the topology of the Service Overlay, independently and irrespective of how the actual routing and physical links are positioned.

Although Service Overlay Networks are not bound to any one type of network or transport technology, today a Service Overlay Network is typically built using IP technology. Additionally, the overlay links can be overlapped at the physical layer even though they are completely separated at the overlay layer. Non-overlay traffic or other overlay links may pass through a part of a whole group of IP-layer links. To obtain satisfactory performance, the overlay nodes need to continuously probe the network, so as to obtain updated information on the status and performance of the overlay links [Adami et. al 2009].

Overlay networks can be created by end-hosts as an application or service requirement in order to fulfill its actions, or it can be built by intermediate network nodes residing in the backbone network as a means to optimize or regulate network traffic. The former is usually a critical requirement by some VoIP, file sharing, gaming and collaborative user applications: These construct the overlay without any support from the backbone or any intermediate node, forming an application level transport layer over which packets are exchanged. The latter is often undertaken by network operators, content providers and third parties primarily to provide a better quality of service and experience to end users, keeping the peering structure largely hidden to end hosts. The discussion of overlay networks in this report is focused on end-user overlay networks, although it does not completely exclude backbone overlays.

Peer-to-peer networks form the largest and most common class of end-user overlay networks today. The term “"peer-to-peer" has come to be applied to networks that expect end users to contribute their own files, computing time, or other resources to some shared project [Oram 2001]. While the history of peer-to-peer networking is almost as old as the Internet itself, the phrase “P2P” came into popular existence when it was colloquially used to describe socially disruptive application overlay networks that performed filesharing by end-users. The resulting legal and technical repercussions are still ongoing at the time of this writing. A large portion of active P2P filesharing users engage in exchanging copyright infringing media and software piracy, while many ISPs around the world throttle bandwidth pertaining to certain kinds of P2P traffic in an effort to stem network load.

However, P2P networking technology has also been legitimately promoted and deployed for various uses such as VoIP-based communication, video-on-demand, distribution of non-copyrighted or royalty-free media, supplying various kinds of software updates and engaging in online multiplayer gaming.

Unstructured P2P networks lack any precise control over the network topology or file placement. The network is formed by nodes joining the network following some loose rules [Lv et. al 2002]. Flooding is the predominant search technique in unstructured peer-to-peer (P2P) networks. If performance is measured as the number of exchanged messages per distinct response, flooding with small time-to-live performs well in regular networks [Gkantsidis et. al 2005]. However, its performance deteriorates as the time-to-live increases, or if the topology of the underlying network is not regular [Gkantsidis et. al 2004]. In addition, flooding has poor granularity [Ritter 2001], and generates large loads on the network participants [Lv et. al 2002].

Structured P2P networks possess a network topology is tightly controlled and files are placed at specified locations that makes subsequent queries easier to satisfy. In highly structured systems, both the structure of the P2P network and the placement of files are precisely determined [Lv et. al 2002]. Structured overlays conform to a specific graph structure that allows them to locate objects by exchanging O(lg N) messages where N is the number of nodes in the overlay. Structured overlays can be used to construct services such as distributed hash tables, scalable group multicast/anycast, and decentralized object location [Dabek et. al 1999]. These overlays are highly resilient; they can route messages correctly even when a large fraction of the nodes crash or the network partitions [Castro et. al 2002].

With the advent of social networks and instant messaging, a new variation of the P2P network, called the Friend-to-Friend (F2F) network, has emerged. F2F is a completely decentralized architecture in which two computers can communicate only if their owners know one another. Constraining the connections to friends-only solves many of the security problems of the peer-to-peer architectures. Groups can easily build their own ad-hoc networks and collaborate without the need for any servers or third-party services [Galuba 2009]. However F2F networks are predicated on the fact that friends and contacts need to be authenticated, and as such, an external authentication service is usually involved if the network itself does not provide any such service.

Apart from P2P networks, overlays are often used as a means to introduce a new technology or as a means of access control into a restricted network. Overlay technology is relied upon as a transitionary technology to introduce IPv6 into IPv4 networks. 6to4 [Carpenter and Moore 2001] and 6rd [Townsley and Troan 2010] are good examples of how islands of IPv6 access networks can be interconnected using relay routers over an IPv4 backbone network. The end-user and device communicate via IPv6, with the mechanism overlaying the IPv6 packet routing and end-to-end architecture directly over the IPv4 transport network. In effect, the entire IPv4 network serves as a link-layer for the IPv6 network. DualStack-lite Internet Draft [Durand et. al. 2010] describes how, using IPv6 as the default network, IPv4 networks and packets can be supported via a tunneling architecture. Teredo [Huitema 2006] as well as IPv6 Tunnel Brokers [Durand et. al 2001] support provisioning IPv6 addresses to end-devices independently of their IPv4 network topology.

3. Services Today

Many popular ubiquitous services today can be scoped into one of the two following categories:

- Traditional custom service implementations. Such services have comprised of "fat clients", where applications run directly on the device. The implementation of the application relies exclusively on software technology supported by the hardware and operating system provided by the vendor. Such service implementations tend to be monolithic by design and exist both at compile as well as runtime as a single executable. This is a common approach for many services and applications that run in desktop computers as well as mobile laptops. Network interactions also tend to be straightforward, with the applications opening connections and exchanging information with servers using well-known communication protocols. Many examples of such services abound, such as Skype for VoIP-based telephony, or a multitude of messaging clients. Many of these provide the ability to query the device, operating system or 3rd party software extensions to update a user's mood or status with their current location based on GPS information, or providing URLs of current pages the user is browsing as well as updating in real-time what music the user is listening to.

- Services that comprise "thin clients" and widgets that leave the bulk of the complexity and functionality of the actual services themselves on the infrastructure external to the device. The client residing in the device is often a web browser or a derivative which provides a remote access interface. In many cases, the infrastructure is accessed via the web, run by a third party content or service provider. Many such websites use cookies stored in the local cache to save stateful information for the convenience of returning users, such as user authentication and last observed activity. Online email services such as Google Mail, photo sharing sites such as Flickr and online music streaming services such as Grooveshark and Last.FM are examples of such services.

Increasingly, many services today blur the distinction between the above two categories owing to user mobility, network connectivity, richness of device capabilities as well as social communication platforms become more widespread and powerful. Consequently, this report looks at how collaborative service platforms, as well as technological factors such as communication stacks and technology platforms are helping to drive future services.

4. Web Application Platforms and Example Services

While computing power and storage capacities have increased, especially in mobile devices, we are still witnessing a proliferation in the consumption and demand in the uptake of online content and service provision. This could be attributed to factors such as inexpensive cellular data connections, widespread availability of wireless broadband as well as the richness of social and content-sharing platforms. Many of these platforms are continually evolving, particularly with the increased consumer interest in social networks, and many offer web-based APIs, allowing the possibility of combining orthogonal services into a sort of "meta service" or mashup. Some of these have been rather simple, accessed simply via a web browser, such as

- Flickr users having the capability of sharing video from other websites such as Vimeo and YouTube directly from their Flickr photostream,
- Collaborative and news websites allowing readers to participate in leaving comments on stories, after being authenticated using Facebook Connect
- Real-estate agencies using Google Maps APIs to outline the location of available houses on sale.

As cloud computing technologies gain commercial importance, social networks become more mature and the notion of an always-connected device or user becomes a fact in the future, services are anticipated to further push the boundaries into how such service platforms are employed. The devices themselves seamlessly weave into the fabric of what is offered, with little or no complexity to the end user, while customizing themselves to the need at hand. The notion of exactly where applications launched from the device actually reside will be invisible/unimportant to the user: these could range from simple collaborations of thin applications which harness web-based APIs and browsers to fully realise the service mashup, or they could be full fledged applications, parts of which execute on the device while other parts are distributed into the service provider's platform, or to other devices that belong to the realm of control.

Some examples abound today that can give an insight into future developments for using service provider platforms as a basis for service development such as Facebook, Google and Nokia's Ovi.

4.1 Facebook and Facebook Places

Facebook remains the dominant platform for social networking today, boasting more than 500 million users as of July 2010 of which 150 million active users access their pages using mobile devices [Facebook 2010]. In addition to deploying Facebook Connect, that allows third-party websites to authenticate users based on their Facebook credentials, the platform hosts hundreds of applications which exploit the social nature of the website to share content, play games and perform various kinds of collaboration. The recently announced Facebook Places [Sharon 2010] brings this one step closer with location based services, by interacting with the mobile device's GPS receiver, with a web browser that supports both geolocation and HTML5. Facebook Places allows a Facebook user to share their locations with friends whilst similarly being able to see where other Facebook friends using Places are. Facebook Places allow friends to be tagged together, to perform group activities such as making hotel or restaurant reservations directly from the application itself.

4.2 Google APIs and Google Latitude

Google has heavily invested in browser-based technologies to lure user away from desktop applications towards cloud-based solutions that can be accessed by users irrespective of terminal or location. Google Docs, Mail, Wave and Buzz are just a few of the collaborative and social communication tools that are provided. Google Latitude was launched in 2009 which provides functionality similar to Facebook Places. Using the Google Latitude app, a user is able to update his location in realtime, allowing selected contacts and friends to view his current co-ordinates. Google Latitude uses GPS as well as IP-based geolocation and maps the resulting location onto Google Maps. Conversely, it also allows the user to view the location of his contacts who use Google Latitude. A location alert can be triggered when friends are nearby. An additional feature of Google Latitude is that upon the user's explicit consent, a history of visited locations can be stored. The location data can be subsequently retrieved via the Google Latitude API [Gasson 2009].

4.3 Nokia Ovi Service, SDK and Messaging

In addition to having a repository through which applications can be downloaded or bought for smartphones, Nokia's Ovi can be perceived as a collaborative service, providing many services to mobile devices and PCs such as media sharing, geolocation, synchronization and backup, an application store and until recently, a file upload/download/mirror service. A music player service allows unlimited music downloads for subscribers and enforces DRM by allowing playback from registered devices. Transfers to other media or devices is disallowed for unpurchased songs.

The web-based Ovi SDK is a toolkit that allows creating mobile apps quickly and easily and provides access to Ovi APIs, such as the Ovi Maps Player API and the Ovi Navigation Player API [Nokia 2010a]. Ovi also provides a messaging service. While many smartphones from various vendors today allow multiple standalone VoIP and IM applications, newer Nokia smartphones attempt to merge traditional phone contacts with those from VoIP and IM services, thereby creating a single view of all contacts reachable from the smartphone, be it via a cellular voice service or via the Internet through Wi-Fi or a cellular data service. The Contacts shortcut in Maemo5-based Nokia N900 launches an application through which users can be reached via phone, Skype Video and Google Video services, while a Conversations shortcut merges SMS, email and IM chats into a single view. Newer Symbian3-based smartphones such as the N8 do this differently since social networking and IM-based chat are managed via the Ovi platform: The phone owner signs in to Ovi, provides the necessary credentials for particular social or IM networks to allow the Ovi server to manage these connections on behalf of the device [N8 2010].

The recently announced Ovi App Wizard also provides an easy way for non-technical users to rapidly create and publish apps for the Ovi Store [Ovi 2010]. The Wizard facilitates the creation of content-based apps with RSS and Atom feeds for video, audio, text and images for blogs, YouTube and Twitter.

5. Communication Technologies and Protocol Stacks

The ready availability of processing power and networking capabilities in the immediate environment of a user or group of users also serves as a motivation to approach the notion of a future service as a dynamic and component-oriented application instead of the previously monolithic design. Such an option would provide extremely flexible opportunities for an application to reconfigure or re-adapt its communication, storage and processing abilities to automatically take advantage of any available infrastructure.

Future Services additionally are expected to be able to employ a range of communication methods for connectivity and reachability. These include various kinds of overlay networks as well as directly accessing a device's existing communication stack. In this section we discuss the kinds of transport mechanisms atop which different kinds of services and applications communicate and interact.

5.1 Web Technologies

Active development of AJAX (Asynchronous Javascript and XML) technologies has led to a stable development platform for the implementation of web applications. AJAX was initially developed as a means of overcoming limitations in allowing flexible page rendering by a web browser from a web server. Today, AJAX refers to a group of enabling technologies, such as HTML, CSS, XML, Javascript and JSON (Javascript Object Notation) to collectively provide an interactive experience for a user to use an application through a web browser. AJAX-based web applications have the ability to present themselves as belonging to the device they are accessed from, while in reality remain server-side services.

Historically, AJAX technologies started taking root during the browser wars of the 1990s with ActiveX, Java and the Javascript development activities finally converging to produce a means by which any web browser can perform asynchronous requests on a web server with the exchange of XML –based objects. AJAX received widespread attention when it was chosen as the technology of choice for web-based APIs and services from Google.

In essence, AJAX hides the underlying HTTP based communication between a client and server as a series of object calls in Javascript to invoke operations at the server using a well-known URI. Such calls tend to be asynchronous using the HTTP GET and HTTP POST operations. An XMLHttpRequest object [van Kesteren 2010a] is required at the client end to open connections to the server, construct well-formed requests, keep track of state changes, and handle responses received using a callback function. While AJAX traditionally relied on code written in Javascript, many alternatives exist today in Java, C, C++ and many others. In effect, this creates opportunities for many clients to interact with web servers without being web browsers themselves. These applications also render the boundaries about what a web application really is, indistinct: An application, could in effect, remain standalone and native to a specific platform for functionality such as video and audio while using AJAX to communicate with a webserver. A good example of this would be the Google Talk plugin for web browsers (available only for specific platforms such as Intel-based Mac OS X machines and Windows PCs) that allow voice and video chat, while the text chat client remains embedded in the browser and is AJAX-based.

Another communication technology that is rapidly gaining momentum for web-based application is HTML5 [Hickson 2010a]. As of this writing, HTML5 is still work-in-progress by the World Wide Web Consortium, its aim being an evolution of the current HTML4 by extending some of its current features while introducing new additions.

For web-based application development, HTML5 proposes to introduce scripting mechanisms and APIs in addition to web browsing and page rendering [van Kesteren 2010b]. Currently, Javascript is widely described as the scripting language of choice. The APIs allow the ability for clients to perform messaging, for embedding machine readable data into HTML documents, for allowing a server to access the audio, video and image capturing devices located at the client and for web storage. With HTML5, the work is also set for the introduction of WebSocket, a technology that specifies both an API as well as a protocol. WebSockets allow the ability to bootstrap an existing HTTP connection into a birectional, TCP-based communication channel between the client and the server [Hickson 2010b].

5.2 Emerging Technologies for Proximal Services

From a technological perspective, the interaction between end users and their immediate hardware platforms for the consumption of services has radically shifted away from the multi-user, single-host, workstation-like computing paradigm towards the single-user, multi-device embedded computing approach.

It is highly conceivable that in future, a single user would employ multiple devices residing within proximity (such as a Body Area Network or a Personal Area Network) to interact with another user or a computing server residing elsewhere in the Internet. The service itself could be constituted of several independent distributed software components executing in tandem within these disparate devices to take advantage of specialised embedded hardware functionality unique to each device. Towards this end, several wireless communications solutions can be proposed.

WiGig is an industry initiative launched in mid 2009 by a consortium of companies called the Wireless Gigabit Alliance. The WiGig effort promises wireless streaming speeds of up to 7 Gbps using the 60GHz frequency band [WiGig 2010]. The first specification was completed in Q4 2009, and WiGig Alliance opened its adopter program in Q2 2010. Designed from the ground up to address requirements varying from high performance to low power, WiGig aims at covering a borad range of devices ranging from desktop and laptop computers to low power handheld devices and battery operated consumer electronic equipment. The specification includes support for both IP data as well as streaming HD video and audio. In May 2010, a press release was issued stating that the WiGig Alliance and the Wi-Fi alliance will share technology specifications aimed at creating and fostering the next generation of Wi-Fi networking based on WiGig.

Bluetooth version 3.0 [Bluetooth 2009a] was standardised in early 2009 as a means to overcome the low data rates seen in earlier versions, and to exploit the availability of high-speed WiFi radios found in the majority of devices which need high-speed data transfer. In effect, Bluetooth 3.0 offers a new higher speed short-range data transfer mechanism, atop an ad-hoc 802.11 WiFi medium. This catapults Bluetooth 3.0 into a viable alternative transport mechanism for TCP/IP, as RFCOMM and L2CAP provide both connection-oriented and connectionless modes, while SDP offers a highly effective discovery mechanism for Bluetooth-enabled peripherals. As of this writing, Bluetooth 3.0 has begun seeing uptake among several tablets and laptops.

5.3 New Communication Technologies for Low Power Devices

While Bluetooth 3.0 was squarely aimed at penetrating the high-speed data transfer market, the recently announced Bluetooth 4.0 encompasses the Bluetooth Low Energy technology [Bluetooth 2009b]. Bluetooth Low Energy is aimed at communication requiring minimal power consumption and preserving the longetivity of small battery powered equipment. The standard stipulates data rates up to 1Mbps while keeping latency under 3s. Although most uses of Bluetooth Low Energy are expected to be short range, the specification nevertheless supports a range of up to 100 meters. Bluetooth 4.0 supports both single mode (meaning low energy mode only) or dual mode (meaning the device will support high speed data transfer using Bluetooth 2.1 or 3.0 in addition to low energy). The advent of Bluetooth technology into the low power arena puts it in square competition with both near field technology based on RFID, as well as low power technology being touted for sensor networks, notably ZigBee [ZigBee Alliance 2007] and 6LowPAN [Montenegro et. al 2007].

ZigBee has currently seen widespread deployment, particularly with healthcare, smart metering and home automation. Traditionally ZigBee has been touted as a low-cost and low-powered solution for sensors. ZigBee standards are issued by the ZigBee Alliance, and while IEEE 802.15.4 mesh is used as the underlying technology, the specified protocol stack comprising an application layer sitting directly atop the network layer has been criticized as being proprietary and non-interoperable with IP-based technology. In a bid to address this concern, the ZigBee Alliance in July 2009 announced the development of an IP based stack specification called ZigBee IP, which was aimed at the Smart Energy market, that will connect ZigBee-based smart energy sensors directly to the IP world.

6LowPAN is an IETF initiative to bring native IPv6 support to low power sensor networks. Just like ZigBee, 6LowPAN uses IEEE 802.15.4 as the underlying technology. The IETF 6LowPAN working group was formed as early as 2005, and has already resulted in the publications of two RFCs. As opposed to ZigBee, 6LowPAN does not define an application layer protocol. As an alternative, work is ongoing in the IETF to define the Constrained Application Protocol (CoAP) [Shelby et. al 2010]. CoAP is being developed within the IETF CoRE working group, to provide a framework for resource-oriented applications intended to run on constrained IP networks.

6. Software and Device Technology Platforms

In this section, some of the major operating systems, development platforms and software frameworks which aid in the creation of platform-specific and device-centric services are examined.

6.1 Android

Android is a mobile device platform distributed by the Open Handset Alliance. Since its debut, it has been positioned as a fully integrated mobile "software stack" that consists of an operating system, middleware, user-friendly interface and applications [Open Handset Alliance 2007]. Android has been built atop the Linux 2.6-series kernel. Language- wise, the system provides C and C++ libraries that can be used by other parts of Android's system components. Developers use the tools and APIs provided by the Android SDK for platform-specific application development. Athough Android applications are Java-based, facilities exist that allow applications developed in other languages such as C and Python to run in Android [Android 2010a], [Kohler 2009].

The approach undertaken for the design and deployment of end-user applications provides much better stability compared to the traditional Unix way of installing packages. All code in a single Android package is bundled into an archive file and considered to be one application. By default, each application is assigned a unique Linux user ID and runs in its own Linux process. Android starts the process when any of the application's code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.

Each process has its own virtual machine (VM), so application code runs in isolation from the code of all other applications [Android 2010b].

Android Java applications use a clean-room Java Virtual Machine, named Dalvik. Dalvik was designed so as to use less CPU cycles and allow less power consumption compared to using regular JVM bytecodes. At the time of writing, the latest Android version, codenamed "Froyo", just-in-time compilation was introduced in Dalvik that boosted the performance of executing Java applications by a factor of 2-5. At the same time, Froyo introduced the V8 engine for its web browser, allowing for better response times with Javascript-intensive web pages [Android 2010c]. Currently, the Android development environment includes a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE. Additionally, the newly introduced App Inventor for Android, a web-based visual programming tool, allows novice users without programming experience to create Android applications. App Inventor provides rich access to GPS, accelerometer, and orientation data, telephony services like phone calls and texting, speech-to-text services, contact data, persistent storage, and Web services such as those provided by Amazon and Twitter [Claburn 2010].

6.2 iOS and Xcode

The iOS is the operating system powering all Apple's mobile devices which include the iPhone, the iPod as well as the iPad. iOS was originally introduced for the iPhone, the latest version being iOS 4.1. iOS shares a common heritage and many underlying technologies with the Mac OS X operating system that powers Apple's laptop and desktop computers. The kernel in iOS is based on a variant of the same basic Mach kernel that is found in Mac OS X. On top of this kernel are the layers of services that are used to implement applications on the platform. the Core OS and Core Services layers contain the fundamental interfaces for iOS, including those used for accessing files, low-level data types, Bonjour services, network sockets, and so on. The Media layer contains the fundamental technologies used to support 2D and 3D drawing, animation, audio, and video. The uppermost Cocoa Touch layer is comprised of several kinds of application-level frameworks providing the fundamental infrastructure used by an application. For example, the Foundation framework provides object-oriented support for collections, file management, network operations, and more. The UIKit framework provides the visual infrastructure for an application, including classes for windows, views, controls, and the controllers that manage those objects. Other frameworks at this level provide access to the user's contact and photo information and to the accelerometers and other hardware features of the device [Apple 2010].

Application development for the iOS platform is predominantly in the Objective-C language. The iOS SDK facilitates this process by providing tools and development environments. Apple's Xcode Integrated Development Environment, the recommended way for developing applications for Mac OS X, also remain the development platform for creating iOS apps. The current iOS SDK 4.1 release includes the Xcode IDE, iOS Simulator, and a suite of additional tools for developing apps for iPhone, iPad, and iPod touch. Early versions of iOS did not perform multitasking which severely limited the iPhone's capabilities for running more advanced applications. However, this limitation was lifted in iOS 4, allowing true multitasking.

6.3 Symbian

The Symbian platform is a complete operating system, aimed at the smartphone market, and is maintained by the Symbian Foundation. The operating system is written in C++, and presents a modular, microkernel-like architecture with its EKA2 kernel. Symbian provides separate kernel and userspace facilities, with its EKA2 kernel being designed towards single-user, multitasking, pre-emption and priority-based execution model [Sales and Tasker 2009]. While the device drivers may sit in either the kernel or user-space, the filesystem and communications stacks reside in the user-space [Morris 2007]. The entire Symbian platform code was released as open source in February 2010 [Symbian 2010], while the first phones utilizing the fully open Symbian^3 platform debuted later the same year.

Application development for Symbian platforms employ the Symbian SDK. Currently, the Symbian SDK for Symbian ^3 supports C++ and Java programming languages. A plugin is required for Python-based development. Additionally, the SDK supports development with the Web Runtime (WRT) towards AJAX applications as well as with the Qt framework for cross-platform applications. An emulator is provided as part of the SDK, together with debugging and diagnostics tools. [Nokia 2010b].

6.4 Maemo and MeeGo

The Maemo platform has been derived from the Debian Linux platform, and was developed by Nokia as a Linux-based platform for mobile touchscreen tablets and smartphones. Many parts of Maemo are consequently based on open source code, with early versions of the platform leveraging GNOME software and libraries. At the time of writing, the latest version of Maemo is Maemo5, designed to run on the Nokia N900 smartphone. Because Maemo has its roots in the Linux world, it inherits many of its virtues, such as native multitasking, commandline and system level access to much of its hardware, the support of multiple programming languages such as C, C++, Java and Python, and the interest of many Linux developers worldwide who wish to port their open-source applications to the Maemo platform.

The development environment for Maemo is currently restricted to the installation of a scratchbox environment which mandates the need for a (preferably Debian- or Ubuntu-based) Linux desktop system. This scratchbox is commonly referred to as the Maemo SDK, and features a sandboxed command-line or X11 environment that can be customized to the needs of the developer using shell scripts. The compilation target can be swapped at the developer's discretion, from an emulation platform that mimics the behavior of the target environment and some hardware on the x86 platform, to a full fledged cross-compilation process which creates an ARM-processor based executable designed to run natively on the device itself. While the former approach is sufficiently adequate for the creation of simple graphical or stand-alone applications, the latter is necessary for applications needing hardware accelerated graphics support, wireless networking with Wi-Fi or Bluetooth and camera and sensor-based applications.

Such an approach to development on Maemo is mitigated by the fact that it is usually relatively trivial to port many existing Linux and Unix applications to the Maemo platform, and by the fact that Maemo distinguishes itself from many of the major smartphone platforms with the wide range of functional multimedia codecs, working Flash, CSS and Javascript web browser support, and uninhibited access to popular VoIP and videoconferencing applications like Skype and Google Talk (with video supported for both).

The recently announced MeeGo platform is a joint activity by Nokia and Intel that combines the evolution of the Maemo platform with that of Intel's Moblin project [MeeGo 2010]. MeeGo is aimed at becoming the core enabling platform, not just for Nokia's future smartphones, but also for netbooks, vehicular entertainment and control systems as well as consumer electronic products such as a television. While the two platforms that MeeGo is rooted upon are based on the Debian and Fedora Linux distributions respectively, MeeGo itself is being promoted as a Linux distribution independent of others, as an open source project hosted by the Linux Foundation [MeeGo 2010]. The architecture of the MeeGo platform reveals a layered approach, with three layers. The MeeGo OS Base layer contains the Linux kernel and core services along with the Hardware Adaptation Software required to adapt MeeGo to support various hardware architectures. The MeeGo OS Middleware layer provides a hardware and usage model independent API for building both native applications and web run time applications. The MeeGo User Experience layer provides reference user experiences for multiple platform segments [Van De Ven 2010].

6.5 Qt

The Qt object-oriented framework is written in C++ and was initially a cross-platform widget toolkit for creating graphical user interfaces. Qt's role has expanded since then to encompass various other features, such as networking and protocol functionality, database modules, a webkit that interacts with XML and web content, a scripting library based on Javascript as well as communication via the D-Bus IPC mechanism. In effect, Qt today is regarded as a full-fledged application framework that could be used to develop applications for a variety of operating systems. At the time of writing, the latest Qt version is Qt 4.7. Qt comes equipped with the Qt Creator, which is a complete environment for creating applications with Qt that comes equipped with both development as well as debugging support.

The Qt framework plays a very central role to smartphone platforms developed by Nokia. Because it is a full-featured cross-platform framework, application developers are encouraged to use Qt for maximum portability between Nokia's Symbian and Maemo/Meego platforms. Additionally, Qt has been announced as the foundation of Meego's UI and is the API for developing applications in all upcoming Meego-based Nokia devices [Nokia 2010c]. Qt Creator is recommended as the integrated development environment (IDE) to use for creating MeeGo applications [Spencer 2010].

7. Ongoing Related Research

This final section of the report looks at some ongoing projects and research, the areas of which are of interest to the work that should be done in WP4. These research areas cover the following areas:

- How devices, their services and capabilities can be discovered
- How various transport technologies can be exploited
- How overlay networks can be constructed

7.1 Ontology models and Semantic Web

An ontology model can be expressed as a formal representation of domain-specific knowledge, capturing entities, events, services, properties and ideas in a well-organized manner. According to one well-cited source, an ontology is a formal explicit description of concepts in a domain of discourse (classes, sometimes called concepts), properties of each concept describing various features and attributes of the concept (slots, sometimes called roles or properties), and restrictions on slots (facets, sometimes called role restrictions). An ontology together with a set of individual instances of classes constitutes a knowledge base [Noy and McGuinness 2001]. Ontology models are often used to describe real world environments and situations. Often they are flexible in incorporating new information or properties. Consequently they are often used in conjunction with web services and semantic web technologies for a wide range of research activities.

Relevant research in this area shows successfully how user requirements are captured and stored for eventual service composition, how service registration and discovery for embedded components and services can be accomplished, and how service and device collaboration can occur in a smart space .

7.1.1 DynamiCos

The DynamiCos project [Silva et. al 2009] aims at providing a framework that can provide automated runtime service composition mechanisms that can deliver a personalized service delivery. To achieve this automated support, DynamiCoS defines ontologies (domain conceptualisations). The framework allows different service developers to publish their semantically annotated services in the framework. These semantic descriptions have to refer to the framework's ontologies. End-users may have different domain or technical knowledge, which implies that their service request interfaces have to be defined accordingly. DynamiCoS tackles this problem by defining a service request that supports different user interfaces. A service request consists of a specification of goals the user wants the service to achieve. A goal is likewise used to describe services, specifying the activities (or operations) the services can perform. Goals of users and services are specified according to the framework ontologies (representation of the domain of knowledge), this allows matchings to be found, whenever a service realises the user goals.

7.1.2 NoTA

NoTA (Network on Terminal Architecture) has been an ongoing research activity in Nokia Research Center for several years, but it was officially unveiled for public release in 2007, with Release 3.0 [Nokia 2010d]. The main idea behind NoTA was to find new ways in which an embedded design architecture could be designed in a modular way, with each individual module functionally comprising both hardware and software resources loosely decoupled from other modules. These modules can reside within the same chip or could reside off-chip elsewhere within the device. This allows independent development as well as testing of NoTA modules by separate vendors prior to integration.

The architecture introduces 2 kinds of nodes: A Service Node (SN) which both publishes and consumes events (and services), and an Application Node (AN) which only consumes events (and services). Nodes communicate using a publish/subscribe scheme for events. Nodes can also communicate using a streaming interface. Messaging and streaming among nodes is facilitated by a logical Interconnect. Communication among NoTA nodes is communication agnostic, with the transport network stack mechanism being abstracted away from the actual high-level communication constructs. This allows the NoTA subsystem to easily adapt to any transport or physical interconnect. Presently, apart from the MIPI Alliance's UniPro high-speed interface for interconnecting integrated circuits, SNs and ANs possess the ability to communicate using TCP, FIFOs, Bluetooth and USB as transports [NoTA 2008]. In effect, this allows inter-node communication between ad-hoc devices. Service Interface Specifications in NoTA use an ontology model, with WSDL used to describe the service interface of an SN. When TCP is used as a transport, link-local multicast is used for the device discovery.

NoTA has been ported to major embedded and desktop platforms such as Symbian, Maemo, Android, iOS, Linux, BSD and Windows. Future research aims at focusing on inter-device solutions to seamlessly access services among devices and NoTA networks, in addition to exploiting the architecture's transport agnostism [Leppänen 2009].

7.1.3 Smart-M3

The Smart-M3 project explores the representation of smart spaces comprising devices and software, as distinct entities that can interact with each other using semantic web technology. A Smart Space is an environment that has an associated digital representation in which relevant real-world information is stored and kept up to date. The project aims at intelligent service and device collaboration within a defined area, such as a Smart Home or a meeting room, by taking into account environmental and contextual data using a tuple space mechanism.

Each participating device or service in Smart M3 uses an M3-agent, also known as a knowledge processer (KP) to interact with a semantic information broker (SIB). A smart space in Smart-M3 is defined as a named search extent of information, where the information is stored in one or more SIBs. In the simplest case, one SIB will store all information in a smart space, but there is a possibility of connecting multiple SIBs to make up a smart space. The information in the smart space is stored as an RDF graph according to some defined topology [Luukkala and Honkola 2010]. Queries in this sense are ontology driven, but are not strictly bound to any one ontology model.

The communication between KPs and SIB is transport independent with multiple transport mechanisms being supported by the SIB. Such mechanisms include TCP/IP, HTTP, XMPP, Bluetooth as well as NoTA. Additionally, Smart M3 allows the notion of an application in a smart space to differ from the concept of a traditional application. Instead of a monolithic application running on a single screen, the smart space applications are better seen as scenarios that can be executed to meet the goals of the user [Luukkala and Honkola 2010]. Consequently the composite smart application may be visualized as multiple interacting smart spaces that enable cross-domain service mashups.

7.2 Platform Composition

When two or more wireless mobile devices converge into a physical space, many opportunities for building ad-hoc collaborative services and applications for the benefit of their users, abound. This is particularly so given the amount of computational and storage power modern wireless mobile devices possess. Platform composition research aims at doing just that: To be able to compose these devices into a unified platform that can deliver collocated services. Effective composition of such platforms aim at exploiting the individual strengths of each device as well as overcoming any individual limitations.

7.2.1 Composition Framework

The *Composition Framework* research conducted at Intel Research [Pering et. al 2009] integrates standard computing components to support effective collaborative work by wirelessly combining the most suitable set of resources available on nearby devices. The developed prototype provides a specific implementation of Platform

Composition along with the user interface necessary to invoke standard platform services. By design, it is orchestrated around utilizing existing standards to support familiar applications on ad-hoc sets of devices. Although the underlying services and protocols used to share data among devices are not themselves new, the system instead focuses on the centralization and coordination of the sharing process.

The research delves into how ad-hoc collaborations could be undertaken by looking at a design space composed of three axes: The composition granularity for the collaboration (either event, object, or service-based), the sharing model of the resources that users interact with (independent, coordinated or mirrored) and the resource referencing that specifies how devices and services discover, address and reference each other (ad-hoc, familiar or well-known). The Composition Framework architecture consists of four major components: framework core, user interface, network discovery, and service modules. D-Bus is used to facilitate communication between the modules. The core components are implemented in Java. Each individual service for sharing a resource is specified by an XML service descriptor file, which encodes basic properties of the service (name, icon, etc.), and provides details on how to probe, invoke, monitor, and disconnect the service. Some services, such as storage sharing, are implemented using asynchronous operating system calls, while others, such as display sharing, are implemented by invoking a standalone client process. Services are handled using an explicit client/server model based on commonly available standard systems.

7.2.2 Context Card

Context Card is a sensor platform that is able to use context-aware composition to overcome the wireless discovery process, selection and connection establishment process through sensing [Lyons et. al 2009]. Context information is supplied by sensors on a mobile device, and the project examines how these sensors can also be used to facilitate discovery through spatial sensing, and by representing unique aspects of their state. This allows for better characteristics with which users can distinguish each device.

The Context Card platform also communicates with the aforementioned *Composition Framework* through the D-Bus IPC mechanism. The Framework subscribes to sensor events originating from the Context Card platform to perform wireless discovery. The combined architecture uses layer-2 networking to distribute device names as well as service and context information to reduce overheads imposed by traditional methods that share information over layer 3. In addition to using context as a mechanism for the user to manage the discovery process and control compositions, the context was also used for the services involved in a composition. For example, relative spatial positions reported by the Context Card and subsequently advertised by the Composition Framework may be used to automatically configure the positioning of devices in the construction of an extended desktop, as opposed to manual configuration.

7.2.3 CompUTE

The *CompUTE* project [Bardram et. al. 2010] aims at creating a runtime infrastructure for device composition. The project uses the idea of a composite device, which is one device made up of a composition of several separate devices working together in concert. The platform supports three types of interaction scenarios among users and devices: One user to several proximal devices, several users to several shared devices, and lastly device composition within a smart space. The *CompUTE* architecture is optimized towards supporting the use of shared and extended desktops and displays for users. A central Gateway is relied upon for service registration, discovery and publication. A *CompUTE* client resides on each participating device.

CompUTE is restricted to Windows XP, and uses UPnP for discovery and SOAP for remote invocations. Microsoft's .net is used as an implementation platform. The current prototype mimics an extended shared desktop across the participating devices. The user has the ability to navigate mouse events and supply keyboard interaction across all the desktops, access a shared clipboard, move objects and files between displays and so on. Usability trials conducted on test groups reveal the viability of this project for collaborative work.

7.3 Reflection and Overlay Adaptation

It has become very commonplace for wireless and mobile devices to form overlays over which services are executed. Particularly for purely wireless networks, the likelihood that the overlay's architecture and link quality will deteriorate because of evolving network conditions, effects of battery performance, variations in physical topology and overlay arrangement, increases. Additionally, while early overlay networks, such as peer-to-peer networks were tightly bound to one specific type of usage and application, overlays are beginning to be used as generic transports to various types of applications today. This implies that overlay architectures need to have very extensible adaptation mechanisms. In addition to parametric and algorithmic adaptation to cope with changing network and environmental conditions, overlays also have to contend with application, service and user requirements.

Two highly effective techniques to cope with adaptation are using component-based engineering and reflection. Component-based engineering promotes the ability to reconfigure and readapt specific parts (or components) of an overlay at run-time. Reflection is a technique that allows software to introspectively inspect its own behaviour and properties at runtime, and manipulate portions of itself when certain criteria are met.

7.3.1 Overlay Adaptation in Juno

Juno is a configurable middleware designed to address the heterogeneity of next-generation content distribution [Tyson et. al 2008]. Within the Juno project, a sub-project is studied in which a middleware-based approach is investigated that furthers this idea towards an extensible architectural adaptation for overlays [Tyson et. al. 2009].

The implementation of this idea resulted in a Java-based framework. The work looks not only at how, using a context engine, an overlay can adapt to perform optimally for the application using it, but also how the entire overlay can be treated as a pluggable component that can be replaced, if necessary by a new overlay network. Abstractions are introduced at multiple levels that allow aspects of the overlay's functionality to be represented from a very coarse sense (such as the overlay as one unit) towards a very fine-grained sense (such as placing every method or algorithm behind an independent abstraction). Consequently, these abstractions allow implementations of components as well as constructions of composites of components which can be controlled and adapted based on decisions being fed to the middleware at run-time from a context engine. A number of overlays were implemented using this component-based approach which include CHORD, SCAMP, BitTorrent, TBCP and Pastry, and case studies verified the feasibility of the architectureal reconfiguration.

8. Conclusion

In this document, some of the key enabling technologies, architectural design choices, and the end-user perspectives were presented. There exists a wealth of technologies for use, and each is applicable to ubiquitious services and service overlay with certain limitations. This is most evident in the areas of wireless, power and energy constrained devices and their connectivity.

The future work will concentrate on making a selection on what technologies and techniques to follow, and what are most feasible design models for the project. The current situation in the field of supporting future services with dynamic overlay architectures, substantial related work exists, but work on this specific area has only just started, and has to proceed based on drawing parallels from the results of work in similar areas.

9. References

[Adami et. al 2009] Adami, D., Callegari, C., Giordano, S., Nencioni, G., and Pagano, M. 2009. Design and performance evaluation of service overlay networks topologies. In *Proceedings of the 12th international Conference on Symposium on Performance Evaluation of Computer & Telecommunication Systems* (Istanbul, Turkey, July 13 - 16, 2009).

[Android 2010a] Android NDK. *Android Developers Website June 2010.* http://developer.android.com/sdk/ndk/index.html#overview

[Android 2010b] Application Fundamentals. *Android Developers Website October 2010.* http://developer.android.com/guide/topics/fundamentals.html

[Android 2010c] Android 2.2 Platform Highlights. *Android Developers Website 2010.* http://developer.android.com/sdk/android-2.2-highlights.html#Platform Technologies

[Apple 2010] iOS Overview. *Apple Developers Website July 2010.* http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//apple_ref/doc/uid/TP40007592

[Bardram et. al. 2010] Bardram, J. E., Fuglsang, C., and Pedersen, S. C. 2010. CompUTE: a runtime infrastructure for device composition. In *Proceedings of the international Conference on Advanced Visual interfaces* (Roma, Italy, May 26 - 28, 2010)

[Bluetooth 2009a] Bluetooth SIG. *Core Specification v3.0 + HS, April 2009.* http://www.bluetooth.com/SiteCollectionDocuments/Core_V30__HS.zip

[Bluetooth 2009b] Bluetooth SIG. *Core Specification v4.0, Dec 2009.* http://www.bluetooth.com/SiteCollectionDocuments/Core_V40.zip

[Carpenter and Moore 2001] Carpenter, B., Moore, K. 2001. Connection of IPv6 Domains via IPv4 Clouds. *IETF RFC 3056, February 2001,* http://tools.ietf.org/rfc/rfc3056.txt

[Castro et. al 2002] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. 2002. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 299-314.

[Claburn 2010] Claburn, T. 2010. Google App Inventor Simplifies Android Programming. *InformationWeek July 2010.* http://www.informationweek.com/news/smb/mobile/showArticle.jhtml?articleID=225702880&subSection=News

[Dabek et. al 2003] Dabek, F. Zhao, B. Druschel, P. Kubiatowicz, J. Stoica, I. 2003. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of 2nd Int'l. Wksp. Peer-to-Peer Systems (IPTPS 2003),* Berkeley, California, USA, Feb. 20-21, 2003.

[Durand et. al 2001] Durand, A., Fasano, P., Guardini, I., Lento, D. 2001. IPv6 Tunnel Broker. *IETF RFC 3053, January 2001,* http://www.rfc-editor.org/rfc/rfc3053.txt

[Durand et. al 2010] Durand, A., Droms, R., Woodyatt, J., Lee, Y. 2010. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. *IETF Internet-Draft August 2010,* http://www.ietf.org/id/draft-ietf-softwire-dual-stack-lite-06.txt

[Facebook 2010] Facebook Statistics. *Facebook Website, July 2010*. http://www.facebook.com/press/info.php?statistics

[Gasson 2009] Gasson, M. 2009. Normality Mining: Results from a Tracking Study. *FIDIS (Future of Identity in the Information Society) Technical Report D12.10, June 2009.*

[Galuba 2009] Galuba, W. 2009. Friend-to-Friend Computing: Building the Social Web at the Internet Edges. *LSIR-REPORT-2009-003*, http://lsirpeople.epfl.ch/galuba/papers/f2f.pdf

[Gkantsidis et. al 2004] Gkantsidis, C. Mihail, M. Saberi, A. 2004. Random Walks in Peer-to-Peer Networks. In *Proceedings of IEEE Infocom 2004*.

[Gkantsidis et. al 2005] Gkantsidis, C. Mihail, M. Saberi, A. 2005. Hybrid Search Schemes for Unstructured Peer-to-Peer Networks. In *Proceedings of IEEE Infocom 2005*.

[Hickson 2010a] Hickson, I. 2010. HTML5, A vocabulary and associated APIs for HTML and XHTML. *W3C Working Draft 24 June 2010*. http://www.w3.org/TR/html5/

[Hickson 2010b] Hickson, I. 2010. The WebSocket protocol. *IETF Internet-Draft September 2010.* http://tools.ietf.org/id/draft-ietf-hybi-thewebsocketprotocol-02.txt

[Huitema 2006] Huitema, C. 2006. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). *IETF RFC 4380, February 2006.* http://tools.ietf.org/rfc/rfc4380.txt

[Kohler 2009] Kohler, D. 2009. Introducing Android Scripting Environment. *Google Open Source Blog, June 2009.* http://google-opensource.blogspot.com/2009/06/introducing-android-scripting.html

[Leppänen 2009] Leppänen, T. 2009. NoTA ecosystem – past, present and future. In *Presentations of 2nd International NoTA Conference.* San Jose, California USA Sept 30th- Oct. 1st 2009.

[Luukkala and Honkola 2010] Luukkala, V., Honkala, J. 2010. Integration of an Answer Set Engine to Smart-M3. In *Proceedings of 3rd Conference on Smart Spaces, ruSMART 2010 and 10th International Conference on NEW2AN 2010,* St. Petersburg, Russia, August 2010.

[Lv et. al 2002] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. 2002. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international Conference on Supercomputing* (New York, New York, USA, June 22 - 26, 2002). ICS '02. ACM, New York, NY, 84-95.

[Lyons et. al 2009] Lyons, K., Want, R., Munday, D., He, J., Sud, S., Rosario, B., and Pering, T. 2009. Context-aware composition. In *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications* (Santa Cruz, California, February 23 - 24, 2009). HotMobile '09.

[MeeGo 2010] FAQ. *MeeGo Website 2010.* http://meego.com/about/faq

[Montenegro et. al 2007] Montenegro, G., Kushalnagar, N., Hui, J., Culler, D. 2007. Tramsmission of IPv6 Packets over IEEE 802.15.4 Networks. *IETF RFC 4944, September 2007.* http://tools.ietf.org/rfc/rfc4944.txt

[Morris 2007] Morris, B. 2007. The Kernel Services and Hardware Interface Layer. *The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS,* ISBN: 978-0-470-01846-0. Wiley, Inc.

[N8 2010] Nokia N8-00 English User Guide, August 2010.

[Nokia 2010a] Ovi for developers. *Nokia Developers Website, 2010*. http://developer.nokia.com/Develop/Web/Tools/Ovi_developers.xhtml

[Nokia 2010b] Symbian SDKs. *Forum.Nokia Website 2010,* http://www.forum.nokia. com/Library/Tools_and_downloads/Other/Symbian_SDKs/

[Nokia 2010c] Meego and Qt. *Qt.Nokia Website 2010.* http://qt.nokia.com/ products/platform/meego/

[Nokia 2010d] NoTA project. *Forum.Nokia Website 2010,* https://projects.forum.nokia.com/NoTA/wiki

[Nota 2008] NoTA World website. Getting started, user guide to NoTA. 2008, http://www.notaworld.org/documentation/tutorials

[Noy and McGuinness 2001] Noy, N. F., McGuinness, D. 2001. Ontology Development 101: A Guide to creating your first Ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, March 2001

[Oram 2001] A. Oram, Ed. 2001 *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc.

[Ovi 2010] Ovi Blog May 2010. *Try out the new Ovi App Wizard!.* http://blog.ovi.com/2010/05/03/try-out-the-new-ovi-app-wizard

[Pering et. al. 2009] Pering, T., Want, R., Rosario, B., Sud, S., and Lyons, K. 2009. Enabling Pervasive Collaboration with Platform Composition. In *Proceedings of the 7th international Conference on Pervasive Computing* (Nara, Japan, May 11 - 14, 2009).

[Ritter 2001] Ritter, J. 2001. Why gnutella can't scale. No, really. http://www.darkridge.com/~jpr5/doc/gnutella.html

[Sales and Tasker 2009] Sales, J., Tasker, M. 2009. Symbian OS Internals/1. Introducing EKA2. *Symbian Developer Wiki, 26 May 2009.* http://developer. symbian.org/wiki/index.php/Symbian_OS_Internals/1._Introducing_EKA2

[Sharon 2010] Sharon, M. E. 2010. Who, What, When, and Now...Where. *The Facebook Blog, August 2010.* http://blog.facebook.com/blog.php?post=418175202130

[Shelby et. al 2010] Shelby, Z., Frank, B., Sturek, D. 2010. Constrained Application Protocol (CoAP). *IETF Internet-Draft September 2010.* http://www.ietf.org/id/draft-ietf-core-coap-02.txt

[Silva et. al 2009] Silva, E., Pires, L.F., van Sinderen, M. 2009. Supporting Dynamic Service Composition at Runtime based on End-user Requirements. In *Proceedings of 1st International Workshop on User-generated Services* (Stockholm, Sweden, November 24, 2009)

[Spencer 2010] Spencer, B. 2010. Qt Creator. *Meego Developers Website*, February 2010, http://meego.com/developers/getting-started/qt-creator

[Symbian 2010] Symbian Completes Biggest Open Source Migration Project Ever. *Symbian Website, February 2010.* http://www.symbian.org/news-and-media/2010/02/04/symbian-completes-biggest-open-source-migration-project-ever

[Townsley and Troan 2010] Townsley, W., Troan, O. 2010. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) -- Protocol Specification. *IETF RFC 5969 August 2010,* http://www.rfc-editor.org/rfc/rfc5969.txt

[Tyson et. al 2008] Tyson, G., Mauthe, A., Plagemann, T., El-khatib, Y. 2008. Juno: Reconfigurable Middleware for Heterogeneous Content Networking. In *Proceedings of the 5th International Workshop on Next Generation Networking Middleware (NGNM)* (Samos Islands, Greece 2008)

[Tyson et. al 2009] Tyson, G., Grace, P., Mauthe, A., Blair, G., and Kaune, S. 2009. A Reflective Middleware to Support Peer-to-Peer Overlay Adaptation. In *Proceedings of the 9th IFIP WG 6.1 international Conference on Distributed Applications and interoperable Systems* (Lisbon, Portugal, June 09 - 11, 2009).

[Van De Ven 2010] Van De Ven, A. 2010. MeeGo Architecture. *MeeGo Website February 2010.* http://meego.com/developers/meego-architecture

[van Kesteren 2010a] van Kesteren, A. 2010. XMLHttpRequest. *W3C Candidate Recommendation 3 August 2010*. http://www.w3.org/TR/XMLHttpRequest/

[van Kesteren 2010b] van Kesteren, A. HTML 5 differences from HTML 4. *W3C Working Draft 24 June 2010.* http://www.w3.org/TR/html5-diff/

[WiGig 2010] Wireless Gigabit Alliance Website, Oct 2010. http://www.wigig.org/faqs/

[ZigBee Alliance 2007] ZigBee Alliance, ZigBee Specification 2007. http://www.zigbee.org/Products/DownloadZigBeeTechnicalDocuments.aspx