Odin: Context-Aware Middleware for Mobile Services

Thiranjith Weerasinghe
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand
Email: twee003@aucklanduni.ac.nz

Ian Warren
Department of Computer Science
University of Auckland
Auckland, New Zealand
Email: ian-w@cs.auckland.ac.nz

***Abstract*—Mobile devices such as smart phones are increasing permeating society. With strides in computational power, coupled with the ability to connect to other small devices, smart phones are able to host novel services. To address the repetitive problems associated with mobile service development, namely service reachability, scalability and availability, we have developed Odin, which is a middleware platform for mobile service provisioning. Beyond providing a provisioning solution, Odin conserves scarce resources such as network bandwidth and device power supply. However, Odin has previously lacked an ability to take into account operational context. In this paper, we present context-aware extensions to Odin that further optimise resource usage. Augmented with support for context types that include location, performance, power and network, Odin is able to propagate context information to applications and dynamic adapt the middleware's behaviour. Novelty of the work lies in a solution whose device overhead is very low, and one that offers a coherent approach to context dissemination and adaptation. Based on quantitative evaluation, context-aware Odin's low overhead is demonstrated along with significant gains in resource conservation.**

I. Introduction

Mobile devices such as PDAs and smart phones are increasingly becoming integral part of our daily lives. The recent advances in technology have enabled them to be equipped with GPS, accelerometers, cameras and multiple network interfaces such as Bluetooth, Wi-Fi and 3G. Such developments, along with substantial improvements in processing capacity, have allowed mobile devices to shift their role from service consumer to *provider*, enabling new and unique types of services.

To demonstrate the potential of mobile services, consider a patient monitoring service (PMS) [1]. A mobile patient could be monitored by a smart phone augmented with body sensors. Remote healthcare professionals would consume the service to observe the patient *less intrusively* than using conventional monitoring equipment. This would enable data to be sent on an *as needed basis*, leading to reduction in costs and efficient bandwidth usage. In addition, PMS could enable health professionals to remotely *control monitoring* of the patient's essential signs.

While creation of mobile services on todays devices is technically feasible, the lack of middleware support causes developers to repeatedly address unique challenges posed by mobile service provisioning. Such challenges stem from services' inherent mobility and the constrained resources of small devices, and include reachability, availability, scalability and heterogeneity. Reachability is concerned with maintaining connectivity between a roaming service and its clients [2]. Availability is an issue because of the limited power supply of mobile devices. Despite advances in computational power, mobile devices remain limited in this respect relative to desktop and server hardware, hence the need for scalable infrastructure. To address these challenges, we have developed Odin, an intermediary-based middleware for mobile service provisioning [2].

Being aware of dynamic operational conditions pertaining to mobile service provisioning would enable Odin to better optimise resources. Operational attributes, such as device location and network signal strength, are collectively referred to as *context* [3]. For example, a PMS could switch over to a low-power network interface such as Bluetooth [4] upon detecting low battery levels and a nearby intermediary based on its current location. This would help to keep the device operational for longer, hence improving service availability and allowing the device to be used by its user for other tasks. Similarly, the service could switch to a higher bandwidth network interface such as Wi-Fi or 3G based on usage, thereby effecting scalability and cost. An ability to detect fluctuations in wireless signal strength could help the PMS to switch points of network attachment or to select a reliable network interface based on previous knowledge; in both cases the motivation for the adaptation would be to improve service availability.

Odin has previously lacked the ability to leverage contextual information. This paper presents our research in extending Odin with context-awareness support to improve the quality of mobile service provisioning.

II. Odin

Odin is a middleware solution that facilitates the design, development and deployment of mobile services [2]. It extends the Jini Surrogate Architecture specification to allow mobile devices to expose their services as a Jini federation. As shown in Figure 1, Odin comprises three key components: i) a service running on a mobile device known as the *device-service*, ii) a *surrogate* that represents the device-service in a fixed-network, and iii) a fixed intermediary called the *surrogate-host* (SH) that facilitates communication between a device-service, its surrogate and the clients consuming the mobile service.

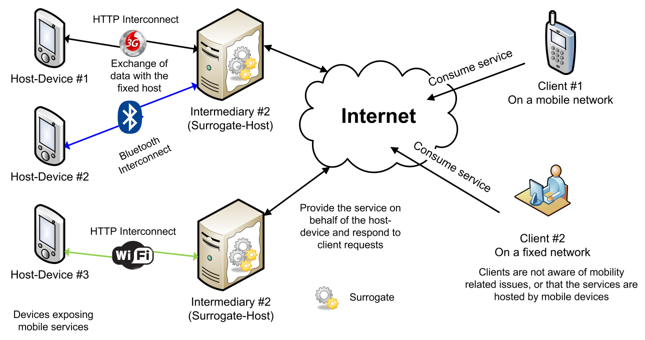
HTTP Interconnect
Exchange of data with the fixed host
Host-Device #1
Host-Device #2
Bluetooth Interconnect
Intermediary #2 (Surrogate-Host)
Internet
Consume service
Client #1
On a mobile network
Consume service
HTTP Interconnect
Host-Device #3
Intermediary #2 (Surrogate-Host)
Provide the service on behalf of the host-device and respond to client requests
Client #2
On a fixed network
Clients are not aware of mobility related issues, or that the services are hosted by mobile devices
Surrogate
Devices exposing mobile services

Figure 1. High level overview of Odin architecture

A device first registers with the SH by making available its surrogate as a JAR file. The SH will then register the surrogate as a Jini service, which will enable clients to discover and consume the device-service. All client requests will be first handled by the surrogate, which can communicate with the device if necessary. Communication between a device and the SH occurs over an *interconnect channel* (IC).

Odin provides HTTP and Bluetooth based interconnect channels to allow devices to leverage their multihoming capabilities [5] when communicating with the SH. The use of HTTP interconnect also allows Odin to overcome the reachability problem introduced in Section I. This is accomplished by the SH piggybacking service requests on replies to periodic HTTP keep-alive messages initiated by the device [2], [5]. Therefore, Odin allows mobile services to be provided independent of a particular mobile network operator. In addition, Odin supports device/user initiated (proactive) and reactive *vertical handover* between heterogeneous network interfaces; such as Bluetooth and IP based Wi-Fi/3G networks. The reactive handover transparently switches over to an available network if the current interconnect fails, without any data loss, thus improving service availability [5].

Odin also provides the ability to migrate a surrogate from one SH to another during runtime, transparently to clients [1]. With the aid of *surrogate-migration*, a patient's PDA hosting the PMS could switch from using Wi-Fi to a nearby SH with Bluetooth support under low power conditions to improve service availability [4]. Similarly, switching to a more able SH would help to better address scalability, when the current SH is under heavy loads. However, currently, the device-service has to manually initiate a request to migrate its surrogate to an alternate SH.

The use of an intermediary allows Odin to mask device mobility from clients. The additional computational power of SHs can also be leveraged by service developers offloading computationally intensive tasks from the device to its surrogate. The effect of this is to preserve battery life and to better address scalability. For detailed information on the architecture and design rationale for Odin, we direct the reader to [1], [2].

III. Related Work

In this section we focus on mobile specific context-aware systems found in the literature. All investigated systems support three common tasks [6]; i) *context acquisition*, which involves gathering contextual information (CI), such as device location and weather information, from context sources; ii) *context processing*, which is concerned with how CI is modeled and represented, and iii) *context management* that looks at how other applications and software modules interact with the system and use CI. Context-sources that provide CI can be hardware sensors (e.g. GPS), software modules (e.g. weather service) or logical entities that infer CI based on multiple sources (e.g. planned route based on location, time and calendar information).

The *Network Abstraction Layer* (NAL) [7] and di Costa's *Aspect-Oriented Middleware Architecture* (AOMA) [8] are two systems that are deployed directly within a mobile device. *Content Distribution Framework* (CDF) [9] and *Contory* [10] are examples of distributed frameworks that act as context brokers to collect CI from multiple devices. *Context-aware Mobile Service Platform* (CA-MSP) [11] aims to improve end-to-end quality of services hosted on a mobile service platform similar to Odin. Unlike other systems, which can gather context information both within the device and from external sources, AOMA and NAL are restricted to sources within the device they are deployed on.

A. Context Acquisition

Contory and AOMA acquire CI through polling their context-sources, whereas the others support both polling and subscription based mechanisms. Periodic polling is potentially wasteful and reduces battery life. Conversely, a subscription based mechanism can be used to acquire context as and when it changes. Hence, subscription-based mechanisms are preferable to conserve device resources.

Other than CDF and AOMA – where the former represents context sources as services whilst the latter uses aspect oriented techniques – the remaining systems lack support for adding context-sources at runtime.

B. Context Processing

Once acquired, raw CI must be modelled in order to better facilitate context management. There is no universally accepted standard for context representation, but a set of widely used approaches can be found in the literature [6]. For example, NAL uses a custom XML based markup scheme, whilst CDF uses OWL [12], an ontology based model. The other systems use object oriented models to represent context. Of these, the XML approach is relatively computationally expensive, and this is an issue for mobile devices. Object oriented models become harder to maintain and reason with as context models become more complex. Conversely, ontology based models provide richer semantics and expressiveness to perform reasoning with complex context models [6], [13]. Of the investigated systems, only Contory takes the *quality of context* (QoC) [14] into account when processing CI. Use of QoC attributes, such as freshness and precision, can be used to resolve ambiguities when conflicting data are present.

C. Context management

Except for AOMA and CA-MSP, the surveyed systems provide a means of exposing CI to be used by other entities (both software running within the device, and on other machines). However, only the former systems support dynamic adaptation of middleware to optimise behaviour according to the current operational conditions. For example, CA-MSP uses CI to improve its *quality of service* (QoS) by dynamically performing vertical handover. AOMA is the only system that allows an application to specify adaptation strategies at runtime. However, both AOMA and CA-MSP perform the reasoning logic within the devices, thus using valuable device-side resources. This may impact other applications running on the device, diminish battery life and lower service availability.

NAL, CDF and Contory can notify external applications of changes in CI, enabling them to augment their capabilities. Contory and CDF allow clients to subscribe and query both raw and processed CI, whereas NAL only exposes the raw CI. However, none of the systems provide the ability for clients to perform complex queries on CI within the systems themselves. Moreover, they lack support for aggregating CI and have limited expressiveness. Acharya et al has proposed a generic formalised method to address these issues by enabling clients to extract CI at different granularities and to offload computation logic to context-aware servers [15].

IV. Context-Aware Odin

A. Design and Architecture Overview

Figure 2 illustrates the three main stages to support context-awareness within Odin.

1) Context Acquisition: As shown in step 1 (Figure 2), Odin supports CI acquisition from context-sources residing within both the device and its SH. CI captured by the device is sent to the SH, via the currently active interconnect channel, where all the reasoning takes place. To improve CI management, various context-attributes are grouped into conceptually similar entities called *context-types*. Odin defines both the meaning and representation format for each supported context-type, as shown in Table I.

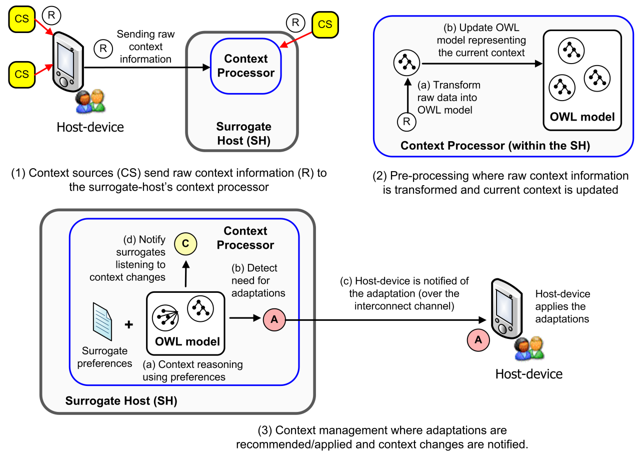
CS
R
CS
Sending raw context information
R
Host-device
Context Processor
R
CS
Surrogate Host (SH)
(1) Context sources (CS) send raw context information (R) to the surrogate-host's context processor
(b) Update OWL model representing the current context
(a) Transform raw data into OWL model
R
OWL model
Context Processor (within the SH)
(2) Pre-processing where raw context information is transformed and current context is updated
(d) Notify surrogates listening to context changes
C
Context Processor
(b) Detect need for adaptations
Surrogate preferences
OWL model
(a) Context reasoning using preferences
A
Surrogate Host (SH)
(c) Host-device is notified of the adaptation (over the interconnect channel)
Host-device applies the adaptations
A
Host-device
(3) Context management where adaptations are recommended/applied and context changes are notified.

Figure 2. Key components within context-aware Odin

Table I
Context-types and attributes supported by Odin

Context Type	Description	Attributes
Location	Location of device/SH.	Latitude and Longitude in decimal degrees.
Performance	CI that determine the load within the device and SH.	Overall CPU usage (%age), memory usage by the SH/device-service in MB, free system memory (%age).
Power	Power consumption within the device.	Remaining battery life as a percentage.
Network	Interconnect specific attributes (Wi-Fi, 3G and Bluetooth) within the device.	Received signal strength in decibels, IC type, whether IC is alive and/or active, bandwidth in Mbps.

Odin provides service developers with an API to abstract context-acquisition and which decouples context-sources from Odin. For example, developers specify and interpret in a standard form (decimal degrees), even when the context-source natively represents it in a different format (e.g. decimal minutes seconds). Combined with clearly defined context-types, this allows heterogeneous context-sources to expose information to Odin without any ambiguities.

Service developers specify the supported context-sources to Odin, and the middleware takes care of transparently acquiring CI. The developers can specify whether Odin should poll (including the frequency to poll) and/or subscribe to notifications from supported context-sources.

2) Context Processing: Upon receiving raw CI, the *context-processor-module* (CPM) within the SH maps the data onto an OWL model (step 2 in Figure 2). OWL was chosen for representing the context-model due to its richer semantics, greater expressiveness, support for advanced reasoning capabilities and existence of well developed third party tools [6], [12], [13]. Consequently, OWL enables context to be modeled, validated and reasoned more easily compared to alternatives discussed in Section III. Odin performs all context processing within the SH, freeing the device's CPU from computationally intensive reasoning tasks, and therefore prolonging battery life.

3) Context Management: The CPM performs two types of context management tasks; i) handling developer-defined *context notifications*, and ii) *middleware-level adaptations*.

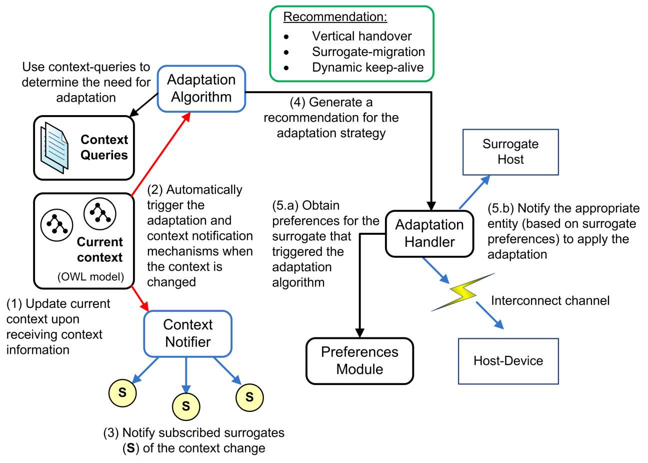
Recommendation:
Vertical handover
Surrogate-migration
Dynamic keep-alive
Use context-queries to determine the need for adaptation
Adaptation Algorithm
(4) Generate a recommendation for the adaptation strategy
Context Queries
Surrogate Host
(2) Automatically trigger the adaptation and context notification mechanisms when the context is changed
(5.a) Obtain preferences for the surrogate that triggered the adaptation algorithm
Adaptation Handler
(5.b) Notify the appropriate entity (based on surrogate preferences) to apply the adaptation
Current context
(OWL model)
Interconnect channel
(1) Update current context upon receiving context information
Context Notifier
Preferences Module
Host-Device
S
(3) Notify subscribed surrogates (S) of the context change

Figure 3. Context-aware processing within Odin

The context notification mechanism allows CI to be easily shared between different surrogates and device-services based on context-type. Each device-service/surrogate can subscribe to the CPM for updates concerning desired CI (step 3 in Figure 3). Odin allows service developers to control the visibility of CI exposed by a device-service to other surrogates deployed within the same SH.

With the aid of the context notification mechanism, surrogates can listen to changes in CI and utilise them to augment their service offerings. For example, this allows the parcel tracker service to be notified whenever the device location is changed. This is done without the involvement of service developers, thus minimising their development efforts.

Section IV-B presents more details about the specific adaptations that are enabled by Odin's context awareness provision.

B. Adaptation Support

Odin supports three adaptation strategies to dynamically change the middleware's behaviour.

i. *Proactive vertical handover* transparently switches the interconnect channel used by the device to communicate with its SH. This helps to optimise both device-side and SH resources. For example, to address scalability, the CPM will automatically migrate the surrogate to a more able SH if the current SHs load exceeds specified thresholds.
ii. *Middleware initiated surrogate-migration* dynamically migrates a surrogate to a different SH. This adaptation optimises bandwidth, power consumption and improves service availability based on a devices multihoming capabilities.
iii. *Dynamic keep-alive management* adjusts the frequency of sending keep-alive messages. This helps to optimise bandwidth usage and conserves power consumption at the device-side when the service is inactive.

Odin allows service developers to specify rudimentary policies as name-value pairs to determine how the adaptations are performed. For example, a policy used with the PMS might specify Wi-Fi as the preferred interconnect in order to minimise service costs. The CPM will perform vertical handover, based on the minimum required signal strength specified in the policy, if the Wi-Fi signal strength degrades.

As shown in step 5 of Figure 3, upon detecting the need for adaptation, a recommendation is forwarded to the appropriate entity (SH or device). The recommendation contains the information necessary to apply the adaptation.

V. Evaluation

In this section, we present a quantitative evaluation of the impact of adding context-awareness support to Odin. A simple service deployed on a HTC Hero smartphone running Android OS 2.1 was used for hosting a simple mobile service. The SH was deployed on a fixed machine running Windows Vista 64-bit OS on an Intel Core 2 Duo 3.66 with 3GB of RAM. The device and the SH were communicating using the HTTP interconnect over a 54Mbps 802.11g wireless network.

A. Performance impact of Context-Sources

Several experiments were conducted to monitor the device's battery life and the service's memory usage. In addition to a baseline test where no context-source was used, four experiments were conducted, each using one of four context-sources: power, location, network and performance. These sources map to the four context-types defined in Table I (e.g. the location context-source monitors the longitude and latitude). Finally, an additional test was performed with all four context-sources being enabled simultaneously. CI were polled from each context-source every 5 minutes. The results captured over a 24 hour period are summarised in Figures 4 and 5.

According to Figures 4 and 5, use of context-sources consumes more power and memory. The results reveal that memory usage remains approximately the same (24 MB) when using one or more context-sources. Therefore, simultaneous use of context-sources has minimal impact on the performance of a device.

As evident in Figure 4, the baseline experiment resulted in 20% power remaining after a 24-hour period. There was

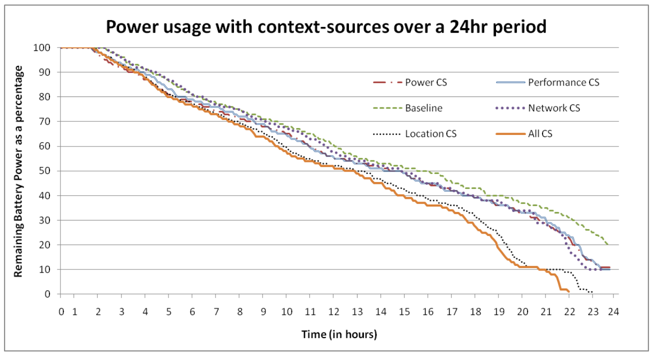

Figure 4. Power usage by context-aware Odin

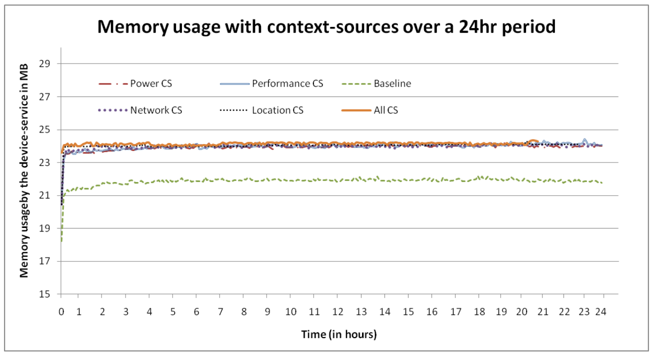

Figure 5. Memory usage by context-aware Odin

only 3% power left when using the phone's internal GPS as the location context-source, whilst other sources had 10-11% power remaining at the end of the experiments (when run individually). However, when used in combination, the device lasted for only 23 hours, before exhausting its power supply.

B. Adaptation responsiveness

Adaptation overheads can be separated into two categories; average time taken to determine an adaptation by CPM (*identification time*), and the time taken to apply the adaptation by the receiver (*application time*). Table II summarises the overheads for the three adaptation strategies introduced in Section IV.

The identification time involves processing CI as well as checking the context model using the adaptation algorithm (steps 1 - 4 in Figure 3), which are performed within the SH. Therefore, the identification times given are only experienced by the SH. The only quantifiable effects on device resources are those shown previously in Section V-A. Compared to other adaptations, the dynamic keep-alive mechanism takes a relatively small identification time since adaptation only depends on the surrogate load. Conversely, both the vertical handover and surrogate-migration strategies depend on multiple CI such as locations and network properties of both the device and current/other known SHs.

Table II
ADAPTATION RESPONSIVENESS

Adaptation Strategy	Identification time (ms)	Application time (ms)
Vertical handover	720 ms	20 ms (select new IC if both interconnects are active)
Surrogate migration	860 ms	591 ms (to migrate to a new SH)
Dynamic keep-alive	69 ms	12 ms (to compute the new keep-alive period)

Table III
COMPARISON OF KEEP-ALIVE MECHANISMS

Mechanism	Power left after 24 hrs	Message size (bytes)	No. of messages sent (in 24 hrs)	Total bandwidth consumed (KB/24 hours)
Dynamic Keep-alive	10%	150	321 (338)	47 (49)
Fixed Keep-alive	10%	150	6780 (17280)	993 (2531)

C. Benefits offered with dynamic-keep alive

The total number of messages and the power consumption was measured when using both the dynamic keep-alive mechanism (range of 5 seconds to 5 minutes) and a fixed keep-alive period of 5 seconds. Table III summarises the results for an idle service. The numbers in brackets in the last two columns refer to the expected values.

Note that both mechanisms send less than the theoretically expected number of messages. This is due to network latencies and processing carried out within both the SH and device. In addition to the 6% saving in battery life, the results clearly demonstrate that the use of dynamic keep-alive helps to minimise bandwidth usage by over a factor of 20. Consequently, this can have significant benefit in terms of reducing costs associated with running a mobile service.

In a typical usage scenario, where a mobile service is prone to long periods of inactivity, the dynamic keep-alive mechanism would send more messages than that shown in Table III due to fluctuations in device usage. Based on the observed results, we hypothesise that bandwidth savings would remain significant.

VI. FUTURE WORK AND CONCLUSIONS

A. Future Work

Currently Odin supports only there specific adaptations and lacks support to infer new information based on the current context. We are currently investigating to extend Odin to support aggregation of CI and inference support based on the ideas outlined in [15]. This would help to enhance the expressiveness of surrogate queries beyond simply registering to be notified when a particular context-type is changed. For example, surrogates would be able to specify when to be notified based on multiple pieces of CI; such as when another SH is in proximity of its device's Bluetooth range.

In addition, we are improving the reliability of context-reasoning algorithm with the use of QoC attributes mentioned in Section III. Overall, these enhancements will help to enrich context provisioning within Odin, to make it more flexible for service developers to utilise CI.

B. Conclusions

In this paper we have presented context-awareness extensions to Odin, our middleware for mobile service provisioning. Our approach distinguishes itself from existing solutions in two ways. First, we perform context reasoning within the SH to prolong service availability by reducing power consumption. Second, our solution supports both context notifications - to augment mobile services - and three middleware-level adaptations. Furthermore, context-aware Odin supports context acquisition through both polling and subscription mechanisms and provides a well defined API to assist service developers to easily plugin heterogeneous context-sources to the middleware.

The adaptations help tackle challenges faced by mobile service provisioning – such as reachability, service availability and scalability – by making best use of available resources based on current operational conditions. In addition, experiments reveal that the dynamic keep-alive adaptation in particular helps to reduce bandwidth and therefore minimises service provisioning costs. The quantitative analysis further demonstrates that adaptations are identified and applied in a timely manner. Though the use of context-sources slightly reduces the devices battery life, we believe the aforementioned advantages provide a greater benefit in improving quality of service overall.

REFERENCES

[1] T. Weerasinghe and I. Warren, "Empowering Intermediary-Based Infrastructure for Mobile Service Provisioning," in *Proc. IEEE Asia Pacific Services Computing Conference (APSCC' 09)*. Singapore: IEEE Computer Society, December 2009.

[2] A. Meads, T. Weerasinghe, and I. Warren, "Odin: A Mobile Service Provisioning Middleware," in *New Zealand Computer Science Research Student Conference*, Auckland, New Zealand, April 2010.

[3] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, 2000. [Online]. Available: http://www.dsv.su.se/fuse/int8/docs/context.pdf

[4] E. Ferro and F. Potorti, "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12–26, 2005.

[5] A. Meads, A. Roughton, I. Warren, and T. Weerasinghe, "Mobile Service Provisioning Middleware for Multihomed Devices," in *Proc. 5th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob' 09)*. IEEE Computer Society, 2009.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad-Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, June 2007.

[7] A. Peddemors, I. Niemegeers, and H. Eertink, "An Extensible Network Resource Abstraction for Applications on Mobile Devices," in *Communication Systems Software and Middleware 2007*. IEEE Computer Society, January 2007, pp. 1–10.

[8] C. M. da Costa, M. Strzykalski, and G. Bernard, "An Aspect Oriented Middleware Architecture for Adaptive Mobile Computing Applications," in *Proc. In 31st Annual International Computer Software and Applications Conference (COMPSAC '07)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 81–86.

[9] P. Pawar, A. van Halteren, and K. Sheikh, "Enabling Context-Aware Computing for the Nomadic Mobile User: A Service Oriented and Quality Driven Approach," in *Proc. In IEEE Wireless Communications and Networking Conference*. IEEE Communication Society, March 2007, pp. 2531–2536.

[10] O. Riva and C. di Flora, "Contory: A Smart Phone Middleware Supporting Multiple Context Provisioning Strategies," in *Proc. In 26th IEEE International Conference, Workshops on Distributed Computing Systems*. IEEE Computer Society, 2006, pp. 68–73.

[11] P. Pawar, K. Wac, B. J. van Beijnum, P. Maret, A. V. van Halteren, and H. Hermens, "Context-aware middleware architecture for vertical handover support to multi-homed nomadic mobile services," in *Proc. In ACM symposium on Applied computing (SAC' 08)*. New York, NY, USA: ACM, 2008, pp. 481–488.

[12] D. Mcguinness and F. van Harmelen, "OWL Web Ontology Language Overview," W3C, W3C Recommendation, February 2004.

[13] T. Strang and C. Linnhoff-Popien, "A Context Modelling Survey," in *Proc. In 1st International Workshop on Advanced Context Modelling, Reasoning And Management (Ubicomp 2004)*, 2004.

[14] K. Sheikh, M. Wegdam, and M. van Sinderen, "Middleware Support for Quality of Context in Pervasive Context-Aware Systems," in *Proc. In 5th IEEE International Conference on Pervasive Computing and Communications Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 461–466.

[15] A. Acharya, N. Banerjee, D. Chakraborty, K. Dasgupta, A. Misra, S. Sharma, X. Wang, and C. Wright, "Programmable presence virtualization for next-generation context-based applications," in *Proc. In 2009 IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–10.