Generic and Open Media Access for Next Generation Wireless Networks: A Case Study

Mahesh Sooriyabandara, Member, IEEE

Abstract—**A number of technology factors including increased connectivity to a diverse range of devices, variety of wireless access technologies (e.g. cellular, WLAN, WPAN, sensor), and highly reconfigurable software and hardware components have been driving telecommunications systems to provide ubiquitous and transparent services. At the same time, dynamically reconfigurable and/ or adaptive communication systems that are capable of operating in a variety of modes increases the heterogeneity of operating communications environments when used by users (including applications and devices) that have conflicting capabilities and requirements. Generic Application Programming Interfaces (APIs) are a key technology component to realize flexible and intelligent management of available modes of operations and adaptation of applications because they enable access and control of heterogeneous technologies in a uniform way. The case study presented in this paper explores the concept of generic APIs in the context of adaptive and reconfigurable communications systems. A generic API called Unified Link Layer API (ULLA) developed by a consortium of European researchers (from industry and academia) is reviewed to demonstrate the benefits offered by such a technology to the dynamic optimization of wireless multimedia systems.**

Index Terms— **Generic API, wireless communications, streaming multimedia, optimization, reconfigurability**

I. INTRODUCTION

THE increasing demand for transparent access to different communications services will increase the presence of reconfigurable and adaptive systems in future telecommunications networks. This trend will be driven by a number of different technology factors such as increased connectivity to different devices, utilizing a large number of radio access technologies (WAN, LAN and PAN) and in the longer run software radio solutions. Today, it is common to find communication devices equipped with more than one link technology that enable several communication modes. Most of these devices often come with proprietary interfaces thus limiting the application's ability to exploit access

opportunities made available by multiple communication modes. To resolve this problem there are a number of attempts to unify media access especially in the wireless and mobile domains [2].

In the foreseeable future, not only will we see the arrival of highly reconfigurable (both hardware and software) communication devices but also a new generation of adaptive and intelligent services that are capable of exploiting the multimode and reconfigurable capabilities of such devices. Research communities around the world are studying architectures for reconfigurable and adaptive systems. While some of these research efforts are focused on defining frameworks and architectures to realize reconfigurable systems (e.g. E2R [6] , XG [7]), others are looking at specific technology components including software radio solutions (e.g. SDR [8], OMG [7]), modular and flexible protocol stacks, flexible execution environments and generic Application Programming Interfaces (API).
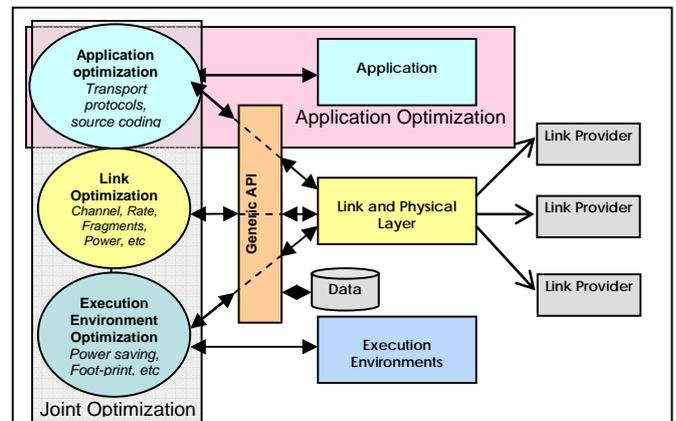

Fig. 1. **Terminal based dynamic optimizations**

Generic link layer APIs [1] are in particular central to future systems since communication modes of devices are probably best abstracted at this layer of the OSI model. As shown in Fig. 1, a generic Link Layer API could facilitate dynamic optimizations at several levels: application, Link Layer, execution environments, etc. At application level, algorithms and protocols (this includes both application and transport layers) could be configured to optimize application operation according to user preferences and context. At the link level, appropriate communication modes (i.e. this refers to available radio access technologies or options within them) could be

selected. Execution environment (i.e operating system and related extensions in a reconfigurable system) optimization refers to the ability to configure (e.g. protocol stacks) the way in which an application is executed with a device. This allows dynamic reconfiguration of protocol stacks. More details on a software reconfigurable architecture can be found in [4].

This paper will look at the role of generic Link Layer APIs to facilitate optimization processes in multimode devices by evaluating one such generic and open Link Layer API called the ULLA (Unified Link Layer API) currently being developed by a European Union funded research project [1]. Section II of this paper describes generic API concepts, ULLA design and discusses short-term and long-term application scenarios. Sections III and IV provide an overview of a prototype implementation of the ULLA framework and an adaptive multimedia streaming application to illustrate the application of the ULLA generic API framework in a optimization scenario. Sections V and VI discuss the results and future work by considering the ULLA as a migration path for fully reconfigurable systems of the future.

II. GENERIC API

A. State-of-the-Art

Almost all of the Link Layer APIs available today are specific to a particular technology and/or operating system. Therefore, the short-term goal of any Generic Link Layer API would be to resolve the complexity and interoperability problem associated with a diverse range of APIs and methods used for accessing communication interfaces; also to provide real and useful triggers, handles for different smart, context-sensitive and link/network awareness to enable the development of "cognitive applications" [1]. This problem has been known for some time, and there has been some prior work on uniform and generic link layer APIs reported in the literature. The Odyssey framework and APIs developed in the Glomo research programme are some of these. The Odyssey framework defines a simple interface to allow applications to access link information in a generic manner. However, it is platform dependent and does not provide an interface to configure links. Under the Glomo research programme, a number projects have developed APIs to enable link and radio access in a uniform manner. These APIs had been designed with mobility and routing in mind. In addition to those there are also widely used uniform APIs such as Linux Wireless Extensions (LWE), and the Windows management APIs supported via by the NDIS driver interfaces. The LWE provides access and control to 802.11 based technologies on Linux devices. However, it is not extensible to other technologies or operating systems. Window management APIs and NDIS interface on the other hand provide an extensible and generic interface to network drivers, but support only Ethernet based network interfaces and are mainly used for power management and plug-and-play support. In general, none of these solutions are generic enough to address the diverse requirements of heterogeneous and reconfigurable

wireless networks such as extensibility, scalability, and platform/ link technology independence. On the other hand, the ULLA API considers requirements derived from a diverse range of application domains (e.g. mobile, wireless and embedded) and accommodates for extensibility for future technologies. A detailed state-art-review of a range of currently available Link Layer APIs can be found in [2].

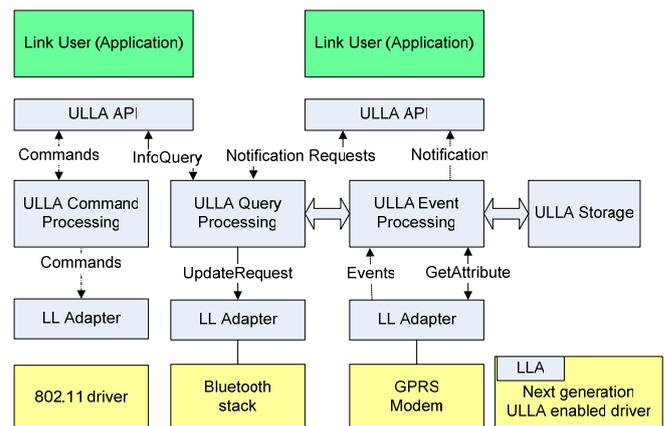### B. ULLA Design Rationale

Fig. 2 ULLA architecture block diagram

The ULLA framework conceals the technology specific details of link technologies from applications by abstracting link technologies through a generic "link" class which represents communication service provided by link providers. Link providers are devices or software entities that provide a communication facility. Link users (where link users can include any higher layer protocols, middleware or application software) access and control the available communication devices and modes using the generic and simple API provided by the ULLA framework. Technology specific details are captured by extending this generic class using class inheritance techniques. The services offered by the ULLA framework falls into three categories:

- Queries: Queries could either be synchronous information requests or asynchronous notification subscriptions. An SQL based query language called UQL (ULLA Query Language) has been used to specify complex queries in a generic and standard manner. Query processing functionality in the ULLA framework translates the requests received from link users and returns appropriate link information back to the link users.
- Events: Events refer to the asynchronous notifications forwarded towards the link user by the ULLA framework based on a criteria specified by the link user using the UQL. Event processing functionality forwards appropriate link configurations, measurements, statistics and other link events in an asynchronous manner to applications.
- Commands: Commands allows link users to configure links and link providers in a standard way.

A data storage that keeps link specific information centrally, allowing efficient information access by Link users,

is also defined by the ULLA framework. An entity called Link Layer Adaptor (LLA) is defined to provide a proxy interface to integrate legacy drivers to the ULLA framework.

Non functional requirements include extensibility, platform-independence and scalability which are essential to the generic API concept. The extensibility refers to the ability to extend functionality to support multiple types of links provided by the diverse range of legacy and future access technologies. The ULLA adopts the class inheritance concept of the Object Orient Design (OOD) principle to support this extensibility. Platform independence is provided through encapsulating the core functionality using an open and standard set of C based interface functions. Scalability or support for devices with different resource capabilities is achieved by using different ULLA core profiles. The initial proposal considers a three tier profile system to support very small sensor type devices up to highly capable laptop type devices. In addition to that the framework is comprised of several other concepts such as role management to handle privileges and link management to resolve conflicting configuration requests from different applications.

C. Usage scenarios for Generic Link Layer APIs

There are a number of application scenarios applicable to multi-mode devices or reconfigurable terminals when they operate in a dynamic communication environment by roaming around and connecting to different networks or by switching among various available communication modes. The following list illustrates example applications of the ULLA API.

1) Optimization support

Wireless terminals may perform optimization of different system aspects including application performance, power management, security management, connection management, mode selection, content adaptation, link adaptation and transport adaptations. This often requires accurate and timely information regarding estimated (or actual) performance, cost, security and resource utilization characteristics retrieved locally from the device and/or remotely from other network nodes. For example, the multimedia streaming application described in section III performs real-time adaptations using notifications about channel characteristics or infrastructure changes received through ULLA.

2) Intelligent Connection Management

In addition to optimization support, intelligent connection management schemes could significantly benefit from using a generic API because it allows discovery and control of available communication modes allowing seamless inter- and intra-technology handovers. Especially, opportunistic or cognitive connection management algorithms could use information gathered overtime to learn and reason about the operating environment when selecting a suitable connection.

3) Context Aware Applications

The ULLA framework enables implementation of advance context awareness within applications by allowing application to extract spatial, temporal, location, or user-specific contextual information about the operating environments.

4) Service and resource management

In general terms, reconfigurable systems will require performance related capabilities as well as preferences related to cost and security to perform service or resource management functions in a dynamic manner. Service adaptations may be performed by changing or tuning the protocol stack modules to optimize the user experience Execution environments could also be configured to provide balance the trade-off amongst performance, power consumption and security. Hence application preferences are achieved by reconfiguring appropriate modes as well as the most appropriate execution of the software (and configurable execution environment hardware) supporting the mode functionality. Further, generic APIs allow management of the capabilities of the devices such as communication parameter settings and scheduling of software and/or firmware updates. For example, information regarding links available for software download and notification of available networks or link performance related information would be used for device management.

III. MULTIMEDIA PROTOTYPE

The performance of multimedia streaming applications is heavily dependent on the loss and congestion characteristics of the operating environment. This is especially true for wireless systems where the dynamics of wireless environments are time varying and dependent on the activity of other wireless users. For example, packet loss that arises from link impairments or congestion often degrades the quality of streaming video impacting upon end user experience. Accurate characterization of links is often required to realize optimal adaptations of multimedia applications under such conditions. Optimizations could be implemented at the link level or application level (or any other layer of the OSI model). For example, a video application could monitor available channel usage and vary streaming bit rates accordingly to optimize user experience under heavy link congestion. Alternatively, applications can monitor channel usage or interference level of wireless links and decide on the selection of a more suitable alternative channel. Reconfigurable systems could perform more advance protocol adaptations or change the operational mode thorough replacing protocol modules completely.

To demonstrate usage of Generic APIs for dynamic optimization in the context of non-reconfigurable adaptive systems, this section presents a prototype of multimedia streaming application that implements a dynamic channel selection algorithm in a wireless LAN environment. The scenario uses a multimedia optimization agent to characterize channel conditions by retrieving link information and events. It is also capable of configuring channels in real-time using the command interface provided by the ULLA API.

A. Test Platform

The test platform that we have developed and used for evaluation includes three laptops where one laptop runs a video server and streams video over a WLAN link (IEEE802.11b) to a client running on another laptop. The third laptop is used as an interferer to deteriorate the WLAN link quality. It is envisaged that this optimization application scenario captures common performance problems associated with wireless multimedia streaming in LAN environment where packet loss results from interference or link impairments.
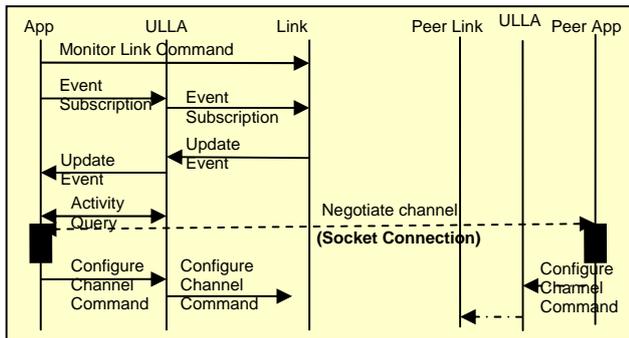
B. Algorithm

Fig. 3. Simplified message sequence diagram to illustrate operation of the Dynamic Channel Selection Algorithm

Fig. 3 illustrates the Dynamic Channel Selection algorithm implemented using the services exposed by the ULLA framework. The aim of the algorithm is to select the best available channel (i.e. less congested channel with minimum interference) by monitoring all available wireless frequency channels. For this purpose, a channel monitoring agent associated driver software that continuously and periodically monitors usage or activity of the different available channels in the 802.11 network was used. The application initiates link monitoring using the command interface of the ULLA API (this action is marked as the "Monitor Link" command in the Fig. 3). The agent at server-end subscribes for event notifications of channel quality (e.g. channel quality may be measured using noise level) reaching a certain threshold (either at the local or client ends or both). An example query that could be used to setup a notification subscription in this of scenario would be "*SELECT NoiseLevel from IEEE802_11 where NoiseLevel > -70*" (the action marked as "subscribe Event" in the diagram). On receiving event notification, the server agent then queries information regarding usage (activity) on alternative channels (shown as query activity in the diagram). The format of the query would look like "*SELECT activity, frequency from IEEE802_11Channel*". The agent then selects a suitable alternative channel frequency with the aim of providing best possible communication for the multimedia streaming application. Finally, it requests both server and client ends of the wireless link to change channel to

a selected frequency (shown as "Negotiate Channel" in the diagram) over a TCP/IP socket connection.

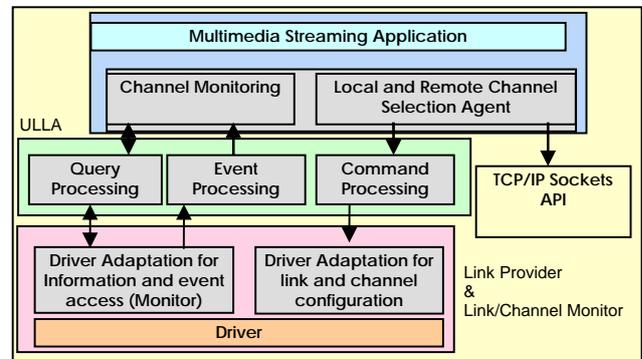### C. Multimedia Streaming Prototype Architecture

Fig. 4. Overview of adaptive multimedia streaming application

The prototype that we have developed for real-time performance adaptation of multimedia streaming applications constitutes a modified video streaming application and adapted 802.11 wireless network drivers (link adaptors). They communicate with each other using the interfaces and services provides by the ULLA generic API framework. The architecture of the prototype is shown in Fig. 2. The VideoLAN [10], a publicly available video steaming tool was used as the application. The adaptive component of the prototype constitutes two mechanisms: one for channel monitoring and the other for channel configuration. The channel monitoring is achieved by a software agent that utilizes the query and event interface provided by the ULLA. The channel selection agent is responsible for coordination of channel configuration of remote and local ends. It again uses the ULLA command interface to configure respective ends. In this particular implementation, the request to configure the remote end is sent over a TCP/IP connection established using the standard INET socket interface resulting in client-server software architecture. The LLA component is constituted two modules one for standard link provider functionality (i.e. access to link information and control capability) and the other for channel monitoring.

D. ULLA API Implementation

This section describes specific the ULLA implementation used for this evaluation. This implementation is not yet fully compliant with the ULLA API framework which is still being defined by the Gollum consortium. The implementation described in this paper only an example prototype implementation at the user space. Although the consortium wishes to make recommendations on the ULLA implementation options, it is envisaged that there could be several different implementation options for this framework. Note that the Gollum consortium, [1] that is defining the

ULLA framework, will make public the ULLA API interfaces and associated functionality and provide a reference implementation by the end of 2006. The choice of the implementation described in this paper partly depends on the target platform and multimedia prototype selected for the evaluation.

The implementation as shown in Fig. 5 incorporates a shared library (e.g. libulla.so) that implements all core functionalities of the ULLA described in section II-B. It uses an off-the-shelf database solution, the PostgreSQL 8.0.2 [9] server at the back-end as ULLA data storage for storing the relevant link information. This particular implementation of ULLA benefits from the built-in capabilities of the Postgres database at the expense of a relatively large footprint.
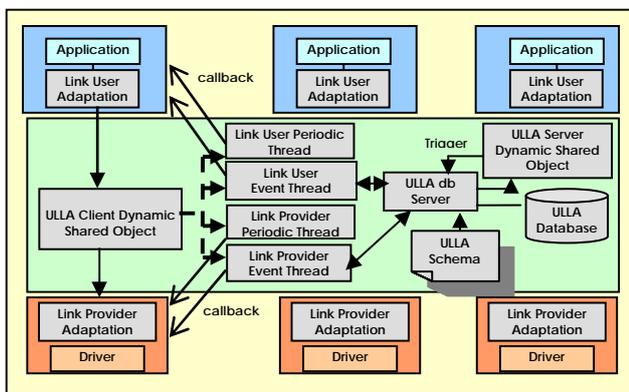

Fig. 5 Example Linux implementation of ULLA framework

The libulla library includes interfaces exposed to link users (e.g. applications) as well as link providers (normally ULLA compatible network interface drivers). This implementation expects applications to dynamically load the ULLA library in order to access generic API services. Also note that the use of off-the-shelf database like Postgres may only be suitable for high-end devices with adequate resources (e.g. laptops). In the case of resource limited devices, a light-weight or proprietary database solution could be used. The IPC mechanism used for the command interface and for internal ULLA core processes is based on a generic asynchronous message passing scheme. For Linux platforms the IPC uses System V shared memory and semaphores. The necessary shared memory segment is set up at initialization of the first process (i.e. a LP) and subsequent processes (i.e. LU and other LP) use the existing IPC shared memory block. On Linux platforms the UNIX socket method is used to access the Postgres backend and this is used in the ULLA core to retrieve information from the database as well as to obtain asynchronous notification triggers. The Postgres database access functions are used for this purpose and are contained in a shared object library (libpq).

To adapt the network driver to the ULLA framework, an LLA that utilizes Linux wireless extensions version 17 was used to interface with hostap driver (version 0.3.9) [11] of 802.11 devices. The LLA implements several commands including *scanAvailableLinks* and *setChannel* which can be

executed using the command API provided by ULLA. The modified driver returns the throughput and noise level information using the wireless extensions iwspy mechanism. A separate wireless card was used for channel monitoring which gathers network measurements and statistics by passively listening to traffic from 802.11a, 11b and 11g networks by operating in monitor (rfmon) mode.

IV. RESULTS & ANALYSIS

In this section the performance improvement offered by the optimization algorithm based on the ULLA API is presented. However, the main goal of evaluating the multimedia prototype implementation described above was to demonstrate use of a Generic Link Layer Application Programming Interface (API) for dynamic optimizations of wireless and mobile multimedia devices. These results could be extended to predict implications of using this API in other potential applications including connection management in multimode devices and reconfiguration management in reconfigurable systems which were described in section II-C.

Fig. 6. Performance improvement of multimedia streaming application using Dynamic Channel Selection algorithm

The Fig. 6 above compares video performance with (left) and without (right) the adaptation algorithm described in section III-B. The scenario begins when the video streaming is at an acceptable quality level as the neighboring channels are less active and the data experiences minimum or no packet loss. At this stage the video is free from defects. When an adjacent channel is used to exchange large amounts of traffic, this causes noisy interference on the channel being used for the video transmission. Packets are dropped at the client end due to errors and this becomes very evident in the form of artifacts and other degradations in the received video stream (as evident from the right window in Fig. 4). During high-interference periods, the video simply freezes since there are too many packet losses due to errors. When the adaptive algorithm is used, as soon as a certain noise threshold is exceeded on the link, the server device performs a scan of the

other WLAN channels available to assess usage levels and find a better channel to mitigate the error and loss behavior. The channel with the least potential interference levels is selected and the server then instructs the client to change channel along with itself. When at least one channel with suitable quality is available, this ensures that the video continues at acceptable quality levels (as apparent from the left window in Fig. 6)

The performance improvements observed above were made possible by the rich set of features of the ULLA which enables the characterization of links which facilitates detection of the effects that could impact application performance together with identification of their possible causes. It also enables applications to configure the links when necessary, for instance to provide more or less resilience to interference or other impairments.

In addition to this, the initial results on the performance of the ULLA implementation have indicated that it is capable of meeting real-time requirements of multimedia scenario without excessive use of the system resources.

V. CONCLUSIONS

The results have shown that the ULLA generic API framework enables the realization of dynamic optimization algorithms for adaptive multimedia streaming applications. Some of the benefits offered by such a generic API to dynamic optimization of non-reconfigurable systems are:

[1] It eases the implementation of adaptive applications by providing a set of powerful and simple interfaces to application developers to characterize and control wireless systems and environments.

[2] It provides monitoring and notification (event based trigger) capabilities to enable dynamic and intelligent adaptations including cross layer optimizations.

[3] It will act as a powerful tool to build cognitive resource management applications including terminal based connection management and device reconfiguration.

In addition, a number of important aspects regarding the use of link information (and methods used for collecting them) for performing adaptations have been identified during the prototyping activity. For example, it has been identified that relying solely on instantaneous values of performance (such as activity and noise levels) and generic statistical parameters as provided by legacy software network interface drivers could be misleading and undesirable. The details of these findings are beyond the scope of this paper and are the subject of on going investigation. If required, generic API defined by the ULLA framework can allow applications to query measurement capabilities and to configure measurement methods implemented by link providers enabling retrieval of statistical and measurement information with high accuracy.

The analysis of ULLA framework has shown that the capabilities of a generic Link Layer API can facilitate not only the advanced and real time adaptations with in a single communication mode but also reconfigurations of multiple modes using the interfaces exposed to a particular application.

VI. FUTURE WORK

Generic and extensible API frameworks such as ULLA can provide the means to cope with non-reconfigurable as well as reconfigurable devices enabling them to obtain meaningful events and to control operational modes in a uniform way. For example, ULLA could provide a migration path from multi-mode to fully reconfigurable devices. However, it will be important to understand potential implications of new requirements that will emerge from future reconfigurable systems as well as other non-technical drivers (e.g. market, industry). Therefore, future research will focus on exploration of evolution of the Generic Link Layer API concept to cater for requirements of emerging and future reconfigurable systems; and to explore how such frameworks could be exploited to realize advance and intelligent communication architectures and technologies (e.g. mechanisms for information exchange, reconfiguration management, resource management, context awareness) especially in the context of reconfigurable systems with in projects such as E2R [6].

REFERENCES

[1] Gollum Project website http://www.ist-gollum.org accessed 10/04/2006
[2] "Generic Open Link Layer API for Unified Media access", Gollum state-of-the-art report Chapter 4 – Link-Layer APIs http://www.ist-gollum.org/docs/Gollum_D21_State_of_the_Art_Public.pdf accessed 10/04/2006
[3] T. Farnham, A. Gefflaut, A. Ibing, P. Mähönen, D. Melpignano, J. Riihijärvi and M. Sooriyabandara; "Toward Open and Unified Link-Layer API"; IST Mobile Summit 2005, Dresden, Germany, June '05
[4] Georganopoulos, N.; Farnham, T.; Burgess, R.; Scholer, T.; Sessler, J.; Golubicic, Z.; Buljore, S.; "Terminal-centric view of software. reconfigurable system architecture and enabling components and technologies", IEEE Communications Magazine, Volume 42, Issue 5, May 2004 Page(s):100 – 110
[5] XG project website, http://www.darpa.mil/ato/programs/XG/ visited 10/04/2006
[6] "End-to-End Reconfigurability" (E²R) Integrated Project website, (http://e2r.motlabs.com/) accessed 10/04/2006
[7] Object Management Group (OMG) www.omg.org accessed 10/04/2006
[8] SDR Forum website, http://www.sdrforum.org/ accessed 10/04/2006
[9] PostgresQL database http://www.postgresql.org accessed 10/04/2006
[10] VideoLAN, http://www.videolan.org accessed 10/04/2006
[11] Hostap wireless driver http://hostap.epitest.fi/ accessed 10/04/2006