GOLLUM White Paper

Version 0.4

The GOLLUM Consortium

June 2006

http://www.ist-gollum.org

Scope

This document describes the concept of a Unified Link Layer API (ULLA), which has been developed by an EU-funded research consortium project named "GOLLUM" (Generic Open Link Layer API for Unified Media access). ULLA targets mobile applications that need to manage a variety of wireless adapters.

The purpose of this document is to discuss the benefits ULLA could bring to HW platform providers, OS vendors and Independent SW Vendors (ISV). The GOLLUM consortium would like to get feedback about the ULLA concept so that a reasonable decision can be made concerning the best path for its exploitation at industrial level.

The intended audience for this document is twofold. Executives may quickly grasp the key characteristics of ULLA in the short executive summary, whilst a detailed view of ULLA is also provided for a technical audience.

Document History

0.1	17 May 2006	Created during Brussels F2F
0.2	5 June 2006	Merged MS contribution
0.3	7 June 2006	Comments from RWTH and TREL
0.4	9 June 2006	Final draft

Table of Contents

Executive Summary

The increasing variety of wireless adapters available on mobile terminals makes it more and more difficult to develop applications that can manage all of them. No single API exists nowadays to control wireless links ranging from cellular to IEEE802.11x, IEEE802.16x, Bluetooth, UWB, Zigbee and so on.

The GOLLUM project – a research consortium that includes RWTH Aachen University, Microsoft (European Microsoft Innovation Center), STMicroelectronics, Toshiba, Telefonica, Materna and the University of Cantabria – has been addressing this issue. GOLLUM proposes a Unified Link Layer API (ULLA) that can be used with existing and future wireless adapters. ULLA is OS independent and has been prototyped on WinCE, Windows XP, Linux and TinyOS.

ULLA provides a consistent wireless link layer programming framework that has a twofold approach:

- for ISV and OS providers, ULLA offers both an API and an extensible data model that abstracts wireless link differences; by using ULLA, applications can configure links and receive asynchronous notifications in a standardized way;
- for wireless hardware providers, ULLA provides a framework where device drivers have a unified way of exporting adapter features in an OS independent way.

ULLA exposes the characteristics of available wireless links using a database abstraction. A subset of SQL is used to retrieve link characteristics, statistics, available bandwidth, power consumption estimates and link QoS capabilities. Typical types of applications that could benefit from ULLA are:

- Connection Managers (e.g. UMA or IEEE802.21 agents) that are responsible for selecting the most appropriate wireless link depending on user or operator specified policies, including handover support;
- Multimedia clients and servers that need to adapt the characteristics of their generated traffic to varying wireless link conditions.

ULLA is generic enough to accommodate future emerging requirements, for example from the Cognitive Radio research area. Nevertheless, the GOLLUM project has demonstrated that ULLA is applicable and useful already in today's platforms that range from sensor network nodes to typical mobile phone designs and even laptop computers.

Feedback about the ULLA approach, related business models and impact on existing HW/SW architectures, is needed from the different stakeholders to better evaluate its potential. The GOLLUM consortium believes that the benefits of ULLA will become clearer once there is sufficient adoption of the API at industrial level. The options to raise industry awareness about ULLA are currently being investigated, one of them being the creation of a Special Interest Group (SIG).

Further information about ULLA and code examples that show how it is supposed to be used by applications can be found in the following sections of this white paper.

Introduction

In the OSI stack model, the data link layer (also simply called the link layer) is layer two of the seven defined layers, lying between the physical and the network layers. The main purpose of the link layer is to transfer data across the physical link joining adjacent nodes. Usually a link layer API is used to access information related to the corresponding link.

There has been a considerable amount of work devoted to the definition of programming APIs that enable applications to retrieve link layer information. However, three main problems remain. First, most wireless technologies offer different types of APIs for accessing link layer information. For example, in most operating systems the programming interfaces for Wireless LAN are completely different from the Bluetooth ones, which are both different from the APIs for accessing cellular system information. On Windows XP or Windows CE for example, Wi-Fi links are accessed via NDIS, Bluetooth through a socket as well as a proprietary interface, and cellular links are accessed via the TAPI or Extended TAPI interfaces. These APIs differ greatly between the different operating systems. A second major problem is the lack of uniformity in the information obtained from these APIs. Typically, only technology-specific data can be obtained without any abstraction, making comparison of different links difficult. Often measurements have only interface card or device-specific meaning; even within the same technology. This is due to variations in the hardware and drivers from different manufacturers. Finally, the existing link layer API's have very limited support for asynchronous notifications. This forces applications to rely on polling in order to detect changes occurring in the link conditions. Even when these kinds of notifications are supported, they tend to be hard-coded and rather narrow in scope, limited to basic connectivity related events. For example, the Network Driver Interface Specification (NDIS) provides asynchronous notifications only for a limited set of predefined events (BIND, UNBIND, etc…). It does not provide any generic notification mechanism(s) that allow applications to define their own notification criteria (e.g. "Notify me when there is a network with a bitrate higher than 2 Mbits/s").

In addition to these fundamental issues, existing link layer APIs do not offer support for gathering and using statistical information regarding link attributes. Observing only present attribute values makes link-aware programs vulnerable to transient behavior, often causing expensive adaptation procedures to be initiated by momentary changes in link conditions. Collecting smoothing and time averages would allow programmers to select the temporal granularity they wish their programs to operate on. This way they would not have to implement this kind of functionality from the ground up for each application.

This paper presents a solution to these problems by introducing a *Unified Link Layer API (ULLA)*. ULLA is an operating system independent interface, offering a unified, technology-independent view of the wireless[1] links and technologies available on a

[1] This white paper mostly focuses on wireless link technologies. Nevertheless, the design of ULLA does not distinguish fundamentally between wired and wireless technologies, and can be used to access information on and configure fixed links as well.

platform. The interface supports not only querying for information from the various links, but also the issuing of commands to change the configuration of the links and wireless interfaces. It also supports flexible asynchronous events for signaling changes in the conditions of the wireless channels, based on configurable criteria.

The remainder of this document is organized as follows. The next section discusses the main motivations for a Unified Link Layer API and the scenarios where it can offer advantages over the existing state-of-the-art. The ULLA reference architecture is presented and the API usage is then shown by means of typical examples using the C programming language.

Why ULLA?

As wireless channels are inherently dynamic in nature, adapting to changes in channel conditions is required to provide a positive experience to users. Increasingly, wireless channels are highly heterogeneous in terms of the communication capabilities that they offer. Considerable research effort has been directed towards the development of content adaptation and protection techniques, and methods for mitigating the adverse effects of horizontal and vertical handovers. However, there exists one major obstacle that has severely limited the introduction of such techniques in heterogeneous wireless environments, namely the difficulty of obtaining information from wireless network interfaces, particularly if more than one technology is to be supported.

ULLA (Unified Link Layer API) is a first attempt at fixing this problem by providing a simple and uniform way to access link layer information independently of the targeted technologies. From a programmatic point of view, ULLA offers the following features:

1. A technology independent programming interface enabling applications to query and configure links. For this purpose the API defines the notion of information query and commands.
2. An asynchronous programming model allowing applications to flexibly specify and register link notifications using chosen link characteristics. These notifications can be periodic or based on the occurrence of events happening at the link layer.
3. A model that will enable future technology to be integrated without having to redefine the existing API.

The remainder of this section focuses on the presentation of several scenarios where such a Unified Link Layer API (ULLA) would provide a definite advantage over the existing situation.
Readers interested in API details can jump directly to page 13.

Cross Layer Optimizations

The term “Cross Layer Optimization” encompasses all techniques and methods that allow OSI networking layers to adapt their behavior based on the perceived state at lower or higher layers of the stack. In recent years, a variety of solutions for wireless cross-layer optimization (CLO) have been proposed. CLOs employ multiple parameters at different layers of the protocol stack that are jointly optimized to meet varying application requirements. However, these solutions are often developed in an ad-hoc fashion, limiting their widespread use as standard mechanisms. Therefore, the ability to control radio “knobs” (**[1]**) in an intelligent way requires that interfaces are defined to expose controllable parameters. ULLA can be an important enabling technology for CLOs by providing a means to access a wide variety of link layer parameters. We strongly believe a component such as ULLA could help optimize network operations across the protocol stack.

Transport protocols such as TCP are adversely affected by wireless link error and delay patterns, leading to throughput reduction and energy waste in a mobile terminal. TCP was designed at a time where wireless links were not widespread and hence is based on the assumption that errors are caused by congestion in the underlying network As a consequence, traditional TCP stacks deal with errors by drastically slowing down the rate at which they transmit data. In many wireless environments, an error in the transmission of a TCP packet can actually be caused by a transient RF issue. For such an error slowing down the overall rate of transmission is the opposite of what should be done; resending the packet as soon as possible is usually a better solution. If this specific adaptation of the TCP layer could be performed on a link type basis, a ULLA would enable the TCP adaptation for any type of links. Information provided by ULLA related to handoffs, network disconnection and packet loss could be used to differentiate congestion problems from true packet losses, and improve timer management in TCP (see, for example, **[4]**) and other protocols sensitive to jitter and other changes in network characteristics.

Adaptation to changing link characteristics can be effective at the application layer, too, as presented for example in **[6]** and **[7]**. In multimedia streaming scenarios, source encoding parameters can be dynamically varied, based on link statistics; PHY/MAC parameters can also be controlled accordingly. In **[2]**, a CLO example is presented that uses real-time transcoding in a Wireless LAN environment. Delay constrained applications (like wireless VoIP) may wish to enforce robustness by adding FEC streams (**[3]**), when the packet error rate goes above an acceptable level. Such applications could benefit from ULLA asynchronous notifications in order to delay their network requests until the required network characteristics are fulfilled (bandwidth, latency, etc.).

Another important advantage of using ULLA would be to enable CLOs in a more technology independent way. By providing a single API that works over different technologies, ULLA can enable CLOs to be defined without targeting a specific type of link. Hence the defined CLOs would be effective over all technologies that ULLA covers without requiring any modification.

Network-aware applications

Considerable research effort has emerged for creating frameworks for context-sensitive applications. Context sensitivity can be defined as the ability of a device or an application to adapt its behaviour or configuration to changes in its environment such as its network conditions or its location (logical or physical). However, current implementations of context-sensitivity have turned out to be extremely difficult, and it has even been argued that “generic artificial intelligence” is necessary to make the concept work **[5]**.

We see ULLA as being an enabling technology for building generic context management systems. It would enable the reception of information about the user context that the network(s) could provide through a unified interface, something that is impossible with present-day technologies. The type of context that ULLA could provide, may include location information (either in terms of absolute or relative coordinates in case of cellular systems or ultra-wideband links, or in logical terms using the information about the networks detected in the case of WLANs, for example), information about user mobility, and about other devices surrounding the user.

Connection Management

With the increased number of supported network interfaces, owners of mobile devices are confronted with the difficult task of deciding which network to use when they wish to transfer data. At present, due to the limitations in the IP stack, most mobile devices can only handle a single active connection at a time (per link). A Connection manager is typically understood as the agent responsible for deciding which network to use according to the current network conditions experienced by a device and the requests performed by users. However, due to the multitude and disparity of link layer APIs, existing connection managers operate on a very limited amount of information (essentially driven by profiles) and cannot easily adapt to new technologies. Moreover, the lack of a notification mechanism often forces them to rely on a costly solution such as regular polling. The existence of ULLA could enable the development of more intelligent connection management schemes based on information inferred from both the link layer and end-to-end connections. Using ULLA, the connection management implementation could also become independent of a particular link layer technology, partially transportable between operating systems and simplified thanks to the support for flexible asynchronous notifications.

As a proof of concept, a Connection Manager based on ULLA has been prototyped. At the lowest level of its architecture, ULLA provides dynamic access to link information as well as dynamic and configurable notifications. In between user applications and ULLA lies the Connection Manager with the following roles:

1. Handle and prioritize the connection requests coming from multiple applications.
2. Establish connections upon requests from applications, using the “best possible” path in the current networking environment. Application connection requests are composed of a destination network and a profile defining the minimal requirements that must be fulfilled by the requested connection. The CM uses the information about available network links reported by ULLA and static configuration information on the device (dial up, VPN configuration, etc.) to compute the possible path alternatives allowing the device to reach the requested destination while satisfying the profile. The CM selects the optimal path among the different alternatives based on predefined user preferences. Once a connection has been established the CM notifies the requesting application and uses the ULLA support to monitor the first hop of the established path (link used by the connection).
3. Monitor the “quality” of established connections. Every connection uses a first hop link. If the link characteristic exceeds the limits of the connection profile, the whole connection will also exceed the condition defined in the connection profile. The CM uses the support of ULLA to register notifications when such events occur. In such a case, the notification is processed by the Connection Manager and a CM notification is forwarded to the application that can decide to react accordingly.

4. Try to re-establish a new connection when a link disappears. The Connection Manager will also monitor the disappearance of the link involved in an established connection. In case a link is broken (due to loss of coverage for example), ULLA will report a notification to the CM. The CM can then proactively try to find an alternative path to the requested destination. In order to enable an application to react to such an event, a "ROAM" notification is sent to any application that uses that particular connection. This feature is optional and is decided by the application.

So far the experience has been positive. The code developed for the Connection Manager does not rely on any specific library to deal with link discovery and statistic collection. As a consequence, it could integrate a new type of wireless link without being modified. However, due to specificities in the configuration of IP, the code still has to differentiate in the way IP connections are created and handled. The current implementation of the Connection Manager will be extended to provide other more advanced features based on ULLA.

Mobility Support - Roaming and handover support

Currently there are many activities involving the definition of standards allowing for the transparent handover of a connection between different radio technologies (vertical handover). This is especially true for 802.11 and cellular networks. Several standards in this domain have been accepted or are currently under development.

Established in 1998, the 3rd Generation Partnership Project (3GPP) is a collaboration agreement between a various telecommunications standards bodies. Defined as part of 3GPP release 6, Unlicensed Mobile Access (UMA) defines a parallel radio access network known as a "UMA Network" that interacts with the mobile core network using the current mobile network standard interfaces. Through the introduction of a new UMA Network controller connected to the core network, traffic arriving from an IP network is actually relayed to the core network thus enabling mobile stations to transparently roam from wireless to standard network. For a seamless integration with existing mobile networks and unlicensed spectrum networks it is also needed to define a UMA-enabled handset with dual-mode operation capable to operate within both networks.

At a lower layer, the IEEE working group 802.21 (WG 802.21) is working on a standardization to enable handover and interoperability between heterogeneous network types including both 802 and non 802 networks. This standard focuses on the way session handoff takes place between different networks. It defines a set of link layer events that are used to gather relevant information about lower layer network conditions. These events are then used to ensure minimal disruption of service when the handoff takes place.

All these approaches strive towards the same goal: to provide a mechanism for transparent handover of connections. In all of these, a key attribute is the ability to prepare for the handover in advance by implementing a "*make before break*" policy. This allows for multiple network connections to slightly overlap in time to make sure that no connectivity gap occurs. This is especially true for vertical handover where different technologies are involved. Such policies require the ability to monitor links in order to detect on time that link quality is fading to the point that a new connection should be considered and configured. ULLA enables such a scenario. Its asynchronous notification model provides an easy way to monitor connections and detect problems in a link-agnostic way. Through link monitoring based on ULLA, some form of motion prediction could also be implemented in order to help with the decision-making process.

It is interesting to note that the 802.21 proposal defines similar concepts as the ones proposed in ULLA. The main difference between the 2 approaches is that the events defined in by the 802.21 WG are static while ULLA provides a language that is flexible enough to express any kind of network condition based on the supported link attributes. As a consequence, ULLA could actually be used to implement the 802.21 standard. Similarly, the UMA approaches will require an entity such as a connection manager that will monitor existing links and take decisions to roam and select the most adequate technology to use according to the set of available links.
At the network layer ULLA could be used to optimize mobility solutions by providing preliminary notifications when the link quality falls below a threshold, so that the handover process can be anticipated. This would help to reduce handoff latency.

Routing protocols

Sensor networks and wireless mesh networks have a similar need for routing protocols that can adapt to dynamically changing RF conditions. Indeed, relying on path metrics such as the number of hops often leads to poor performance since it does not take into account specific RF effects such as channel overlap or poor signal reception. To perform well, routing algorithms for wireless ad-hoc networks must consider more complex path metrics that will take into account the RF conditions of the different links involved in a path. In [8], the authors present a new path metric adapted to wireless mesh networks. This path metric, called Weighted Cumulative Expected Transmission time (WCETT), takes into account different aspects of links associated to a path such as bitrate, error rate, used channels and latency. These values are combined in such a way that paths with less overlapping channels get an advantage over paths with overlapping channels, while still taking into account the overall cost of the transmissions on every hop of the path. Obviously, ULLA would be an appropriate technology to simplify the process of gathering the information necessary to compute the path metrics. Instead of relying on proprietary and cumbersome interfaces, ULLA would provide a simple yet efficient interface. Additionally, the use of the ULLA would greatly simplify the extension of mesh networks to heterogeneous technologies since it would guarantee a uniform interface regardless of the underlying technology.

ULLA and Software Defined Radio (Cognitive Radio)

Software Defined Radio (SDR) is an emerging technology defined as a radio communication system whose channel modulation and demodulation wave form are defined in software. A SDR is a radio that can be programmed so that the physical and MAC layers can be loaded similarly to how a normal program is loaded onto a machine. The effect of SDR is that the same hardware can be used to access different types of wireless networks. For example, a device equipped with an SDR could be used as a GSM phone or to access a Wi-Fi network.

In 1999, Joseph Mitola extended SDR with the term ‘cognitive radio’ to define a SDR that is aware of its environment and can dynamically adapt by changing its frequency, coding or modulation scheme. The process of dynamic adaptation will allow devices to choose the type of radio spectrum that fits best their needs (bitrate, latency, cost, etc.) according to their current environment. Cognitive radio is an emerging solution to the problem of radio resource management. The currently allocated spectrum in the unmanaged license band is often overcrowded, leading to poor performance. Cognitive radio devices will be able to continuously sense the spectrum in order to identify unused frequency bands and thus provide the best RF channel to a mobile device in the face of a dynamically changing RF environment.

ULLA could be considered as a first step towards cognitive radio. Presently, most devices are not equipped with SDR but instead embed multiple radios. By providing a uniform API to access these different radios, ULLA could enable cognitive algorithms to be deployed on existing standard radio networks. This would enable intelligent algorithms to be defined and deployed in the near term without the need for SDR. A classical example used when talking about cognitive radio is the ability to infer from past experience that a network will soon be available. With cognitive radio, this can be realized by constantly sensing the RF spectrum and maintaining historical information. Similarly, a ULLA enabled device could monitor the availability of links during the hours of a day in order to build a time map of the available links. Such a map can then be used to implement prediction algorithms that allow devices to defer network accesses until the foreseen connection is available. Such algorithms will be easily extended to deal with SDR and allow not only different types of links but the complete baseband stack to be modified.

ULLA architecture

The Unified Link Layer API may be implemented differently on various platforms, but the general SW architecture includes a library to be linked to by applications, an ULLA core and a set of Link Layer Adapters that wrap legacy drivers, as shown in Figure 1.

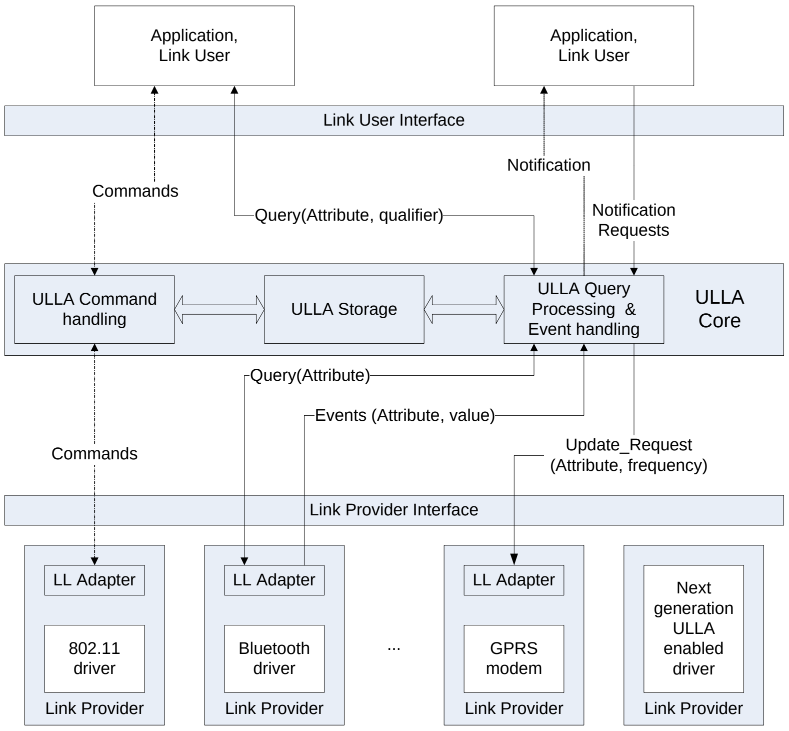


Figure 1: ULLA architecture block diagram

The ULLA library, which may be used by multiple applications at the same time, interfaces to the ULLA core in a platform dependent way. The ULLA core includes Command Processing, Event Processing and Query Processing blocks. A Database back-end may be used to store link related information according to the GOLLUM schema, also referred to as ULLA Storage. Not shown in Figure 1, a Link Manager (LM) block is responsible for handling potential conflicts among multiple LUs according to pre-defined policies. A suitable interface is defined to allow the insertion of 3rd party Link Managers.

Link Layer Adapters (LLAs) are SW modules that are loaded by the ULLA Core in a platform dependent way. These blocks are responsible for translating ULLA commands into driver specific methods as well as exporting driver specific events towards the ULLA core. LLAs also fill the link and link provider tables in the ULLA Storage by properly manipulating the proprietary statistics exported by the driver.

Abstracting links to deal with different technologies

The interface provided by ULLA is fixed, in the sense that the operational signatures described are not expected to change often. The mechanisms used to deal with different technologies and to keep ULLA “future proof" rely on the flexibility of the data structures passed via the interfaces.

To describe links (and other supported abstractions like link providers), ULLA uses the notion of class following an object-oriented approach. A class describes a set of attribute names and types as well as a set of commands. The highest abstraction level is offered by three base classes, *ullaLink*, *qosLink*, and *securityLink* for links, and one, *ullaLinkProvider*, for link providers. These classes have been defined in such a way that they can apply to different kinds of technology. To guarantee uniformity in the API, all LLAs are required to support the mandatory base classes *ullaLink* and *ullaLinkProvider* as they offer an abstract, technology-independent view of links and LPs. The additional *securityLink* base class unifies common attributes and commands in the security area. For instance, several technologies do not support link layer security, so the implementation of the respective class is not mandatory but recommended for the links that can provide such a feature. Other technology-specific attributes and commands can be offered in additional classes offering lower level of abstraction.

In the following we use a hierarchy of wireless LAN –related classes to illustrate this approach, even though similar considerations are valid for other wireless technologies. Attributes and commands that have meaning only within the IEEE 802.11 family of technologies are offered via the *80211Link* and *80211LinkProvider* classes. Together they offer the functionality to query or modify 802.11 specific attributes (RTS/CTS threshold value, RSSI, etc.). Further attributes and commands that have meaning only within the different 802.11 extensions, such as IEEE 802.11k for radio resource measurements, are then exposed via additional classes (*80211kLink* and *80211kLinkProvider*). Vendors providing the LLA to support a particular wireless interface are free to choose which optional classes they support, and can even offer their own classes to access implementation-specific functionalities that have meaning only for the particular vendor’s products.

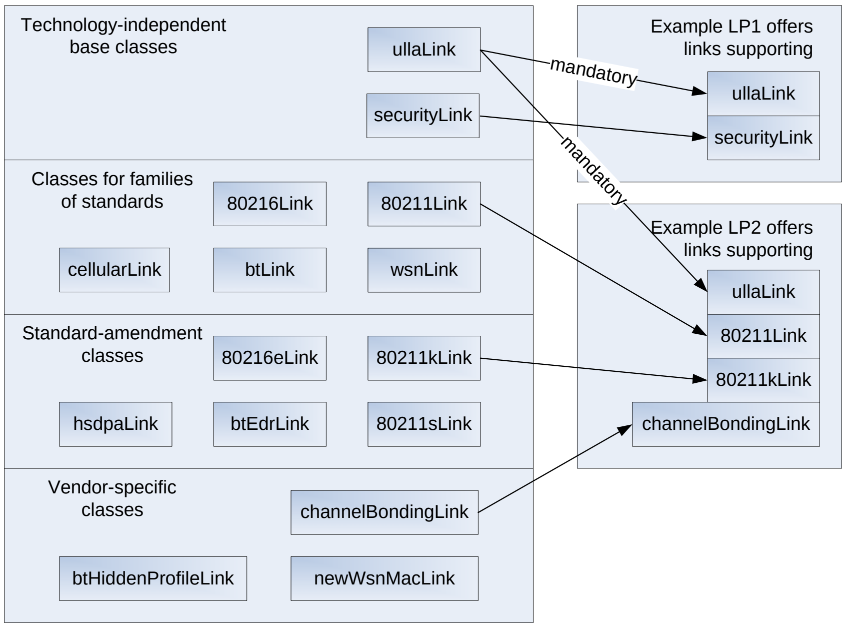


Figure 2: ULLA class hierarchy

This approach, illustrated for the link classes of Figure 2, has several advantages. First, it offers the presently lacking abstract view over different links and link providers. Applications for which this level of abstraction is sufficient can always operate on the base classes, without caring about the possible extensions supported. More technology-aware applications, such as various diagnostic tools, can then choose a lower level of abstraction, and still maintain portability across different technology vendors. Since the link provider is responsible for informing the ULLA core about the classes it supports, including their contents, new base classes can be added without a need for updating the implementation of the ULLA core.

This mechanism allows the core implementation to remain untouched even though new wireless technologies continue to appear. The functionality offered towards the link user should also retain its validity in the foreseeable future.

ULLA usage example

In this section some examples of typical ULLA use cases are presented from an application programmer's point of view.

Using ULLA

In order to use ULLA, an application needs to register itself with the ULLA core.

```
#include <ullalu.h>

LuDescr_t foo;
ullaResultCode err;

foo.name = “bar”;
foo.description = “testing LU registration”;
foo.version = “1.0”;
foo.profile = ULLA_PROFILE_BASE;

/* register with ULLA core */
err = ullaRegisterLu(foo, ULLA_ROLE_STD_LU);
```

Table 1: LU registration

When registering, the Link User specifies its role. In the example above it is a standard link user, with no special privileges. Other roles include the Connection Manager and the Link Manager, for which OS-specific authorization mechanisms may be used inside the ULLA core.

Knowing which links are available

A mobile terminal that wants to know which wireless links are available for communication would use ULLA as shown below.

```
ullaResult_t res;
ullaResultCode err;
int i, value;

char *query = “SELECT linkId, networkName, txBitrate \
             FROM ullaLink”;

/* sends query to ULLA core, synchronous call */
err = ullaRequestInfo(query, &res);
/* here we parse the query result and print out results */
while (ullaResultNextTuple(res) != ULLA_NO_MORE_TUPLES_ERROR) {
      /* do something with query result */

}
```

Table 2: ULLA query

In this step, the LU may also retrieve information related to the authentication mechanisms needed to use the wireless link.

Connecting to a specific link

When the LU wants to establish a layer-2 connection, it sends a connect command to ULLA as shown below.

```
CmdDescr_t my_cmd;
ullaResultCode err;
int linkId;
char errString[LEN];

my_cmd.id = linkId;
my_cmd.class = “ullaLink”;
my_cmd.cmd = “connect”;

/* fire the command with 2 seconds timeout*/
err = ullaDoCmd(&my_cmd, 2000);
/* error checking */
if (err != ULLA_OK ) {
       /* get the reason for connection failure */
       err = ullaGetErrorString(errString, LEN);
       printf(“Error: %s\n”, errString);
}
```

Table 3: ULLA command example

In this example, a synchronous command is activated (ullaDoCmd) with a timeout of two seconds. If the command fails, the reason can be found by using the *ullaGetErrorString* function.

It should be noted that the connect command refers to bringing up the link at the layer 2 of the protocol stack, i.e. IP configuration has still to be performed. Such an action is typically the responsibility of a connection manager.

Handover management

ULLA can be used to trigger handovers among various wireless links. The code example below shows how this can be done.

```
RnDescr_t my_notif;
RnId_t linkDown, linkUp;
handleNotification_t my_handler;
ullaResultCode err;
/* link down condition definition */
char *down_cond = \
       “SELECT linkId FROM ullaLink WHERE rxQuality < 20”;
/* link up condition definition */
char *up_cond = \
       “SELECT linkId FROM ullaLink WHERE state = ‘registering’”;

/* the notification handler routine called by the ULLA core */
void *my_handler(RnId_t rnId, ullaResult_t res, void *privdata) {
       /* parse the notification and handle it */
       /* … */
}
my_notif.count = 1;
my_notif.period = 0;
my_notif.handler = my_handler;
my_notif.privdata = NULL;
/* register our notification handler and our trigger conditions */
err = ullaRequestNotification(&linkDown, down_cond, &my_notif);
err = ullaRequestNotification(&linkUp, up_cond, &my_notif);
```

Table 4: Handover support in ULLA

In this example, an “up” condition and a “down” condition are defined as UQL[2] strings, which can be used to trigger a handover. Such conditions correspond to a new wireless link being discovered and to a degrading wireless link, respectively. The two conditions are passed to the ULLA core by means of the *ullaRequestNotification* function, where the LU-implemented callback handler is passed as an argument. Whenever one condition is met, the LU handler is invoked by the ULLA core in an asynchronous way.

In addition to the described event-based notifications, which are fired upon occurrence of pre-defined conditions, ULLA supports periodic notifications. The LU simply provides a period unequal to zero and ULLA will invoke the callback function periodically. The number of notifications to be fired is specified using the count-variable. Periodic notifications can be used to monitor certain characteristics, e.g., to perform historical analysis later on.

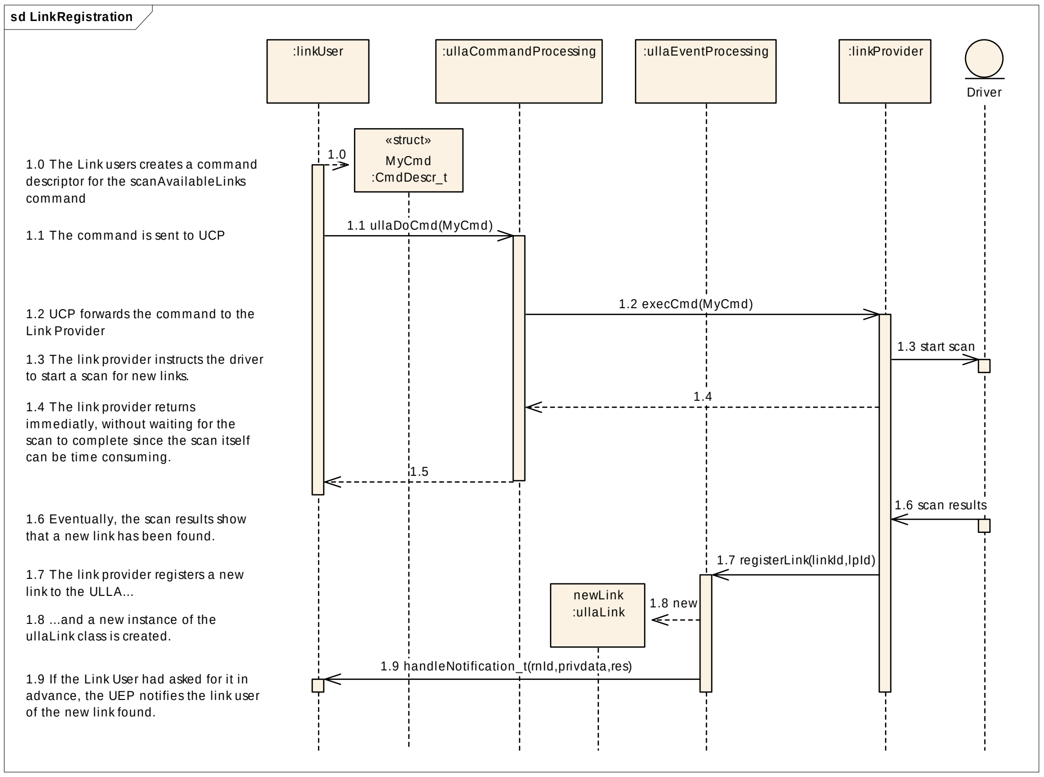
[2] The ULLA Query Language, or UQL for short, is used to express queries, which can also be used when registering notifications and is a subset of the well-known Structured Query Language (SQL).

Integrating a new wireless driver with ULLA

As seen in the "ULLA architecture" section, existing drivers can be wrapped through a Link Layer Adapter that is loaded by the ULLA core.
The LLA is responsible for:

- registering one or more LinkProvider classes in the ULLA storage;
- executing commands given by the ULLA core using legacy driver calls;
- maintaining and updating ullaLink and LinkProvider classes with the relevant attributes, which may be calculated using driver specific calls;
- implementing a reflection interface that can be used either by the core or by the link user to know which classes are supported by the LLA.

The following sequence diagram shows the process of scanning for new wireless links and the consequent ullaLink object creation performed by the link provider.



1.0 The Link users creates a command descriptor for the scanAvailableLinks command

1.1 The command is sent to UCP

1.2 UCP forwards the command to the Link Provider

1.3 The link provider instructs the driver to start a scan for new links.

1.4 The link provider returns immediatly, without waiting for the scan to complete since the scan itself can be time consuming.

1.6 Eventually, the scan results show that a new link has been found.

1.7 The link provider registers a new link to the ULLA...

1.8 ...and a new instance of the ullaLink class is created.

1.9 If the Link User had asked for it in advance, the UEP notifies the link user of the new link found.

Figure 3: Scan for available links.

It can be noticed that the link provider can execute commands asynchronously.
It notifies the link user when new links are found, after creating and populating the corresponding ullaLink entry in the ULLA storage.

Conclusion

Accessing link layer information in a uniform and simple way will increasingly become an important feature if applications are to dynamically react to varying network conditions. In this paper we have presented a Unified Link Layer API that addresses the issues related to existing link layer APIs, mainly their non-uniformity, complexity and lack of asynchronous notification mechanisms. We detailed a set of practical scenarios that would benefit from ULLA. Not only will such an API simplify the access to link layer information and enable heterogeneous networks to be automatically supported, it will also facilitate the integration of future radio technologies. The integration of such an API in mobile OSes would stimulate the creation of applications providing enhanced user experience by exploiting more efficiently the diversity of available wireless and wired links.

ULLA has been prototyped by the GOLLUM consortium using Linux, WindowsCE and TinyOS on mobile phone, PDA and sensor node platforms, respectively. Its usability has been assessed on a number of applications, including adaptive Multimedia streaming, smart Connection Management and flexible routing.
Link Layer Adapters have been developed for GPRS/UMTS, Wireless LAN, Bluetooth and ZigBee, proving that existing technology can be easily accommodated in the ULLA architecture. On top of this, an API guidebook is being prepared that shows how the API can be used in detail and how future wireless standards can be added. It is foreseen that ULLA can also be an enabler for emerging research areas like Cognitive Radio.

Further information about ULLA can be found at http://www.ist-gollum.org.

References

[1] M. Zorzi and R. Rao, “Perspectives on the Impact of Error Statistics on Protocols for Wireless Networks”, IEEE Personal Communications, vol. 6, pp.32-40, Oct. 1999.

[2] G. Convertino, D. Melpignano, E. Piccinelli, F. Rovati, F. Sigona, “Wireless Adaptive Video Streaming By Real-time Channel Estimation And Video Transcoding”, in *Proc. IEEE Inter. Conf. on Consumer Electronics*, Jan 2005.

[3] J. Rosemberg, H. Schulzrinne, “An RTP Payload Format for Generic Forward Error Correction”, *IETF RFC2733*, Dec. 1999.

[4] W. T. Raisinghani, A. Kr. Singh and S. Iyer, “Improving TCP performance over Mobile Wireless Environments using Cross-Layer Feedback”, in *Proc. ICPWC-2002*, New Delhi, 2002, pp.81-85.

[5] T. Erickson, “Some Problems with the Notion of Context-Aware Computing,” ACM Commun., vol. 45, no. 2, Feb. 2002, pp. 102–04.

[6] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker, “Agile Application-Aware Adaptation for Mobility”, *in Proc ACM Symposium on Operating Systems Principles*, Saint Malo, France, Oct 1997.

[7] J. Sun , J. Tenhunen and J. Sauvola, “CME: a middleware architecture for network-aware adaptive applications”, In *Proc. 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Beijing, China, 1:839 – 843

[8] R. Draves and J. Padhye and B. Zill ,“The architecture of the Link Quality Source Routing Protocol”, MSR Technical Report MST-TR-2004-57.

[9] R. Draves and J. Padhye and B. Zill, "Routing in Multi-radio, Multi-hop Wireless Mesh Networks". Proc. of ACM MobiCom 2004, Philadelphia, PA, Septembre 2004.

[10] IEEE 802.21 Working Group. **http://www.ieee802.org/21/**