ICT-MobileSummit 2009 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2009
ISBN: 978-1-905824-12-0

# A Case for Generic Interfaces in Cognitive Radio Networks

Vinay Kolar[1], Petri Mähönen[1], Marina Petrova[1], Mahesh Sooriyabandara[2], Janne Riihiärvi[1], Tim Farnham[2]

[1]*Department of Wireless Networks, RWTH Aachen University,*
*Kackertstrasse 9, 52072, Aachen, Germany*
*Tel: +49 2407 575 7032, Fax: +49 2407 575 7050,*
*e-mail: {vko,pma,mpe,jar}@mobnets.rwth-aachen.de*

[2]*Telecommunications Research Laboratory,*
*Toshiba Research Europe Limited,*
*32, Queen Square, Bristol BS1 4ND, UK*
*Tel: +44 1179069830, Fax: +44 1179060701*
*e-mail: {mahesh, tim}@toshiba-trel.com*

**Abstract:** Cognitive wireless networks are envisioned as a solution for intelligent ubiquitous networks that are capable of adapting to the dynamic environment through programmable radio devices. Each functional module and protocol accomplishes such learning by observing the various environmental, networking and application related parameters and adapts its behaviour to improve the performance. While access to a variety of parameters provide a large margin to improve the performance, it hinders the practical use of the system by restricting portability. In this paper, we argue that standard generic interfaces can be effectively used to overcome the above problem and enable seamless communication between modules at different layers and radio technologies. We examine the various parts of a cognitive wireless network, categorise different types of generic interfaces, analyse the suitable abstractions, and propose architectural principles for a cognitive radio system. We discuss four such categories of generic interfaces that abstract different layers of networking, application utility and policy enforcement. Various business benefits arising from such standard generic interfaces are briefly discussed. We believe that the proposed architecture and the benefits greatly enhance a feasible realisation of cognitive wireless networks.

**Keywords:** Generic Interfaces, Cognitive Radio, Cognitive Wireless Networks

## 1. Introduction

Cognitive radios and cognitive wireless networks [1, 2, 3, 4] are envisioned as solutions to enable ubiquitous networks that are capable of learning from the environment, adapting to the dynamic environment and, thus, increase the system performance. Dynamic Spectrum Access (DSA) techniques in the context of cognitive radios are expected to increase the efficient use of scarce spectrum. The core of such systems involves: (a) devices that are capable of communicating across different networking technologies; and (b) a set of modules that provide the capability for learning and adaptation. A key feature of cognitive radios is that devices are reconfigurable so that they allow re-programming the radio behaviour in real-time. Note that the device is not necessarily terminal equipment, but can also be other equipment such as base station or access point towards infrastructure. Naturally software defined radio is particularly suitable platform for implementation of such devices.

 

Most of the research projects and practitioners are also positioning a cross-layer optimisation to the centre of such devices. It is expected that cognitive radio devices could provide efficient use of cross-layer optimisation to enhance their operational efficiency. Cross-layered methods and modules are particularly promising as they could unleash the full use of information that resides at the various levels of protocol stack.

While cross-layered optimisation in cognitive radio networks has gathered a lot of attention (e.g., [5]), much less emphasis has been given to solving practical problems that still block the advancement towards Cognitive Radios (CR). One of the main challenges to cope is *heterogeneity*: there exists a large number of different proprietary radio interfaces and protocol entities – even in the case where functionality of building blocks of the radio systems are very similar. As a result, wider deployment of CR prototypes has been very slow; and sometimes completely blocked, due to fact that although the functionality of each block may be exactly the same, often the interfaces and APIs (Application Programming Interfaces) are highly different and proprietary. In many cases, the interfaces are not made openly available for any third parties. The difference in the proprietary interfaces often means that one is required to rewrite large parts of the code when moving from platform to another. A good example is the current situation with WiFi (IEEE 802.11) network interface cards where access to certain simple but essential information is not often available. The APIs and interfaces are highly proprietary, and only very rudimentary information is made available to programmers.

We have worked during the last couple of years in various EU projects, especially ARAGORN [6] and GOLLUM [7], on developing technologies which provide standard generic interfaces. These technologies will enable seamless communication between protocol and hardware entities at different layers and enhance the portability. In addition to providing portability, such decoupled modules shorten development cycle and ease the implementation overhead of the developer. In this paper, we discuss the need for such generic interfaces and describe shortly our initial high-level architecture of the generic interfaces that are designed for CR/SDR systems.

## 2. Background and Principles

Providing generalised access to various information, possibly across different layers and protocol entities, is naturally a key requirement for efficient system that provides cognitive resource optimisation capability. Standardised interfaces are not, of course, new idea, and any modern radio interface and system standard includes well defined interfaces. However, typically many of the management functionalities inside of the devices and between some elements are typically proprietary. Especially APIs have not been standardised, and apart of OBSAI (Open Base Station Architecture Initiative) [8] there has not been much activity in this domain. This is understandable since the proprietary approach has been until now driving many business models. Recently discussion towards CR and SDR interface definition has been started by various standardisation groups most notably by IEEE SCC41/P1900 and ETSI. SDR Forum has also been an early player in this domain [9].

Our approach is very much complementary to the above ones, since we are working on providing the basic mechanisms for general information exchange in cognitive wireless systems, instead of focussing on interfaces for a particular type of radio device or

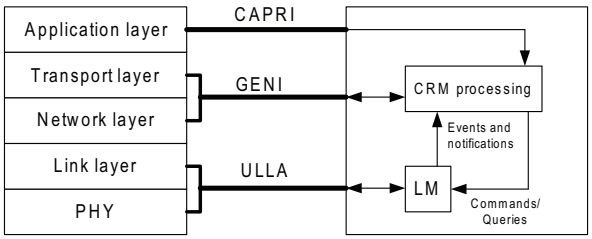


Figure 1: A basic positioning of interfaces and Cognitive Resource Manager (CRM).

network layer. The desired core properties for our API are that they should be: (a) generic, i.e. platform/technology independent; (b) as transparent as possible; and (c) extensible to support future technologies and platforms. Our two key goals are to hide away different proprietary access APIs and interfaces, and to provide an interface to describe the goals of application (users).

## 3. Interfaces and Basic Architecture

There are three main categories of interfaces we have considered. This list is naturally not exhaustive, but each of the considered categories are archetypical and can be seen as a starting point towards other similar interfaces. We are working on developing following three interfaces:

- Universal Link Layer Interface (ULLA) facilitates the interaction with the link layer. In some sense one can see ULLA as a sort of abstraction layer, with the difference that this mechanism is extensible, provides a specific API and developed specifically to support SDR/CR operations.
- Generic Network Interface (GENI) on the other hand enables the detailed monitoring and configuration of the transport and network layers. Much like ULLA, it provides these functionalities through generic and portable APIs.
- Common Application Program Requirement Interface (CAPRI) provides an interface between applications and cognitive radio optimiser. It is a mechanism for applications and users to define their goals and objectives in a quantifiable manner.

Furthermore as part of our architecture we have defined a Common Control Channel (CCC), which provides support for the cognitive radios especially in the context of spectrum management. The characteristics and the functionalities of the CCC are beyond the scope of this paper, and that work is more closely related to exiting state of the art that is done by SCC41/P1900.4.1 and IEEE 802.11k committees.

In Figure 1, we show the functional location of the interfaces towards the classical protocol stack. The CRM processing unit is a part of so called Cognitive Resource Manager (CRM), which is responsible for the main coordination of cognitive radio resources and thus utilises the defined APIs and interfaces for information exchange [10, 11].

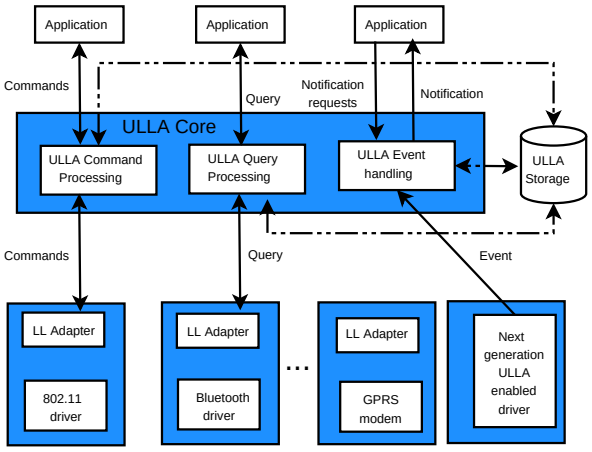


Figure 2: The basic components of ULLA architecture.

## 4. ULLA

The role of the Universal Link Layer API is to expose radio link level behaviours to the CRM in order to enable intelligent resource management though cross-layer optimisation. For this purpose, a number of functional and non-functional requirements for the Generic Link Layer API have already been identified [7]. For instance, statistics querying, event-notifications, link/radio configuration are several key services expected from this interface. ULLA is a technology-independent link layer API that is developed to solve the complexity and interoperability problems related to the large number of different APIs and methods for accessing heterogeneous communication interfaces. It abstracts radio and link layer specific details and provides a *generic querying mechanism* to enable management of radio links in a flexible manner. These features of ULLA would enable the CRM to gather information and update settings of the various links connected to the interface in a unified manner.

The ULLA API facilitates access to link-layer functionality and information in a technology independent manner. It provides an abstraction from specific link technologies to the applications or other Link Users (LUs) by regarding a link to be generic means of providing a communication service. In this context, links are made available and configured through Link Providers (LPs) to permit abstraction from specific platforms and technologies. Link users that benefit from ULLA services include, but are not limited to, any higher layer protocols, middleware or application software.

Figure 2 shows the different components of the ULLA. The *ULLA core* consists of three main components. The *UllaQueryProcessing* module is in charge of analysing the queries and notification requests coming from Link Users. The *UllaCommandProcessing* module handles commands and forwards them to the corresponding Link Provider. The *UllaEventProcessing* module takes care of handling events arriving from the Link providers (new link arrivals, new value for a link characteristic, etc.), in particular, for the evaluation of registered notification requests. Finally the *ULLA Storage*, represented outside of the ULLA Core, is an optional component used to cache link characteristics collected from the Links and Link Providers in order to avoid access to the drivers or hardware for each query. The main services provided through the ULLA are:

- Queries: ULLA provides a generic querying mechanism allowing applications to

retrieve link information in a technologically independent fashion. A query language called ULLA Query Language (UQL), a subset of SQL, is used.

- Commands: ULLA supports a mechanism that allows applications to configure and manage links in a standard way through the use of commands than can be called from user level applications.

- Events: ULLA provides support for asynchronous notifications based on user defined link criteria. For example, it is trivial with the ULLA to enable a notification when specific link received signal strength goes under a certain threshold.

It should be noted that ULLA itself does not provide a large static API towards links. Instead, it provides a very small number of functional API commands. The feature and attribute richness of the interface is provided through the UQL, and commands are given as parameters for those few API calls. The ULLA query language is used to request information from the ULLA and request notifications, i.e. to indicate an event that should trigger an asynchronous notification. The present reference implementation of UQL is a subset of SQL with minor semantic differences and a few additions to SQL. For example, the basic keywords used by UQL are:

- SELECT: used to select what attributes should be returned by the query. Select should be followed by a list of attributes defined as attribute in the class selected by the FROM statement.

- FROM: specifies the class that should be used. UQL will limit the number of class accessed in a query to a single class. Most of the time, the $ULLA\_Link$ class will be specified. Queries targeting specific links could use derived classes though.

- WHERE: filters the responses returned. The WHERE statement is followed by a clause specifying a Boolean condition, (e.g: bandwidth $>$ 10 Mbits/s). Multiple clauses can be concatenated with the AND keyword.

Moreover we provide basic logical operations AND, OR, and NOT. Other aggregator functions could be used on the returned attributes to simplify the application work, e.g. AVG (compute average), MAX (compute maximum), MIN (compute minimum), SUM (compute sum), COUNT (returns the number of attributes returned).

In addition, ULLA defines a common information model to integrate diverse information coming from heterogeneous radio interfaces available on a device. This simplifies common management of radio/link resources. Link-aware applications (or Link Users) can obtain clear and consistent information about performance, configuration and capabilities of communication resources through this data model.

## 5. Extending beyond ULLA with GENI

The key novelty of our approach has been to build a very lightweight system that implements query engine and language, which allows us to build and extend many different kind of interfaces and APIs. As the main command structure, parser and query engine stay unchanged those can be reused and the syntax of interface programming stays unchanged. Thus for example GENI that provides data models and interface towards network and transport layers reuses same core infrastructure and components defined

in ULLA specification. In other words, GENI simply extends ULLA with new providers (service) and information (class) definitions for defining the notion of transport layer connections and network layer routing information. Instead of traditional Link Layer Adapters, GENI introduces Transport and Network layer adapters to interface with the TCP/IP stack of the host operating system. For the transport layer, a new provider is defined for UDP and TCP assuming for now that these are the only transport layer protocols we consider for implementation.

## 6. CAPRI

Cognitive radios need to be aware of different application and user goals. In order to make multi-objective optimisation, we need to provide mechanism to handle objectives quantitatively. One very natural way is to use utility functions (objective functions) for modeling these goals. Due to this CAPRI interface need to provide slightly different and richer set of functionalities than other interfaces that have been discussed above. Fundamentally, CAPRI is designed to support utility-based optimisation [12, 13, 14].

Different applications can have widely varying requirements in terms of network connectivity. For interactive web-browsing bandwidth is the most important quantity provided that the delay does not become prohibitive, whereas for VoIP low (residual) packet error rates along with low and stable delay are more important. Streaming media applications in general feature a complicated throughput dependant behaviour in the quality of the streamed media due to the use of certain fixed codec rates. Adding the complication of multiple applications sharing a connection over a link or a path, it becomes clear that for optimisation of the system performance quantitative expression of these application interests is needed.

A flexible solution to the problem is obtained through the application of utility functions. We associate each application with a function $U$, evaluated on the collection of the various measurable attributes of the connection (such as throughput, delay, error rate etc.), resulting in a single real number, the utility of the connection for the application. In simple cases the utility functions might be functions of a single attribute only, but we foresee them more typically as being dependent on a number of them.

The CAPRI is fundamentally based on the capability of expressing application preferences through it. Thus the basic interface will be of the form: "*attach_utility*(utility specification, application ID)". If such a function is invoked by the application itself, the latter argument can, of course, be omitted. It is included to allow dedicated operating system components to associate utilities to applications, allowing legacy applications and other software components not supporting CAPRI directly to be integrated seamlessly into the same optimisation framework. The semantics of *application ID* will, in general, be operating system specific, although we expect simple process number or PID to be the most common choice. The argument *utility specification* is a string containing the description of the utility function in a language basically forming a subset of the textual notation for mathematical expressions typically used in modern computer algebra systems. This part of the definition work is still on-going part of our research.

We cannot, of course, expect ordinary users to operate on utility specifications directly. They are meant to represent the lowest, most expressive layer of ways to express application preferences, used by developers with necessary expertise or domain knowledge (on behaviour of multimedia codecs, for example). We expect that on top of

the basic CAPRI a layer of macro-like behaviours will be developed to further abstract the most common types of utility functions. Additionally, we expect various deployed systems to come up with additional mechanisms for inferring user satisfaction to the application behaviour by means of various HCI techniques. Utility functions form in each of these cases the basis of expressing the desired behaviour to the CRM where it can be reasoned about automatically, but will largely remain hidden from users and even most developers.

## 7. Discussion and Conclusions

We have already defined most of the core part of the architecture and interfaces. Moreover, there is an early reference implementation for ULLA and related query engine components. Our current measurements show that the system can be implemented with low usage of resources and thus is suitable also for embedded devices. The performance evaluation has shown that interface is able to operate in real-time fashion so that events notification and information exchange can be performed at line-speed. Part of the work has been also presented towards standardisation groups and has been made available to public [15].

We have also paid a lot of attention to understand underlying commercial consequences of such interfaces. As discussed above the general API approach has a strong potential not only to lower development costs in typical industrial environment, but also enables new business opportunities by opening possibilities for different stakeholders in flexible fashion. It should be also noted that the flexible interface technology what we are offering is not a large monolithic approach, which would force stakeholders (e.g. equipment or chipset manufactures) to open *all* internal interfaces and functionalities towards everyone. On the contrary, our approach allows general access towards proprietary functionalities, which still can be kept proprietary and confidential. Naturally standardisation may require that some specific and very generic functionalities must be provided, e.g. reading for received signal strength, in a predefined manner.

We believe that the outlined query engine based information exchange and interfaces API approach is providing not only a future proof way to implement extendable APIs for cognitive radios and software defined radios, but also takes into account business realities in a fashion that has not been considered by previous attempts. We are currently continuing our work towards making a minimal reference implementation that includes all the major components of generic APIs, and plan to make performance evaluation by using a low-cost CR/SDR platform.

## Acknowledgements

This work was financially supported by European Union (ARAGORN project). We acknowledge also the partial support from DFG and RWTH Aachen through UMIC-research center facility. We thank other project partners for their valuable discussions and feedback.

## References

[1] J. Mitola and J. Maguire, G.Q., "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, pp. 13–18, Aug 1999.

[2] D. Clark, G. Partridge, J. C. Ramming, , and J. T. Wroclawski, "A knowledge plane for the internet," in *Proc. of SIGCOMM 2003, Karlsruhe, Germany*, 2003.

[3] P. Mähönen, "Cognitive trends in making: Future of networks," in *Proceedings of IEEE PIMRC 2004, Barcelona, Spain*, vol. 2, pp. 1449–1454, 2004.

[4] R. W. Thomas, L. A. DaSilva, , and A. B. MacKenzie, "Cognitive Networks," in *Proc. of IEEE DySPAN 2005, November 2005*, pp. 352–360, 2005.

[5] A. de Baynast, P. Mähönen, and M. Petrova, "ARQ-based cross-layer optimization for wireless multicarrier transmission on cognitive radio networks," *Computer networks*, vol. 52, pp. 778–794, March. 2008.

[6] *The ARAGORN Project website*, (`http://www.ict-aragorn.eu`).

[7] *The GOLLUM Project website*, (`http://www.ist-gollum.org`).

[8] *OBSAI-Open Base Station Architecture Initiative*, (`http://www.obsai.org`).

[9] *Software Defined Radio Forum*, (`http://www.sdrforum.org`).

[10] P. Mähönen, M. Petrova, J. Riihijärvi, and M. Wellens, "Cognitive Wireless Networks:your network just became a teenager," in *Proc. of IEEE INFOCOM 2006 (poster sessions), Barcelona, Spain*, 2006.

[11] M. Petrova and P. Mähönen, *Cognitive Resource Manager: A cross-layer architecture for implementing Cognitive Radio Networks*. in Cognitive Wireless Networks (eds: Fitzek F. and Katz M.), Springer, 2007.

[12] J. Riihijärvi, M. Wellens, and P. Mähönen, "Link-Layer Abstractions for Utility-Based Optimization in Cognitive Wireless Networks," in *Proceedings of CROWNCOM'06*, (Mykonos, Greece), June 2006.

[13] Z. Cao and E. W. Zegura, "Utility max-min: An application-oriented bandwidth allocation scheme," in *Proc. of IEEE INFOCOM'99*, pp. 793–801, 1999.

[14] T. Harks and T. Poschwatta, "Utility fair congestion control for real-time traffic," in *Proc. of 8th IEEE Global Internet Symposium, co-located with IEEE INFOCOM*, (Miami, FL, USA), pp. 85–90, March 2005.

[15] "Unified Link Layer API (ULLA): Open-source project."
Available at `http://sourceforge.net/projects/ulla/`.