RECONFIGURATION IN SOFTWARE RADIO SYSTEMS

Apostolos A. Kountouris, Christophe Moy

Mitsubishi Electric, ITE-TCL
Rennes, FRANCE

***Abstract*--Reconfiguration, dynamic or static, partial or complete is an essential part of software radio technology. Thanks to it, systems can be designed for change and evolution. In a sense "change" becomes part of the mainstream system operation. In this paper issues relating to the required device-level support will be covered. Device level support implies appropriate hardware and software architectures as well as design approaches accounting from one hand for the supported reconfiguration scenarios and from the other for the device specific constraints. We have opted for a customized and thus lightweight component-based approach using as guidelines a typical software-upgrade scenario for "bug-fixing" and device performance enhancement.**

I. INTRODUCTION

Reconfiguration, dynamic/static, partial/complete is an essential part of software radio technology [1]. Thanks to it systems are designed for change and evolution. In other words "*change*" becomes part of the mainstream system operation. Recent work in European Union R&D projects (i.e. the TRUST, CAST projects), the SDR Forum and lately WWRF, clearly shows that the concept of reconfiguration especially in the context of mobile cellular networks is a complicated business. Reconfiguration still raises questions on the required *system-level support* both at the reconfigured devices and at the network side [2].

Over the past decade previous work has concretely demonstrated the technical feasibility of the *Software Defined Radio* approach in the design of radio communications equipment. The project *SpeakEasy* was a turning point. This previous research and experimentation has resulted in a deep understanding of the SWR technology and its potential applications. Initially effort has mainly focused on issues relating to the area of *increasingly software implementation*. The advantages of this approach are numerous. In addition, the flexibility of increasingly software implementations offers potential advantages especially when the radio equipment is considered as part of a network as is the case in cellular mobile radio networks. This potential can be concretely exploited through equipment reconfiguration. In the evolution path towards and beyond 4G this potential flexibility can be useful in many technically challenging as well as commercially attractive use scenarios. Moving towards more *cognitive* (i.e. intelligent) radios [3] the WWRF vision indicates that in 4G SWR technology will play a key role. This is because in future networks (or network of networks) access *transparency for the user*, *service quality* and *network management optimization* will necessitate to consider reconfiguration as part of the mainstream system operation.

In this paper we shall attempt to analyze and discuss the software radio issues relating to reconfiguration and more specifically the required device-level support. The rest of the paper is organized as follows. First a brief overview work related to reconfiguration will be given. Next, the hardware platform we use for experimentation and prototyping will be described. On top of this reconfigurable architecture a *component* based design approach is employed to develop the needed reconfiguration mechanisms. Our approach can be tailored to handle various types reconfiguration scenarios necessitating more or less network involvement. Several case studies undertaken by our laboratory will also be described. Finally, some conclusions will be drawn.

II. PREVIOUS WORK AND A PLAUSIBLE ROADMAP

In this section a rapid overview of software radio literature related to reconfiguration is given. Space limitations do not permit to be more exhaustive in our review.

A. Previous Work

[4] gives an overview of Japanese R&D in SWR. For reconfiguration the main application targets is multi-mode and multi-service operation as well as remote upgrades for *performance enhancement* and *bug-fixing*. Depending on the reconfiguration type, reconfigurability has to be designed in both the physical and higher layers. The interest in Japan for SWR stems from the assessment that in the 4G era 3G will coexist with 4G to ensure coverage in urban and rural areas respectively. Users will require greater transparency for access as well as greater service integration; this is the single terminal trend. In addition the different backbone networks should be able to cooperate. The final goal is, thanks to the reconfiguration capabilities, to offer the user what is called in the paper *best* communication; best in terms of quality, price, coverage.

In [5] a scheme for a *parameter controlled* reconfiguration and a prototype is presented. This scheme targets multi-mode and by extension multi-service terminal operation based on a common hardware platform. The baseband

functional blocks in the transceiver chain are created so that common aspects in the different modes are factored by functional block *parametrization*. Some parameters control the specific ways that blocks can be connected or by-passed. In a sense the software architecture is static since the software for all modes is resident at all times. The mode of operation is selected by downloading the specific parameter values for each air-interface. Though flexibility is constrained, this approach by virtue of its simplicity is robust, reliable and can provide for fast mode switching.

In [6], [7] the terminal design approach of project CAST is presented; it intends to give terminals more flexibility by making future extensions possible. In this approach the decisions on the system flexibility are transferred from design-time to run-time. Two elements are basic. First, a modeling of both hardwired and software functions as well as logical and physical connections by means of *object oriented* techniques (UML); second, hardware and system support for the dynamic instantiation, by means of a resource controller (RSC), of signal processing chains on the available hardware. The proposed target architecture combines Java technology to DSPs and FPGAs and hopes to provide for partial and complete system reconfiguration.

In [8] Moessner et al. describe a complete *network-wide* architecture framework to support three typical reconfiguration scenarios in a mobile cellular network. These scenarios are terminal boot, multi-mode operation and software upgrades. This architecture aims to support partial or full reconfiguration of all protocol layers as needed, control and management of the reconfiguration process, and finally, control and monitoring of the network nodes in respect to their configuration that may change over time. The following elements are described: (i) the terminal software architecture, (ii) the network entities for reconfiguration control. A CORBA based solution is suggested for realizing a configuration software bus within the terminal to connect the two terminal functional parts i.e. configuration and radio related parts.

In [9] the approach taken by the EU project TRUST is presented and a thorough analysis of the reconfiguration problem is given helping to grasp its high complexity.

Finally, in [10], [11] both papers present algorithms and techniques for the "blind" identification of the air-interface standard/modulation by a receiver. Such schemes will permit the terminals to become more intelligent and thus more independent so remove from the network both the responsibility and the workload for reconfiguration (i.e. processing and messaging).

B. A Plausible Deployment Roadmap

From the above discussion the complexity of the general reconfiguration problem becomes evident. Different scenarios necessitate different types of reconfiguration, partial or total, static or dynamic, with or without network implication. Furthermore, the way reconfiguration capabilities will be deployed in the future is not yet completely known. More experimentation is needed to help standard bodies, regulation authorities and business actors, define some kind of deployment roadmap. Past experience shows that technologies evolve from simple towards more complex applications and on a need basis. The scenarios on software upgrades for bug-fixing and performance enhancement as well as algorithm dynamic change (i.e. algorithm diversity) within a single mode of operation will be deployed first. These schemes permit download and reconfiguration signaling through logical/physical channels existing within the mode of operation. Next will come simple robust schemes for multi-mode/multi-service operation without or with minimal network implication. Alternative uplink air-interfaces could be used whenever a mode of operation disposes only of a downlink, e.g. DAB. During this period device reconfiguration mechanisms will mature, a higher reliability of the reconfiguration processes will be attained and regulation issues will become more clear. At the same time the move towards 4G will advance network interoperability. This fact will push forward software radio applications for dynamic mode switching under network control. This will enable dynamic spectrum and network resource management, more intelligent air-interface selection for "best" communication and service integration. Progress in the domain identification algorithms will contribute in making the reconfigurable radios more independent and will help to lower the impact of reconfiguration on the network.

In any case the first step will be to design devices (both terminals and basestations) to support the required reconfiguration scenarios. As [6], [8] and [9] show, this implies new approaches in hardware organization, software architecture and reconfiguration interfaces.

III. A SWR Experimentation Platform

The hardware platform used for our experimentation in software radios and reconfiguration of the radio operation is shown in FIG. 1. A quad TIC6201 DSP processing board interfaces to the analog world through mezzanine cards for A/D, D/A conversion. These cards include bypassable digital frequency translation components for down/up-conversion (DDC/DUC). Most of the operation parameters are under software control through well defined APIs and hardware interfaces. The platform disposes of a fast ethernet connection and has a bi-directional R/T data streaming interface as well as a separate interface more appropriate for control signaling. Through these interfaces remote hosts can interact with the platform both for development and in the context of demonstration applications. In our demonstrations the MATLAB environment running on a host is connected to the DSP platform

for R/T application data visualization and application control. This setup effectively permits to study and demonstrate different types of reconfiguration scenarios and the interactions of the various entities in a cellular network when air-interface reconfiguration occurs.

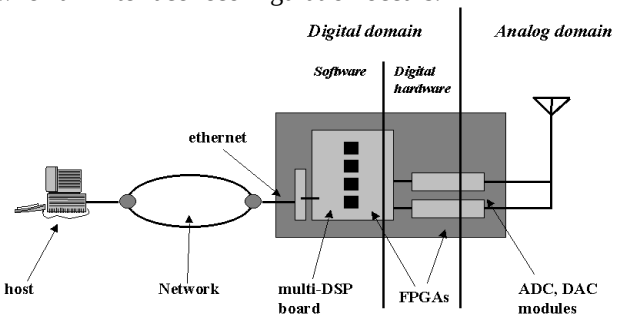

FIG. 1: MERLIN - A SWR experimentation hardware platform

As it will be explained later on, the platform resources are represented in the software domain by software abstractions acting as components. In this way application software and platform hardware are modeled in a uniform manner. Though we currently use available technology (processors, A/D/A converters) we concentrate in working out solutions to system-level problems anticipating the rapid evolution in these technologies.

IV. COMPONENT BASED DESIGN

In this section a component based approach will be discussed. Components have been used in the software engineering community for quite sometime. A component based approach is to be considered as an *extension* and complement to the object oriented approach. Components are a design approach to enforce and achieve reuse (Meyer [12]). In a wider sense components are also the means to achieve system *extensibility* and *evolutivity* after deployment (Szyperski [12]). For a system, extensibility refers to the addition to new elements to the existing ones and evolutivity to the replacement of old ones by new ones. These are precisely the goals of reconfigurability and reconfiguration in software radios. More details on components can be found in [12], [13].

A. Components, Composition and Configurations

A component can be defined as a completely *encapsulated* behavior representing a *unit of change*. Change may occur when the system is operating (hot change) or when it is stopped. Change has to be supported by some means of dynamic linking and late binding. Though basic object oriented design concepts (e.g. encapsulation/information hiding) apply to component design as well, a component is not necessarily a class, it can be a collection of tightly coupled peer classes. With adequate rules, discipline and some basic system support, C language can be used to build components though using some OO language will certainly help. Programming practices allowed by many OO languages have to be avoided (here is where discipline is capital). Examples are the use of global variables (compromise encapsulation) and inheritance (compromises extensibility and evolutivity).

Components imply *composition* which is a recursive operation. Through composition more complex components are built from simpler ones. The full system may be considered as the top-level component. Here *configuration* enters into play. Webster's defines configuration as: *"...something (as a figure, contour, pattern, or apparatus) that results from a particular arrangement of parts or components..."*. For our purposes we interpret this definition to say that a configuration gives a static view of the system's structural and functional aspects which define the system operation. Structural aspects relate to the interconnection of the various system components.

Re-configuration is the process of changing a system's configuration by modifying either its structure or the functional aspect of one or more of its components or by changing both structure and function at the same time. As already stated this is what components offer: extensibility and evolutivity.

At a first time in our work we consider the simple case that structure (component interfaces and interconnections) will not change and that change will concern the algorithms that implement the component behaviors. Extensions to handle more general reconfiguration cases are envisioned.

B. Configuration and Configuration Data

We make the distinction between *configuration* and *configuration data*. A configuration describes a system state of affairs while the configuration data is a machine representation of that state. Assuming that a configuration change does not modify the system structure, i.e. how the system components are interconnected, the configuration data may consist of a combination of the following elements:

- parameter values for each hardware and software component with some degree of genericity in its design;
- state variable initial values for each software component whose process is not stateless;
- binary code data representing the implementation of software component functions and binary data bitstreams representing hardware component implementations on reconfigurable hardware (e.g. SRAM-based FPGA); these account for the reconfigurability offered at the chip level;

When also structure is allowed to change via reconfiguration, the configuration data must also include a machine representation of component interconnection information as well as execution scheduling information.

Configuration data stored using an explicit storage format

form *configuration records* that can be further structured as a *configuration database*. The term database implies some form of *indexing*. In our case the index key consists of a configuration identifier unique for each configuration. It should be noted that unique component identification is a feature of current component frameworks. The *supported configurations* is the set of all configurations for which there is a record in the configuration database.
Thanks to this database, references can be obtained to the entire data collection or individual parts of it making it easy to access either the entire system configuration information or the configuration information of specific subsystems, or individual components. This is the work of the *configuration manager* described next.

C. Reference Architecture

Our reference architecture is shown in FIG. 2. It is generic enough to represent the basic reconfiguration architecture elements and their interaction. It also provides for future evolution by representing the cases that reconfiguration control and reconfiguration data will be distributed across a network. The reconfigurable transceiver software consists mainly of the following software entities: the transceiver (TRx) that implements the signal processing tasks and the configuration manager (CMan) that is responsible for the static or dynamic configuration (and reconfiguration) of the system.

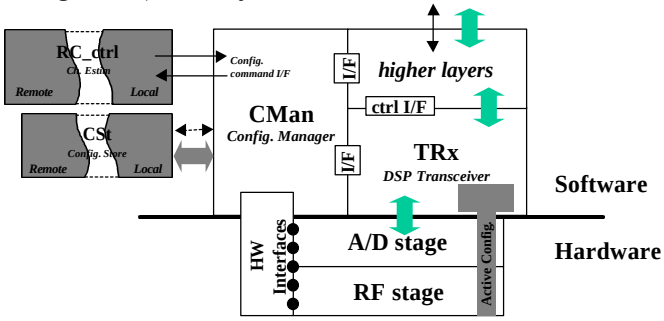

FIG. 2: Reference architecture for reconfigurable radio devices

Of importance are the *control interfaces* (I/F) presented to the *configuration manager* (CMan) by the software and hardware (A/D, RF) components allowing CMan to control their configuration related aspects. The CMan is a hierarchical entity distributed in the system. In an a sense reconfiguration is a recursive process starting at the top-level component and propagating down to the leaf components. Each component is responsible for its own re-configuration and is sensitive to only a part of the configuration data. For *on-line* re-configuration, special sequencing and synchronization is needed to control how configuration data propagates through the system and thus preserve consistency.
Finally, depending on the case, the *reconfiguration controller* (RC_ctrl) and the *configuration store* (CSt) may be distributed entities in the sense that processing intelligence and the associated data may be distributed across several physical entities. Consequently specific communication paths are implied. For instance this may be the case in a cellular network where terminal function depends on decisions taken at the network side, i.e. the device is under network control.
In this case these distributed entities have a remote and a local part in respect to the terminal equipment. The local part acts as a proxy for the remote part. When the device, from the standpoint of (re-)configuration, is completely independent the remote parts disappear. In this case the reconfiguration decision making and deployment are performed locally using locally stored configuration data.

D. The Configuration Cache

At this point it is useful to describe into more detail how the configuration data is organized and as well as the mechanisms that govern access to this data. The *configuration store*, CSt, defines the following: how configurations are stored, where configurations are stored, and how the configuration data are accessed.
We chose to organize the CSt as a 3-*level cache* (i.e. a configuration cache). The first level (L-CSt1, L standing for local) corresponds to configurations stored in execution memory (processor internal memory). These configurations are in a sense pre-installed and ready for execution after some initialization. Switching between such configurations does not impose significant overhead delay since it only necessitates diffusing parameter values to the concerned components and resolving pointer references of software component functions. The second level (L-CSt2) corresponds to configurations stored in secondary (processor external) memory. Switching to these configurations necessitates first bringing the configuration data into execution memory (the 1-st level) using some data transfer mechanism (e.g. background DMA). The transferred data replaces some other 1-st level configuration. Switching then continues as previously described.
The third level (R-CSt, R stands for remote) corresponds to configurations stored at some remote site. Such configurations can be transferred either directly to the 1-st level if a reconfiguration was requested or to the 2-nd level if only an update of the locally stored configurations is requested. This transfer requires the establishment of a communication link (wireless or not) based on some transfer protocol that guarantees error-free data delivery.

E. Implementation Aspects

Existing component infrastructures, like for example Java and Java Beans, due to their genericity are quite heavy-weight; they offer a lot more capabilities than we actually need at the expense of system resources. For implementa-

tion we were inspired by the ideas developed by Stewart in [14] namely the *port based object* (PBO) abstraction. A port based object is a component combining the following elements: (i) a context independent I/O interface (ports), (ii) an object which is viewed as an abstract data type offering encapsulation and (iii) a process that implements the component behavior. This process is represented by a finite state machine and according to its state the appropriate object methods are called. *Replacement independence* at the algorithm level is the byproduct of object encapsulation and the context independent I/O interface. In addition to the PBO abstraction the *port-based framework* offers a basic and uniform execution environment for both preemptive and non-preemptive implementations.

The implementation approach of [14] being specific to a domain i.e. reconfigurable robots, uses a constrained type of components (*port-based objects*) and so it has certain restrictions in terms of the reconfiguration capabilities sought in software radios. Changing the configuration constants of generic components readily implements the parameter-controlled type of reconfiguration. In addition, *replacement independence* at the algorithm level allows for software upgrades and bug-fixing scenarios provided that the upgrades influence only the component internals and they do not have any type of structural impact to the rest of the system.

Structural impact can be as simple as requiring an extra input port for the new algorithm to work, or as complex as adding/removing components requiring both component interconnection and execution scheduling modifications. The port-based implementation model can be easily extended to cope with such structural impact inherent in other software radio reconfiguration scenarios. One extension consists in implementing I/O port interfaces as objects where ports can be dynamically added/removed and connections between old components and new ones can be dynamically created. A second extension represents the component scheduler as an object where processes can be added and removed dynamically. Hence, in a system constructed as a hierarchy of components, the system scheduler is viewed as a hierarchy of component schedulers.

Currently we stick to the constrained implementation whose basic premises are defined in [14] by adding the needed support for on-the-fly reconfiguration while keeping the needed infrastructure overhead low. For this we have opted for a C based implementation without any RTOS support. Components can be software or hardware. Hardware components are represented in the software domain by means of abstraction components providing a *componentized* interface to the actual hardware.

V. CASE STUDIES

In the past two years, using the software radio platform described earlier on, we carried out in our laboratory several experiments covering increasingly software receiver implementations for various standards. Lately we shifted our focus on reconfiguration applications.

Our first experimentation consisted in building a full simplified UMTS-FDD downlink (from IF to BB) including the RAKE receiver and turbo decoder blocks. All functions including carrier recovery, timing adjustment and frequency translation from IF to BB, were implemented in software. This case study served mainly to benchmark the capabilities of our platform on a demanding air-interface.

The second experiment consisted in implementing entirely in software the modem for various mobile communication standards namely GMSK, 3pi/8 offset PSK, QPSK and Frequency Hopping-FSK for GSM, EDGE, UMTS and Bluetooth respectively. The goal was to demonstrate the single platform implementation of a wide panel of modulation schemes for standards that will most probably be present in the multi-mode terminals and basestations of the future. In this experiment reconfiguration granularity is coarse and the reconfiguration process consists of a network initiated switch command triggering complete reconfiguration by paging-in the required application binary file (image) from external processor memory using DMA or the host disk via an ethernet connection. More details and results can be found in [15].

A third experiment, partly conducted as student internship projects, consisted in fully implementing in software receivers for broadcast AM/FM. In the FM case three different demodulation algorithms were implemented. These alternatives permitted to automatically switch from one algorithm to another based on a simple SNR based criterion. This demonstrates the possibilities of trading-off processor cycles (thus power) for better performance under varying reception conditions. In addition changing demodulation algorithms necessitates principally the reconfiguration of the demodulator component block which in turns necessitates the reconfiguration of hardware components by changing their parameter values. For instance, reconfiguring from a real to a complex IQ demodulation scheme necessitates reconfiguring the Rx module from real to complex operation, changing the sampling frequency and the data format. This experiment revealed not only the potential of algorithm diversity in service quality enhancement but also the subtleties of *configuration dependencies* and consistency. Reconfiguring a single component while preserving operation consistency necessitates reconfiguring other system components.

Finally, the "bug-fixing" scenario was further studied in the case of an EDGE receiver. The sampling time adjustment function was implemented entirely in the software domain by means of an interpolation process combining pulse shaping and polyphase filtering. This function was componentized for replacement independence. System software provided for remote binary code downloading from a remote host via a TCP/IP connection. A bug was

simulated in the timing adjustment function and was subsequently corrected by downloading only the binary code corresponding to this function. Complete knowledge of the target system memory map dispensed us with the need for dynamic linking facilities. Reconfiguration was either interactive (remote operation and maintenance) or automatic after stopping the system. We also tested the case of reconfiguration without stopping the system by background downloading into memory locations provided for this effect. Interference with the system was minimal and the transition (switch) to the new operation was seamless. This experiment increased the interest for reconfiguration scenarios where transition from a current configuration to a new one could be implemented incrementally.

An interesting concept relevant to software radios is the concept of algorithm diversity. An example is given by Laster in [16] for the case of GMSK demodulation for which different algorithms exist. Instead of using a unique demodulation algorithm yielding good performance on average, thanks to software radio and reconfiguration more flexible schemes can be envisioned. In the future we shall experiment with this concept because it corresponds to one of the interesting and short term applications of software radio reconfiguration in a cellular network.

VI. CONCLUSIONS

Reconfiguration in software radios may have different manifestations depending on the targeted use-case scenarios. In its most general form it is a quite complex problem. To navigate through the problem space we had to define typical use-case scenarios and a plausible roadmap for the deployment of reconfiguration capabilities. We chose to first address the issues relating to the design of radio equipment supporting reconfiguration. Such platform support is a basic point and of great interest from the standpoint of equipment manufacturers. A component based design approach for the representation of hardware platform capabilities and software architecture is estimated as necessary.

We hope in the future to be able to treat within a more *formal* framework issues relating to the compositional aspects of component based architectures. Finally, an important open issue is the required network-wide architecture and simultaneous management of reconfiguration and inter-network handoff. Resolving such questions will necessitate close collaboration between all implicated actors, manufacturers, operators, regulators and standard bodies. Experimentation will provide valuable feedback for standardization.

VII. ACKNOWLEDGMENTS

We would like to thank Trium R&D for supporting our investigation of reconfiguration aspects of software radio.

VIII. REFERENCES

[1] W. Tuttlebee, *Software Radio Technology: A European Perspective*, IEEE Comm. Mag., Feb. 1999.

[2] J. Pereira, *Re-Defining Software (Defined) Radio: Re-Configurable Radio Systems and Networks*, IEICE Trans. on Comm., vol. E83-B, no. 6 pp. 1174, 2000.

[3] J. Mitola III, G.Q. Maguire Jr.., *Cognitive Radio: Making Software Radios more Personal*, IEEE Pers. Communications , vol. 6, no. 4, Aug. 1999, pp. 13-18.

[4] N. Nakajima, R. Kohno, S. Kubota, *Research and Developments of Software-Defined Radio Technologies in Japan*, IEEE Communications Magazine , vol. 39, no. 8 , Aug. 2001, pp. 146 -155.

[5] H. Harada, Y. Kamio, M. Fujise, *Multimode Software Radio System by Parameter Controlled and Telecommunication Component Block Embedded Digital Signal Processing Hardware*, IEICE Trans. on Communications, vol..E83-B, no.6, pp.1217.

[6] D. Lund, B. Honary, *Design and Maintenance of Physical Processing for Reconfligurable Radio Systems*, 12th PIMRC'01, v. 1, pp. 96-99, Sept. 2001.

[7] D. Lund, B. Honary, K. Madani, *Characterising Software Control of the Physical Reconfigurable Radio Subsystem*, in Proc. IST Mobile Communications Summit 2001, Spain, Sept. 2001.

[8] K. Moessner, S. Gultchev, R. Tafazolli, *Software Defined Radio Reconfiguration Management*, in Proc. 12th PIMRC'01, vol. 1, pp. 91-95, USA, Sept. 2001.

[9] M. Mehta, N. Drew, G. Vardoulias, N. Greco, C. Niedermeier, *Reconfigurable Terminals: An Overview of Architectural Solutions*, IEEE Comm. Magazine , vol. 39, no. 8 , Aug. 2001, pp. 82 -89.

[10] C. Roland, J. Palicot, *A Blind Recognition of the Transmitted, Signal for a Self- Adaptive Re- Configurable Terminal*, in Proc. IST Mobile Communications Summit, Spain, Sept. 2001.

[11] H. Ishii, S. Kawamura, T. Suzuki, M. Kuroda, H. Hosoya, H. Fujishima, *An Adaptive Receiver based on Software Defined Radio Techniques*, in Proc. 12th PIMRC, vol. 2, pp. 120-124, USA, Sep. 2001.

[12] B. Meyer, C. Szyperski, Software Design Magazine: *Beyond Objects* column, http://www.sdmagazine.com

[13] C. Szyperski, *Component Software, Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

[14] D.B. Stewart, R.A. Volpe, P.K. Khosla, *Design of Dynamically Reconfigurable Real-Time Software using Port-based Objects*, IEEE Trans. on Soft. Engineering, Vol. 23, Iss. 12 , Dec. 1997, pp. 759 -776.

[15] C. Moy, A. Kountouris, L. Rambaud, P. LeCorre, *A Reconfigurable Radio Case Study: A Software based Multi-standard Transceiver*, proc. VTC Fall, 2001.

[16] J.D. Laster, *Robust GMSK Demodulation Using Demodulator Diversity and BER*, Ph.D. Thesis, Virginia Tech, 1997.