Performance Evaluation of the Functional Description Language in a SDR Environment

Shi Zhong[1], Craig Dolwin[1], Klaus Strohmenger[2], Bernd Steinke[2]
Note 1 Toshiba Research Europe Limited, 32 Queen Square, Bristol, BS1 4ND, UK;
{shi.zhong, craig.dolwin}@toshiba-trel.com;
Telephone +44 117 9060700; Fax +44 117 9060701

Note 2 Nokia Research Center, Meesmannstr. 103, 44807 Bochum, Germany;
{klaus.strohmenger, bernd.steinke}@nokia.com

ABSTRACT

In this paper we present and evaluate an XML based Functional Description Language (FDL) to communicate configuration information for Software Defined Radio (SDR) equipment. To demonstrate the feasibility of using FDL, we have implemented a proof-of-concept platform based on the Real-Time Research Platform (RTRP) and a Nokia 770 Internet Tablet. The platform has been designed to emulate an SDR terminal where the RTRP implements the functionality of a future reconfigurable SoC. To emulate the network an Open Mobile Alliance Device Mangement (OMA DM) server has been implemented on a laptop, while an OMA DM client operates on the Nokia 770. Both are linked via a WiFi connection.

The proof of concept demonstrates the downloading of an FDL document describing the IEEE 802.11a RAT and its interpretation into a set of object codes and configurations for a DSP, FPGA and GPP. Two FDL files have been created to describe the IEEE 802.11a transmitter and receiver. The overhead for parsing the 802.11a transmitter and receiver FDL descriptions were measured. The reconfiguration overhead for deploying the whole 802.11a baseband RAT using the FDL language was evaluated.

1. INTRODUCTION

The gradual de-regulation of spectrum [1] encourages spectrum to be utilised more efficiently. An increase in spectrum efficiency can be achieved by switching to an under used part of the spectrum and/or by altering the Radio Access Technology (RAT) to match the environment e.g. in an automobile a cellular system might be more appropriate while at the home or in the office WLAN or UWB maybe cheaper. A number of methods have been proposed to support the re-configuration of the terminal. A high level approach is to allow intelligent terminals, also known as Cognitive Radios, to decide locally which communication method is best i.e. it would determine which part of the spectrum it would work in and which RAT to use. To ensure that separate pieces of equipment work together constructively and avoid interference with one another each terminal uses a policy set down by a network based intelligence. This policy can be updated over the air using a policy description language. Using this approach the operator has no direct control over configuring each terminal. A more low level and direct approach to managing the system is to instruct each terminal to reconfigure to a specific technology. This is the approach we investigate through the use of the Functional Description Language (FDL). Clearly taking this approach could require significantly more communications across the network than the policy based approach so in this work we evaluate the overhead imposed by sending a fine grained RAT description for IEEE802.11a using FDL. In practise the FDL description could be at a courser level of granularity e.g. merely specifying GSM or UMTS rather than specifying the individual processing blocks as described here.

We initially outline the requirements for FDL and then describe in more detail the XML schema. A brief description of the OMA DM protocol and the platform is given and then the IEEE802.11a RAT is described. Finally we summarise how large the FDL files are and how long it takes to reconfigure the hardware.

2. FUNCTIONAL DESCRIPTION LANGUAGE

FDL has been developed to allow hardware platforms from multiple manufacturers to be efficiently reconfigured without the decision making agents having a detailed understanding of the underlying hardware. It is important that FDL supports efficient reconfiguration because one of the main limitations for the commercial use of SDR is the increase in power consumption when reconfigurable components, such as DSPs and Reconfigurable Logic, are used. The FDL has been developed with the following key objectives:

- Scalability
- Support for hard real time applications
- Platform independence
- Efficient representation

The previously referred to *decision making agent* (and often called the Network Reconfiguration Manager (NRM) [2]) would typically be part of a cognitive radio network attempting to reconfigure the network to optimise the available resources. In a system supporting FDL the NRM would construct an FDL file defining a configuration for a group of terminals. The FDL file would then be distributed to each terminal and the terminal would attempt to reconfigure its hardware to implement the new configuration. FDL imposes no constraint on what functionality is being defined so the document may define a single RAT or multiple RATs it might also define a mechanism for scanning the spectrum for activity. A more detailed usage scenario description is given in [3].

FDL is based on XML and is used to describe functional configurations for reconfigurable equipment. It enables RATs to be installed and upgraded at run-time. RATs are defined as a hierarchical flow of signals (data and control) between functions (termed processes in the language) communicating via 1-to-1 or 1-to-many channels. The channels are connected to input and output ports on the functions. They define the input and output data types.

In Figure 1 we show the different planes associated with an FDL description. The *Functional Plane* defines logically how processes are linked via communication channels and ports. The *Operating Plane* refers to the operating environment on each resource supporting a process. The *Communication Plane* defines the environment used to support the communication channels and can be a mixture of device drivers for busses and services supplied by an RTOS. The *Hardware Plane* encompasses the physical resources used to implement the signal processing and communications.

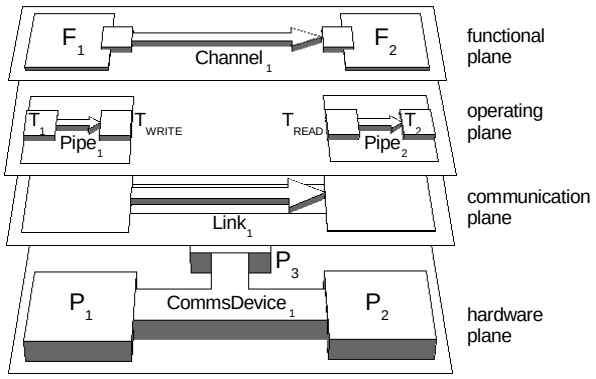

Figure 1 Software radio planes

In addition the FDL description captures precedence and timing information. In Figure 2 we show 5 processes (P1-P5) executing periodically with 3 communication channels (C1-C3). In this FDL document the following timing elements are defined:

- T_{period}, the period over which each process is repeated
- T0, the absolute start time for the 1st period defined in terms of a specific timebase
- deltaT, the earliest time P3 can start relative to the start of the period.

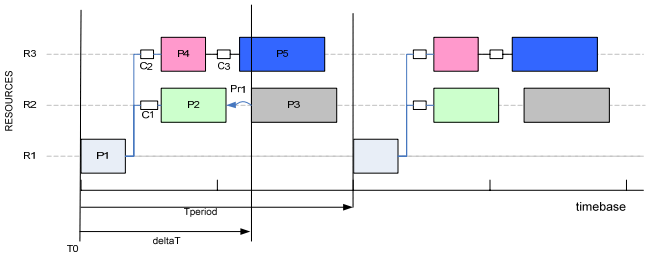

Figure 2 Timing Diagram

An FDL document can be used to describe the fine-grained configuration of a signal processing chain. The **deploy** tag specifies the rat deployment operations. In the deployment operation, a signal processing chain can be represented by an **Alg_Algorithm** XML element. It contains a unique id, a type description, and a execution mode for the algorithm. The AlgDescSchema.xsd XML schema defines a set of rules, data type validation for an algorithm. The signal processing chain (**Alg_Algorithm**) is broken down into sub-components. The main components are subProcesses and subChannels. A channel and process are linked via inputPort or an outputPort. To support a level of hierarchy processes can be associated with one or more sub-processes. A process is equivalent to a functional block such as a channel codec or speech codec. The channels are logical communication links between processes and could eventually, depending on the platform, be mapped to a DMA channel on a shared data bus.

The required elements for defining a signal processing chain are listed in the following:

- Alg_Algorithm. An algorithm description represents a signal processing chain. It contains three attributes.
  - **componentID**. Unique component ID for current algorithm.
  - **type**. The type of the algorithm.
  - **executionMode**. Specifies the execution mode of an algorithm. It can be continuous – (an infinite loop execution mode), or once (execute once then exit).
  - **subProcess**. A functional block used in the processing chain.
  - **subChannel**. The channel is logic link between processes.

A simplified sample of deploying 80211a baseband receiver chain using FDL is illustrated in the following:

```
<deploy>
    <Alg_Algorithm componentID="Baseband80211aRx"
    type="Baseband80211aRx" executionMode="continuous"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AlgDescSchema.xsd">
    <subProcess>
    </subProcess>
    <subChannel>
    </subChannel>
    < /Alg_Algorithm>
</deploy>
```

The attributes and optional elements of a subProcess are:

- **subProcess**. A **functional** block used in the processing chain.
- **componentID.** Unique component ID for the functional block.
- **type**. The type of the functional block.
- **executionMode**. Specifies the execution mode of the functional block.
- **version.** Version of current process.
- **processPara**. Parameters for the functional block.
- **earliestStart.** The earliest start time of the functional block.
- **timeBase.** Timing base for functional block, it could be us, ms, ns, or other defined by user.
- **deadline.** Deadline for finishing current functional block execution.
- **processDepe**. The current process depends on another process.
- **input.** Input port for the functional block.
- **output.** Output port for the functional block.

Each subProcess contains one to many input and output ports. The required elements of an input port are:

- **input**. Input port of a subProcess.
    - **componentID**. Unique component ID for the input port.
    - **type.** The type of the input port.
- **portDataType**. Data type of the port
- **inputConnection**. The reference to a channel, which connects the input port.
    - **refType.** Type of the channel
    - **ref.** Name of the channel

The required elements of an output port are:

- **output**. Output port of a subProcess.
    - **componentID.** Unique component ID for the output port.
    - **type.** The type of the output port.
- **portDataType**. Data type of the port
- outputConnection. The reference to a channel, which connects the output port.
    - **refType.** Type of the channel
    - **ref.** Name of the channel

A channel connects two processes (as shown in **Figure 3**), and the latency, bandwidth, input, and output ports are defined in a channel.

- **subChannel**. A channel connects two processes.
- **latency.** Latency of current channel.
- **peakBandwidth**. Peak band width should be allocated to current channel.
- **writer**. A reference to the output port.
- **reader.** A reference to the input port.

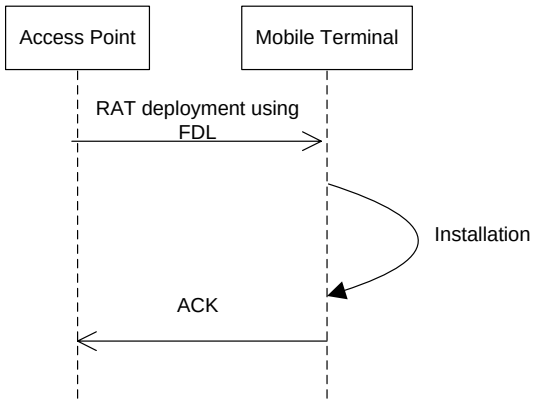

Figure 3: RAT deployment procedure.

Part of the 802.11a baseband receiver processing chain is shown in Figure 4.

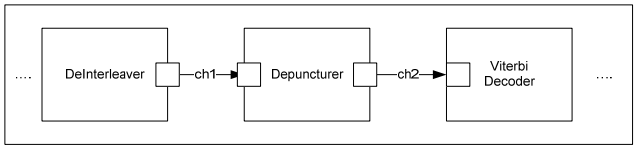

Figure 4: A snapshot of signal processing chain block diagram.

3. REMOTE DEVICE MANAGEMENT

The purpose of remote device management is to deliver configurations and client applications to mobile terminals via device management (OMA_DM) servers [4]. The Open Mobile Alliance (OMA) specification enables the device management features in a vendor independent way. The DM

architecture consists of one or more servers and a client in the mobile device. The server and the client communicate with each other using the OMA DM Protocol, which is based on the SyncML Representation Protocol. The OMA DM also defines a protocol for data synchronisation, named "SyncML Data Sync Protocol".

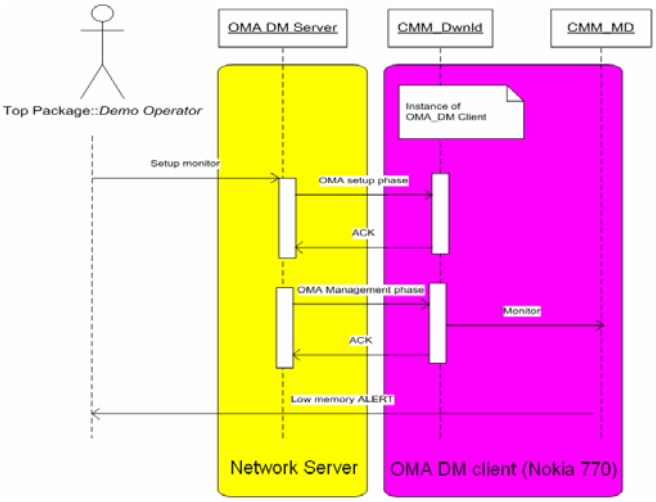

Figure 6 Initialisation of the Communication between OMA DM Server and Client

The OMA DM in a device is represented by a management tree consisting of at least one root node and typically several further nodes. The management tree contains and organises all available management objects. Nodes are entities that are manipulated through the OMA DM protocol. Within the management tree, an interior node can have an unlimited number of child nodes, while a leaf node must contain a value, including null. Each node has a set of run-time properties associated with it, such as Name, Size, etc. The manipulation includes adding a child node, getting the node's properties, replacing this node, or deleting this node. Every node can be accessed directly through a unique Uniform Resource Identifier (URI).

The relation of the node to the device management can be incarnated in two distinct ways: (1) A node might reflect a set of configuration parameters for a device. Actions that can be taken against this node might include reading and setting parameter keys and values. (2) Another node might be the run-time environment for software applications on a device. Actions that can be taken against this type of node might include installing, upgrading, or uninstalling software elements. These actions can be executed on the nodes using the OMA DM protocol management commands [5].

In the device management as applied in the case presented in this paper the OMA DM client is part of the "Configuration Downloads" (CMM_Dwnld) module of the device management framework. The CMM_Dwnld module in general provides the capability to perform downloads of different device management components for the reconfiguration process. It undertakes the management of the downloading procedure. During the initialization of the communication between the OMA DM server and client on the client side the only involved components are CMM_Dwnld and CMM_MD (Monitoring and Discovery) module for monitoring purposes (Figure 6). For the implementation of the test platform, the logical components as indicated in the top row of this figure have been mapped to physical devices as indicated by the shaded background boxes. More information on the implementation is provided in the subsequent section "Proof of Concept Platform".

The device management activities that happen during the transport of the actual configurations, i.e. the delivery of FDL and SPM (Signal Processing Module) files, are

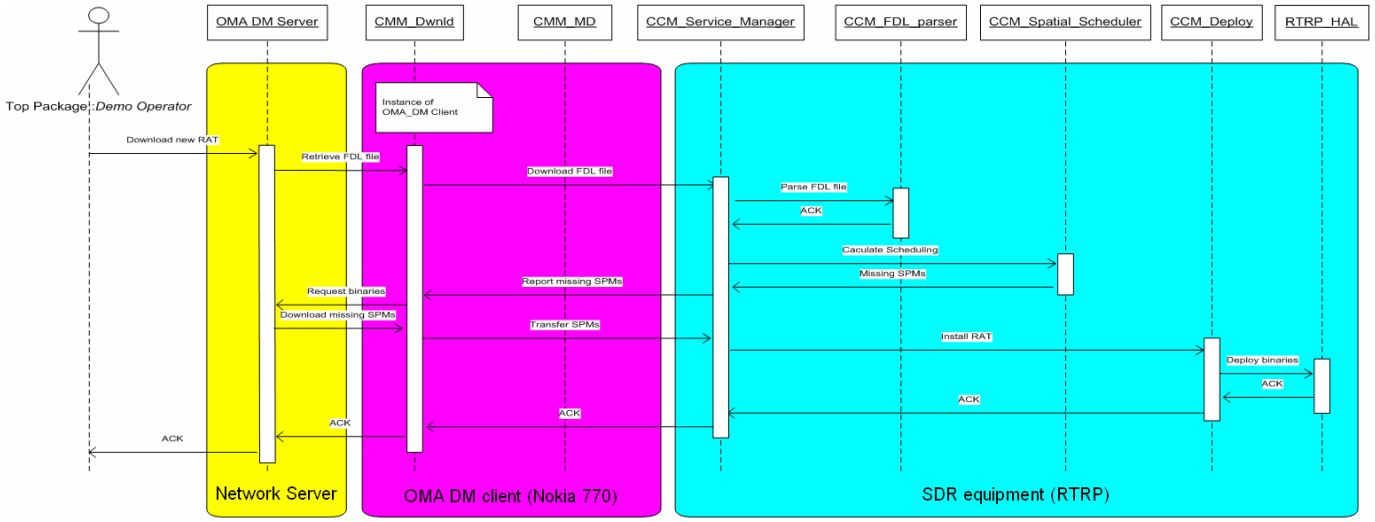

Figure 5 FDL and Signalling Processing Modules (SPM) Download and Deployment Through OMA DM Server

depicted in Figure 5. During *Retrieve FDL file* an FDL file is delivered to the OMA DM client using the ADD command of the OMA DM protocol. The OMA DM client provides the file for further processing to the Configuration Control Module (CCM) components of the device management framework. After the FDL file has been parsed and interpreted it may happen that further downloads are required to implement the device configuration as requested within the FDL file. In this case a *request Binaries* message is sent by the client to the server. The server responds to that by providing the missing SPM files, again using the ADD command. When the OMA DM client has received the SPF files it transfers them to the CCM Service Manager for further processing.

4. PROOF OF CONCEPT PLATFORM

As we have reported in previous papers [6] we have developed a Proof of Concept platform (Figure 7) to investigate and demonstrate the process of reconfiguring a terminal after receiving an FDL document. The platform is comprised of:

1. Laptop containing an OMA Database Manager (DM) server, this is used to store the FDL document and a set of object codes known as Signal Processing Modules (SPM). The laptop represents the NRM.
2. Nokia 770 Internet Tablet containing the Configuration Management Module (CMM) and the OMA DM client. The Nokia 770 is connected to the OMA DM server using its WiFi link and connected to the RTRP using a USB link.
3. Real Time Research Platform (RTRP), this is used to emulate a reconfigurable baseband SoC and contains one or more DSP/FPGA and a PC. It also contains the Configuration Control Module (CCM) this is used to interpret the FDL documents and supports the protocol to request object code from the OMA DM server and install the code on the hardware.

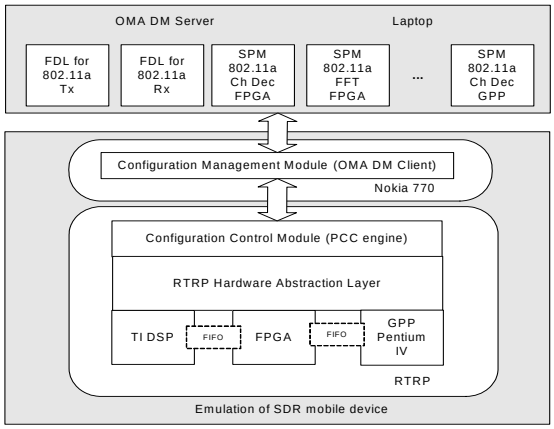

Figure 7 Proof of Concept Diagram

To test the system and evaluate the overhead imposed using FDL we developed a set of SPMs for an FPGA (Xilinx Virtex II Pro) , DSP (TI C6203) and GPP (Intel Xeon) to implement the physical layer for an IEEE802.11a WLAN transceiver. We then created an FDL document to describe the complete IEEE802.11a transceiver see Figure 8. The FDL document is compressed using the GZIP [7] utility before transmission, see Table 1.

802.11a	XML Size (Kbytes)	Zipped Size (Kbytes)	Parsing time (s)		
			Win	Linux	770
Receiver	18.3	1.8	0.12	0.11	0.31
Transmitter	17.6	1.9			

Table 1 IEEE802.11a FDL file statistics

We also measured the time taken to parse both the transmit and receiver documents on three platforms i.e. Windows (Windows® XP, Xeon 2.0Ghz), Linux (Ubuntu Linux, Pentium IV 3.2Ghz) and Nokia 770 (Debian Linux, TI OMAP 252Mhz). Once the FDL code is parsed the appropriate SPM are downloaded from the OMA database. To estimate how much time the downloading would take in a practical system we used typical data rates for three cellular systems (HSDPA, EDGE and GPRS). The zipped SPM sizes were based on a mixture of TI DSP and FPGA as shown in Figure 8. Once the SPM have been downloaded they are un-zipped and installed in the target DSP and FPGA. In Table 2 we summarise the time taken to install all the SPMs onto the hardware. It is clear from these figures that the time taken to install the object code is substantially greater than the time taken download and interpret a FDL file.

	SPM Size[Note 2] (Kbytes)	Install Time (s)	Projected Download Times (s)		
			400KB/s (hsdpa)	29.6KB/s (edge)	10KB/s (gprs)
RX	109	24[Note 1]	0.27	3.7	10
TX	111		0.28	3.8	10

Table 2 SPM download and Install times

[Note 1]: Decompression and Installation time. TI object code: 4 s, FPGA: 20 s.
[Note 2]: Size after being compressed using GZIP

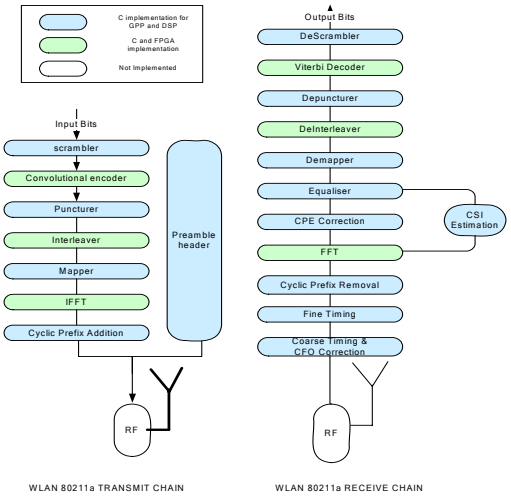

Figure 8 IEEE 802.11a processing chain

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described the practical implementation of FDL for the IEEE802.11a WLAN RAT. We have also evaluated the overhead associated with communicating and interpreting FDL documents and shown that even with a relatively complicated configuration the overhead is quite small when compared to the Over The Air (OTA) downloading of object code and its installation onto the hardware. The work so far has assumed relatively simple mapping between processes and resources and this has allowed us to use a rather trivial temporal and spatial scheduling scheme. Future work could focus on more complicated RATs e.g. cellular and further investigate the problem of power sensitive dynamically scheduling on a resource limited platform.

Another improvement is planned on the security of the OMA DM client. The current implementation runs on a Linux distribution with normal access control. Future implementation will use the Security Enhanced Linux (SELinux) offering Role Based Access Control (RBAC) and Type Enforcement (TE) as mechanisms to decide about access requests. The impact of the increased security on the performance has to be investigated.

6. ACKNOWLEDGEMENT

This work was performed in project E2R II which has received research funding from the Community's Sixth Framework programme. This paper reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein. The contributions of colleagues from E2R II consortium are hereby acknowledged.

7. REFERENCES

[1] Ofcom, “Business Radio Trading & Liberalisation, a consultation on proposals to liberalise and simplify business radio licensing (including measures to extend spectrum trading”, 6 July 2006

[2] Holland, Oliver, Muck, Markus, et.al "Development of a Radio Enabler for Reconfiguration Management within the IEEE P1900.4 Working Group," New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on , vol., no., pp.232-239, 17-20 April 2007

[3] R. Burgess, S. Mende. The Role of Configuration Data and a Configuration Control Module in an End-to-End (E2R) Software Radio System. 14th IST Mobile & Wireless Communications Summit: Dresden, 19th - 23rd June 2005

[4] S. Lin, S. Jiang, H. Lin, J. Liu: An introduction to OMA Device Management, http://www-128.ibm.com/developerworks/wireless/library/wi-oma/

[5] OMA Device Management Protocol. Candidate Version 1.2 – 28 Jun 2005

[6] Shi Zhong, Craig Dolwin, Rollo Burgess, “A Software Defined Radio Proof-of-Concept Demonstration Platform”, SDR forum technical conference, November, 2006.

[7] GZIP home page, www.gzip.org.