OULU UNIVERSITY OF
APPLIED SCIENCES

Teemu Vaattovaara

ANALYSIS OF MODEM INTEGRATION IN OPEN SOURCE SMARTPHONE PLATFORMS

Teemu Vaattovaara

ANALYSIS OF MODEM INTEGRATION IN OPEN SOURCE SMARTPHONE PLATFORMS

Teemu Vaattovaara

Master's Thesis
Spring 2011
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author(s):	Teemu Vaattovaara
Title of Master's thesis:	Analysis of Modem Integration in Open Source Smart Phone Platforms
Supervisor(s):	Kari Laitinen, Jyrki Kaski
Term and year of completion:	Spring 2011
Number of pages:	55 + 64 pages

Current mobile phones are considered as multipurpose and powerful handheld devices suitable for many activities. This thesis is analyzing the oldest, and still the most important, functionality of a mobile phone; telephony communication. The specific topic is how the traditional and rather stable peripheral, like a cellular modem processor, is integrated to the latest open source application environments. In these environments legacy functionality meets the open source as a newcomer, and it may create some interesting solutions for an open source community, business partners and the industry overall. Traditionally the mobile phone, and especially the modem development, has been a closed source and proprietary design; it can be assumed that open source environments are enforcing different business scenarios even for a traditionally closed sourced modem development and co-operation. This thesis includes two white papers to propose a more efficient software architecture for the modem integration that could be standardized with and complement an open source implementation. The conclusion of this thesis contains some of the questions found during standardization discussions; what kinds of problems and challenges there are when initiating a standardization for some technology area.

Keywords
Adaptation, baseband, cellular, integration, interface, modem, standardization

CONTENTS

1 INTRODUCTION.................... 9

2 OPEN SOURCE MOBILE PLATFORMS.................... 11

2.1 Short history of open source mobile platforms.................... 11

2.2 Software licenses.................... 13

2.2.1 Apache Software License (ASL).................... 13

2.2.2 GNU Generic Public License (GPL).................... 14

2.2.3 Eclipse Public License (EPL).................... 15

2.3 Android.................... 15

2.3.1 Introduction.................... 16

2.3.2 Software architecture.................... 16

2.4 MeeGo.................... 19

2.4.1 Introduction.................... 19

2.4.2 Software architecture.................... 19

2.5 Symbian.................... 22

2.5.1 Introduction.................... 22

2.5.2 Software architecture.................... 22

3 MODEM SERVICES INTERFACE.................... 25

3.1 AT command interface.................... 26

3.1.1 Interface implementation.................... 27

3.1.2 Standardization.................... 28

3.2 Nokia Wireless Modem API.................... 29

4 MODEM SERVICES.................... 30

4.1 Introduction to modem services.................... 30

4.1.1 SMS Protocol.................... 31

4.1.2 Android.................... 32

4.1.3 MeeGo.................... 34

4.1.4 Symbian.................... 35

4.1.5 Summary of findings.................... 36

4.2 Modem System Services.................... 36

5 STANDARDIZATION AND REUSABILITY.................... 37

5.1 Business need for standardized modem interface 37
5.2 Shortcomings of AT commands 38
5.3 Open Modem Interface initiative 40
5.3.1 Open standard 42
5.3.2 Benefits for modem suppliers 42
5.3.3 Benefits for application engine platforms 44
5.3.4 Benefits for product integrators 44
5.3.5 Technical characteristics 45
5.4 Device Interconnect Protocol based industry standard 46
6 CONCLUSION 47
6.1 Technical findings 47
6.2 Open standardization initiative and industry feedback 47
6.3 Personal findings 48

TERMS AND ABBREVIATIONS

API	Application Programming Interface is a software interface that can be used by clients to access some functionality provided by a service. Application Programming Interface is typically specific for application environments and operating systems.
Application Engine	Software and hardware responsible for running applications and providing an environment for other peripherals to join.
DIP	Device Interconnect Protocol, part of NoTA architecture
Hardware/ Physical interconnect	Physical connection between two hardware peripherals, e.g., USB connection, MIPI standardised connections, shared memory, or some proprietary nonstandardised connection.
H_IF	High Interconnect Interface
H_IN	High Interconnect
Interface	See definition of API. Interface is more abstract compared to API. Interface can exists between two independent peripherals, and it may contain a definition of hardware interconnection. In a context of this thesis, interface is used to refer to cross-peripheral software interfaces, whilst the API is used to describe internal interfaces within the application environment.
L_IN	Low Interconnect
Modem	Peripheral that handles communication between the device and base stations. Typically, it is a separate and independent hardware chip that is connected to an application processor by some hardware interconnection.
NoTA	Network On Terminal Architecture
Peripheral	An independent hardware and software component that is connected to a host through some hardware interconnection and software interface. Typically the peripheral requires a

	device driver that communicates with hardware by using a manufacturer specific software interface.
Subsystem	A collection of software components that create a logical and functional entity. For example, a UICC subsystem may contain UICC software interfaces, UICC services, and a UICC hardware driver.
Telephony Integration API	API that is typically part of middleware layer and abstracts the underneath modem software and hardware interfaces, e.g., Android Radio Interface Layer (Android Developers - Telephony), MeeGo oFono (Intel, Nokia) and Symbian CTSY Dispatcher.
Telephony Middleware	Middleware components that are not specific to a certain modem, but is a common software asset and part of platform delivery, e.g. Android Java Telephony Manager, MeeGo oFono and Symbian Etel.
Telephony Middleware API	API provided by Telephony Middleware and used by application layer clients, e.g. Android Telephony Manager API, MeeGo oFono API and Symbian Etel API
UICC, SIM	Universal Integrated Circuit Card; an evolution of the SIM card, it can contain more identification services (e.g., for NFC or multimedia services) than traditional SIM for wireless networks.

PREFACE

I would like to thank everybody involved. Completing this thesis, and attached whitepapers, has been a long and painful road. Special thanks go to my colleagues and co-writers Martin and Jyrki, who have been giving precious contribution to whitepapers and spending countless hours in teleconferences discussing these topics. There have been tens of different people working around this subject, and hopefully this work will continue among those persons. The topic is such that it cannot be handled within a single company, but requires innovative experts from different areas, and from many different companies in the telecommunication industry.

Last but definitely not the least, I would like to thank, and apologize, my family. Thereby I promise never ever study anything... …related to this degree anymore. ;)

Best Regards,
Teemu Vaattovaara
Thursday January 20th 2011, when the night has fallen.

1 INTRODUCTION

This master thesis is focusing on software integration of cellular functionality on open source mobile phone platforms. Cellular functionality consists of application engine applications, middleware, middleware APIs, hardware abstraction layers and a cellular stack on a modem processor. Those are the most critical layers regarding this analysis. Android, MeeGo and Symbian have been selected as reference platforms. All these platforms use an open sourced middleware or interfaces, making it possible to make high level comparisons between their architectures. However, there are some differences about how much details can be included from each platform; this is due to the public documentation available and to the source codes. Most of the platforms have published enough information to give an overview of telephony integration, and to see some differences in implementation and software architecture. This analysis is focusing only on high-level software architecture.

Due to the nature of modem integration, this documentation has its focus on the modem supplier's and product integrator's viewpoints, and does not provide any end user related analysis. The end user analysis should be focusing more on the application and user interface layers. After all, modem software architecture and hardware are always hidden from the end user. Modem functionality is visible only as some line on a device technical specification, like a list of supported radio access technologies.

More than the modem features itself, it is more important to understand the changes that are evolving through the mobile device industry. Mobile devices are shifting from closed source platforms to more open environments, thus introducing new software and hardware combinations for modem integration.

The hypothesis for this study is that all modem integration architectures are similar at a principal level, but due to differences in software architectures, APIs and implementations, these cannot be easily reused across different platforms. For example, using Symbian CTSY Dispatcher adaptation in MeeGo or Android is either

extremely difficult or not possible at all. However, as the principal is the same, there are some solutions, which can be applied to achieve a more reusable codebase and architecture. This would gain significant industry-wide benefits for cross-platform modem development, integration and business opportunities.

Modem features are mostly based on the 3GPP standardization for the cellular connectivity. Typically, other more system-integration-level services are forgotten. This thesis, and especially the attached white papers, give a brief understanding of modem integration outside the cellular protocol services.

MeeGo, Symbian and Android have some fundamental, non-technical differences that have some impact on how modem integration can be done. These differences define how the platforms split their software asset between open source and proprietary components, and what kind of philosophy they have as a business model. Therefore this document contains an introduction to different licensing models and introduction to each platform's telephony integration layers for overview and comparison purposes.

2 OPEN SOURCE MOBILE PLATFORMS

Open source mobile platforms follow similar software architecture solutions for modem integration. There are some differences how software asset is divided between the closed and open source software.

2.1 Short history of open source mobile platforms

There have been open source mobile platforms before Android was launched. 'Limo', 'Moblin', and 'Openmoko' have existed for some years without gaining any significant market share in the mobile OS market. These platforms did not cause very much distraction for proprietary mobile platforms. The discussion related to open source mobile phones was really started after the Open Handset Alliance was formed, introducing the Android platform in 2007. Android is supported by Google Inc. and it was therefore gaining significantly more publicity than any previous open source platforms. Since the Android launch, Nokia has introduced two open source mobile phone platforms; Symbian Foundation and MeeGo (formerly known as Maemo). Maemo is famous from Nokia Internet Tablets, but mobile phone functionality was not published until 2009, when Nokia N900 mobile device was launched. Nokia and Intel announced 15 February 2010 that Maemo and Moblin will be combined as one platform – MeeGo (1).

All of these three platforms have fundamentally same basic architecture as shown in figure 1 and figure 2. They all can contain multiple engines, and from this document's perspective, the most important ones are the application and modem engines. The application engine contains naturally all the application layer responsibilities, such as call handling and messaging user interfaces. The application layer is using services from a middleware, like *Telephony middleware*, which is responsible for abstracting Modem HW driver from the application layer. The abstraction is done by the middleware by using the integration APIs, thereby providing consistent and easy-to-use modem services regardless of the modem hardware below.

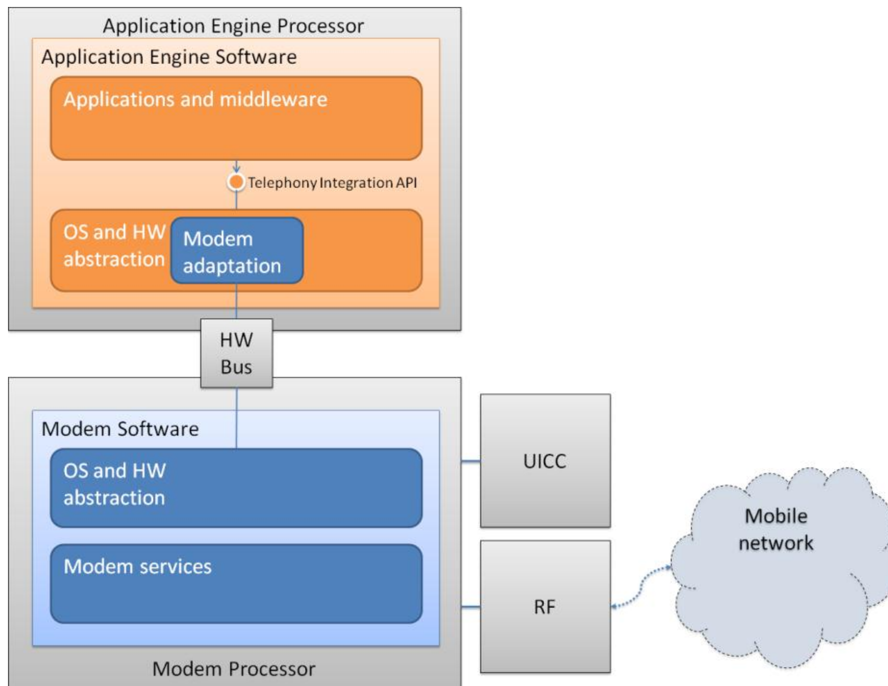

FIGURE 1. Application Engine controls modem processor through some hardware interconnection. Modem handles all signaling and low-level messaging from/to network.

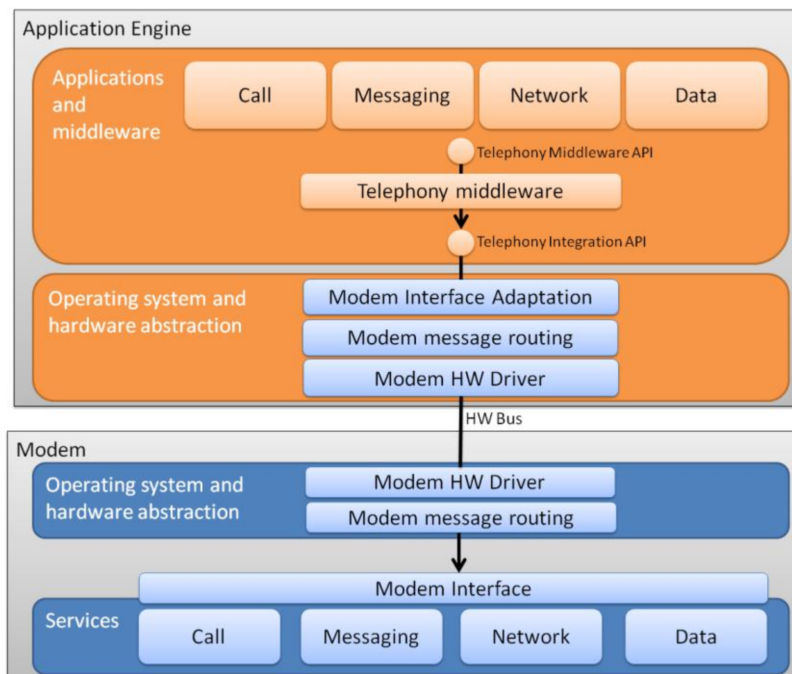

FIGURE 2. Overview of a smart phone telephony architecture where modem supplier responsibility is on components below telephony integration API

The application engine and the modem are connected by some hardware interconnection, which can be practically anything that has enough performance for high data rate packet connections. In figure 2, it is assumed that both the modem control and the modem audio are transferred through the same physical HW bus, and therefore, in that diagram context, *Modem HW Driver* is used to refer to both the modem control driver and the modem audio driver.

The modem contains its own operating system, which does not have any dependencies to the operating system used in the application engine. The only software related dependency, between those two environments, is the modem software interface that must be used by the modem driver in the application engine. The modem software interfaces are described in more detail in chapter 3.

2.2 Software licenses

An open source software has always some licensing model associated with it. A license may, or may not, allow a proprietary derivative implementation based on the open source code. Therefore, the license used has a big impact on possible business scenarios for open source components. An open source component can affect surrounding components as well. For example, it is allowed to dynamically link and use an LGPL licensed component, without making one's own components open sourced. This is not possible if the purpose is to link and use functionality from GPL licensed component.

Android, Symbian and MeeGo have chosen different licensing models. This section shortly describes the fundamentals of the licensing model for each platform, and how it affects the cellular adaptation and its business scenarios.

2.2.1 Apache Software License (ASL)

Android, as an application environment, uses the ASL license. The Linux kernel of Android is licensed under GPL, but due to the virtual machine and Java runtimes, it was possible to choose a different license model for the applications and the middleware.

ASL is a business friendly license allowing the manufacturer and developers to decide whether they want to publish their own modifications or not. This efficiently allows the proprietary software to be implemented for Android without any need to give away own inventions or competitive advantages.(2.)

This applies even to the core Android functionality implemented by Google Inc; there are some internal classes especially in the telephony service area, which are kept as a closed source software and without public APIs.

2.2.2 GNU Generic Public License (GPL)

MeeGo is fully based on Linux and therefore the chosen license has a much wider impact than, for example, in Android where the virtual machine allows different licensing for the hardware abstraction and the application environment.

The use of GPL version 2.0 licenses allows MeeGo to use numerous existing open source projects as the core of the application environment. The advantage of this is that a smaller effort is needed to build up an application environment, but the obvious drawback is the requirement to publish all derivative work under the same GPL license as the rest of the system. GPL license does not allow any derivative work to be a closed source. Even a function call to a GPL licensed code is handled as a derivate work, and therefore it is not possible to use functions from the GPL code without licensing one's own component under the same license. This applies only to function calls, and therefore the GPL code can be used if the usage is going through some messaging and hardware bus. Therefore the GPL licensing requirements does not apply to a modem software stack. (3.)

Due to the ideology of the Linux environment, MeeGo has its telephony stack implemented under the GPL version 2. This makes MeeGo a more open platform than its competitors, and it ensures that all the work related to the MeeGo telephony stack is, and will always be, an open source.

There is one problem with the GPL licensing; this license model is not very business friendly if a manufacturer does not like to publish their proprietary implementation as

open source. The modem, like any other low-level hardware component, may and will have some manufacturer specific innovations, and therefore GPL is suitable only for those modems that are very generic and provide the functionality as published by 3GPP. This may decrease the interest to participate into MeeGo telephony development if a company has some inventions or other proprietary advantages that must be kept secret. At the worst case, this causes unnecessary and inefficient software layering to be done only to avoid the GPL licensing requirements, and therefore it makes the telephony stack even more complicated as it already is.

2.2.3 Eclipse Public License (EPL)

Symbian Foundation uses the Eclipse Public License. This license allows manufacturers to develop proprietary software if the software only links to an EPL licensed code. It is not possible to modify the EPL licensed source code without making the changes public.(4.)

The EPL license fits quite well to a layered software architecture where there is no need to make changes to an upper layer when implementing lower, for example hardware abstraction layers. Well-defined integration APIs support this licensing model by encouraging the collaboration and contribution if API or upper layer software changes are needed. For example, if some modem supplier finds out an improvement in the API which is licensed under EPL, then they must contribute their changes to the API, but they can still keep their own adaptation layer implementation as a closed source. This kind of co-operation model fits perfectly with the EPL license; EPL supports a proprietary code, but at the same time it makes sure that all the common improvements are contributed to a platform itself.

2.3 Android

Android is the most well-known open source mobile platform, although it does not provide a full open source access to middleware and application level. Android's open source is limited to the Linux implementation, and most of the Dalvik Virtual Machine is kept as a closed source.

2.3.1 Introduction

As stated earlier, Android was born when the Open Handset Alliance was formed in 2007. The key company behind Android is Google Inc, but there are many more telephony industry and operator companies involved.

"The first truly open and comprehensive platform for mobile devices. Google Inc., T-Mobile, HTC, Qualcomm, Motorola and others have collaborated on the development of Android through the Open Handset Alliance, a multinational alliance of technology and mobile industry leaders."(5.)

In this press release Open Handset Alliance describes Android as a truly open mobile device platform. Android provides the reference implementation for the compatible AT-based modems of Hayes. Therefore, Android can be used as a mobile phone platform out-of-the-box without a major investment to re-implementing the telephony abstraction or the modem hardware driver layers. Of course, this is valid only when using the same or a very similar hardware as in the reference platform. In practice, there is always a need to make an adaptation software for the chosen hardware. Android is designed to be as portable as possible by using the already well-supported, and easily portable, Linux kernel.

On the other hand, Android also allows manufacturers to implement the proprietary telephony abstraction, and therefore Android is not as open as MeeGo. The Android licensing model is more flexible for modem suppliers, but on the other hand it does not allow flexibility on the application environment, which is controlled by Google.

2.3.2 Software architecture

From the modem integration's point of view, the most important component in the Android software asset is the "Radio Interface Layer" aka Radio Daemon (RILD) component within Linux OS. RILD is a Linux user space component and can be implemented as a proprietary component. RILD communicates with the modem hardware through the Linux kernel modem hardware driver. In the Android reference

platform, the modem hardware bus is a shared memory, but in practice it can be anything else as well.

The application environment (Dalvik Virtual Machine in figure 3) telephony manager is connected to the Radio Interface Layer by using a socket connection. This socket connection is reserved only for this communication and it cannot be redirected or expanded for any other clients. RIL Daemon is a generic code and provides a communication channel between the Telephony manager and supplier specific RIL implementation. RIL Daemon is a very thin component and acts as a connector between these two components. All of the telephony specific modem abstraction functionality is located in the supplier's RIL component.

A header file named *ril.h* defines numerous errorcodes, callstate values, radio states and datastructures for voice call, data call, SIM card access etc. This file is approximately 3500 lines long and contains all possible modem specific functions, states and datatypes.

As shown in figure 4, *Telephony manager* uses the Radio Interface Layer underneath, and supplier RIL, as specified in *ril.h* file. Therefore there is a logical dependency between the class *android.telephony* and supplier RIL through the *ril.h* header file.

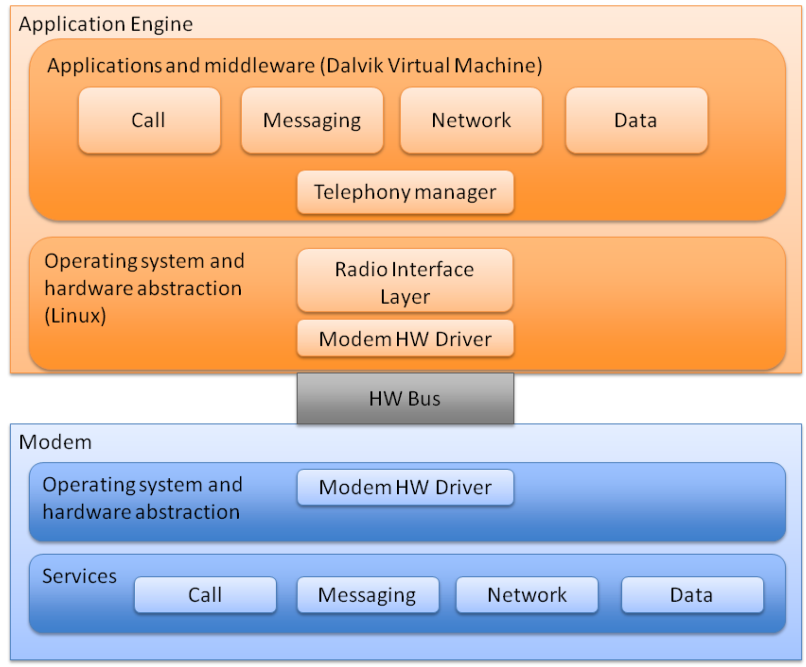

FIGURE 3. Over view of Android telephony architecture

Supplier RIL can use any hardware interconnection and software interface to make a modem connection. This is supplier specific and depends on the modem hardware and software stacks.

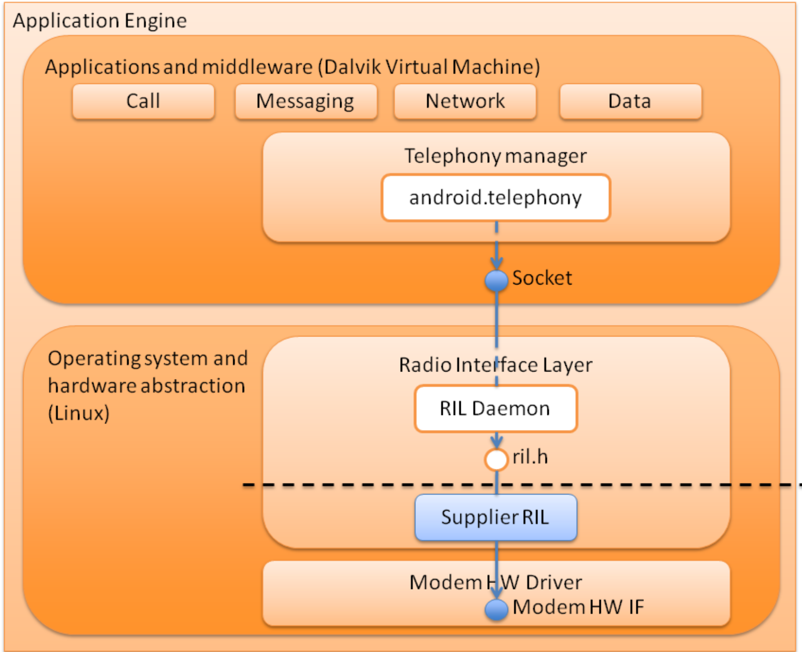

FIGURE 4. Basic communication flow from Telephony manager to supplier RIL through RIL Daemon

2.4 MeeGo

MeeGo has its origins in Nokia's Maemo and Intel Moblin platforms. Maemo has been used in Nokia Internet tablets N770 and N810. Those devices have been implemented only for web browsing, and there has not been a cellular functionality until the announcement of N900.

2.4.1 Introduction

N900 introduced a new improved Maemo 5 version, including a telephony stack and an integrated cellular modem peripheral. Unfortunately N900 cannot be used as a reference platform for this study due to its closed source telephony stack.

MeeGo includes a new oFono telephony stack replacing Maemo and Moblin telephony stacks with an open source alternative. This document uses oFono as it provides enough information for a high level comparison with the other platforms. There is very little documentation about oFono, but the source code is provided including, even other than the AT command, a modem implementation.(6.)

oFono is licensed under GPLv2, and it includes a high-level D-Bus API for use by telephony applications of any license. oFono also includes a low-level plug-in API for integrating with Open Source as well as third party telephony stacks, cellular modems and storage back-ends. The plug-in API functionality is modeled on public standards, in particular 3GPP TS 27.007 "AT command set for User Equipment (UE)."(7.)

As written in the announcement above, oFono focuses on the AT command based modem software interface. This should make it easier to compare the oFono architecture with the similar functionality of Android.

2.4.2 Software architecture

Referring to figure 5, oFono follows the same principles as the other platforms. oFono provides an abstraction for the underneath modem implementation and provides APIs for the middleware and applications. The most visible difference, compared to Android, is a fully utilized Linux environment without any virtual machines. Therefore, oFono can easily be run in the desktop Linux environment as well.

The oFono modem plug-in equals the Android Supplier RIL implementation where all the modem supplier specific logic is implemented. This abstracts different modem implementations from the oFono middleware. oFono itself, like the Android telephony manager, has to focus only on a common middleware telephony functionality and APIs for the applications.

As shown in figure 6, oFono provides the oFono API as a D-Bus API. D-Bus is an inter-process messaging protocol that can be used between applications and processes to communicate with each other. More information about D-Bus can be found on the webpage freedesktop.org. (8.)

oFono itself uses the Modem Plugin API to communicate with the different modem plug-ins. *The Modem Plug-in API* is divided into headers based on the features. There exist separate headers for sms, call, networking etc., and not only a single combined header as in Android.

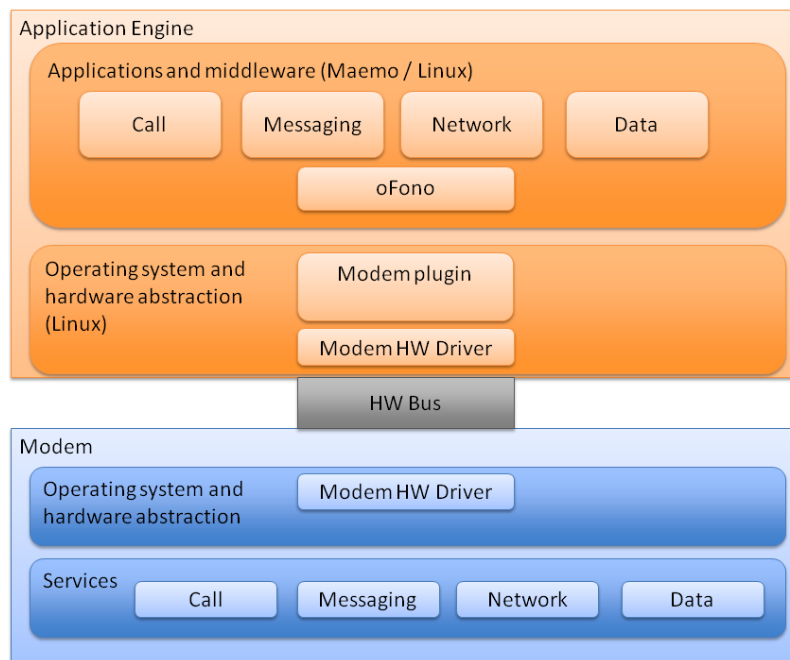

FIGURE 5. MeeGo telephony architecture overview

Otherwise, the modem plug-in responsibility is aligned with Android, as the plug-in has to make a needed conversion from *the Modem plug-in API* to a modem specific message format, and to maintain and handle the actual communication with the modem.

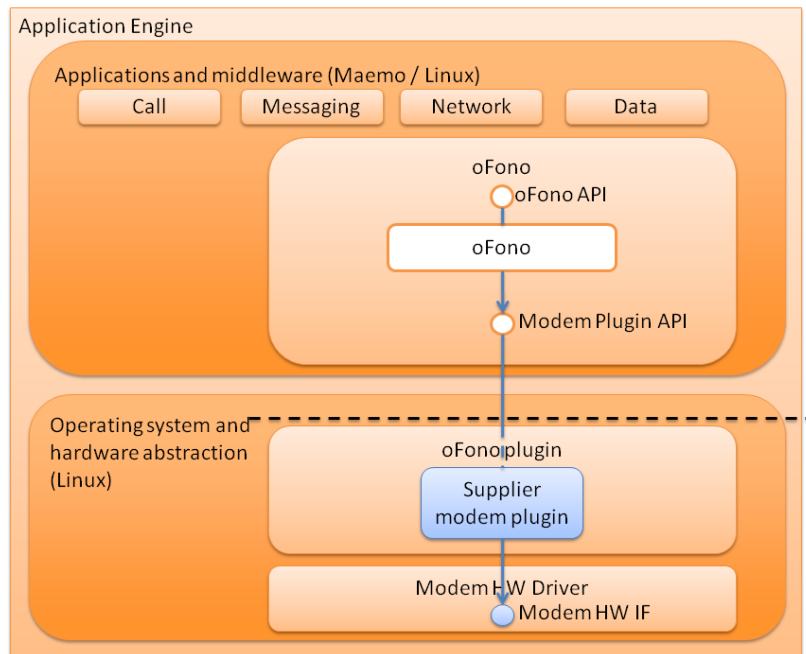

FIGURE 6. oFono APIs and a supplier implementation

2.5 Symbian

Symbian has a very long history as a mobile device operating system. Symbian is the oldest platform of all of these three, and provides all the features required from a modern smart phone from the end user's and network operator's perspective.

2.5.1 Introduction

There is no need to go through the Symbian history considering the scope of this document, because there is a comprehensive history described on the Symbian Foundation webpage. The most relevant event for this documentation happened when Nokia acquired Symbian Ltd in 2008 and formed an open source Symbian Foundation (9), therefore allowing a more public exposure of the telephony stacks, including SHAI layers.

During the writing phase of this thesis, Nokia announced further that the licensing model of the Symbian Foundation will be changed, and the operational model is moving towards a more closed licensing model. This can be interpreted as a failure to introduce Symbian as an open source platform, and as a shift to focus the Symbian development as an internal source code asset. (10.)

2.5.2 Software architecture

As described in figure 7, Symbian has a slightly different architecture on the application and middleware compared to Android or MeeGo; Symbian separates the SMS stack from the main telephony functionality. The telephony server contains functionality for call and network handling, GSM and WCDMA specifics, packet data connections and SIM control and SIM application toolkit.

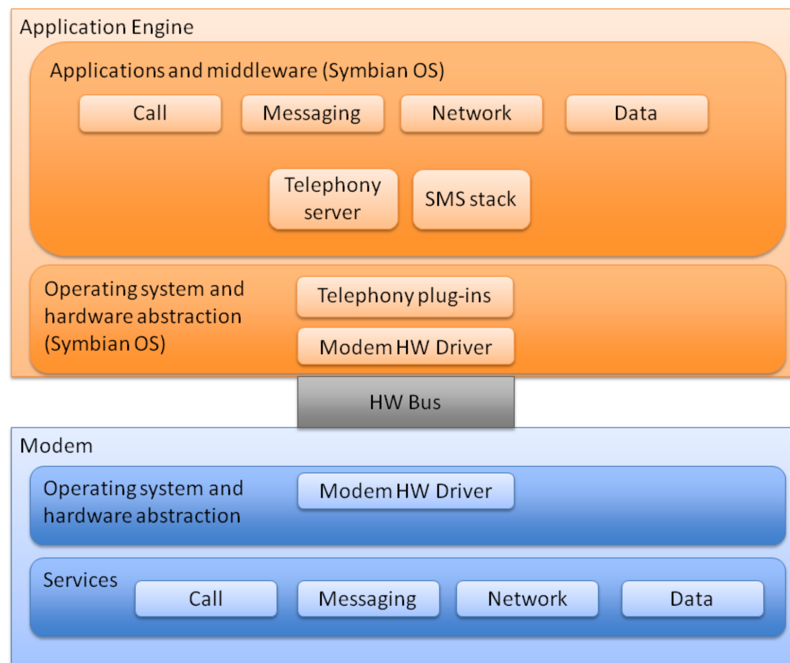

FIGURE 7. Symbian platform telephony architecture overview

As shown in figure 8, there are two options how to integrate a modem functionality underneath the Telephony server (Etel). A legacy solution is to implement the telephony subsystem (Licensee TSY) as a TSY Plugin API component. Due to the architecture simplification and for easier integration, Symbian Foundation has introduced a new layering by creating a Dispatcher component and a simpler Dispatcher API to be used as the integration API. CTSY Dispatcher is a new solution provided by Symbian Foundation and therefore the only valid option for any new modem adaptation where no legacy implementation exits. The Dispatcher takes care of the IPC communication with the Common TSY and provides much simpler APIs for the modem integration. (11.)

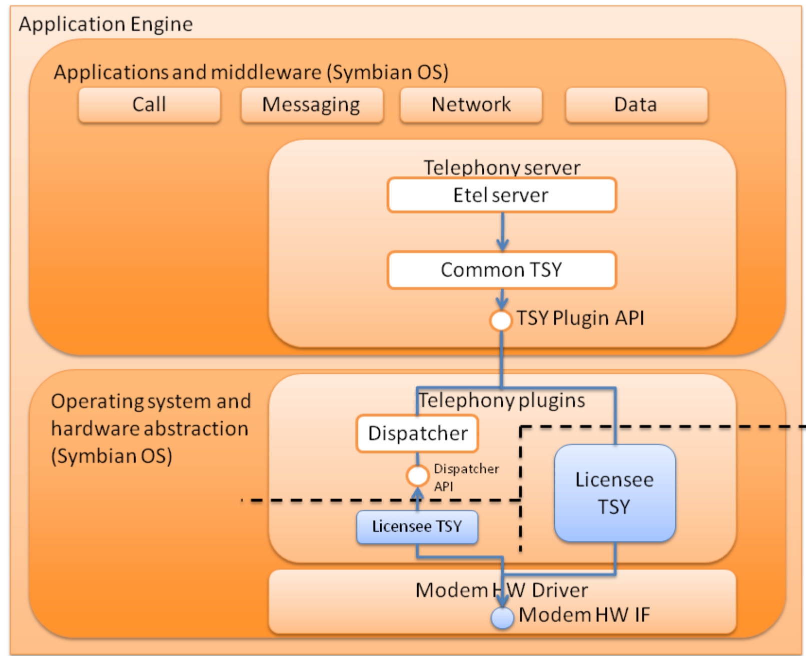

FIGURE 8. Symbian telephony stack components and a supplier implementation

3 MODEM SERVICES INTERFACE

Modem software interfaces are very specific to a modem supplier and operating system that is used within a modem. The most famous software interface is the AT command based communication, but there are numerous different supplier proprietary specifications as well. No public documentation exists about any other than the AT command based interface, and therefore this document uses AT commands to describe some parts of the modem interface.

There are no specific requirements about the modem software interface. Typically, a modem software interface contains only simple control messages, and therefore there are no strict performance or bandwidth requirements. Most requirements, or preferences, come from the modem use cases, which may require communication schemas described in figure 9, figure 10 and figure 11.

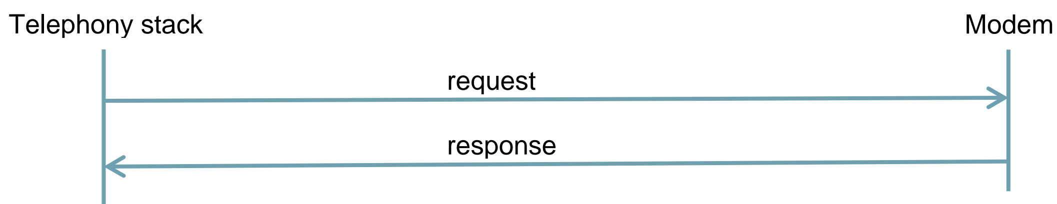

FIGURE 9. Telephony stack receives one response to a request

The scenario in FIGURE 9 completes an action in a single response. This can be applied only to such use cases that are completed almost instantly and they can not have any other state changes afterwards.

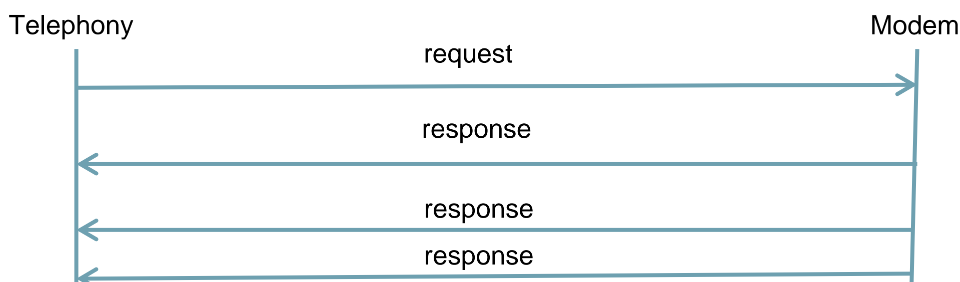

FIGURE 10. Telephony stack receives multiple responses to a single request

A call creation use case may follow the sequence described in figure 10. A telephony stack requests the call creation and gets responses about the connection being established, successful connection and disconnection.

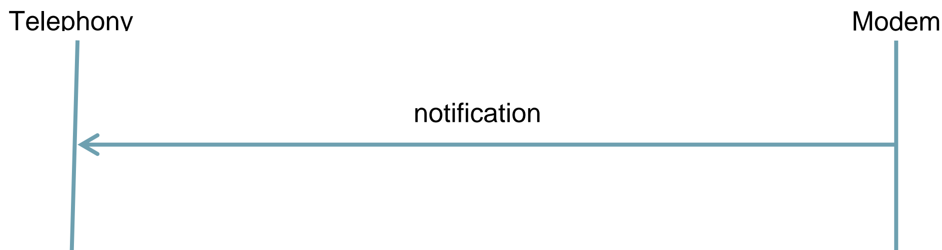

FIGURE 11. Modem notifies telephony stack about some modem originated event

A notification always originates from a modem. This is typical when receiving an SMS or an incoming call. Typically, the notification requires some further communication from the telephony stack to complete the use case. For example, the notification of the incoming call requires that the telephony stack request either a call answer or a rejection.

3.1 AT command interface

The AT command interface has been a standard way to control modems as PC peripherals. Hayes published the first AT command sets for their modem in the late 1970's, and the standardized AT command set has evolved since. Now the AT command set consists of tens of different commands and parameters.

The AT commands have been a standard way to access modem features, and therefore it is the reference communication interface for example in Android and other mobile phone platforms. Ever since the first versions of the Hayes AT command set, there have been manufacturer specific additions to the AT commands. Therefore the AT command implementation and support is somewhat fragmented across the industry. In addition to the missing standardized AT commands, there can be some customized commands for special hardware or software features.

The AT commands are technically old-fashioned as a modern device internal communication framework. They have their origins as a more high-level communication protocol, and therefore they are lacking support for the more in-depth communication protocol requirements, such as in-built service discovery, multiplexing, security, broadcasting and non-blocking, asynchronous communication for more advanced and optimized communication between an application processor and a modem.

Traditionally, the AT commands and data are transferred through the same communication channel, and therefore there are specific commands to switch between the data and the control mode. On modern AT command based devices there are virtual channels, which help the AT command multiplexing on parallel channels, but it still has its limitations due to the legacy design.

3.1.1 Interface implementation

AT commands are standard text strings containing prefix, command and parameters. AT commands behave as a synchronous messaging interface, and typically there is one AT interpreter in a device that will handle communication further. AT commands require virtual channel multiplexing, on both application and modem side, if there is a need to have asynchronous-like messaging functionality, where multiple clients can send and receive AT commands without being interrupted by the other clients.

read command for checking
current subparameter values
subparameter
command line prefix
extended commands are
delimited with semicolon
command line
termination character
ATCMD1 CMD2=12; +CMD1; +CMD2=,,15; +CMD2?; +CMD2=?<CR>
basic command
(no + prefix)
extended command
(prefixed with +)
subparameters
may be omitted
test command for checking
possible subparameter values

FIGURE 12. Basic structure of a command line (12)

Figure 12 is quoted from the 3GPP specification AT command set for User Equipment, and it demonstrates how an AT command string is constructed. Typically

AT commands are not as complicated as in this example. For example, a dial command to a number 12345678 can be expressed as simple as ATDT12345678 (12).

However, due to its textual format, AT commands can become very complex to parse if multiple commands and parameters are included in a single command. Even more challenging is the handling of the return codes. Usually, AT commands have specified certain, numerical, response strings, but in most of the cases the response is plain OK or ERROR. This implementation is feasible for traditional point-to-point communication as in a PC-to-modem connection, where the client and server can communicate synchronously and maintain each other's states. The communication becomes more complicated when a multi-client and multi-server communication is needed by using an asynchronous communication schema.

3.1.2 Standardization

In addition of the original Hayes AT command set, there are additional AT commands, defined by the ETSI and ITU-T organizations. It must be noted that a standardized AT command set is only a part of the truth, as there are manufacturer specific proprietary commands as well. Proprietary AT commands are not standardized and therefore it causes fragmentation between and inside the platforms even if the modem integration is 'AT compatible'. There must be a specific telephony stack for each modem supplier even if they all support the same standardized 3GPP AT command set.

More importantly 3GPP 27.007 AT COMMAND SET FOR USER EQUIPMENT contains commands that are not related to a modem only. There are some commands, for example, to control battery status, keypad and lights, which are not related to a modem at all. Therefore that specification cannot be used solely as a modem interface specification, but instead as a specification for a whole terminal from the PC software's point of view.

AT commands should be reallocated in the way that they form an external interface towards PC applications, and the application – modem interface is specified by using some more feasible and efficient technology.

3.2 Nokia Wireless Modem API

Nokia Wireless Modem API has been published as an alternative modem software interface. This interface contains at least the same functionality as the AT command interface, but the messaging format and the communication protocol are different from the AT multiplexing. This interface is not included in this analysis, but more information can be obtained from the address www.wirelessmodemapi.com (13).

4 MODEM SERVICES

The modem peripheral provides telephony services, but it requires some system level services to make a successful modem integration to an application environment. These system integration services are valid for any peripheral, like Bluetooth or WLAN, and not only for a modem peripheral.

4.1 Introduction to modem services

The most obvious part of modem services is all functions related to 3GPP standards and cellular communication. A short messaging service is used as an example to demonstrate some of the differences between these platforms. The 3GPP functionality contains hundreds of pages of the 3GPP standardization organization, and therefore it is not possible to be fully included in this thesis. Those services can be roughly divided into the feature groups shown in table 1 below:

TABLE 1. High level grouping of 3GPP features

Feature	Description
AT interpreter	AT interpreter services to comply with 3GPP requirements for AT command handling.
Assisted GPS	3GPP functionality of AGPS services.
Call handling	Call control and handling. (e.g. ATD, ATA, ATH, ATS0)
Data transfer	Packet data services (e.g. AT+CGDCONT, AT+CGDATA)
Network handling	Network services (e.g. AT+CREG, AT+COPS, AT+CLIP)
SMS handling	SMS control, sending receiving (e.g. AT+CSMS, AT+CMGS, AT+CNMI)
Supplementary services	Supplementary services e.g. for handling network owned configuration data.
UICC control	UICC control, UICC phonebook, (e.g. AT+CPIN, AT+CPBx)

It is not possible to go through all of those feature groups and therefore only SMS handling and specifically basic SMS sending is used as an example of the message sequences for each platform. This should demonstrate the communication flow and the related software components at a sufficient level. The AT command interface for SMS service is defined in the 3GPP 27.005 specification (14).

4.1.1 SMS Protocol

The basic SMS functionality, from the mobile equipment's and a network's points of view, uses two key message types; SMS-SUBMIT and SMS-DELIVER (15).

SMS-SUBMIT is used when the message is sent from the device to a short message service centre (SMSC). This is the so called SMS MO (mobile originated) messaging and sent every time a user is sending an SMS. In case of concatenated messages, one long message may require sending of several SMS-SUBMIT messages. Depending on the selected character set, it may require even more, because Unicode characters require at least twice the space compared to GSM7bit characters.

SMS-DELIVER is used when the message is forwarded from SMSC to a receiving phone. This is SMS MT (mobile terminated) messaging and visible on the phone UI when a device indicates about the new received SMS message. Concatenated messages appear as one single message, even if it is received as a multiple SMS-DELIVER message.

As shown in figure 13, in addition to those two messages there is a third message type; SMS-STATUS-REPORT. This is sent from SMSC to an originating phone, when SMS-SUBMIT contained a request for a status report. These messages are typically visible as delivery reports on a phone UI containing a timestamp and receiver address information. Requesting SMS-STATUS-REPORT is typically a user changeable setting on the originating phone UI.

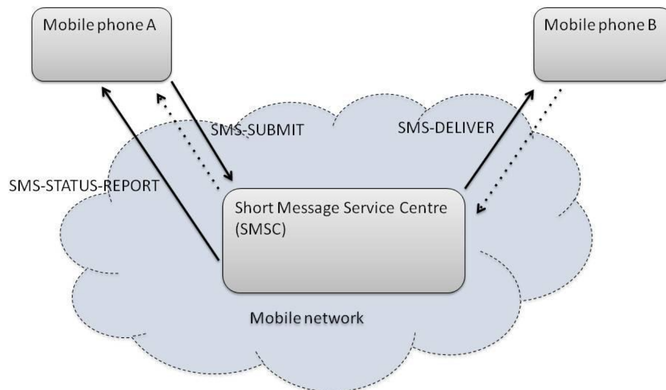

FIGURE 13. Overview of message flow between a mobile phone and an SMSC

All of this communication between a device and an SMSC is handled by a modem, and a modem is responsible for providing a software interface that can be used by an application environment modem adaptation layer, and further by a telephony stack. Typically, application environment interfaces have a higher abstraction and therefore most of the 3GPP functionality and modem interfacing details are hidden inside a telephony middleware or lower layers.

4.1.2 Android

Android ril.h defines the SMS message sending as described below. This specifies the communication content between the telephony manager and a supplier RIL as shown in figure 4 on page 18.

```
* RIL_REQUEST_SEND_SMS
*
* Send an SMS message
*
* "data" is const char **
* ((const char **)data)[0] is SMSC address in GSM BCD format prefixed
* by a length byte (as expected by TS 27.005) or NULL for default SMSC
* ((const char **)data)[1] is SMS in PDU format as an ASCII hex string
* less the SMSC address
```

** TP-Layer-Length is be "strlen(((const char **)data)[1])/2"*

```
* TP-Layer-Length is be "strlen(((const char **)data)[1])/2"
*
* "response" is a const RIL_SMS_Response *
*
* Based on the return error, caller decides to resend if sending sms
* fails. SMS_SEND_FAIL_RETRY means retry (i.e. error cause is 332)
* and GENERIC_FAILURE means no retry (i.e. error cause is 500)
*
* Valid errors:
*  SUCCESS
*  RADIO_NOT_AVAILABLE
*  SMS_SEND_FAIL_RETRY
*  FDN_CHECK_FAILURE
*  GENERIC_FAILURE
*
 * FIXME how do we specify TP-Message-Reference if we need to resend?
 */
#define RIL_REQUEST_SEND_SMS 25
```

(16.)

The SMS application in Android uses Java Telephony Manager API to access SMS functionality. Telephony Manager contains SMS Manager class, which handles all SMS specific functions. In this case, SMS Manager forwards the function call through the socket to the Radio Interface Layer and a supplier RIL. The middleware function for sending an SMS in Android begins in the following way:

```
public void sendTextMessage( String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)
```

(17)

A supplier RIL implements SMS sending, and in Android reference implementation this is handled as an AT command-based communication towards the modem. It must be noted that Supplier RIL is responsible for both, the AT command conversion and message routing, including connection creation, deletion and multiplexing.

4.1.3 MeeGo

In MeeGo, a modem chipset vendor has to implement oFono SMS submit function as described below. This function is used by oFono when calling a supplier specific modem plug-in.

```
void (*submit)(struct ofono_sms *sms, unsigned char *pdu,
int pdu_len, int tpdu_len, int mms, ofono_sms_submit_cb_t cb, void *data);

struct ofono_sms {
        int flags;
        DBusMessage *pending;
        struct ofono_phone_number sca;
        struct sms_assembly *assembly;
        unsigned int next_msg_id;
        guint ref;
        GQueue *txq;
        time_t last_mms;
        gint tx_source;
        struct ofono_message_waiting *mw;
        unsigned int mw_watch;
        struct ofono_sim *sim;
        GKeyFile *settings;
        char *imsi;
        const struct ofono_sms_driver *driver;
        void *driver_data;
        struct ofono_atom *atom;
 };
```

(18.)

ofono_sms structure contains much more detailed information compared to the Android SMS sending function and structure. oFono links SIM to a SMS sending request and this probably has at least two reasons. One reason is that the reading of the SMSC address is handled by using SIM instead of passing the SMSC address as a parameter. Another functionality may be related to storing of a message reference

(TP-MR) value to a SIM elementary file. This functionality is something that seems to be missing in the Android interface.

4.1.4 Symbian

Symbian CTSY Dispatcher defines the supplier interface in mltsydispatchsmsinterface.h header file (19) in the following way:

```
class MLtsyDispatchSmsSendSmsMessage : public MLtsyDispatchInterface
        {
        public:
        static const TInt KLtsyDispatchSmsSendSmsMessageApiId = KDispatchSmsFuncUnitId +
7;
        /**
         * The CTSY Dispatcher shall invoke this function on receiving the
EMobileSmsMessagingSendMessage
         * request from the CTSY.
         *
         * It is a request call that is completed by invoking
         * CCtsyDispatcherCallback::CallbackSmsSendSmsMessageComp()
         *
         * Implementation of this interface should allow a client to send a SMS message
         *
         * @param aDestination The mobile telephone number.
         * @param aSmsTpdu The SMS TPDU.
         * @param aDataFormat The SMS data format.
         * @param aGsmServiceCentre The service centre number.
         * @param aMore  This indicates whether the client is going to send another SMS
immediately
         *          after this one.
         *
         * @return KErrNone on success, otherwise another error code indicating the
         * failure.
         */
        virtual TInt HandleSendSmsMessageReqL( const RMobilePhone::TMobileAddress&
        aDestination,
          const TDesC8& aSmsTpdu,
          RMobileSmsMessaging::TMobileSmsDataFormat aDataFormat,
          const RMobilePhone::TMobileAddress& aGsmServiceCentre,
          TBool aMoreMessages
         ) = 0;
        }; // class MltsyDispatchSmsSendSmsMessage
```

(19.)

Symbian is somewhere between the Android and MeeGo definitions. The most visible difference is the usage of Symbian data types and classes.

4.1.5 Summary of findings

A quick comparison of only one service, and one specific function, is not enough to give a good understanding of similarities and differences. However, already this one example indicates that a source code or a binary reusability across all of these platforms is impossible to achieve. All of these examples have slightly different ways of partitioning parameters between a middleware and a supplier specific component. Symbian uses Symbian specific data types and classes; thus it is impossible to reuse a code from there. In addition to a reused code, there should be similarities on operating system services, processes, IPCs and frameworks. As a result of these findings, it is not feasible to continue to analyze the rest of the modem use cases at a source code level.

4.2 Modem System Services

Some analysis still has to be concluded outside the telephony area. To make a successful integration of the modem processor, there must be some system level services to control the various aspects of the actual modem processor at a system integration level. The most evident system service is a processor communication framework for software and hardware interfaces, such as AT multiplexer which enables the actual communication channels between these two peripherals. The others are the modem boot control to control the processor startup, cellular audio connectivity, software image transfer and energy management, including battery connectivity. In addition to the system level services, there are some supporting services especially for R&D and manufacturing purposes.

All of these system services are somewhat forgotten when discussing a modem integration either in a scope of Android, MeeGo or Symbian. The nature of these services is a common low-level system integration instead of the cellular communication, and probably therefore the main focus is always concentrated on the ‘real’ modem functionality, instead of these supporting services. Anyway, the same findings should be applicable for these services. There is no efficient way to increase the reusability in system service areas, and compared to the 3GPP protocol services,

there is even more fragmentation on the implementation because of the lack of specifications and requirements from the standardization organizations.

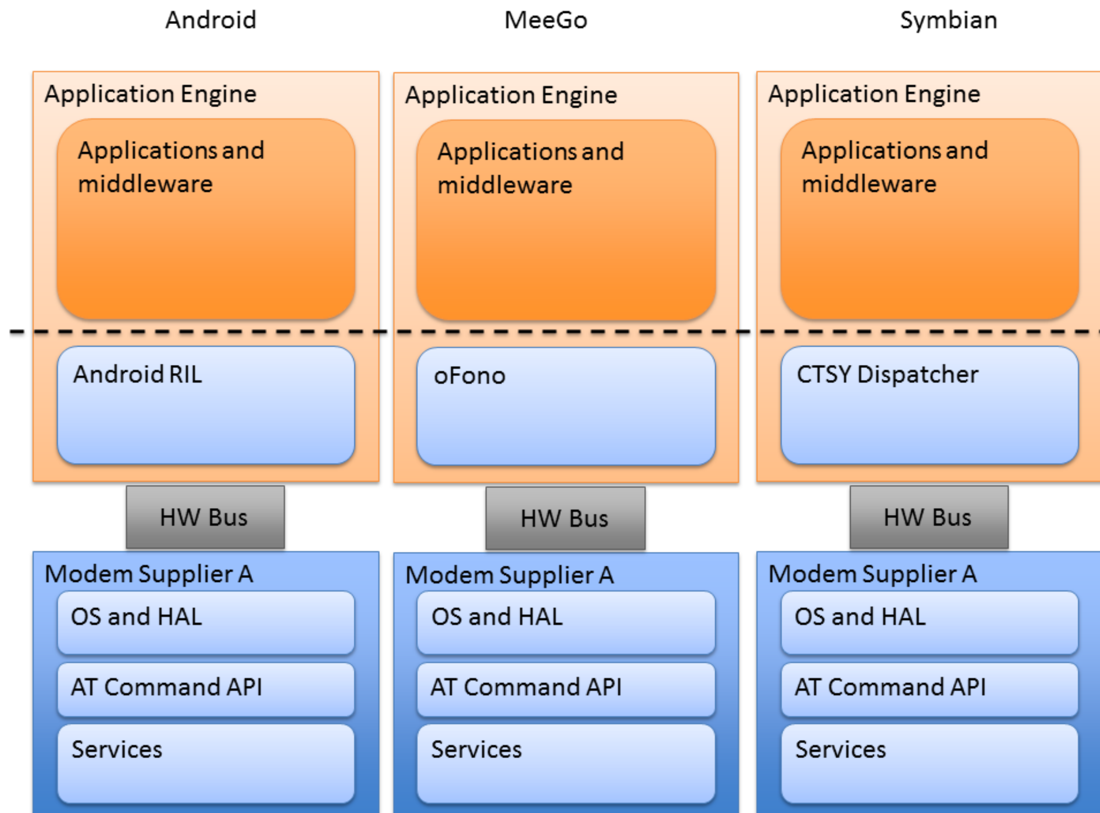

FIGURE 14. Modem supplier responsibility areas

5 STANDARDIZATION AND REUSABILITY

Standardized modem integration gives clear benefits across the mobile-device industry. An improved reusability enables such benefits that are not possible to achieve otherwise, and more importantly, these benefits are equally available for all the roles in the mobile device integration.

5.1 Business need for standardized modem interface

These rather cursory findings discussed in the previous chapter have led to an idea of improving the reusability by defining a lower level standard interface below the middleware integration APIs. This idea, and purpose, is similar to another cross-platform standardization work like OpenGL for graphics and OpenMAX for audio integration.

When looking at different integration layers in Android, MeeGo and Symbian, it can quite easily be seen that there is no easy way to make the modem integration more cross-platform supported on an application engine side. This is due to the fact that each platform uses its own unique software architecture, operating system and licensing model. More importantly, the interfaces between each layer are not consistent across the platforms. There can be some similarities in the MeeGo and Android modem integrations when using lower level Linux layers. However, different licensing models and telephony abstraction APIs at an application level anyway require a separate implementation, even if OS and the programming languages are the same.

Figure 14 highlights the responsibility areas for a modem supplier. A modem supplier has to implement and maintain the same amount of application engine adaptation layers as the amount of the platforms are that they are providing their modem hardware and software asset to. This is a major investment for a modem supplier as they need competences for many platforms, architectures and programming languages.

This problem is even more visible when including all global mobile phone environments; RIM, iPhone, Windows Mobile, Samsung Bada, and proprietary platforms. This problem is at some level valid also for desktop PCs, laptops, notebooks, tablets and any other device containing cellular modem functionality. The business trend is shifting towards the integration of a cellular modem into devices out of traditional mobile devices, and those devices need similar modem drivers or adaptation layers.

5.2 Shortcomings of AT commands

The interface based on AT commands is the only existing interface that defines a somewhat standardized functionality for the modem integration. This is the most wide-known and supported interface by any modem supplier. (12.)

The history of AT commands originates from external desktop PC modems where a communication has been simple and straightforward. AT commands have been transferred within a data stream, and the AT interpreter is switched between a data and a command mode. The traditional AT command interpreter blocks any other commands while the previous command is still being processed. This do not allow parallel communication and makes the adaptation behave as a synchronous communication protocol; a client has to wait until the command is processed because the server is not able to reply, or there is a risk that the state machines go out of synchronization.

It is possible to build an AT modem adaptation and AT modem interpreter in the way that it has a better support for an asynchronous communication in a multi-client environment by using multiple channels for the AT commands. The AT multiplexing protocol is specified in 3GPP 27.010 (20).

Multiplexing allows the creation of virtual AT command channels between an adaptation and a modem. Each virtual AT command is handled as a regular AT command channel as specified in 3GPP 27.007. By using this kind of multiplexing mechanism, call control, unsolicited messages and data flow can be separated into different streams and they do not have any impact on each other's states.

The AT command multiplexing causes some special cases that have to be handled in an adaptation and in a modem software. 3GPP 27.007 specifies some commands that are used to control how the AT interpreter behaves. Due to the virtual channels, there has to be as many virtual settings for each channel. For example, changing an error-verbose value on one channel should not have any impact on another channel error-verbose value. This also applies to e.g. S-register settings and NVM commands that are handled by the AT interpreter itself. There are also differences in how various modem suppliers create, and allocate, their virtual channels. Some suppliers may use only two channels for control and data, while some other's can create a separate channel for each feature area. This potentially makes the adaptation fragmented as the same multiplexer does not fit all suppliers.

The AT multiplexing protocol and AT messaging syntax is a valid and already well-known way to communicate with a modem. However, modem hardware and software are becoming more complex due to new radio technologies and features, and therefore it should be considered whether there is a better and more feasible way to implement modem multiplexing and message protocols.

It is extremely difficult to define a modern modem security, file system and device boot up control interfaces built on the top of string based AT commands. If AT commands would not exist already, it would be impossible to drive them as a standard, modern and device internal communication protocol as an advanced smartphone modem communication framework.

The AT command specification is already fragmented, and there is no single specification that could thoroughly specify the modem interface. Because of historical reasons, AT interface contains commands that are not relevant for an application engine and a modem communication. 3GPP specification, as named in User Equipment specification, focuses on the whole device instead of only modem hardware.

5.3 Open Modem Interface initiative

At a general level, the most appropriate solution to make an application engine and modem integration easier and more efficient is to initiate an open standardized modem interface specification. Such an interface is proposed as the appendices 1 and 2 in this thesis and it is called ‘Open Modem Interface’. Open Modem Interface would contain the system services, the multiplexing and the routing capabilities and the message specifications for a call creation, SMS, networking, system integration services and other 3GPP standardized functions.

To make it more efficient from the business perspective, Open Modem Interface should also contain standardized hardware interfaces. This would enable a more flexible integration into any device.

As can be seen in figure 15, Open Modem Interface allows a modem supplier to concentrate only on modem software as long as the application environments are able to provide community supported adaptation layers. The modem supplier has to implement a layer of Open Modem Interface services and a hardware abstraction in the way that it fits the supplier's internal hardware and software architecture.

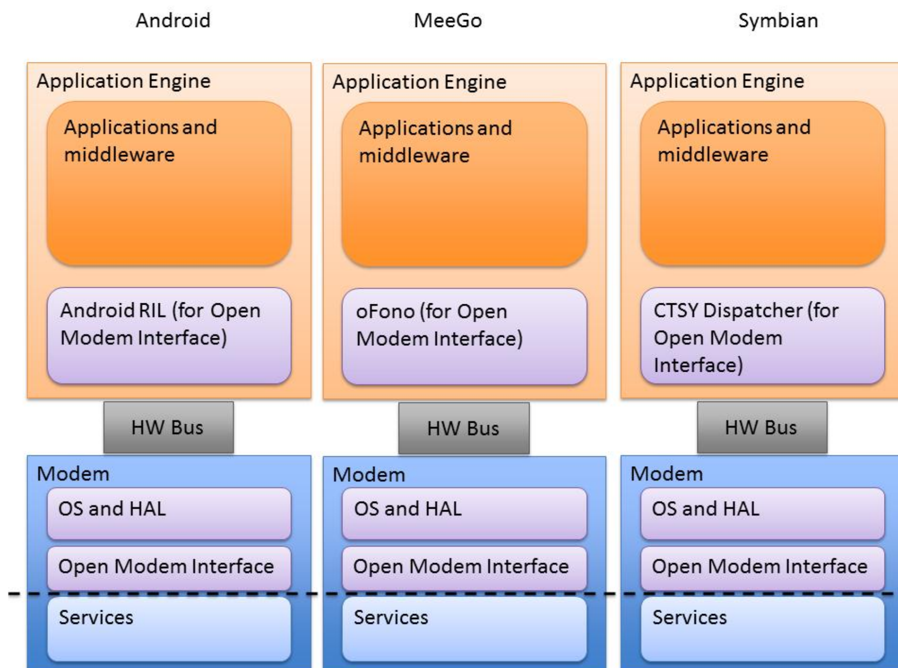

FIGURE 15. Open Modem Interface responsibilities

5.3.1 Open standard

Open standardization allows a rapid interface development and a light-weight interface management. Platform integrators and modem suppliers can contribute to Open Modem Interface specification and participate an interface road mapping process in a co-operative way. Adding new features must be a straightforward process in such a way that supplier-specific additions or exceptions are not needed, or the need for supplier-specific additions are minimized.

Open Modem Interface must be kept as a platform and OS agnostic. The licensing model should be kept such that it allows a proprietary implementation, but also makes sure that all specification improvements and changes are contributed back to a specification.

5.3.2 Benefits for modem suppliers

The most obvious and concrete benefit of adopting Open Modem Interface from the modem supplier's perspective, is the possibility to focus only on the modem hardware and software asset. As already described in figure 15, one modem implementation fits perfectly well any application engine that has the necessary adaptation layer available. When Open Modem Interface gets more publicity, it can be assumed that a compatible adaptation layer is available for all mainstream platforms, and even for a closed source. This will improve the overall efficiency of the R&D because there is no need to create competencies for various platforms. A supplier can focus on their key competences and own innovation areas. This will also improve the time-to-market because there is one complicated layer less to productize.

A standardization organization can provide a robust test environment in the way that it can be used by all suppliers. In this case, there is a significant saving because the same test tools can be used to verify a modem delivery for Android, MeeGo, Symbian or any other platform. Test tools can be seen as one more application environment and the test cases verify the modem functionality against the standardized specifications. Of course, this way the adaptation layer testing is left on the platform's or the

product's integrator responsibility, but because of its open source nature, it can be assumed that Open Modem Interface adaptation quite easily becomes mature in each of the platforms. The same maturity aspect applies to a testing tool, which becomes mature and is improved by contributions from different parties; platforms, product integrators and modem suppliers.

From the business perspective this can open new opportunities as it allows a modem supplier to sell the modem effortlessly to any platform. This is a great advantage, because it decreases the risk of taking part in new products. There is no need to have application environment specific competences, and the scheduling risk is decreased while the quality is improved.

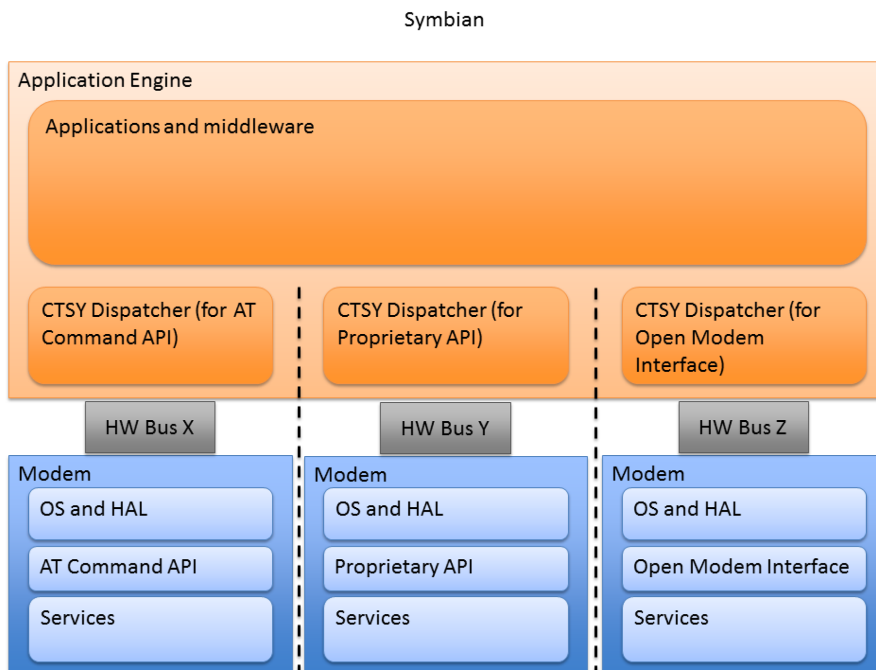

FIGURE 16. Open Modem API is like any other modem adaptation

A drawback of this solution is that there has to be an implementation of the Open Modem Interface layer on a modem software asset. Depending on the actual Open Modem Interface specification, this layer should be very simple, thin and straightforward to implement. An investment for a new layer in the modem software asset is considered quite small compared to an investment required when Open Modem Interface is not adopted, and the adaptation has to be implemented on an application engine side.

5.3.3 Benefits for application engine platforms

From the Android, MeeGo or Symbian platform's perspective standardized adaptation is only another modem adaptation as described in figure 16. There is no need to redefine the adaptation layers or APIs, because Open Modem Interface is at the same level as AT commands or some proprietary interface.
There is still need for RIL, oFono and Symbian CTSY Dispatcher because some modem suppliers may still prefer to have their own proprietary adaptation and modem interfaces. Open Modem Interface should be seen as an alternative integration API for those modem suppliers who do not have any interest to become involved with the adaptation layer implementation. Therefore this proposal do not replace or change anything on the application engine side.

5.3.4 Benefits for product integrators

Product integrators have similar benefits as modem suppliers. A product integrator gets more freedom to choose between the modem suppliers because they all fit the rest of the system easily. This way the modem interface is abstracted from the supplier selection, and the selection can be based purely on features, performance and other aspects without any need to concentrate on a risk of the platform compatibility, work effort estimations, quality and time-to-market.

Figure 17 demonstrates how a product integrator can freely choose between modems X, Y and Z. Symbian CTSY Dispatcher may be already well productized by using some other Open Modem Interface compatible modem. Therefore the decision about the new modem supplier can purely be based on features such as size, power

consumption, price and other technical features. This also efficiently prevents lock-ins to a specific supplier and should increase the competition and productivity in the industry. At the same time, it also increases the business opportunities for modem suppliers, because they can more easily provide their modems to new environments.

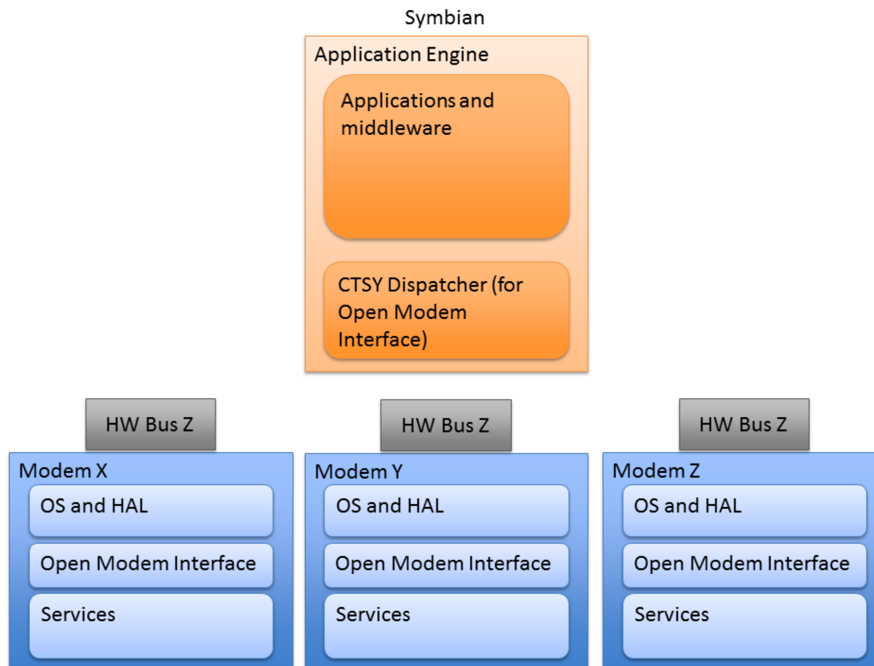

FIGURE 17. Open Modem API increases possibilities for product integrators

5.3.5 Technical characteristics

This section describes some technical requirements and implementation ideas at a conceptual level. The following statements can be considered as key features for Open Modem Interface:

- Open – free to use and to contribute.
- Multiplexing – the multiplexer must be efficient and easy to extend. It must support asynchronous communication between multiple clients and communication in multi-peripheral configurations.
- Detailed – the specification must contain enough of the detail level, including the message sequences, in the way that fragmented implementation is not possible and not allowed.
- Simple – the specification should have high abstraction level in the way that only the needed information is passed between an adaptation and a modem.

Complex data structures must be avoided to make the implementation of the adaptation layer easier. Hardware or OS dependencies are not allowed.

- Expandable syntax – the message format must be expandable in the way that new features can be added easily and approved by using lightweight processes.
- Runtime configurability – there should be a mechanism to make a configuration handshaking between an adaptation and a modem. This would allow shared configuration information in the way that the adaptation layer can identify and differentiate its behavior based on the features supported by a modem.
- Supplier specific extensions not preferred – this would obsolete adaptation compatibility between suppliers. However, there might be some benefits of having a supplier extension specifically in the R&D feature area or for testing purposes.

5.4 Device Interconnect Protocol based industry standard

One candidate for the industry standard modem integration is a solution which is based on NoTA Device Interconnect Protocol. DIP is an open source software architecture, where a hardware abstraction, networking layers and services are separated. NoTA is initiated by Nokia Research Center for easier integration, and its purpose is to provide an easy and a harmonized way to integrate any peripherals and services together. (21.)

Due to importance of the subject of this thesis, separate white papers have been created in conjunction with this master thesis. These white papers, that are provided as appendices 1 and 2 in this thesis, describe business and technical aspects of the proposed NoTA-based solution and define some example interfaces for those basic use cases that are already analyzed within this thesis. The white papers conclude this thesis work by adding detailed descriptions of the challenges, architectures and technical solutions for the integration problems.

6 CONCLUSION

The master thesis conclusion is divided in the way that it highlights some technical problems, standardization related discussions and concerns as well as some personal findings related to this thesis work.

6.1 Technical findings

The lessons learnt from this thesis are that most of the modem adaptation layers and the modem software assets are similar when looking at the high level features and software partitioning. The differences become visible when a more detailed implementation is compared between the platforms. For example, supporting an AT command based modem in all platforms requires a major work effort even if the features and the basic principles are the same. Each middleware and the telephony integration APIs are behaving somewhat differently, therefore causing some complexity for the cross-platform reusability. There is no easy way to increase the reusability on the adaptation layers, and therefore an industry standard and a harmonized modem interface seems to be the only relevant and technically feasible solution for the increased reusability. As described in the attached white papers (see Appendices 1 and 2), there are significant benefits of using the industry-defined and harmonized modem interfaces.

The DIP based solution is a strong candidate, because it already provides a basic framework for the hardware abstraction, communication, routing and service definitions. It is already open sourced with a flexible BSD license, thereby supporting any licensing model and software architecture partitioning. This technology proposal is a promising candidate, and it will be studied further in more detailed studies.

6.2 Open standardization initiative and industry feedback

The most difficult part of the Open Modem Interface has been the initiative itself. There were surprisingly many non-technical problems to be solved, before the industry discussions and a standardization proposal could be proceeded with.

There are many stakeholders to whom the problems and solutions have to be sold. Typically, each of those stakeholders has somewhat different viewpoint; the justifications and the benefits have to be highlighted in a different way. This means that typically different kind of materials and presentations had to be used, when discussing this idea with the application engine OS providers, the product integrators and the modem suppliers.

One of the most critical questions, when initiating these kinds of discussions is to emphasize the nature of the discussion. In other words, these proposals are really targeted as a long-term solution proposal, without interfering the existing discussions about ongoing projects, interface definitions and architectural decisions. Long-term targets typically conflict with the short-term targets, and there have to be detailed plans how to drive these discussions forward in a coherent way. There are always risks that the long-term targets does not get sufficient focus due to the higher priorities in short-term plans; this is something that is probably evident in all software projects and not only in this specific modem integration case.

Now there is some kind of an industry-wide agreement that the problems are there and something should be done in the context of the first whitepaper (see Appendix 1). The technical solutions are open – and the biggest concern is how to initiate such a standard that is applicable for all parties. One part of that discussion will be the operational model of the standardization; what is the forum to control and maintain these modem interfaces.

These discussions will continue, and it can be estimated to be a long process before there will be open standard interfaces, standardization forum, adaptations and modems using this new technology.

6.3 Personal findings

My decision of including Android, MeeGo and Symbian into this study was somewhat too ambitious, because it meant even more work when analyzing the

software assets, and eventually forced me to decrease the depth of the analysis, in order to keep the work effort reasonable. The inclusion of so many platforms was a wrong decision from the detail level's point of view, but on the other hand it was necessary to include multiple platforms, because the final findings would have been difficult to justify based on only a single platform. The generic understanding of Android, MeeGo and Symbian adaptation layers helped me to justify and write the white papers more deeply and with better reasoning.

My initial hypothesis for this work was to focus on the application environment adaptation layer source codes. I planned to study software architectures, classes, methods, parameters etc. to see what kind of implementation differences there are. However, I quite soon realized that each of those platforms are similar at the principal level and very different at the implementation level. I started to consider whether such implementation comparison provides any real benefits in a thesis. Based on those initial findings, there are no significant differences between the different telephony stack implementations from the product creation's point of view. Symbian seems to be ahead of the rest due to its mature and complete feature offering, but on the product integration's point of view, those implementation details are not critical on the whole product's perspective. Therefore, I adjusted my focus area on reusability questions such as how easy it is to reuse a modem engine between these different platforms.

This view proved to be a very good decision, because it led to the creation of two whitepapers and the solutions how the industry wide benefits can be achieved for all parties. The parties, in this context, mean all application environments, including closed source platforms, product integrators, any application engine, any operating system or a modem supplier. If the industry standard open modem interface is initiated and adopted, it will have a significant impact across the business environment, and it is a meaningful and a valuable result in a master thesis. Personally, I am happy with the contents and the workflow of the white papers.

Overall, I am satisfied with the result – this is not a final and a complete solution to all integration problems, but it gives a glimpse into the modem integration challenges

and demonstrates how the existing open source platforms can benefit of the proposed solution. In the best case, this work will result as an industry standard interface for the modem integration, which can be considered as a major achievement. This work will continue with more detailed specifications of the modem services, and some ideas are already visible in the second white paper (see Appendix 2).

REFERENCES

1. Nokia, Intel. Nokia Press Release.
   Nokia
   Available at:
   http://www.nokia.com/press/press-releases/showpressrelease?newsid=1384419.
   Date of data acquisition 1-Jun-2010
2. Apache Software Foundation. Apache License, Version 2.0.
   Apache Software Foundation
   Available at:
   http://www.apache.org/licenses/LICENSE-2.0.
   Date of data acquisition 3-Jun-2010
3. Free Software Foundation. The GNU Operating System.
   GNU General Public License, version 2.
   Available at:
   http://www.gnu.org/licenses/gpl-2.0.html.
   Date of data acquisition 11-Aug-2010
4. Eclipse Foundation. Eclipse Public License - v 1.0.
   Eclipse Foundation.
   Available at:
   http://www.eclipse.org/org/documents/epl-v10.php.
   Date of data acquisition 11-Aug-2010
5. Open Handset Alliance. Industry Leaders Announce Open Platform for Mobile Devices.
   Open Handset Alliance.
   Available at:
   http://www.openhandsetalliance.com/press_110507.html.
   Date of data acquisition 11-Aug-2010
6. Intel, Nokia. oFono.
   oFono Open Source Telephony
   Available at:
   http://www.ofono.org

Date of data acquisition 23-Sep-2010

7. Intel Nokia. Announcing ofono.org.
oFono Open Source Telephony.
Available at:
http://ofono.org/blogs/holtmann/2009/announcing-ofonoorg.
Date of data acquisition 23-Sep-2010

8. freedesktop.org. freedesktop.prg - Software/dbus.
freedesktop.org
Available at:
http://www.freedesktop.org/wiki/Software/dbus.
Date of data acquisition 23-Sep-2010

9. Symbian Foundation. Important dates in Symbian History.
Symbian
Available at:
http://www.symbian.org/about-us/history-symbian.
Date of data acquisition 02-Apr-2010

10. Symbian Foundation to Transition to a Licensing Operation.
symbian.org.
Available at:
http://www.symbian.org/news-and-media/2010/11/08/symbian-foundation-transition-licensing-operation.
Date of data acquisition 23-Jan-2011

11. Cellular Baseband Services Guide in Symbian OS Guide
Symbian Foundation
Available at:
http://developer.symbian.org/main/documentation/reference/s%5E3/doc_source/guide/Telephony/index.html
Date of data acquisition 01-Aug-2010

12. 3GPP. 3GPP specification: 27.007.
3GPP A Global Initiative
Available at:
http://www.3gpp.org/ftp/specs/html-INFO/27007.htm.
Date of data acquisition 01-Oct-2009

13. Nokia. Wireless Modem API.
Wireless Modem API
Available at:
http://www.wirelessmodemapi.com/.
Date of data acquisition 01-Oct-2009

14. 3GPP. 3GPP specification: 27.005.
3GPP A Global Initiative
Available at:
http://www.3gpp.org/ftp/specs/html-INFO/27005.htm.
Date of data acquisition 01-Oct-2009

15. 3GPP specification 23.040.
3GPP A Global Initiative
Available at:
http://www.3gpp.org/ftp/Specs/html-info/23040.htm.
Date of data acquisition 01-Oct-2009

16. Google Inc. Android Open Source Project - ril.h.
android.git.kernel.org
Available at:
http://android.git.kernel.org/?p=platform/hardware/ril.git;a=blob;f=include/telephony/ril.h;h=e921fa60d5e9566868ecacfaf5cd3034a845badb;hb=HEAD.
Date of data acquisition 10-Dec-2010

17. Android Open Source Project - Telephony framework.
android.git.kernel.org
Available at:
http://android.git.kernel.org/?p=platform/frameworks/base.git;a=tree;f=telephony/java/android/telephony;h=96491f218ff91854ba602eecef42fa9ddb72ea38;hb=refs/heads/master.
Date of data acquisition 10-Dec-2010

18. oFono. network/ofono/ofono.git.
git.kernel.org / ofono
Available at:
http://git.kernel.org/?p=network/ofono/ofono.git;a=tree;hb=ddd5582a5476c6d595d6b654da3e74824fdbb2a8.

Date of data acquisition 01-Oct-2010

19. Symbian Foundation. mltsydispatchsmsinterface.h.
Symbian Developer - mltsydispatchsmsinterface.h.
Available at:
https://developer.symbian.org/oss/MCL/sf/os/cellularsrv/file/3553901f7fa8/telephonyserverplugins/ctsydispatchlayer/exportinc/mltsydispatchsmsinterface.h.
Date of data acquisition 01-Oct-2010

20. 3GPP. 3GPP specification: 27.010.
3GPP Global Initiative
Available at:
http://www.3gpp.org/FTP/Specs/html-info/27010.htm.
Date of data acquisition 01-Oct-2009

21. NoTA World. NoTA - Architecture.
NoTA
Available at:
http://www.notaworld.org/nota/architecture
Date of data acquisition 11-Mar-2010

APPENDICES

Appendix 1. Open Modem Interface Benefits for the Mobile Device Industry (24 pages)

Appendix 2. Open Modem Interface Proposal Based on Device Interconnect Protocol (40 pages)

Forum.Nokia

Open Modem Interface Benefits for the Mobile Device Industry

White Paper

Document created on December 7, 2010
Version 1.0

NOKIA

Forum.Nokia

Table of contents

1. Introduction 3
2. Modem integration challenges 6
3. Modem integration architecture standardisation 8
   3.1 Software architecture 9
   3.2 Features and services 10
   3.3 Interface specifications 10
   3.4 Interface management 11
4. Open standard modem services 12
   4.1 Modem Protocol Services 12
   4.2 System services 13
      4.2.1 Audio services 13
      4.2.2 Boot and state control service 13
      4.2.3 Configuration service 13
      4.2.4 File system service 13
      4.2.5 Power management service 14
      4.2.6 Security service 15
   4.3 UICC service 16
   4.4 R&D services 16
5. Industry-wide benefits 17
   5.1 R&D efficiency 17
   5.2 Software flexibility 18
   5.3 Time to market and quality 19
   5.4 Cost efficiency and cooperation 19
6. Open issues 20
7. Conclusions 21
8. Terms and abbreviations 22

Change history

December 5, 2010	1.0	Initial document release

Forum.Nokia

1. Introduction

Mobile phone platforms do not have efficient reusability possibilities for modem hardware and software adaptations when doing cross-platform integration between multiple device platforms such as Symbian, MeeGo, Android, or any other open or closed proprietary mobile phone platforms.

Android RIL, MeeGo oFono, and Symbian CTSY Dispatcher all provide APIs for supplier-specific adaptation layers on the application environment side. However, the telephony aspect is only one part of the full modem integration effort, and the lack of a sufficient reusability mechanism results in unnecessary and duplicated efforts to integrate modem functionality as a cross-platform implementation. This is especially true for modem suppliers that provide modems to multiple platforms.

Due to differences in software and licensing models on the application engine side, the most feasible solution for easing modem integration is to initiate an open standard specification that defines the open standard modem services interface for modem functionality. This allows modem suppliers to focus on their key areas without any need to develop and deliver application-specific components. It is a solution that will have significant impact across the mobile device industry, because one complex and fragmented subsystem area can be harmonised and standardised, making product integration substantially faster, cheaper, and of higher quality. It will also create new and more beneficial business scenarios for both device integrators and modem suppliers.

This document focuses on industry benefits, generic architecture, requirements, and modem-related services. Therefore, it does not propose any technologies as ready-made solutions. No specific application platform is used as an example, since, on a general and theoretical level, these findings are applicable to any existing platform, including open and closed source.

Diagrams are drawn so that corresponding layers can be easily matched to Symbian, MeeGo, or Android. This proposal and these diagrams apply to proprietary architectures as well, on the assumption that there is sufficient layering available for modem service adaptation and hardware abstraction.

Forum.Nokia

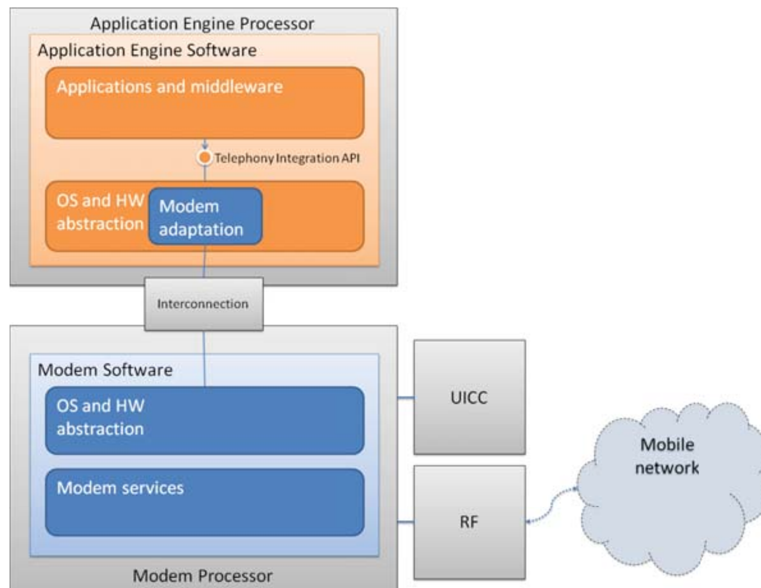

Figure 1: Hardware partitioning of a typical mobile device. The application engine and modem processor are connected through some form of hardware bus. The modem adaptation integrates modem-specific interfaces into telephony middleware APIs. The modem controls cellular network connectivity.

This document does not focus on hardware, although the hardware is closely related to the modem integration, specifically the hardware bus that is used to connect these two systems together. In System-on-a-Chip (SoC) solutions, where these two systems are integrated to the same chip, shared memory can be used as a hardware connection. When the modem is integrated as a separate chip, then the hardware bus can be based on Uniport, UART, USB, or some other standardised or proprietary hardware interface.

This hardware connection selection should be visible only at a very low level in the application and modem software stacks. There are hardware abstraction APIs that are specific to each platform; typically the modem supplier needs to provide the software implementation of the modem hardware connection on the application side.

In addition to the hardware connection, there are dependencies on the modem hardware capabilities. Generally, these are other hardware components or subsystems including UICC, flash memory for file system, GPS for AGPS services, security hardware for authentication and secure storage, and other hardware-related services that have dependencies either inside the modem or on the application engine. Regarding integration details, these hardware solutions are even more important than standardised 3GPP protocol functionality because each modem supplier typically has its own specific implementation. These differences in architecture cause fragmentation between suppliers, and can lead to higher implementation effort on both the application and modem software stacks during product integration.

This document describes the benefits of having standardised interfaces and architecture. It also shows the advantages of having processes to define, control, and deliver all relevant functionality regarding the modem peripheral software and hardware integration.

Forum.Nokia

2. Modem integration challenges

Modem integration consists of all software and hardware integration efforts that are needed when integrating modem functionality into a mobile device. Generally speaking, this functionality usually focuses on 3GPP specifications and the telephony middleware. For example, Android RIL, oFono, and Symbian CTSY Dispatcher focus only on the telephony area, and therefore do not describe the full effort needed for modem integration. Figure 2 shows all key layers of modem integration to any mobile platform.

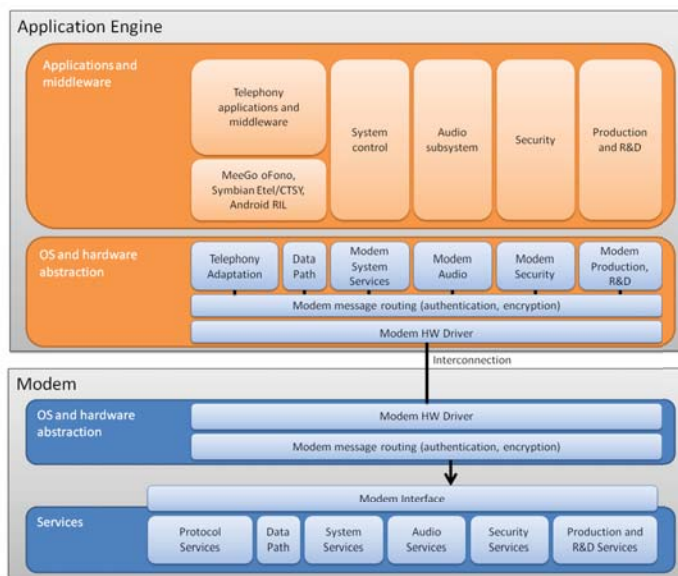

Figure 2: There are different feature areas that need to be handled and specified when integrating a modem peripheral to an application engine.

Modem integration always requires effort on the application engine side because there are no available standardised adaptations and interfaces. These would define an industry-wide mechanism to connect and access modem functionality. Insufficient standardisation means there are many different hardware connections and software stacks for modem integration. The modem software stack, specifically software interfaces and message routing protocol, is then reflected to the adaptation implementation, for example, in the Android RIL or oFono modem plug-in component, which needs to be modem specific.

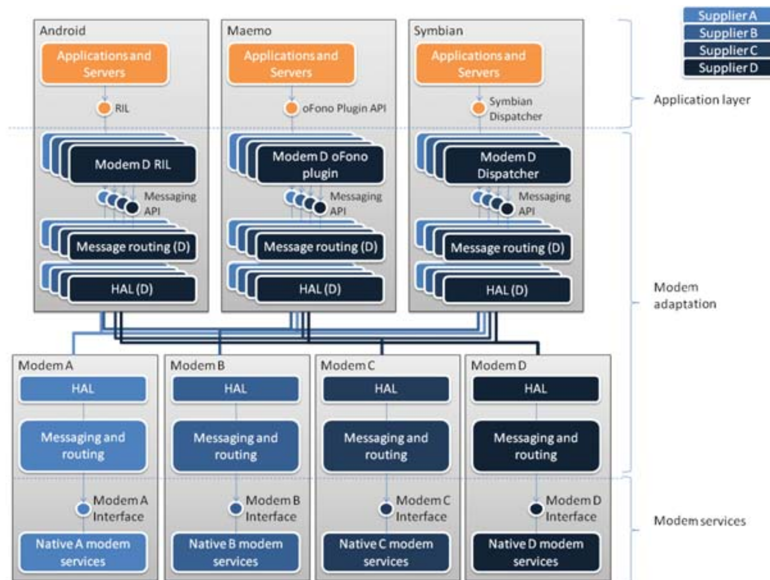

Figure 3: Modem suppliers A, B, C, and D must implement and maintain multiple different adaptation layers (Android, MeeGo, and Symbian), which causes unnecessary work effort and a need for new host OS specific competences and investments.

Implementing an AT command-based modem by using 3GPP 27.007 and 3GPP 27.010 specifications may provide some reusability possibilities across different modem suppliers. However, there is still a significant amount of work needed for integration, because the 3GPP 27 series specification usually fulfills only the telephony area; other areas are handled by using supplier-specific AT command additions, which typically leads to custom software architecture. AT commands and the AT multiplexer are not optimal for current and upcoming modem requirements where more complicated protocols are needed, or when a simple text-based message payload is not enough.

In addition to the AT command interface, there are many different modem supplier-specific interfaces. As shown in Figure 3, the worst case for a modem supplier is that it must implement and maintain multiple application engine implementations just to be able to provide modems for each platform (for example, when supporting one specific modem A, B, C, or D, there need to be separate adaptations for Android, MeeGo, and Symbian). For the modem supplier, this requires a major investment in terms of resourcing, monetary investment, and host OS specific competence development. All of those investments and risks may reflect to a final product program and a product integrator as delayed schedules, unexpected requirements and late architectural changes.

Forum.Nokia

3. Modem integration architecture standardisation

Figure 4 illustrates an overview of a situation where open standard and widely adopted modem integration interfaces, services, and architecture are used. As shown, there exist community-maintained adaptation components for open standard modems, allowing a modem supplier to focus solely on modem deliveries without needing to get involved in the architectures or deliveries of the application engine. A modem can implement an open standard interface as a native interface, as in modems C and D, or keep its native APIs and provide this new interface by using a wrapper layer.

In the interest of simplicity, Figure 4 demonstrates only the telephony area. The full integration affects many more areas, as described later in this document.

The new standardised architecture must comply with the following requirements and characteristics to make it a relevant alternative for any modem supplier-specific interfaces.

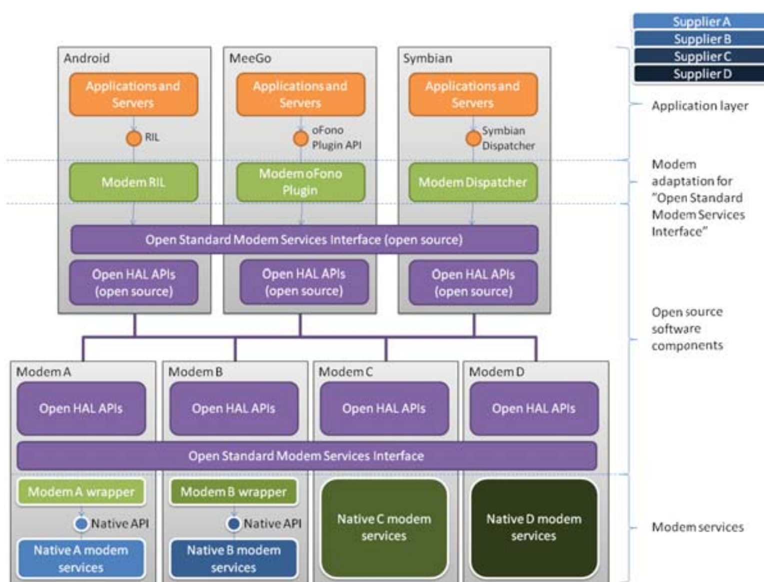

Figure 4: The modem integration effort becomes more focused when adaptations can be reused for standardised modem deliveries.

Forum.Nokia

3.1 Software architecture

The software architecture must be clearly layered to enable an expandable architecture and to define clear functional responsibilities. It must be possible to change or expand the hardware abstraction, message routing, and services without any changes (excluding configuration) to other layers.

The modem integration architecture, including defined services, must be easily integrated to any operating system or programming language. This applies to existing open-source mobile platforms, such as Android, MeeGo, and Symbian, but also to any proprietary application environments and modem software assets.

The architecture must enable reliable security services so that a client and services can authenticate and trust the communication between them. There should be no back doors or other security vulnerabilities, especially in system-critical services such as modem secure storage, production tools, or R&D features. Production tools, for example, may be used to configure and adjust RF band settings through a device's API. This API may be vulnerable to an attacker who has the intention of making the device transmit on an invalid RF band. An example of R&D features is the use of trace facilities that are used to debug device errors during development. These trace facilities may be misused by an attacker who wishes to discover the content and timing of data movements within the device in order to optimise an attack.

The Modem Service interface must be open standard, and it should provide open-sourced implementations by using business-friendly licensees, allowing integration to any closed source or permissive or nonpermissive open-source software environments.

The architecture must provide an efficient configurability for services. It must be possible to discover available services and change the adaptation functionality based on configuration data. Services should be configurable between an application engine, a modem, or some other peripheral. For example, connecting a UICC card on either an application or a modem processor should be a straightforward configuration that doesn't affect the rest of the system.

The architecture must support multi-modem scenarios, where two or more modems are integrated to a single application processor. This is a critical business case, when modems with different radio technologies are combined into a single product. The scenario is significantly more difficult to realize when all modems are using their own proprietary interfaces.

By using standardised interfaces and layers, multi-modem integration should be much easier because, at least theoretically, the modem interoperability, state, and data synchronisation and functionality is much more easier to access through the same interface, instead of supplier-specific interfaces and architectural requirements. This will allow much more flexibility in choosing radio technologies since modems can be changed more easily; for example, replacing Modem B (CDMA) with Modem A (GSM) should not require any major integration effort besides the functional changes between those two radio technologies.

Forum.Nokia

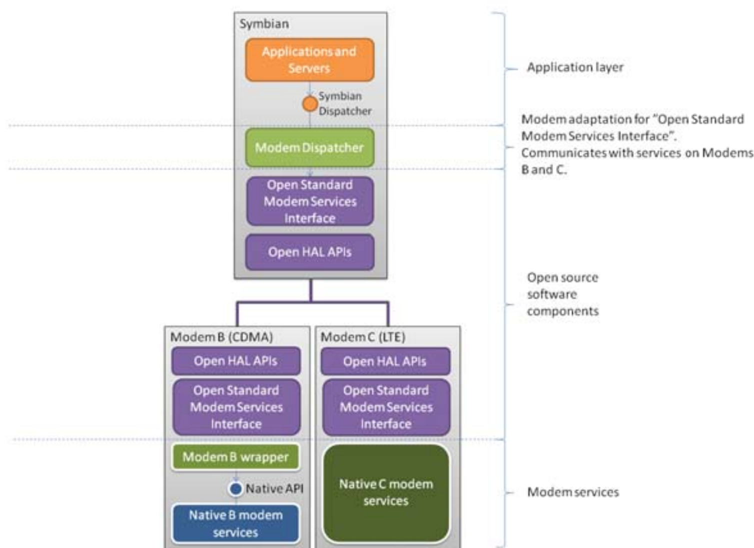

Figure 5: Dual-modem support is critical, especially when integrating LTE functionality into mobile devices. Modem technologies can be a combination of GSM, WCDMA, TD-SCDMA, CDMA, or LTE.

3.2 Features and services

Defined features must cover a majority of modem functionality, and supplier-specific custom interface add-ons should be avoided as much as possible. Features must be configurable so that the modem can implement only the relevant services. Comprehensive specifications combined with a service configurability mechanism should create dynamic, easy-to-adopt modem services, even if modem suppliers have some variation between their deliveries.

Features must naturally contain all 3GPP telephony functionality, as well as other related areas such as audio control and transfer, security services, a file system, and other supporting services. It should be possible to define production- and R&D-related services, as long as the architecture provides a sufficient level of security to prevent unauthorised access.

Features must be expandable in an open-source manner, and it should be possible for suppliers to create their own internal services as well.

3.3 Interface specifications

Interfaces must be defined as well-structured documents, describing functionality, function/parameter content, and sequences. These definitions must be unambiguous so that conflicting interpretations can be avoided. The

content of the interface should be defined so that only application-engine-relevant functionality, parameters, and sequences are included.

3.4 Interface management

Interface management can be handled by some already existing open-source interface management organisation, or a specialised organisation can be created for modem interface management.

Interface management must accomplish the following:

1. Prevent fragmentation so that an adaptation layer can be kept compliant with all modems.
2. Ensure backward compatibility so that an unnecessary migration effort is minimised.
3. Use a lightweight interface management process so that new features can be added without delays.
4. Maintain a roadmap and offer regular releases so that new features can be scheduled and released in a controlled way.
5. Keep all interested parties involved so that each supplier has an equal opportunity to make comments, and propose or discuss interfaces and management processes.

Forum.Nokia

4. Open standard modem services

These services form the core of modem services standardisation. Open Standard Modem Services can be divided into several different categories: protocol, system, and R&D services. Protocol services contain all 3GPP-specified telephony functionality. System services should include all other required system-level services such as audio, security, and boot functionality. Typically these services do not require any network communication, and thus can be seen as local services. R&D services should be reserved for services that are needed during the production, care, and development phases. These services are more likely specific for product integrators.

4.1 Modem Protocol Services

Modem Protocol Services should be defined so that each interface constructs a logical functionality group to make interface management, as well as both application engine and modem implementations, more organised.

Table 1 lists some of the possible Modem Protocol Service specifications as feature groups.

Feature group	Description
AT	AT interpreter service
AGPS	AGPS services
Call	Call control and handling (e.g., ATD, ATA, ATH, ATS0)
Data	Packet data services for control and data path (e.g., AT+CGDCONT, AT+CGDATA)
Network	Network services (e.g., AT+CREG, AT+COPS, AT+CLIP)
SMS	SMS control, sending/receiving (e.g., AT+CSMS, AT+CMGS, AT+CNMI)
Supplementary	Supplementary services

Table 1: Modem Protocol Service specification groups and examples of corresponding AT commands.

Modem Protocol Service is a critical part of the open standard because this specification will replace all 3GPP 27 series AT command specifications as the modem software interface. It is, however, the most problematic issue with the new open standard interface, because 3GPP-specified AT commands are required as a mandatory functionality by any mobile terminal. An alternative architecture for AT commands is proposed in Section 5.2, 'Software flexibility.'

Forum.Nokia

4.2 System services

4.2.1 Audio services

The circuit-switched voice call uplink and downlink audio need to be integrated from a modem to the application engine audio subsystem. This audio path and control should be based on already defined and standardised interfaces—for example, OpenMAX source and sink components. There should be no need to redefine audio control interfaces specifically for a modem peripheral, but rather to define the modem audio service interface as closely as possible to OpenMAX. The focus should be on how to efficiently integrate OpenMAX into modem hardware and software assets by using the same standardised architecture that is used for other modem services. Special attention needs to be taken to ensure sufficient performance for audio signaling.

4.2.2 Boot and state control service

The modem peripheral should use a standard service for a boot-up sequence control and a software image transfer, when needed. This service interface contains generic functions for a modem startup, an image transfer, a shutdown, a state synchronisation, and state changes such as offline mode.

It should have support for special states such as firmware update, care services, and R&D states.

4.2.3 Configuration service

The modem peripheral needs to have some mechanism to make changes to its configuration data. This can be achieved by defining a generic configuration service for peripheral configuration data, which can contain some runtime settings (for operator customisation, for example) as well as some static data that is burned in during manufacturing.

Examples of static data could be a version number, serial numbers, and other identification data. Other, more modem-related data, may consist of IMEI and RF tuning values that must be protected and stored by a secure storage service.

4.2.4 File system service

All modems have some file system services to store their statistical and configuration data. Depending on the amount of stored information, this can also be implemented as some register-based storage rather than an actual file service. By using open standardisation, a modem doesn't need to have its own file system because it can utilise file system services from the application engine. This will yield immediate benefits, since it makes the modem software asset and hardware configuration simpler—there's no need for a modem internal file system implementation or specific hardware for file storage. Chosen file system architecture may impact modem boot service and software update procedures.

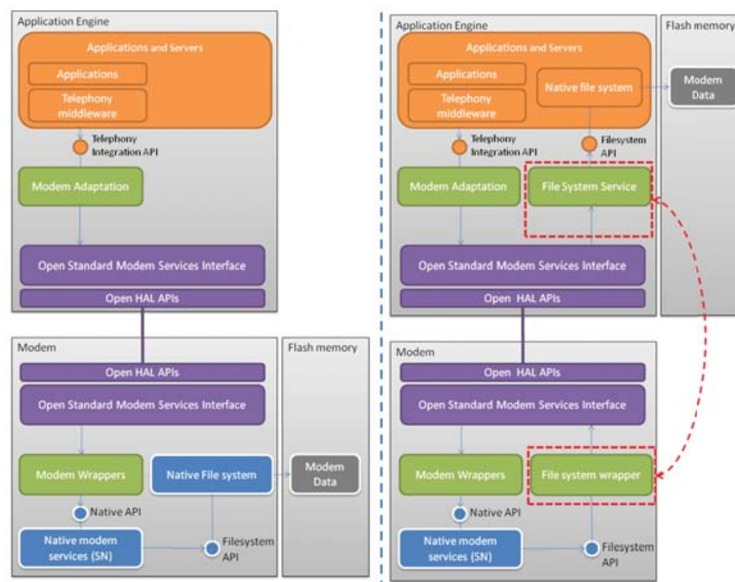

Figure 6: Modem's internal file system compared to centralised file system service implemented on the application engine.

The file system service is strongly tied into the security subsystem, because the modem file system must be protected against unauthorised access. This is required even if the file system is implemented as the application engine file system service or as the own partition on the modem flash. This solution may raise some specific security concerns, especially in platforms that don't have a platform security model available, or those that grant root access for an end user and therefore allow access to the modem-owned data.

4.2.5 Power management service

Power management service can be used to monitor and control power consumption and different states of the peripheral. In its simplest mode, power management can be used to set the peripheral to idle, sleep, on/off, online/offline, or any other power-management-related schemes applicable for peripheral hardware and functionality. Power management can also contain more advanced and independent logic to control power schemes without any need for host control. That independent power management can decrease priorities or even shut down some functionality based on battery level or other measurable signals. For example, it could be possible to decrease network bandwidth or close some other modem services to preserve battery levels for more critical use cases.

Power-management scenarios are specific for peripherals—for example, a camera peripheral has a different kind of power and hardware control than a modem. Therefore, achieving a common power-management service for all peripherals might be a challenging task, unless all states can be defined at a generic level, and the

peripheral can implement only the relevant power-management schemes. Most likely, the modem, Bluetooth, WLAN, and other radio-access peripherals can utilise similar power-management schemes.

4.2.6 Security service

Security services are a crucial part of modem integration, so there must be a service interface for security functionality. This service must allow the modem to authenticate software images and connections between critical clients and services. Furthermore, it must have secure storage for modem-sensitive data. There are common security techniques that can be used as a basis for the modem security architecture.

There are various stakeholders who expect a certain level of protection. These stakeholders include users, operators, service providers, governments, chipset vendors, and manufacturers. They want to ensure that the modem behaves according to the specifications, and that attackers cannot alter the modem's behavior. Various motivations for attackers include financial gain, making a political point, and/or achieving notoriety. These aspects need to be considered when analysing threats to security and building up a trust model. One example of a threat involving the modem is an attempt to block a network and deny service to its users.

Certain security 'valuables' are associated with the modem. These are items of data that an attacker should not be able to alter, such as modem firmware, IMEI, RF band configuration, RF tuning values, and SIM lock code. For example, the IMEI is typically written into the device during manufacturing. In normal use, the device transmits the IMEI to the cellular network to identify itself. An attacker should not be able to alter the device to transmit a fake IMEI to the network.

The security structure must accommodate a variety of hardware configurations and adjust to each configuration. Some specific threats emerge due to the separation of the APE and modem processors, as well as the respective OS used by each. The security architecture will be derived from the consideration of stakeholders, threats, trust, valuables, configurations, and countermeasures. Once the relevant countermeasures have been defined, the APIs for the security service will be written. Implementations will be validated against these APIs.

Modem security should be robust, without flaws or back doors. For this reason it is beneficial to publish a common standard for security and invite others to test it so that its robustness can be verified. For example, in the 1970s the Data Encryption Standard (DES) was widely published and tested. If various security solutions are spread out among a number of manufacturers and vendors, then less effort will be applied to proving each, and the likelihood of one being flawed is higher. Undoubtedly each manufacturer and vendor has detailed legacy solutions that would be difficult to alter quickly; however, it would be possible to align security environments, APIs, and authentication techniques between the application engine and the modem.

Forum.Nokia

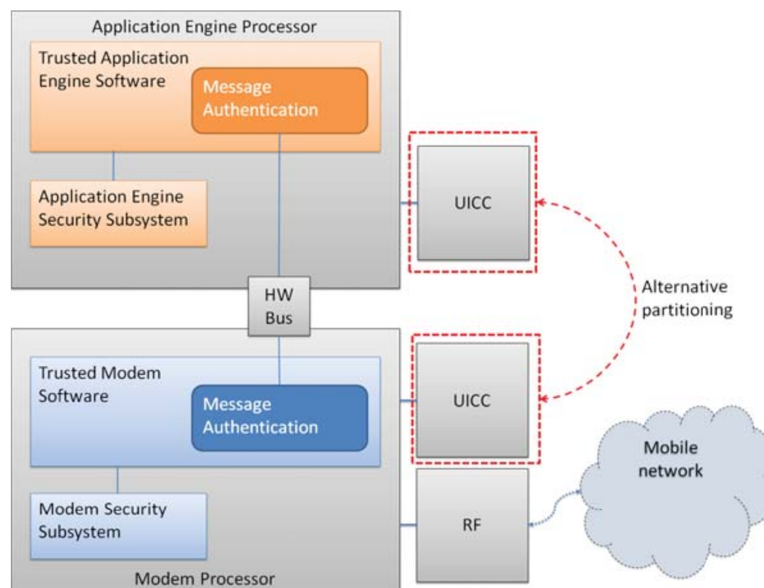

Figure 7: Security subsystems and message authentication

4.3 UICC service

UICC control is needed to access subscriber identity data and implement the Application Toolkit feature. This service may not necessarily be on the modem software, but should be connectable to the application or the modem processor and the software asset. This requirement is critical for use cases where the same UICC needs to be shared between different modems, for example in dual-modem scenarios. UICC services must have a special attention for dual-modem and dual-UICC configurations.

4.4 R&D services

Some services are needed for the R&D phase—typically tracing and debugging services—and should be common for all peripherals, not just specifically for a modem. These are enabled only in the R&D phase and should be disabled in the final product.

These services can have supplier-specific extensions or exceptions. This, of course, will increase the risk that common development tools cannot be used among modem suppliers. Each supplier typically has their own method of development and their own development tools and environments. Therefore, the risk of fragmenting R&D services is not as critical as in any other services.

Forum.Nokia

5. Industry-wide benefits

This section summarises benefits for modem suppliers, OS suppliers as application environment providers, and product integrators. Most of these benefits are applicable regardless of the chosen technology.

5.1 R&D efficiency

The most obvious benefit of having a standardised interface is the redefined responsibility between product integrators and modem suppliers. This architectural partitioning allows a modem supplier to focus solely on modem hardware and software development, and makes the need to invest anything on an application engine side obsolete.

The example in Figure 3 contains three platforms and four modems, for a total combination of twelve separate implementations. Each modem supplier must provide the implementation and the support for Android RIL, the oFono modem plug-in, and CTSY Dispatcher. In practice, there will be even more adaptations for other proprietary platforms such as Samsung Bada, iPhone, Blackberry, and the Nokia S40 device.

The implementation effort is only one part of the application support. There must be a sufficient number of capable personnel, and competencies must cover a wide scope because implementations vary based on the operating system, programming language, and Telephony Integration APIs. All adaptations need to be tested and verified thoroughly, and there is no single testing environment or tool that can be used to test for all adaptations.

All R&D efficiency-related problems could be streamlined and improved by adopting standardised modem connectivity and service interfaces, as described in Figure 4.

If Android, MeeGo, Symbian, and other platforms can provide an open-source, community-maintained adaptation per platform for standardised modem interfaces, then the only integration effort required for the modem supplier is to guarantee a standard functionality as specified. This proposal means significantly less complexity in terms of responsibility, work effort, and overall competence and resourcing. More importantly, this operational and technical arrangement will allow a modem supplier to stay focused on modem content, without any specific need to go into the details on application engine adaptation layers.

The availability of conformance test tools, which would allow a supplier to validate modem functionality against specifications, could minimise testing and verification efforts. This would also mean that a supplier could use any development environment, for example, using Android to develop modems that are used by Symbian products.

5.2 Software flexibility

Open standard software architecture is twofold: It allows application engines to integrate modems in a much more simplified way, but it also enables modems to repartition their functionality as an application engine service. This partitioning may be more feasible on open-source application engines, where a community can share the same application layer implementation and implementation effort. In a closed-source environment, partitioning decisions and implementation of application engine services are probably left as a product integrator responsibility. Such software architecture flexibility allows at least two examples of alternative architectures that are applicable for most application environments: AT command interpreter partitioning and the location of UICC hardware and software stacks.

Because the role of the AT command-based interface can be decreased on a modem asset, it is possible to move the AT interpreter service fully to an application engine functionality. In general, when an open standard architecture is used, there is no longer any need to have AT interfaces or interpreters on a modem. Distributed AT functionality between the application and the modem always requires that there be two separate AT interpreters present in the device; these interpreters must always be in sync, at runtime and as an implemented command support.

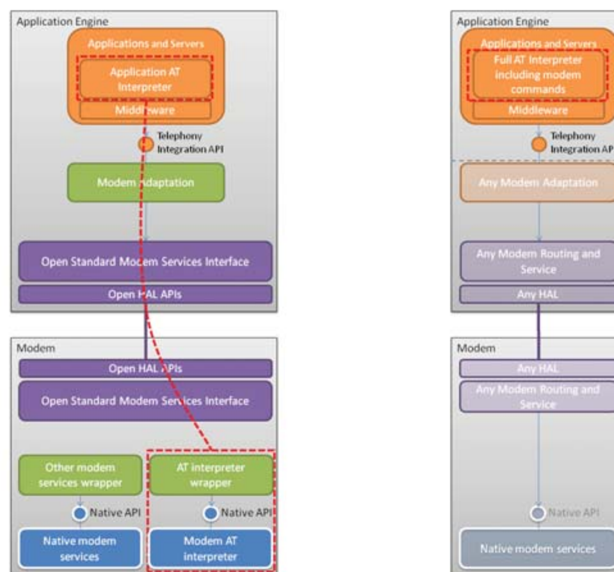

Figure 8: AT command interpreter alternatives; distributed AT interpreter on the left and full AT interpreter on the right

There is no strict requirement that only a modem can support 3GPP AT commands, and therefore it is feasible to implement AT commands solely on the application layer. By using the native middleware APIs, the AT interpreter becomes a modem agnostic and can be used with any modem: proprietary, open standard interface

compliant, or even AT command based. This, of course, requires alignment between AT command features and the middleware APIs, and the definition of new middleware APIs for such features that are not yet supported.

This latter option may require more effort in the short term, but it is definitely the most feasible implementation as a long-term solution, because it would allow an application engine to have one common implementation of the AT command stack, which would then be reusable across any modem hardware and software. The most obvious benefit for modem suppliers is that modem software would no longer need AT command support. Product integrators and platforms would benefit by having only one instance of the AT interpreter, making integration and requirement handling much more straightforward.

This same principle can be applied to a UICC service as well. Moving the UICC hardware and software stack to an application engine will provide a standard open-source UICC interface and software stack for modems to use. This reduces the modem supplier work effort because the UICC can be disabled from a modem software asset when the same service is available from the application environment.

5.3 Time to market and quality

Efficient R&D has a direct impact on time-to-market and quality, because there are fewer components to design, implement, test, verify, and maintain. Product integrators will benefit since they get better visibility to existing functionality, requirements, and new features and, due to widely used adaptation and modem layers, a robust, high-quality implementation. Modem suppliers can focus more on modem software assets, so it can be assumed that modem delivery has better quality than a proprietary solution where both modem and adaptation layers are delivered.

There is, of course, a requirement that a modem adaptation needs to be well adopted among different suppliers and platforms before all of these benefits can be applied.

5.4 Cost efficiency and cooperation

One of the key benefits for product integrators is the ability to increase the number of possible modem suppliers. This will efficiently prevent supplier lock-ins and guarantee sufficient competition between modem suppliers. From a modem supplier point of view, this will increase business opportunities to device integrators and application environments that were not previously possible due to the high degree of integration work effort in the application environment.

Forum.Nokia

6. Open issues

Some open items remain before this can all be realised as an implementation. The most challenging task is to agree on this proposal and to select the technology for a cross-platform hardware abstraction, routing, and services layers.

A standardisation forum or other open-source management processes needs to be defined to control the development, management, and ownership of the newly specified interface.

Forum.Nokia

7. Conclusions

This proposal complements existing adaptation architectures by providing an alternative modem integration layer instead of the traditional middleware APIs. It does not overlap or compete with Android RIL, oFono, or Symbian CTSY Dispatcher, and is therefore a valuable addition to any platform, allowing modem suppliers greater choices in providing their hardware and software to a specific platform.

As a product integrator, this will allow more freedom in selecting modem suppliers and will decrease effort and time-to-market. Open standardisation creates greater possibilities for efficient implementation, architecture, and extended business opportunities for the mobile industry.

Forum.Nokia

8. Terms and abbreviations

Term or abbreviation	Meaning
3GPP	3rd Generation Partnership Project
AGPS	Assisted GPS; uses network information to get a rough estimate of a location, making GPS positioning speedier.
Application engine	Software and hardware responsible for running applications and providing the environment for other peripherals to join.
GPS	Global Positioning System
Hardware / physical interconnect	Physical connection between two hardware peripherals, e.g., USB connection, MIPI standardised connections, shared memory, or some proprietary nonstandardised connection.
Host	Refers to the main chipset, e.g., application processor.
MIPI	The Mobile Industry Processor Interface (MIPI®) Alliance; an open membership organisation that includes leading companies in the mobile industry with the shared objective of defining and promoting open specifications for interfaces in mobile terminals. (MIPI, 2010)
Modem	Peripheral that handles communication between the device and base stations. Typically, it is a separate and independent hardware chip that is connected to an application processor by some hardware bus.
Peripheral	An independent hardware and software component that is connected to a host through some hardware interconnection and software interface. Typically the peripheral requires a device driver that communicates with hardware by using a specific software interface.
R&D	Research and Development; used to describe production, development, and maintenance phases of product creation.
SIM	Subscriber Identity Module
Subsystem	A collection of software components that create a logical and functional entity. For example, a UICC subsystem may contain UICC software interfaces, UICC services, and a UICC hardware driver.
Telephony Integration API	API that is typically part of middleware layer and abstracts the underneath modem software and hardware interfaces, e.g., Android Radio Interface Layer (Android Developers - Telephony), MeeGo oFono (Intel, Nokia) and Symbian CTSY Dispatcher (Nokia).

UICC	Universal Integrated Circuit Card; an evolution of the SIM card, it can contain more identification services (e.g., for NFC or multimedia services) than traditional SIM for wireless networks.

Copyright © 2010 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are trademarks or registered trademarks of Nokia Corporation.

Disclaimer

The information in this document is provided "as is," with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this document at any time, without notice.

Licence

A licence is hereby granted to download and print a copy of this document for personal use only. No other licence to any other intellectual property rights is granted herein.

Forum.Nokia

Open Modem Interface Proposal Based on Device Interconnect Protocol

White Paper

Document created on December 7, 2010

Version 1.0

NOKIA

Forum.Nokia

Table of contents

1. Introduction 4
2. Extension architecture modem integration 7
   - 2.1 DIP and Modem Services 7
   - 2.2 Application and service nodes 9
   - 2.3 Service discovery 9
   - 2.4 Modem security 11
     - 2.4.1 Threat analysis and trust model 12
     - 2.4.2 Extension Architecture/DIP Security Services 14
     - 2.4.3 Modem security startup 16
   - 2.5 Runtime view and performance 18
3. Service definition and implementation 20
   - 3.1 Interface definition format 20
   - 3.2 Stubs and message encoding/decoding 21
   - 3.3 Version compatibility 24
4. Modem services 26
   - 4.1 Modem protocol services 26
   - 4.2 Modem system services 27
     - 4.2.1 Assisted GPS (AGPS) Service 28
     - 4.2.2 Application Toolkit Service 28
     - 4.2.3 Configuration Service 28
     - 4.2.4 RF State Service 28
     - 4.2.5 UICC Service 28
   - 4.3 Modem system services - common services 29
     - 4.3.1 Audio Control and Audio Path 29
     - 4.3.2 Boot Service 29
     - 4.3.3 Data Path Service (DIP streaming sockets) 30
     - 4.3.4 File System Service 30
     - 4.3.5 Power Management Service 30
     - 4.3.6 Security Service 30
     - 4.3.7 Test Service 31
     - 4.3.8 Trace and Debug Service 32
5. Technical benefits 33
   - 5.1 Flexible architecture 33
   - 5.2 Security 33
   - 5.3 R&D efficiency 34
   - 5.4 Quality and time to market 34
   - 5.5 Cost efficiency and cooperation 35
6. Conclusions 36
7. Terms and abbreviations 37
8. References 39

Change history

December 7, 2010	1.0	Initial document release

Forum.Nokia

1. Introduction

This document describes how Extension Architecture (EA) and Device Interconnect Protocol (DIP) can be used to achieve open-source architecture and open-standard specifications for modem integration. The need for open-standard modem integration was described in another document (Open Modem Interface Benefits for the Mobile Device Industry[11]); this paper explores one of the possible technologies.

The key advantages of Extension Architecture, including DIP, are flexibility of hardware integration and service partitioning between the application and the modem processor; easy service definition by using WSDL-based definitions; and tools to generate POSIX-compliant source files and APIs for the adaptation layer and modem implementations. DIP is open sourced under BSD license and therefore allows various business scenarios, even for closed-source platforms.

DIP is already implemented as a proof of concept and available for Symbian, MeeGo, and other Linux distributions. The industry-wide, open-source, modem integration architecture can use existing DIP and define an open-standard modem services interface on top of it.

Extension Architecture and Device Interconnect Protocol (DIP) enable easy integration of any subsystems and their services. DIP consists of the hardware abstraction layer and the communication handling, routing, and service communication layers. DIP is already implemented and published as an open-source implementation, and therefore it is a viable candidate for the modem integration communication and service framework.

Extension Architecture and DIP originated with the Nokia Research Center, but have since become an independent, open-source, free software integration architecture for anyone to use. Extension Architecture defines the services by using the DIP stack and thereby allowing extendable and flexible architecture. Services are defined with WSDL service definitions. The code generator creates POSIX-compliant source code, allowing easy integration to any platform, including proprietary and closed source. It is possible to use WSDL for other programming languages and environments, for example, generating native Symbian C++ or Java™ classes from WSDL.

This document examines Extension Architecture, DIP, and especially service specifications, from a technical point of view, in order to analyse how this architecture can be utilised for modem services. This paper is based on a more generic white paper about the proposal for an industry-wide, open-standard, modem services interface (*Open Modem Interface Benefits for the Mobile Device Industry[11],* 2010).

Forum.Nokia

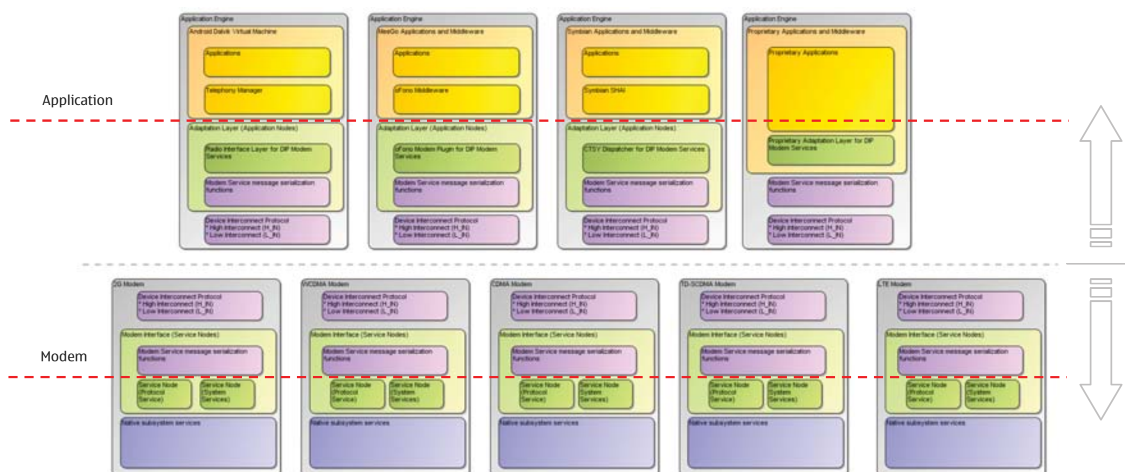

Figure 1: Standardised modem integration improves integration flexibility and efficiency of different radio-access technologies to different application environments.

Extension Architecture is a software architecture solution, whereby subsystems are connected in such a way that the interconnection, service discovery, messaging, and hardware abstraction are handled transparently for the applications and services. Extension Architecture uses DIP for subsystem connectivity. DIP is divided so that High Interconnect (H_IN) handles service discovery, client registration, authentication, and message routing, while Low Interconnect (L_INup and L_INdown) is used to abstract the hardware connection and the transportation protocol.

This partitioning allows flexibility in changing hardware and the transportation protocol without affecting the layers above. It also enables transparent connectivity of multiple peripherals and subsystems, thereby allowing reuse of DIP when connecting cameras, modems, audio, or any other hardware peripherals or software subsystems. On top of DIP, there is the service layer, which will, for example, implement modem-related services as the application and service nodes.

Extension Architecture and Device Interconnect Protocol are described in more detail in the following documents: *Architecture White Paper: Peripheral Abstraction[2]* (2010) and *System Architecture Flexibility[14]* (2010), and on the NoTA World website (NoTA - Architecture[8], 2008).

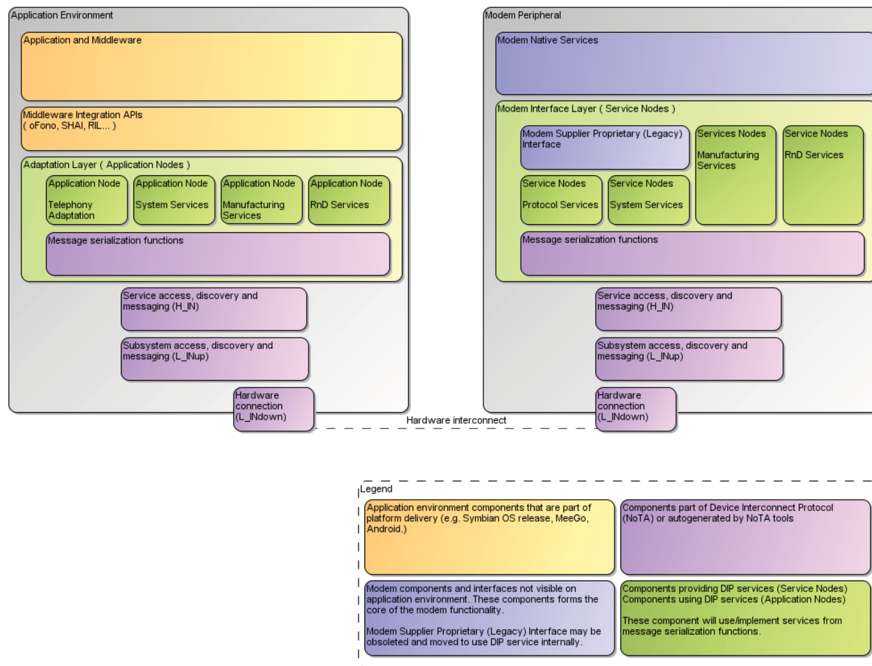

Figure 2: Overview of Extension Architecture using DIP and Service Definitions

Forum.Nokia

2. Extension architecture modem integration

Using just DIP as the standard transportation mechanism will ease the integration effort, because supplier-specific message routing or multiplexing and hardware abstraction can be avoided. This, however, will not improve adaptation for modem-specific software interfaces, which can be the most troublesome adaptation due to its wide feature area, proprietary solutions, and complicated specifications. To take full advantage of DIP, there must be a standardised services interface.

Accordingly, all modem services need to be defined as DIP services. These services should be grouped logically into categories, depending on their functionality.

2.1 DIP and Modem Services

From the DIP layer perspective, integration of the modem peripheral is no different than any other peripheral. Therefore, the DIP layer doesn't need any functional changes due to modem integration. Of course there is a need to implement a modem-specific, low-interconnect layer to connect with the physical interconnection, but it can be assumed to be similar to any other hardware driver implementation.

The DIP modem adaptation on the application environment can be implemented as an Android RIL plug-in, an oFono plug-in, or a Symbian CTSY Dispatcher, without any functional differences compared to any proprietary, or AT-command-based, modem interface. The DIP modem adaptation responsibility is to integrate the middleware integration APIs to the underneath DIP services, thus hiding modem services and communication protocol from the application and the middleware layers. DIP is described in the previously mentioned documents; therefore, this document concentrates only on the modem service layer and specifications.

On the modem side, there need to be similar layers as in the application engine—in other words, the DIP communication stack and DIP services. DIP service nodes can be implemented as a wrapper component from DIP service interfaces to the modem native interfaces, as shown for the protocol and system services in Figure 2.

The other, more efficient alternative is to use the DIP service interface as the native, internal, modem interface, thus removing the old legacy interfaces. From the modem supplier's perspective, this proposal moves the integration API from the application environment middleware integration API to the supplier's own key areas of competence: modem software asset, operating system, and modem service interfaces.

DIP does not define any service interface by itself, and therefore all of the modem-related service interfaces need to be defined by the industry. The focus on modem integration must be on the modem services and adaptation layers on both the application and the modem software assets. These services vary from 3GPP-

specific protocol functionality to more system-level functions such as peripheral boot up, file system, security services, and R&D functionality. The modem services can be roughly categorised into four different logical service areas, as illustrated in Figure 3.

Modem Protocol Services contains all telephony functionality such as call management, networking, supplementary services, data connections, and the other 3GPP cellular functionality. These services form the core of the low-level network communication services and therefore cannot be implemented as application environment services on the host processor. Instead, the implementation must be fixed with the defined application and service node responsibilities.

Modem System Services contains system integration services, which are local and can be partitioned more freely within the device. These can be UICC services, modem configuration and state control services, or some production and R&D functionality specific for the modem use cases. Typically, the integration APIs for these services are not as clear as those for the telephony-related services. Therefore, these services should have the most focus when defining and designing the modem integration architecture and the service interfaces.

Modem System Services will use some of the already-specified common services, for example, audio, boot, and file system services. These types of common services can be common across different peripherals, not just for the modem.
Because the services are still at a very early stage of development, there are no specifications ready, excluding some simple examples. Therefore, this document describes services as simple functionality descriptions instead of detailed specifications.

Modem Manufacturing and R&D Services constructs two additional logical categories. These may be embedded inside the other categories, like the system services for modem configuration functionality and the protocol services for protocol testing or network monitoring. These services have their own characteristics, life line, and different requirements compared to the other services. As such, they should be managed under different logical categories. Typically, each modem supplier, chipset supplier, and product integrator has its own customised development tools and production processes, and therefore these services will be the hardest to define and align across the industry.

Figure 3: Overview of DIP Modem Services categories

Forum.Nokia

2.2 Application and service nodes

On top of DIP, there exist application nodes (AN) and service nodes (SN). In its simplest form, a modem adaptation is an application node using the service nodes from the modem software asset. Communication between these nodes is transparent, so that the application node doesn't know, and doesn't need to know, on which specific peripheral the actual service is located. There are, however, some special configurations where the application node may need to specifically communicate with a certain service node; for example, when implementing dual modem products with two separate modem processors. DIP is meant to be expandable and scalable; therefore, dual modem configuration is simpler and more straightforward to implement when compared to a traditional AT command multiplexing.

The modem service node implementation uses modem internal APIs for the actual functionality. The Open Standard Modem Services Interface is most likely not a native interface within a modem, at least not initially, but instead a wrapper layering similar to modem adaptation on the application engine. . There is no reason why Extension Architecture/DIP cannot be used as a native, modem-internal interface; on the contrary, DIP allows private services to be used so that they are visible only inside the modem peripheral, without being exposed to the application environment. For example, using File System or the UICC interface within a modem software asset, as an internal interface, allows product integrators to easily change the file system or UICC implementations between the application and the modem assets, without causing major extra work on the modem side or with the overall peripheral connectivity. Thus, using DIP as the modem native communication mechanism would increase system flexibility even further.

2.3 Service discovery

Figure 4 gives a brief example of the basic communication flow between the application and modem environments, and also demonstrates the flexibility of Extension Architecture, within the scope of the modem services. To underscore this flexibility, note that this figure is applicable for either a static hardware configuration diagram, where either Modem 1 or Modem 2 is connected; or a runtime diagram, where two modems are connected and communicating at the same time. The DIP layer is capable of connecting and managing communication for the multiple parallel modems, thereby keeping the adaptation layer simpler, even if there are differently configured modems connected at the same time.

In Figure 4, both modems are providing similar protocol services, but are different from flash memory and UICC configuration points of view. Modem 1 contains a UICC, but doesn't have its own flash memory, while Modem 2 has a flash memory, but doesn't have a UICC. The application engine is then providing both File System and UICC services for those modems that need to use them from the application processor side.

The protocol service application node is using the Service Node (Protocol Service) from either Modem 1 or Modem 2. This communication always goes from the Application Node to service on the modem side. From an

adaptation perspective, the Application Node is a normal modem adaptation implementation underneath the middleware integration APIs, and therefore is similar to any other modem adaptation. On a UICC service case, the communication flow varies depending on UICC configuration. The Application Node can communicate with Modem 1 UICC service or with Modem 2 UICC service, which is actually located on the application environment as the UICC Service Node. Regardless of the location of UICC Service, this communication is transparent for the Application Node because DIP is handling the service discovery and routing. Furthermore, this UICC configurability is not visible on the middleware layer, because the modem adaptation layer is still similar to any other modem adaptation implementation, even though it provides such flexibility between adaptation and modem subsystems. On Modem 2 there must be an Application Node for UICC, because Modem 2 needs to access the UICC Service Node on the application environment.

This flexibility enables some other configuration scenarios as well. For example, it is possible to share UICC service between two modems at runtime. Extension Architecture flexibility is not limited to UICC or File System; it can be applied to any local service, where service doesn’t need to specifically locate either on the application environment or the modem subsystem.

Forum.Nokia

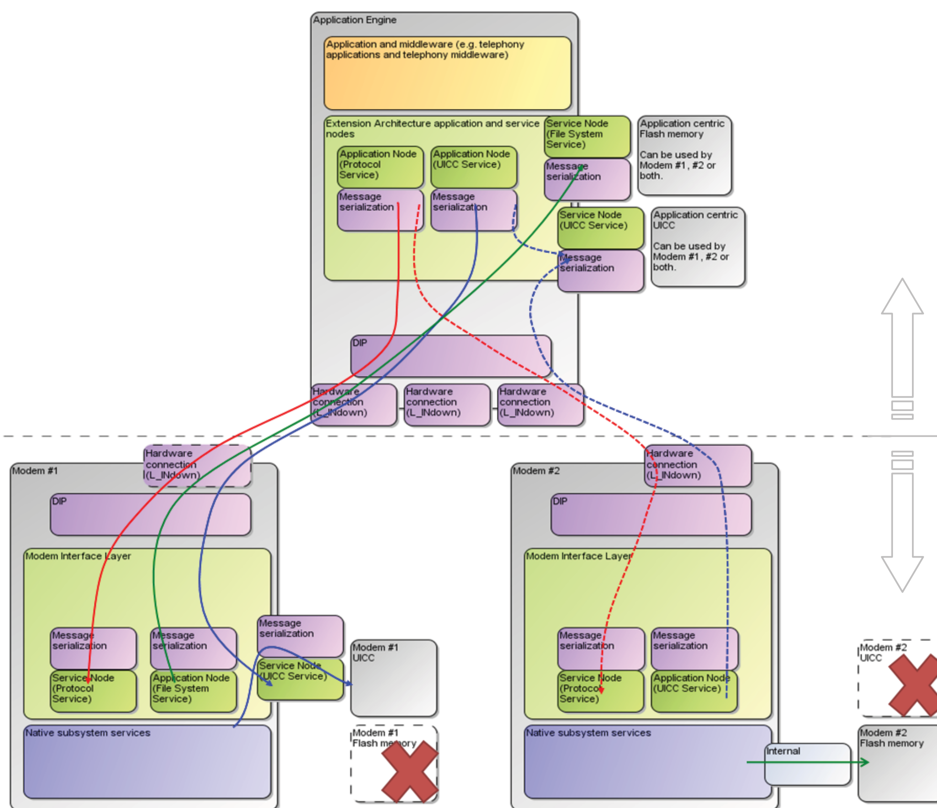

Figure 4: DIP-based modem integration enables different scenarios for hardware partitioning and software compatibility for different service scenarios.

2.4 Modem security

The modem needs security to protect it from tampering. Tampering can occur if malicious software is downloaded to the device or connected to it. To combat these threats, countermeasures need to be implemented in the software and hardware architectures.

The architecture selected needs to accommodate a variety of different platforms, modems, and configurations. The number and type of countermeasures required will vary accordingly.

When comparing the modem engine (typically containing a chip with processor and memory) with other typical engines, it is likely that a greater degree of security is needed. For example, in an audio engine there may be a threat to the audio streams and audio routing. For a modem engine, there is the threat of fraudulent calls on a network, which is more severe. In another example, a WLAN engine may be able to interfere with nearby mobile devices; however, a cellular modem has much greater transmitting power and can potentially interfere with many more mobile devices.

Forum.Nokia

2.4.1 Threat analysis and trust model

In order to qualify these comparisons, an analysis technique is needed. This technique can also be utilised to determine which countermeasures should be employed by the modem in the configuration used. The proposed analysis technique is the construction of a trust model. This technique follows a logical sequence of steps to analyse security threats and decide on countermeasures.

The analysis starts with a consideration of use cases—legitimate use cases that could be exploited by malicious software. One example is transferring RF band settings from the application engine to the modem engine. In this case, malicious software could alter the band settings during the transfer. The consequence would be that the modem starts to transmit on a different band to that desired by the manufacturer or operator.

In such a case, the RF band settings are considered to be valuable. There is financial gain in being able to alter the band settings, which implies that it is a target for malicious software. For this reason, greater security countermeasures are needed.

In contrast, samples of the voice audio signal are not so valuable. There is little financial gain in altering samples of the speech, so it is not a target for malicious software. Here, there is no need for strong countermeasures.

A list of threats is then created for each use case. One threat example is that downloaded software may attempt to mimic a message carrying RF band settings to the modem. A list of countermeasures is created for each threat—in this case, for example, using registration for software servers or authenticating messages between the application and modem engines.

Finally, each countermeasure is implemented as a trust point. This is a specific implementation of a countermeasure. For example, registration of software servers could be done according to the method employed by the extension architecture, and authentication could be according to the AES 256 standard.

In summary, threat analysis follows a sequence of logical steps (use cases, valuables, threats, countermeasures, and trust points) to arrive at a well-thought-out conclusion. Table 1 describes typical legitimate use cases and the corresponding threats.

Use case	Valuable	Threat	Countermeasures
Write IMEI to secure storage in modem engine	IMEI	Hostile application writes alternative IMEI to modem engine	• Activation and registration of services • Routing and security table • Authentication of source SN • Authentication of application engine • Signed software • Modem firewall • Timeout SNs • Deregistration and deactivation of services
Transfer of RF Band Settings from application engine to modem engine	RF band settings	Hostile application uses settings for a different network	
Set frequency and power levels	RF tuning values	Interfere with GSM network by accessing lower-level modem services	
Access GSM network and make a call	SIM lock PIN code and UICC data	Eavesdrop and interfere with data to and from UICC card	
Modem boot	Modem firmware	Flash hostile application into memory	
Update modem firmware	Modem firmware	Flash hostile application into memory	
Write certificates in production	Digital certificates	Flash alternative certificates to defeat software signing	
Enable field test and R&D features	None	Hostile application uses field test functionality for malicious use	
Access service features	Warranty information	Reset warranty information	
Standby operation	None	Hostile application floods modem with requests	

Table 1: Typical use cases and threats

Forum.Nokia

2.4.2 Extension Architecture/DIP Security Services

Turning your attention to Extension Architecture/DIP, it can be seen that this architecture has several security features that can be used. In spite of the configurability and extendibility of the architecture, it can support whichever level of security is needed. A typical configuration may comprise of an application engine and a modem engine, each of which have their own processors and operating systems. Each of these can contain an Extension Architecture Sub-System (SS). Extension Architecture has a uniform way of addressing Sub-Systems and the Application Nodes (AN) and Service Nodes (SN) within them regardless of which OS is used.

Extension Architecture inherently supports some countermeasures. The method of registration, activation, and service discovery of AN and SN nodes prevents malicious applications from gaining access to valuable items. Each engine implements memory separation within its own OS. However, Extension Architecture implements a routing table (socket connection table) which is capable of maintaining the separation of processes across two engines.

Further, where proprietary techniques are used for security, Extension Architecture is extendable to accommodate the interface to these features between engines. Each platform and vendor may implement a different regime for chip identification, 'root of trust', and software signing. However, Extension Architecture can be extended to implement authentication between these systems and secure an interface between them.

A block diagram showing some relevant aspects of Extension Architecture is shown in Figure 5. A sequence diagram showing startup is shown in Figure 6. In this example, there are two Extension Architecture 'Subsystems' (SS), one for the Application engine and one for the Modem engine. Modem security is explained in terms of standard entities that are described in Extension Architecture. The following text largely describes their specific relevance to modem functionality.

Forum.Nokia

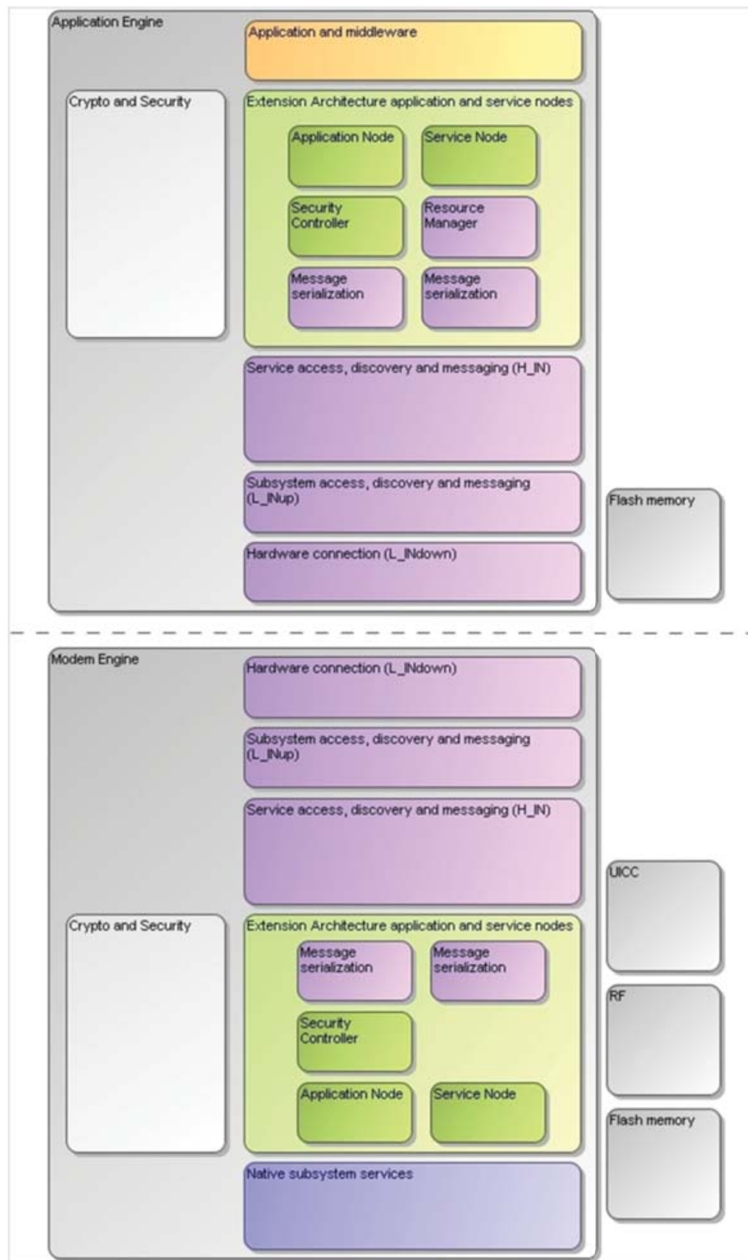

Figure 5: Relevant aspects of Extension Architecture security components and peripherals

Security Controller (SC)

There is one Security Controller (SC) in both the application engine and the modem engine. The SC handles key and authorisation material and verifies the integrity of other entities. It works with on-chip crypto and security functionality, which is separated from the rest of the architecture in the form of a trusted module. Typically, a unique security ID and a 'root of trust' are associated with this functionality. The SC is the first entity to boot in order to verify the security of the Subsystem (SS). Digital signatures are typically used to verify that software images have not been tampered with. After these activities are completed, the rest of the SS is booted.

The SC provides cryptographic and security functionality to RM, SNs, and ANs. Security Association key exchange is controlled by the SC. A security association can be set up between the SC in the application processor and the SC in the modem engine. This allows the engines to trust one another.

Resource Manager Service Node (RM SN)

There is one Resource Manager Service Node (RM SN) in the application engine. The RM SN connects to the H_IN layer of Extension Architecture. The RM SN has an overview of all the ANs and SNs in both the application and modem engines. It implements a security policy to authorise (validate) all the operations of H_IN including node registration, activation, service discovery and connection. Once a connection between two nodes is allowed it is implemented by the H_IN of both the application and modem engines in the form routing tables (socket connection tables).

With respect to security, the application engine can be viewed as the master and the modem engine as the slave. The RM SN in the application engine is responsible for access control to modem services. It liaises with the H_IN in the modem engine. For example, secure storage, could either be located on the application engine side or the modem engine side. During boot up, the modem engine H_IN discovers which services it has available and informs the application engine RM.

2.4.3 Modem security startup

A startup sequence diagram is shown in Figure 6. To begin with, the SCs in the application and modem engines boot up independently and verify their subsystems. In this example, the modem boots from its own flash memory, but in other cases, its image may be downloaded from shared memory (see Section 4.3.2, 'Boot Service,' for this case). Typically, digital signatures would be checked and then used to verify software. If the SC verifies the software successfully then the H_IN and L_IN layers are started (In the application engine, the RM SN is also activated). The application engine H_IN first communicates with the modem engine H_IN (through L_IN layers) and in this example requests authentication. This is one countermeasure against a 'man in the middle' threat between the application and modem engines. A challenge/response technique is used both for the application engine to authenticate the modem engine and vice versa. If this is successful, then a secure tunnel is established between the two engines.

The next step is service discovery. The application H_IN gets permission from RM SN and requests the services available from the modem engine H_IN. It uses this information to construct a routing and security table. In simple terms, this shows which SNs and ANs are allowed to communicate with which other SNs and ANs. The RM SN decides when services can be activated either in the application or modem engines.

Application Engine (Sub-System)
«SW Component» : Application Security Controller
«H_IN» : Application H_IN (Master)
Modem Engine (Sub-System)
«H_IN» : Modem H_IN (Slave)
«SW Component» : Modem Security Controller
1: Boot and verify Sub-System
2: Boot and verify Sub-System
3: Activate H_IN and Resource Manager()
4: Activate H_IN()
5: Request Authentication()
6: Accept Request
7: Request Challenge()
8: Challenge Modem()
9: Respond to Challenge
10: Challenge Application Sub-System()
11: Respond to Challenge
12: Modem Authenticated
13: Application Sub-System Authenticated
14: Setup Secure Tunnel()
15: Secure Tunnel setup
16: Discover SNs and ANs()
17: Reveal SNs and ANs
18: Construct Routeing and Security Table()
19: Configure SNs and ANs()
20: Configure Routeing and Security Table()
21: Activate Modem SNs and ANs()
22: SNs and ANs Activated

Figure 6: Modem security startup

Forum.Nokia

2.5 Runtime view and performance

Extension Architecture/DIP also provides for fast runtime performance by separating secure and nonsecure communications.

A distinction is made between streams and messages. For example, a voice audio stream may be considered of little value and so it is unlikely to be threatened by malicious software. This means that a connection can be set up without the need for constant authentication or encryption. In contrast, messages carrying commands to a Service Node on the modem processor are of higher value. If they involve operations on modem valuables, then, they are likely to be considered a high risk. In this case, authentication of the source of the message is needed as well as detection of any tampering with the message.

Further, Extension Architecture makes a distinction between secure and unsecure nodes. Security attributes are applied to SNs by the Resource Manager so that unnecessary overhead is avoided. Messages that do not require a high level of security do not need to be authenticated.

The preferred model for process allocation is to use a daemon-like approach where the DIP stack is running on a single process and AN/SN nodes in separate processes, presumably in a middleware or application layer context. L_INdown and the hardware interconnect driver can be partitioned between the user and kernel space.

DIP can run on an AN/SN process only in those configurations where there is only one AN/SN node. The peripheral can contain only one instance of the DIP stack, because it is impossible to synchronise two stacks within one peripheral. In some cases it is possible to have multiple DIP stacks, but each of those would have their own service space (meaning that one DIP stack is for Open Standard Modem Services and the other can be reserved for the supplier-specific internal service space).

Unfortunately there aren't any clear performance statistics or analysis available to compare raw data performance when data is transferred through DIP stacks. It can be assumed that the performance should be close to that of similar layered architectures, but more detailed analysis must be made with some reference hardware and software stack. DIP is already in the proof-in-concept state for video and graphics integration and capable of transporting high-definition video streams on a handheld device.

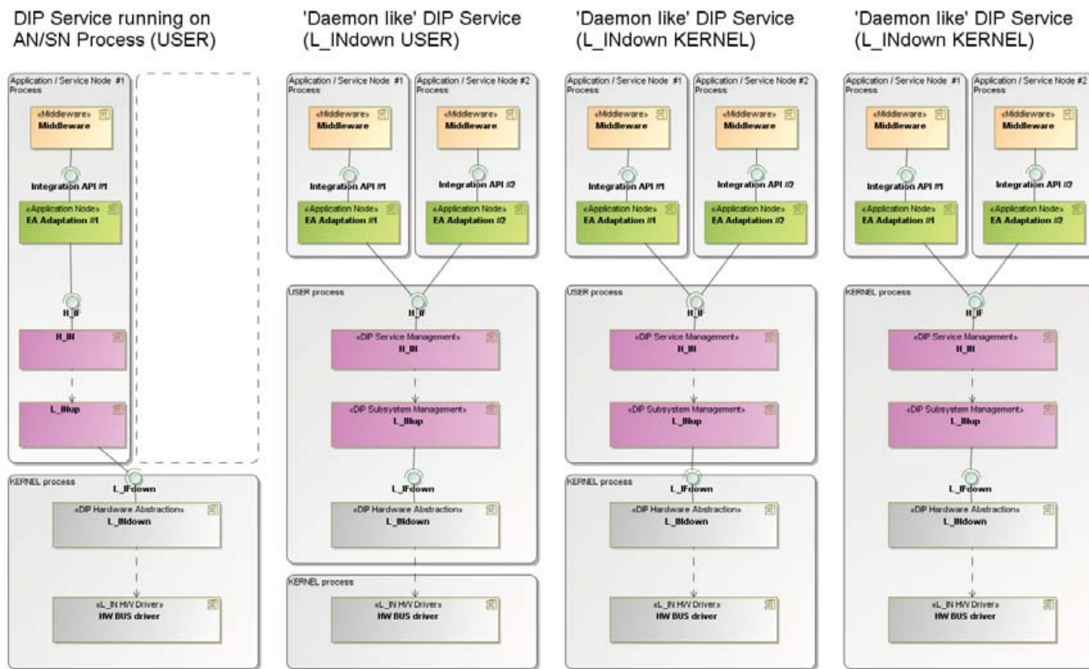

Figure 7: Runtime view of Extension Architecture DIP and AN/SN nodes

Forum.Nokia

3. Service definition and implementation

The following chapter uses a very simple Short Message Service (SMS) use case to demonstrate DIP service definitions and implementations. These principles are applicable for any DIP service, regardless of the subsystem.

3.1 Interface definition format

DIP services are defined as WSDL files. The following WSDL example defines some data types to be used within three messages: a request for SMS message sending; a confirmation message; and a message indicating the newly received SMS. The WSDL file is used to create the header and source files for messaging stubs.

- **sms_common.h** contains all common definitions for both application and service nodes. These are signals IDs, data types, and enumerations.
- **sms_user.h and .c** defines and implements functions used by the application node. It will also contain the skeleton for functions required to be implemented by the application node when processing received messages.
- **sms_service.h and .c** is similar to easms_user.h and .c, but for the service node.

Forum.Nokia

```
<!-- Enumerations and datatypes -->
<types>
  <simpleType name="enum_smsservice_status_t">
    <restriction base="uint8_t">
      <enumeration value="0x00">OK</enumeration>
      <enumeration value="0x01">FAIL</enumeration>
      <enumeration value="0x02">NETWORK_ERROR</enumeration>
    </restriction>
  </simpleType>
</types>

<message name="send_sms_req" code="0x1001" direction="in">
     <documentation>Send SMS-SUBMIT PDU to network</documentation>
     <part name="pdu" type="bdata"/>
</message>

</message name="send_sms_cnf" code="0x1002" direction="out">
     <documentation>Response to SMS sending request</documentation>
     <part name="status" type="enum_smsservice_status"/>
     <part name="tp_mr" type="uint8_t"/>
</message>
<!-- oneway notification message-->
<message name="receive_sms_ind" direction="out">
    <documentation>
    Indication of new MT message.
    </documentation>
    <part name="pdu" type="sms_tpdu_t"/>
</message>
```

3.2 Stubs and message encoding/decoding

DIP contains tools to make application and service node implementation more automated. The stub generator makes functions that will handle all the data serialisation: message encoding and decoding to/from function parameters. This will abstract the actual message format and binary payload from nodes, and consequently node implementation can fully focus on logic instead of message handling. Stubs are generated as POSIX-compliant code, therefore allowing seamless integration to most of the application environments.

Stubs are especially beneficial for modems that are currently using proprietary binary messages and taking a lot of effort to encode and decode messages. Most likely this will not decrease overall binary code size, but it will significantly improve modem and adaptation architecture as the message payload format becomes irrelevant.

The following figures and explanations use SMS to demonstrate the basic message flow, content of an example SMS service interface, and functionality of autogenerated source code.

By using stubs and autogenerated code, it should be rather easy to migrate from any other interface to DIP modem services. The only relevant issue is the parameters transferred between application and modem services because all interfaces must support same amount and same abstraction level of parameters. In this example,

SMS sending is defined so that the modem needs to handle only the PDU data types. The PDU data type needs to be created either in middleware or in the modem adaptation, depending on the application engine telephony integration APIs and software architecture.

Interface functionality and a detailed specification cannot be defined in the context of this document. Final specifications must be created, controlled, and planned in a standardisation forum by all modem suppliers, and in cooperation with application environments.

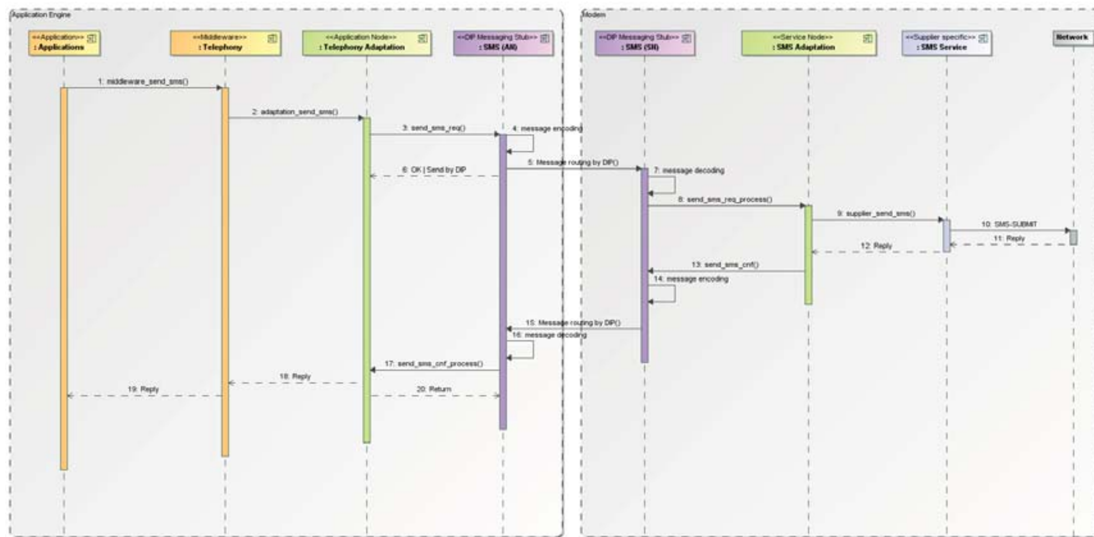

Figure 8: Communication flow between application and middleware (telephony integration API), adaptation (AN), and modem (SN). Most of the stub functionality is autogenerated, and only the processing functions need to be implemented by adaptation layer or modem.

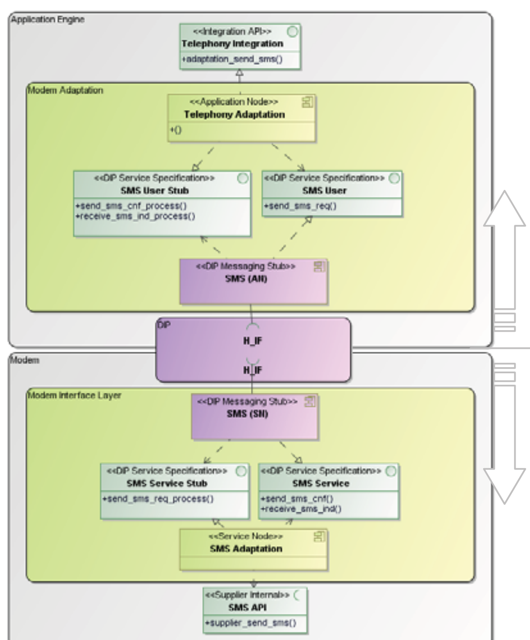

Figure 9: WSDL definition generates four different logical APIs. The User API and Service API are defined as separate header files, and they contain message encoding/sending functions and stubs for message decoding/receiving.

The **SMS User API** contains those functions that are used to encode and send requests to the Service Node. These functions are autogenerated by a stub-generator tool and can be used without any modifications.

```
int send_sms_req( struct context_pointer* context, sms_tpdu_t* pdu );
```

The **SMS User API Stub** defines function stubs that must be implemented by the Application Node. These functions will be called when a response is received from the Service Node and message parameters need to be processed.

```
void send_sms_cnf_process( struct context_pointer* context, status_t status,
enum_smsservice_status status, uint8_t tp_mr);

void received_sms_ind_process( struct context_pointer* context, sms_tpdu_t* pdu );
```

The **SMS Service API** corresponds to the SMS User API, but contains functions for messages sent by the Service Node.

```
int send_sms_cnf( struct context_pointer* context, enum_smsservice_status status, uint8_t
tp_mr );

int receive_sms_ind( struct context_pointer* context, sms_tpdu_t* pdu );
```

The **SMS Service API Stub** defines functions to be implemented by the Service Node.

```
void send_sms_req_process( struct context_pointer* context, sms_tpdu_t* pdu );
```

3.3 Version compatibility

WSDL code generation and the internal behavior of DIP stack message/function handling causes some version compatibility restrictions between application and service nodes. The following examples are focusing on typical *backward compatible* changes. Obviously, removing or changing any existing functions, parameters, or data types will always cause binary and source incompatibility. These scenarios are set up so that both ANs and SNs have their own version of the headers and source codes are compiled separately.

Interface change	Binary compatibility	Source compatibility

Add new enumeration value to a request or response. Enumeration is used as a parameter in a function call.
 	
Version 1.0
`Typedef enum _SOMEVALUES`
`{`
`VALUE_A = 0,`
`VALUE_B = 1,`
`VALUE_C = 2`
`} SOMEVALUES;`
 	
Version 1.1
`Typedef enum _SOMEVALUES`
`{`
`VALUE_A = 0,`
`VALUE_B = 1,`
`VALUE_C = 2,`
`VALUE_D = 3`
`} SOMEVALUES;`
	Binary compatible. AN and SN should revert to a default functionality or return a default error value if unknown enumeration values are received.	AN and SN can be compiled against the new version of the interface header without source break.

Add new function parameter (as the last parameter).
 	
Version 1.0
`function( uint8_t param1 );`
 	
Version 1.1
`function(uint8_t param1, uint8_t param2 );`
	AN (1.0) → SN (1.1)
	Binary incompatible. AN doesn't send enough parameters, and thus correct message stub cannot be found. Error response must be sent to AN by SN error handler.
	
	AN (1.1) → SN (1.0)
	Binary compatible. SN receives only those parameters that are
		AN/SN can't be compiled against new interface headers because function declarations don't match the existing implementation.

	available in the current SN implementation. New param2 is ignored even if it is send by AN.	
Add new function **Version 1.0** - **Version 1.1** `other_function( uint8_t param1 );`	**AN (1.0) → SN (1.1)** Binary compatible. SN can support new function, but AN never calls it. **AN (1.1) → SN (1.0)** Message stub cannot be found from SN. Error response must be sent to AN by SN error handler.	AN/SN can't be compiled against new interface headers because implementation is missing for other_function(...)

It is recommended to implement a version query as the first message to be sent when the application node connects to a service. This allows some flexibility for multiversion support.

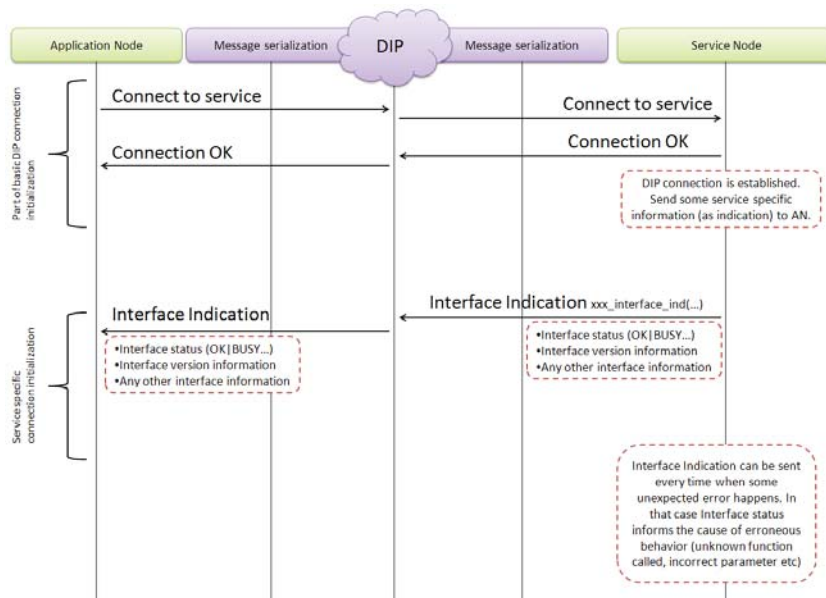

Figure 10: Interface versions should be synchronised during the AN/SN initial connection.

Forum.Nokia

4. Modem services

This chapter defines some characteristics of DIP services related to modem use cases. Depending on the modem software and hardware architecture, all of these services may not be needed, or there may be some need to specify new services specifically for the system and common services area.

4.1 Modem protocol services

Modem Protocol Services are a collection of different 3GPP-related modem services that closely follow 3GPP specifications for protocol communication. Modem protocol services must be defined so that each interface constructs a logical functionality group to make interface management and both application engine and modem implementations more organised. The abstraction level of modem protocol services should follow quite closely to the 3GPP AT command interface because it provides sufficient function and parameter levels for any application engine modem adaptation layer, thus making integration to any Telephony Integration APIs easier. Figure 11 shows high-level architecture and some of the possible services as a feature groups.

Modem protocol services are a critical part of the open standard because this specification can then replace all 3GPP AT command specifications, or any proprietary interface, as a modem interconnection standard. This is one of the most problematic issues with standardisation: 3GPP-specified AT commands are required as a mandatory functionality by any mobile terminal. Therefore, even if DIP modem services are used as an open standard interconnectivity interface, the device must still have AT command support as defined in 3GPP standardisation.

The proposal is to move the AT command interpreter implementation to the application environment , and thus remove that requirement and implementation from the modem peripheral. This proposal for a new AT interpreter architecture is described in the document entitled *Open Modem Interface Benefits for the Mobile Device Industry[11]* (2010).

Forum.Nokia

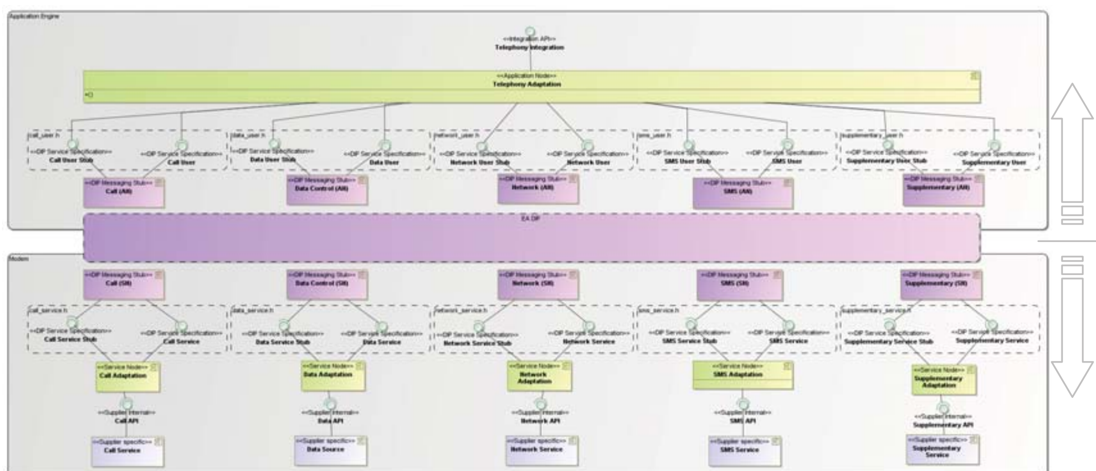

Figure 11: A proposal for Modem Protocol Services feature groups, including all APIs (AN, AN stub, SN, and SN stub)

4.2 Modem system services

System services are focusing on services that are not part of 3GPP protocol communication, but are still a critical part of modem integration from a system integration point of view. These services are typically local services that are needed or provided by a modem peripheral. Due to different modem system architectures, traditionally these have been handled on a case-by-case basis, resulting in a fragmented, nonstandardised implementation between application and modem software assets.

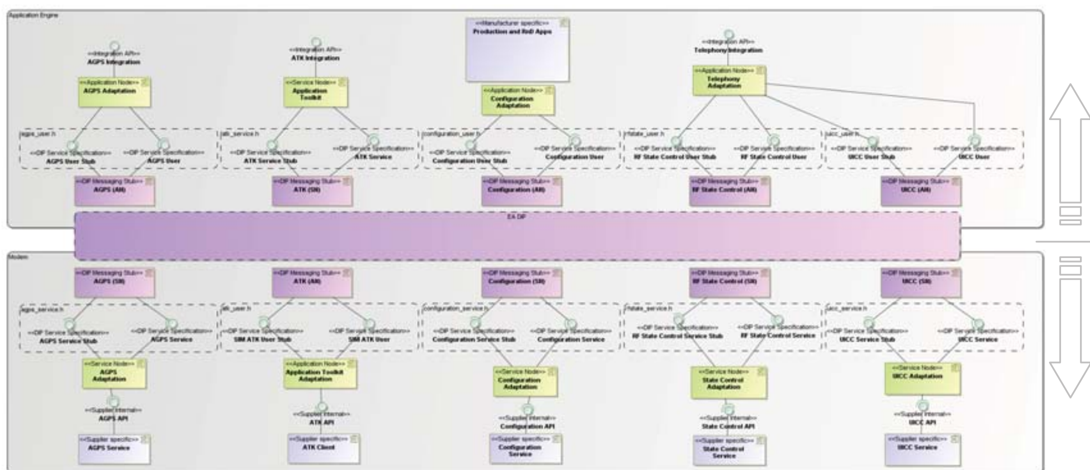

Figure 12: A proposal for Modem System Services feature groups, including all APIs (AN, AN stub, SN, and SN stub)

Forum.Nokia

4.2.1 Assisted GPS (AGPS) Service

Assisted GPS (AGPS) Service does not communicate directly with the network, and therefore can be considered a local service instead of a protocol service. AGPS uses services from the protocol software side, but due to physical partitioning of the GPS peripheral, there might be a need for some flexibility of location of AGPS Service. It could be possible to implement AGPS Service as an application environment service instead of a modem service.

4.2.2 Application Toolkit Service

Application Toolkit Service provides application layer functionality, for example, UI dialogues, for the modem so that the SIM application toolkit features can be supported by the UICC card. This service must be implemented as an application environment Service Node, and typically the UICC Service Node is acting as an Application Node towards the SIM application toolkit.

This service is already defined by 3GPP, but the nature of this service is local rather than protocol, and therefore feasible to have as a system service.

4.2.3 Configuration Service

The modem peripheral needs various configuration data stored during the R&D or production phase. This service allows reading and writing of those values. Typically these values can be RF band settings, RF tuning values, IMEI, manufacturer ID, serial numbers, and other modem configuration and identification data.

Configuration Service must have a close dependency to a security service because the writing of these values must be protected and allowed only in a trusted environment. Depending on R&D or the production environment, the Application Node can be located either in the application environment or on the PC software.

4.2.4 RF State Service

The modem peripheral must be able to be set to offline/online mode. This service may contain more modem-specific state services like controlling network connectivity and radio-access technologies.

4.2.5 UICC Service

UICC Service is closely related to a protocol software and 3GPP specifications, but can still be defined more as a local system service instead of protocol service. UICCs can be physically connected either on a modem or on an application environment, therefore service flexibility is more important than in any other services. This is even more visible for products using multiple UICC cards or sharing one UICC across separate modems. Some UICC partitioning examples are shown in Figure 4.

4.3 Modem system services - common services

Common Services is a logical group within modem system services. These services are such that they can be applied to any subsystem such as Bluetooth, WLAN, or any other subsystems requiring all or part of these services. These services are common for any DIP-enabled peripheral and not limited only to modem peripherals.

The specifications for these services can be hosted outside the 'open standard modem services interface'.

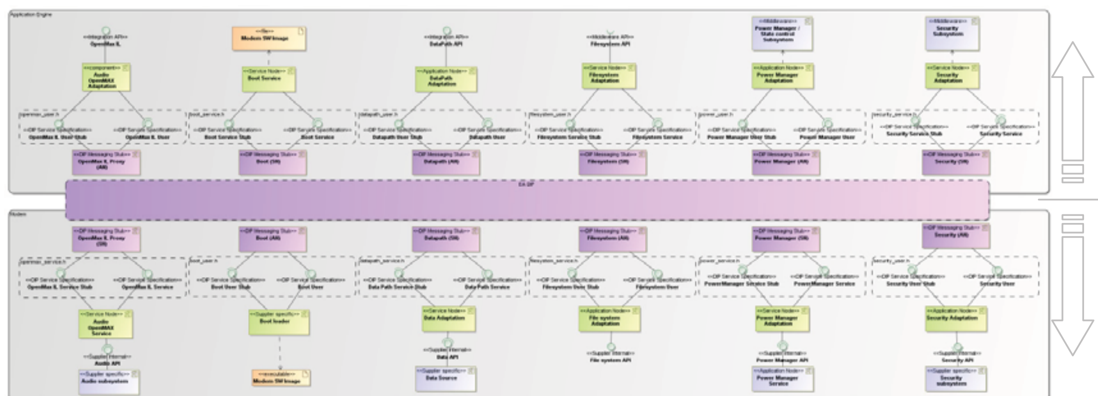

Figure 13: Common Service feature groups, including APIs (AN, AN stub, SN, and SN stub)

4.3.1 Audio Control and Audio Path

The modem audio control and audio path can utilise DIP as an interconnection, but the actual service interface can be based on OpenMAX functions and parameters. This has two benefits: DIP can still be used to achieve interconnection flexibility, but the service is based on an already-standard specification, so there is no need to define a new parallel service for the modem audio control and audio path.

OpenMAX proxy is available from gitorious.org (ilProxy in NoTA[6], 2010).

4.3.2 Boot Service

Boot Service defines how the boot image is transferred to a subsystem and how the service and application node communicates during boot-up sequences. This service is already specified and freely available as an open-sourced implementation and can be used for modem boot-up service, as is (boot in NoTA[3] - Gitorious, 2009).

There are some exceptions that may require special attention. Boot service assumes that a software image is stored in the application environment and transferred on each boot to the subsystem. This may not be true for all modem suppliers because it is possible to store modem software images to the modem flash memory. Therefore, Boot Service needs some changes to allow two different scenarios: modem software image stored in the application environment or modem flash memory.

This has dependencies to a firmware update process. If the flash image is stored in the application environment, a modem software upgrade is transparent to the modem itself. The application environment handles the firmware upgrade procedure and the modem will get its new software image during the next boot up. In another scenario, there is a need to transfer the software image to the modem for flashing. This can use the same interfaces and services as image transfer during boot up, but there must be some additional changes to notify the modem to store its flash image by itself instead of just running it from execution memory.

4.3.3 Data Path Service (DIP streaming sockets)

Data Path Service is a logical name for the raw data connection between peripherals. This service must be able to upload and download at high data rates for GPDS and especially for LTE. DIP provides streaming socket connection as a default service, and therefore can be used as it is in conjunction with the actual data path controlling service, like Data Service for GPDS (Nota Core Overview[9]).

Data Path Service is connection agnostic, and the used connectivity technology, for example, WLAN, GPDS, or LTE, must have its own Data Service for signaling and control.

4.3.4 File System Service

File System Service provides generic file system access for peripherals that do not have their own internal file system service. This service is always located on the application environment, and it may have multiple clients in different peripherals. File System Service is already specified and available (filesn in NoTA[5] - Gitorious, 2010).

Typically, a modem doesn't own a huge amount of binary data, and therefore it is also possible to define such a File System Service that will act as index-based value storage. Similar functionality is used in the Symbian Central Repository[12], where a small amount of data is stored into the database by using identifiers.

4.3.5 Power Management Service

Subsystems need services to monitor and control power management. This can be used to inform subsystems about battery-low condition, charging state, and other states that may have an impact on power consumption. This service is related to RF State Service and can be combined as a one single service.

4.3.6 Security Service

The modem provides some common security services of Extension Architecture:

- Identification of available service and application nodes;
- Configuration of routing and security table (according to instruction);
- Registration and activation of service and application nodes;

- Deactivation and reactivation of nodes according to state transitions;
- Downloading of modem firmware updates.

More advanced security features are provided depending upon configurations:

- Download of modem boot image from application engine shared memory;
- Secure boot and verification of modem subsystem;
- Identification of available security and authentication algorithms;
- Identification and authentication of modem to application engine;
- Identification and authentication of application engine;
- Creation of secure tunnel.

Further, some services are provided in production (such services may be overwritten by user software):

- Identification of modem chip type and chip ID;
- Provisioning of ‘trust’ certificates to modem;
- Flashing of modem software image;
- Writing of IDs to secure storage;
- Writing of tuning values to secure storage.

These common services handle valuables that are unique to the modem, for example, IMEI, RF band settings, RF tuning values, and modem firmware.

Some services are provided for post production activities:

- Enabling of field test and R&D debugging modes;
- Enabling of warranty and service features.

4.3.7 Test Service

Test Service is R&D functionality for the peripheral testing purposes. This service should define a generic service interface to control test cases and test runs. However, it should not define specific tests because these will vary based on the peripheral, service, and supplier.

With modems, it probably makes sense to have categorised test cases instead of one common modem test set: one set of tests to control protocol functionality and the other to test the system and common services.

4.3.8 Trace and Debug Service

R&D work needs various services to support device development, device tracing, and debugging. These must be defined as a common service for all subsystems so that, for example, tracing has consistent functionality across different peripherals (for example, MIPI standardised Open System Trace-based service interface).

The Debug Service may create some peripheral-specific functions, and there should be further investigation to determine whether it needs to be defined as a modem-specific debugging service. For example, modem peripherals may focus mostly on network monitoring and its parameters. In the R&D phase, there can be a need to fine-tune and monitor modem parameters, especially for network connectivity.

Forum.Nokia

5. Technical benefits

5.1 Flexible architecture

One of the advantages of Extension Architecture is that it is quite effortless to change software or hardware partitioning between the application engine and modem. H_IN provides multiplexing, service discovery, and message routing services on top of low interconnect services. This layer ensures that messages are routed correctly between peripherals and application and service nodes, regardless of their actual location. Communication itself is transparent to AN and SN and doesn't require any virtual channel definitions by application or services nodes (which are required when using 3GPP 27.010 AT command multiplexing services).

Extension Architecture/DIP is a generic communication framework, and therefore is not limited to a modem peripheral. It can be used as a low-level hardware communication framework in an embedded system, or as a client/server programming API within a desktop system.

5.2 Security

Extension Architecture/DIP provides a common architecture that can be used by a variety of platforms and modem subsystems. This means it will be open to inspection by a large number of people. The extendable nature of the architecture means there can be a lot of common code utilising runtime service discovery. This, in turn, means there will be numerous opportunities to reuse code between platforms and modem subsystems.

As a result, the code will be subjected to threat analysis and security verification by a number of organisations. Security flaws will be detected and corrected quickly. This contrasts with proprietary security systems, which have fewer resources verifying them and whose strength may depend upon obfuscation.

Further, the aim of Extension Architecture is to be well documented in order for it to be efficiently shared. Part of this documentation will be the rationale behind the system design. There will also be high level drawings and descriptions showing the end to end operation of the architecture. These will be of great benefit to those trying to implement and maintain the software. This is in contrast to legacy proprietary designs where only source code and low-level documentation are available. Often, in these cases, the reader needs to reverse-engineer the system design and reconstruct the rationale in order to understand it.

Extension Architecture provides a way to communicate securely between processors using different operating systems. Each operating system may have its own memory separation and secure Inter Process Communication (IPC). When writing source code for a particular OS, it is possible to dictate the visibility of procedures to one another. Hence, it is possible to limit the number of servers that are able to communicate with one another to a specific set within the OS. However, in addition to this, Extension Architecture provides a way of maintaining process separation across the boundaries of different operating systems. It is possible to dictate a group of

servers which are able to communicate with one another regardless of which OS they reside in. Servers outside of this group are not allowed to communicate with them.

5.3 R&D efficiency

Underlying service discovery, transportation, and networking services are already open sourced and productised by the industry. The DIP stack is available for Linux-based devices, and proof-of-concept implementation exists for other platforms (for example, Symbian) as well.

Message encoding and decoding is autogenerated, and the work effort for such communication should be considerable lower when compared to traditional byte-by-byte message handling.

Implementing the DIP stack and DIP modem adaptation may require a major short-term investment on the modem side, but it will enable benefits that cannot be achieved when implementing multiple adaptation layers for different application environments.

5.4 Quality and time to market

The proposed open-source approach has the benefit of applying the resources of many platforms, OEMs, and modem vendors to a common set of software components. The number of different software components needed is reduced while the number of common Extension Architecture components used by all organisations is increased. Overall, there will be a larger number of people inspecting, testing, and correcting the common code. Common components can easily be reused by different organisations.

The use of well-defined interfaces allows the development of extensive test sets and conformance test tools. Using this common interface, there is an opportunity to prove a new modem independently, on an older, stable platform, before integrating it with a newer platform.

Both modem vendors and OEMs can spend less effort integrating different combinations of modems and platforms. This allows them to concentrate their expertise on improving the quality of the modem solution.

The DIP stack is not just used for modems, and therefore it will be constantly improved and maintained within the scope of other peripherals as well, further improving performance and quality.

Specifications are based on WSDL, and message serialisation is autogenerated, improving specification management and message parsing because there is no need to handle messages as an error-prone binary or ASCII format.

5.5 Cost efficiency and cooperation

Extension Architecture contributes to cooperation and cost efficiency by providing an open-sourced, licensing-free communication framework. The DIP stack and its service descriptions are licensed under BSD, which will allow integration to closed platforms.

Forum.Nokia

6. Conclusions

This document has considered the application of Extension Architecture to modem integration. Extension Architecture provides a consistently layered model for mobile device architecture. This allows for flexibility in the way that services are provided across hardware boundaries within a chipset.

The document examined not only protocol services, but also all services needed for integration. The modem system, manufacturing, and R&D needs were all considered with respect to Extension Architecture. Further, the use of services that are common to typical Extension Architecture/DIP subsystems were applied to the specific modem case.

It was shown that Extension Architecture can accommodate a variety of platforms and modem suppliers while still employing a high proportion of common software components. The security of modem services was analysed briefly in terms of threats and countermeasures. The built-in Extension Architecture/DIP security features were described with respect to the modem case as well as the ability to extend to more advanced security features.

An example WSDL file was used to illustrate an SMS example of how services are defined. Further, each modem service was considered, to show how Extension Architecture can be applied.

Finally, the technical benefits of Extension Architecture were highlighted with respect to modem integration. WSDL definitions have not yet been created for modem protocol services, modem system services, and some of the common services; otherwise, Extension Architecture/DIP is ready to be deployed as the standard modem integration architecture.

Forum.Nokia

7. Terms and abbreviations

Term or abbreviation	Meaning
Application engine	Software and hardware (including a processor and memory) responsible for running applications and providing an environment for the other subsystems to join.
Application subsystem	An Extension Architecture Sub-System (EA SS) implemented within the application engine
AN	Extension Architecture Application Node
Baseband processor	See Modem.
Cellular processor	See Modem.
DIP	Device Interconnect Protocol; a communication framework based on NoTA (NoTA - Architecture[8], 2008).
EA	Extension Architecture; contains DIP and services.
EA Modem Services	Extension Architecture services defined for the modem peripheral use cases: protocol, system, and common services.
Hardware/physical interconnect	Physical connection between two hardware components, for instance, USB connection, MIPI standardised connections, shared memory, or some proprietary nonstandardised connection.
H_IF	High Interconnect Interface
H_IN	High Interconnect
L_IN	Low Interconnect
MIPI	The Mobile Industry Processor Interface (MIPI®) Alliance; an open membership organisation that includes leading companies in the mobile industry with the shared objective of defining and promoting open specifications for interfaces in mobile terminals. (MIPI Alliance[7], 2010)
Modem	Peripheral that handles communication between the device and base stations. Typically, the modem is a separate and independent software asset and hardware block that is connected to an application processor.
Modem Subsystem	An Extension Architecture Sub-System (EA SS) implemented within the modem engine
NoTA	Network on Terminal Architecture (NoTA - Architecture[8], 2008).

Peripheral	An independent hardware and software component that is connected to the device through some hardware interconnect and software interface. Typically, a peripheral requires a device driver that communicates with the hardware by using a specific software interface.
RM SN	Extension Architecture Resource Manager Service Node
SC	Security Controller
SIM	Subscriber Identity Module
SN	Extension Architecture Service Node
Subsystem	Collection of software components that create a logical and functional entity. For example, a UICC subsystem may contain UICC software interfaces, UICC services, and a UICC hardware driver.
Telephony Integration API	API that is typically part of middleware layer and abstracts the underneath modem software and hardware interfaces, for instance, Android Radio Interface Layer (Android Developers - Telephony), MeeGo oFono (Intel, Nokia), and Symbian CTSY Dispatcher (Nokia).
UICC	Universal Integrated Circuit Card; an evolution of the SIM card, it can contain more identification services (for instance, for NFC or multimedia services) than a traditional SIM application for wireless networks.
WSDL	Web Services Description Language (Web Services Description Language[15] [WSDL], 2001)

Forum.Nokia

8. References

[1] *Android Developers - Telephony* (n.d.). android.telephony | Android Developers: developer.android.com/reference/android/telephony/package-summary.html

[2] *Architecture White Paper: Peripheral Abstraction* (2010, June 30). Forum Nokia: www.forum.nokia.com/info/sw.nokia.com/id/8146f7f4-e22c-49b7-9109-2c68393cbd7a/Architecture_White_Paper_Peripheral_Abstraction.html

[3] *boot in NoTA - Gitorious* (2009, Dec). gitorious.org/NoTA: gitorious.org/nota/boot

[4] *Documentation | oFono* (n.d.). oFono Open Source Telephony: ofono.org/documentation

[5] *filesn in NoTA - Gitorious* (2010, Jan). gitorious.org/NoTA: gitorious.org/nota/filesn

[6] *ilProxy in NoTA - Gitorious* (2010, Jan). gitorious.org/NoTA: gitorious.org/nota/ilproxy

[7] *MIPI Alliance* (2010). MIPI Alliance: www.mipi.org

[8] *NoTA - Architecture* (2008). NoTA: www.notaworld.org/nota/architecture

[9] *Nota Core Overview* (n.d.). Nota World: www.notaworld.org/documentation/nota_core#Socket_Types

[10] *NoTA Release 3 Specification Documents* (2010). Forum Nokia: projects.forum.nokia.com/NoTA/wiki#no1

[11] *Open Modem Interface Benefits for the Mobile Device Industry* (2010). Forum Nokia: www.forum.nokia.com/info/sw.nokia.com/id/8493c38e-aed8-4620-97c8-4e24702213d3/Open_Modem_Interface_Industry_Benefits.html

[12] Symbian Foundation (n.d.). *Central Repository*. Symbian Developer: developer.symbian.org/wiki/index.php/Central_Repository

[13] *Symbian OS Communication Architecture* (n.d.). Forum Nokia: wiki.forum.nokia.com/index.php/Symbian_OS_Communication_Architecture

[14] *System Architecture Flexibility* (2010, June 30). Forum Nokia: www.forum.nokia.com/info/sw.nokia.com/id/e1abfc39-6591-4a2e-9068-0deefbb38ff8/System_Architecture_Flexibility.html

[15] *Web Services Description Language (WSDL)* (2001). w3.org: www.w3.org/TR/wsdl

Copyright © 2010 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are trademarks or registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this document at any time, without notice.

Licence

A licence is hereby granted to download and print a copy of this document for personal use only. No other licence to any other intellectual property rights is granted herein.