OULU UNIVERSITY OF
APPLIED SCIENCES

Jarko Poutiainen

MER CELLULAR TELEPHONY ARCHITECTURE

Software Analysis and Description

MER CELLULAR TELEPHONY ARCHITECTURE

Software Analysis and Description

Jarko Poutiainen
Master'sThesis
Autumn 2015
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Jarko Poutiainen
Title of Master's thesis: Mer Cellular Telephony Architecture
Supervisor: Lauri Pirttiaho
Term and year of completion: Autumn 2015 Number of Pages: 66

Modern mobile phones use multi-purpose operating systems similar to those used in desktop computers providing a similar set of features for end users. The essential feature set needed by a mobile phone contains the cellular modem telephony support. This thesis describes and analyses the cellular telephony software architecture of the Mer system.

The software covered in this work is publicly available software. Therefore this work covers how that software can be utilized in other systems than the original system, the concepts of free and open source, as well as software licensing.

The work covers the concept of Android and its cellular telephony support parts that are relevant to this work. A section of this work is dedicated to how the parts of Android, that are relevant to cellular telephony, can be reused. The concept of Mer as a system is described and how it relates to Android.

The main focus of this thesis is in describing the Mer cellular telephony architecture and its parts. This thesis also presents some findings worth noting regarding the Mer cellular telephony architecture. The conclusion chapter contains the personal opinions formed of the system that was analysed and described.

Keywords
Cellular telephony, software architecture, open source, free software, Mer

PREFACE

The idea for this thesis formed during the time I worked for Jolla Ltd. This thesis however is not at the request of the aforementioned company nor by anybody else. It is done out of personal and professional interest. Originally, it was to help me on my work. The findings of this work has been used for project and feature planning. The idea crystallized during the autumn period of 2014 and the work itself was done during the year 2015.

I would like to thank my supervisor Lauri Pirttiaho for his patience. I would also like to thank many of my colleagues who worked with me on the Mer project for Jolla, for their willingness to always lend a hand or share a thought. I would especially like to thank my son for being both patient and understanding despite his age and my wife for support and believing in me completing this work even when I did not.

Oulu, Finland, November 2015

Jarko Poutiainen

CONTENTS

ABBREVIATIONS

AIDL	Android Interface Definition Language
ANSI	American National Standards Institute
AOSP	Android Open Source Project
API	Application Protocol Interface
BSD	Berkeley Software Distribution
CBS	Cell Broadcast Service
CID	Cell Id
DHCP	Dynamic Host Communication Protocol
FOSS	Free and Open Source Software
FSF	Free Software Foundation
GPL	General Public License
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IMEI	International Mobile Equipment Identity
IP	Internet Protocol
IPC	Inter Process Communication
ISI	Intelligent Service Interface
ISO	International Organisation for Standardization
LAC	Location Area Code
LGPL	Lesser General Public License
MCC	Mobile Country Code

MCE	Mode Control Entity
MMS	Multimedia Messaging Service
MNC	Mobile Network Code
NITZ	Network Identity and Time Zone
OHA	Open Handset Alliance
OS	Operating System
OSD	Open Source Definition
OSI	Open Source Initiative
POSIX	Portable Operating System Interface
QML	Qt Modelling Language
RAM	Random Access Memory
RIL	Radio Interface Layer
RILD	RIL Daemon
SIG	Special Interest Group
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service
SSU	Seamless Software Update
TCP	Transmission Control Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
VoIP	Voice over IP
XMPP	Extensible Messaging and Presence Protocol

1 INTRODUCTION

At the heart of every mobile phone lies the support for cellular telephony which enables the user to make, for example, phone calls and send messages. Therefore, still today a key service for any operating system used in a mobile phone is the cellular telephony support. Cellular telephony support covers the applications, the middleware and the cellular modem adaptation software.

Only a few open source based operating systems for mobile devices with cellular modem support exist besides Android. Android is a widely used, well known and documented system. However, Android is mainly developed and controlled by a single company, therefore making any competing solution interesting, especially if it is more “open” or “free” would be appealing. In December 2013 Jolla smartphone sales started using Sailfish as its operating system. The core of the Sailfish OS is based on the Mer distribution which is free and open source software.

Although the source code of the Mer is freely available, hardly any documentation exists or it is inconsistent. The purpose of this work is to describe and analyse the cellular telephony support in the Mer system. What it consists of, how well it covers the functionality required to build a mobile phone and analyse the found caveats. Typically, the implementation of a modem hardware adaptation is not made publicly available, as it is the case with Android. There usually are some requirements regarding the background system, e.g. the used system software and application framework, which the cellular telephony software stack uses. These dependencies are introduced.

Audio handling is not part of this work since audio adaptation, including cellular voice, is usually handled as its own entity. For example, Android has separate interfaces for modem adaptation and audio adaptation.

2 FREE AND OPEN SOURCE SOFTWARE

There are many ways of distributing the software but in principle it can be distributed in two forms: Either providing the source code in an open source form or providing the software as binary and not releasing the source code that the binary is based on. The latter form can be called a closed source. In the closed source the source code is kept as a trade secret and thus not provided freely but it may be available for a payment or possibly not at all. In the open source form the source code is made publically available to everyone to read. However, the licensing affects how “free” the provided source code is. There are many ways of describing what open source is. One way is to consider open source being divided in two: The open source software and free software.

FOSS is an abbreviation of a term Free and Open Source Software trying to include both the terms open source and free software into one. However, the use of this term to cover the free software is questioned by the Free Software Foundation (FSF). (Stallman 2014, date of retrieval 3.11.2015.)

2.1 Open source

Software where the source code is provided publically could be considered by some as an open source software. Here the term “open source” is used in the way defined by the Open Source Initiative (OSI). The Open Source Initiative is a non-profit organisation advocating the benefits of open source development (Open Source Initiative 2015a, date of retrieval 3.11.2015). It was formed in 1998 and one of the first tasks it did was drafting the Open Source Definition (OSD) to define the term open source and distinguish it from the term free software that was considered by some as being philosophically and politically focused (Open Source Initiative 2015b, date of retrieval 3.11.2015).

The OSD basically defines what kind of license can be called an open source license, or in other words under what kind of license the software and the source code should be distributed so that it can be labelled as being “open source”. The OSD specifically dictates that the term open source software does not just mean access to source code but that it must fulfil a list of criteria to be called one.

Among these criteria are the freedom of distribution, the freedom of modification and the freedom of other software, respectively named as “Free Distribution”, “Derived Work” and “License Must Not Restrict Other Software”. The “Free Distribution” criterion specifically dictates that the used license shall not restrict the selling or giving away of the software as a part of other software, basically giving the right to others to reuse the software. The “Derived Work” criterion dictates that the license allows the derived work and modifications. The “License Must Not Restrict Other Software” criterion dictates that the license used must not force the other software released together with the licensed software to be licensed with the same license or as an open source software. In other words, this criterion gives freedom for mixing the closed source and the open source software or in general mixing the software distributed under different licenses. (Open Source Initiative 2015c, date of retrieval 3.11.2015.)

2.2 Free software

Source code, whether distributed as closed or open source, can in a sense be considered free if there is no payment required for example for running the program. However, in this context the “free” does not mean free from payment but rather freedom to use it and to use it in ways other than it was originally designed for. Here the term free software follows the definition published by the FSF. The FSF is a non-profit organisation promoting the freedom to create, modify, distribute and study computer software and it is campaigning against what it sees as a threat to computer users’ freedoms (Free Software Foundation 2015a, date of retrieval 3.11.2015).

The definition of free software provided by FSF consists of four freedoms that the recipient of the software should have for the software to be called free. These freedoms give the recipient the freedom to run the program for any purpose, to study and modify it, as well as the freedom to redistribute it and the modified versions of it. These freedoms require access to source code as otherwise the freedoms could not be considered meaningful. (Free Software Foundation 2015b, date of retrieval 3.11.2015.)

It is difficult to make a difference between what is considered an “open source” and what is considered a “free software”. Essentially the difference between these two terms is in the philosophical nature of the term “free software”, according to Richard Stallman (2015, date of retrieval 3.11.2015). Basically, the idea of free software is that all software should be free for anybody to use and modify it and therefore it is fundamentally against proprietary and closed source software. Open source in distinction advocates the practical benefits of the development model of distributing the software in an open form. Advocates of open source are not as such against the closed source or proprietary software.

Neither open source nor free software are labels against financially benefiting from the software. The business models behind the financial gain between open and free software and that of proprietary (closed source) software of course differ.

2.3 Software community work

The community work can be seen as an integral part of both open source development and free software development. The term community here refers to a group of developers and others alike who contribute to a software project with an agreed goal. This is done in a way that is open to all the members of the community and that, at least in principle, the community is open for anybody to participate in. The principle is that one can, for example, make a fork of the community project software, modify it, use it and provide the modifications back to the community. The community then evaluates the modifications and either accepts or rejects the modification. In the case of rejection an explanation for the rejection should be provided so that the developer may further revise the modification and submit it for re-evaluation. This bi-directional communication then provides the base for the quality benefits provided by the community development model. Furthermore, once the modifications are accepted by the community, these modifications become part of the work of others, whose work is based on the community project, i.e. at least in principle, gaining the benefit of the wider user and developer base and getting a better test coverage by being run on other devices and software platforms than would otherwise be possible.

Both open source and free software movements promote the idea of community work but from different perspectives. The open source movement sees it as a beneficial development model where the developers and users alike benefit from the work done by the community, whereas advocates for free software see the developer community as an integral part of the philosophy of the movement. The difference is in the point of view: Where open source community sees the benefits gained by the developer from the community, the free software community sees the benefits that the community gains from the developers. The licensing models are the manifestation of this difference. As explained later in this chapter the truly "open" license lets the user have access to the source code, modify it and use it without any obligations, whereas a truly "free" software license forces the user to return all the modifications back to the community. It could be said that the difference in the approach between open source and free software communities is that they are two sides of the same coin. On the one hand the community needs to provide something for the members to join in and on the other hand the members need to contribute back to the community for the community to stay alive.

2.4 Fork

A fork is a commonly used term for a copy of a source code of a specific software at a specific moment of time, used as a baseline for parallel work. A fork may be used, for example, when the community is seen as being too slow or reluctant for accepting modifications needed by the developer or developers in their own work. The work done in a fork may later be accepted by the community and be joined back to the mainline. However, the fork may also become a baseline for a new community project aiming towards a goal different from the original community project.

2.5 Free and open source software licensing

A software license is a way to control the use and distribution of the software by legally granting and/or restricting the rights of the recipient of the software. A multitude of different software licenses exists and anybody can write their own license. Licensing can be seen as a way how the principles of open source and

free software are enforced in practice. The software licenses in general can be classified based on how restrictive they are concerning the use and redistribution of the source code.

In principle, any software that is published without copyright can be considered free software. However, releasing the software without any restrictions enables the reuse of that software in a non-free way because it does not restrict the redistribution of that software with a non-free copyright or license. It could be said that the free software licenses can be more restrictive than for example licenses used with proprietary software. For example, in terms of preventing the changing of the used license and in redistribution of the software and any software that is considered as derivative work of that software, the free software license can be more restrictive. In other words, a restrictive free software license is restrictive in terms of changing the license but permissive in use and redistribution under the same license. In general, the free software licenses can be divided into those that do not restrict redistribution or use in anyway and to those that limit redistribution and use in some ways as well as to those that prevent redistribution and use in any way that could be considered as taking away the freedoms discussed in subsection 2.2. Another way to divide the free software licenses is to divide them into permissive and restrictive or copyleft licenses. The difference between these two is similar to that of the open source software and the free software although more concrete in a sense that the ramifications of the used license on how the software can be used and redistributed. (Free Software Foundation 2015c, date of retrieval 4.11.2015; Free Software Foundation 2015d, date of retrieval 4.11.2015.)

2.5.1 Permissive licenses

At least in principle the permissive licenses are licenses that give freedom to use and distribute the software, in anyway wanted. This includes, for example, the freedom of redistributing the modification under another license including non-free license as well as the use of the software together with non-free software. Apache License 2.0 and BSD (Berkeley Software Distribution) 2 and 3 -clause licenses are examples of permissive licenses. (Free Software Foundation 2015c, date of retrieval 4.11.2015.)

Both mentioned BSD licenses give the freedom of distribution and use without the reuse of the license although the unmodified source code must retain the original license. This means that software released using either one of the mentioned BSD licenses can be used as a part of almost any program under almost any license including non-free and free. Only those licenses that are explicitly incompatible cannot be mixed with them. (Open Source Initiative 2015d, date of retrieval 4.11.2015; Open Source Initiative 2015e, date of retrieval 4.11.2015.)

Apache license 2.0 is in principle as permissive as BSD but is considered more complete as it covers, for example, patent rights such as patent termination clause that prevents enforcing patents relating to the licensed software. Apache license 2.0 also expects the use of the same license on unmodified parts of the software. (The Apache software foundation, 2015, date of retrieval 4.11.2015.)

2.5.2 Copyleft licenses

Copyleft licenses are more restrictive than permissive licenses regarding the redistribution and use of the software licensed under copyleft license. The restrictiveness of copyleft license is due to the principle behind such a license. The principle of copyleft license is to guarantee that the freedoms to use and modify the software, is passed along when redistributed with or without changes. The intention is that the freedoms includes any work that is derived from the copyleft licensed work. In short, the principle is to restrict the rights regarding further licensing to preserve the freedoms of the software. GNU General Public License and GNU Lesser General Public license are examples of copyleft licenses. (Open Source Initiative 2015d, date of retrieval 4.11.2015.)

GNU General Public License (GPL) is a strong copyleft license which requires not only the work itself but all the derivative work to be licensed with the same license. In principle, using the software licensed under GPL as a part of another software to form a program means that the whole program becomes covered by the GPL license. (Free Software Foundation 2007a, date of retrieval 4.11.2015.)

GNU Lesser General Public License (LGPL) is less restrictive than GPL and is considered as a weaker copyleft license. Unlike GPL, LGPL permits linking the non-free and free software modules, covered by different license without enforcing LGPL license to cover the linked module. (Free Software Foundation 2007b, date of retrieval 4.11.2015.)

3 ANDROID

Android was in the second quarter of 2015 by far the most used operating system in smartphones. With its over 82% market share it was selling more than all its competitors combined (International Data Corporation 2015, date of retrieval 21.10.2015; Gartner Inc. 2015, date of retrieval 21.10.2015). Although Android is mostly used in mobile phones and similar devices, such as tablets, it is used in other devices as well, such as in televisions and cars. (Google Inc. 2015a, date of retrieval 20.10.2015). Due to the growing ecosystem around Android, it is in a sense becoming the de facto application framework for mobile devices at least in consumer electronics as many hardware design companies are providing Android support for their products. Amongst these companies there are such as ARM, TI, Qualcomm, Freescale and NVidia (Yaghmour 2013, 8). They alone do not make an ecosystem. Many of the major device manufacturers such as Sony, Samsung, HTC, Motorola, LG, Dell, and ASUS have shipped devices running Android (Yaghmour 2013, 8). To complete the ecosystem a number of companies and private individuals have made a number of applications for Android that can be retrieved and installed from for example https://play.google.com/store.

Initially, Android was developed by Android Inc. The Android Inc. was bought by Google Inc. in 2005 (Yaghmour 2013, 1-2). In 2007 Google Inc. made the Android public, together with the founding of Open Handset Alliance (OHA). OHA was said to be devoted to advancing open standards for mobile devices although the role of OHA is somewhat unclear (Yaghmour 2013, 2, 8). The source code is released under open source licenses by Google. However, usually devices using Android are shipped with a mixture of open source and proprietary software. This mixture is possible through the licenses used in Android and partly through the architectural decisions made in Android. (Yaghmour 2013, 10-13.)

Most of the code released by Google to Android Open Source Project (AOSP), is licensed under Apache License 2.0 and some with BSD (Yaghmour 2013, 12). As previously explained in section 2.5.1 of this work, neither Apache License 2.0 nor BSD require the derived work to be published. This means that software based on software under either license need not be published. As mentioned

before, the architectural decisions made in Android enable mixing the software under different licensing models. Linux kernel is licensed under GPL and, as explained in section 2.5.2 of this work, it requires that not only the changes made to the software licensed under GPL must be made public but so too must all the derivative works be licensed under GPL and therefore made public. However, Linus Torvalds, the original contributor of Linux kernel, added a note to a file named “COPYING” in the kernel sources stating that the GPL license is subject only to the kernel sources and therefore any software running on top of the kernel is not considered as a derivative work (Torvalds 1994, date of retrieval 20.10.2015). A good example of how the architecture enables combining different licensing models is the using of sockets by means of communication between different components. Socket communication is considered as a mechanism which is normally used between two separate programs (Free Software Foundation 2015e, date of retrieval 20.10.2015).

In Android devices the most logical part to contain proprietary code would be the Hardware Abstraction Layer (HAL), as that by its nature contains the hardware specific code. Radio Interface Layer (RIL), as an example, is typically provided by the company who designed the modem. AOSP only provides reference implementation of RIL and through both the design and licensing model the hardware vendor may implement and use its own version of RIL without publishing the code. This is because the communication between RIL and the application layer is done through a socket. Thus, the RIL cannot be considered a derived work and the public parts of RIL implementation, which includes the RIL daemon, used in the final products are licensed with Apache License 2.0. The conclusions made in this paragraph are based on analysing the contents of the RIL software repository. (Google Git 2015a, Date of retrieval 20.10.2015).

The development of Android is done in private by Google until it is ready to publish the source code without necessarily any prior announcement of when and/or what is published. In this the Android development differs from the most open source projects. Usually, the open source projects have public development branches and a forum for developers to discuss the upcoming changes. Although the software provided by Google may be open, the development of the software itself

is not and therefore Google can in principle at any point change anything they choose without prior warning. (Yaghmour 2013, 5-6.)

The released Android as such is not likely to be automatically ready for a specific hardware but will require extra work as the official source code provided by Google typically runs only on the so called lead devices. So, Google does not provide hardware specific adaptation software other than those used to develop the new Android release. (Yaghmour 2013, 7, 51.)

3.1 Android Architecture

As shown in the Figure 1, the Android system can be divided into five major parts: The applications and application framework, the system services, the hardware abstraction, the kernel and the inter-process communication (IPC) mechanism enabling the communication between the different parts (The Android Open Source Project 2015a, date of retrieval 21.10.2015). In a sense, to a certain extent, most of the operating systems available could be described in a similar manner. This division can help to compare operating systems as well as help to understand the purpose of the parts of the system and their relation to the rest of the system, as is done later in this and the following chapters.

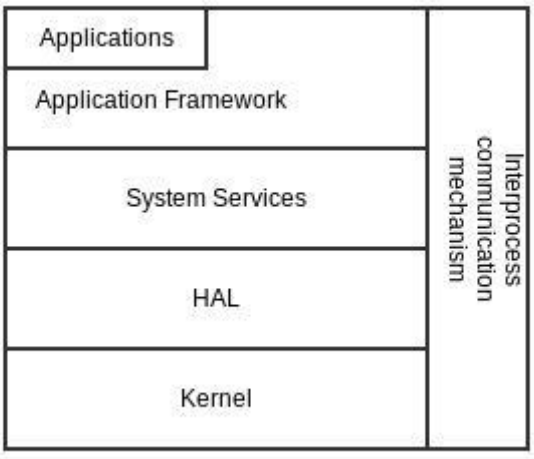

FIGURE 1. Android system architecture

3.1.1 Applications and Application framework

For any operating system to be a complete software platform from the end user perspective, it will need application software, that is, pieces of software carrying out specific functionality such as word processing, a tic-tac-toe game or sending an SMS (Short Message Service) message. To be able to provide these functionalities the application software needs system software to execute because the system software manages and integrates the device capabilities. An application framework is a software framework used by application software developers to develop applications with a standard structure and a means of using system software. Graphical user interface (GUI) is typically developed in an application by using an application framework. Both the application software and the application framework may to a certain degree be operating system agnostic. For an application to be operating system agnostic it needs to have been developed by using the so called cross-platform application framework such as Qt, that can run in two or more operating systems. Android applications are developed using Java language, which is CPU architecture independent. Although Java is a cross-platform computing framework, Android deploy Java in a manner that is different from others. One difference is that it relies on Dalvik or Android runtime instead of Java Virtual Machine. (The Android Open Source Project 2015b, date of retrieval 19.11.2015; Yaghmour 2013, 60-63.)

3.1.2 Inter-process communication mechanism

The main IPC mechanism used in Android is called Binder. The Android Binder is based on the work of the OpenBinder project and although it is inspired by it, it is not a derivative of it but a work of its own. As its name hints at, the purpose of the Binder IPC is to bind the Android processes by providing IPC mechanism so that one process can call a routine of another process in Android. Although Binder provides a means for one application to talk to another, its main purpose is to provide a means for application frameworks, hence the applications, to communicate with the Android system services. The application developers do not need to be aware of the existence of the Binder itself as they can use Android Interface Definition Language (AIDL) as a means to create IPC between the client and a service. AIDL hides the Binder. Binder is very much a part of the fabric of

which the Android is made of. The way it provides system services differs from that of the more traditional Linux approach. One typical way to extend a system's functionality in Linux would be to implement a new daemon to provide a new service. Whereas Binder provides the same by offering a means to add remotely invocable objects that can be implemented with any desired language. These objects may have their own process or share the same process space with other remote services. (Yaghmour 2013, 40-41.)

3.1.3 System services

As previously mentioned for the applications to be executable there needs to be system services available as they manage and integrate the devices capabilities. System services are pieces of software running in the background without a direct interaction with the user. They are typically started during the system boot. System services provide the means of communication with any peripheral devices. They take care of the error handling and the program restart, if so required, as well as the file system access.

In Android the system services are mostly bundled in two servers: the Media server and the System server. The Media server covers everything related to playing or recording the media, e.g. camera services and audio related services. Most of the system services run in the System server, amongst them are Window Manager, Power Manager and Activity Manager. (Yaghmour 2013, 64-65.)

3.1.4 HAL

In any operating system there needs to be a means to connect to the actual hardware while keeping the rest of the software stack hardware agnostic. In Android the hardware abstraction is done in the HAL. Unlike typical Linux distribution the Android does not rely on the kernel device drivers to provide the hardware abstraction but on the shared libraries provided by the hardware manufacturers that may or may not be part of the official AOSP and may or may not be published. In practice the principle of Android HAL is simple. It provides a set of header files that introduce a set messages and/or functions, i.e. a HAL Application Protocol Interface (API). The HAL API needs to be implemented in

the shared library provided by the manufacturer. In the end there is still a need for a kernel device driver to exist, the same as with any Linux distribution. However, due to the way HAL is designed, the actual logic as to how a specific hardware is used can be kept as a proprietary software and regardless remains at least to some extent Android specific because of the HAL API. It is possible to design the HAL implementation so that it can be used, with minor effort, in another Linux distribution by simply structuring the implementation so that the driver and the actual abstraction are separated from the HAL API. (Yaghmour 2013 46-50.)

3.1.5 Kernel

As mentioned earlier, the Android kernel is based on the Linux kernel found at http://kernel.org. As with the most of the Linux distributions, it is patched to customize it to the needs of the specific distribution and in this case for Android. The differentiation in Android is to such extent that the Android user-space components will not run with the "standard" Linux kernel. These differences include IPC mechanisms, such as Binder driver and Anonymous Shared Memory (ashmem), memory management systems such as wake locks and low memory killer, and own logging mechanism. The main reason behind these changes/additions is the aim to provide a better support for small memory mobile devices. For example syslog, the default logging mechanism used in most Linux distributions, uses sockets to send messages and stores the information into files, therefore creating task switches and generates writes to the storage devices. The logging mechanism used in Android uses a circular RAM (Random Access Memory) buffer hosted by a kernel avoiding task switches and file writes, which means that by default the information is lost when the system is shutdown. The changes made to the Android kernel do not affect the way the device drivers are implemented. So at least in principle, Android device drivers should work with the default Linux kernel and therefore with any other Linux distribution. Furthermore, some of these changes may eventually end up being a part of the standard Linux kernel. (Yaghmour 2013, 34-45.)

3.2 Reusing the Android software

As stated before, the Android is the most dominant mobile operating system at the moment. That means it has a broad hardware support, and as previously explained, every operating system, on one hand needs a HAL to make it hardware agnostic and on the other hand means that for every hardware there is a need to implement a corresponding abstraction. From device manufacturers point of view the existing hardware support means reduction in both cost and time to market. Thus, an operating system that could reuse the hardware support made for Android would be interesting to device manufacturers.

The upper software layers of Android are harder to integrate to another Linux distribution than the lower layers. This is due to Android’s structure as the dependencies to Android's special properties tend to increase in upper layers. Binder, The Android IPC mechanism, alone makes it very hard to "tear off" any piece of software from the application layer as it is the main mechanism used to communicate with the system services. As explained earlier, Binder also needs changes to the kernel to work. The application framework that hides the system services from the applications uses the Binder. So although it might be possible in some cases to take a single application or a single piece of software from an Android stack with relative ease, basing a set of features on an Android implementation would require a lot of integration work. It may require an integration of so many other pieces of Android that it could be argued if the distribution where one is trying to integrate, becomes a mere Android fork, keeping in mind the fact that Google may at any stage decide to change the behaviour of any part of the system. Hence, one cannot rely on any community support other than the one built around their own fork. This raises a question if such an approach is worthwhile.

The more feasible way to reuse some parts of the Android and gain the benefit of the large hardware support that exists is to find a part that has the least Android specific dependencies. As mentioned earlier, the upper layers of Android stack have more dependencies of Android specific properties and the lower layers less. Kernel and HAL are the least dependant. Although the kernel changes made to Android may on their own be of value in some cases to someone, it is the HAL

that is more likely to provide the greatest benefit. This is because it brings the hardware support and because of its public, unified interface that enables a faster way to adapt an existing system to desired hardware. Unified does not mean here the same as having the same or similar interface throughout the HAL, but within a set of features it provides a unified interface across a multitude of hardware. Therefore, once a system is adapted to a version of Android HAL, it is adapted to all the hardware that supports that version of Android. The problem is, however, that one does not simply take the HAL binaries and use them. One reason for this is the use of Bionic in Android. The Bionic is a standard C library used in Android as opposed to, for example the GNU C library or glibc, as it is more commonly known (Google Git 2015b, date of retrieval 9.11.2015). The GNU C library is the most widely used C library on Linux distributions (Linux Programmer's Manual 2014, date of retrieval 19.10.2015).

To be able to program, using the C-language, while maintaining the portability of the software, the operating system must have a standard C library which provides support for the standard C-language originally published by American National Standards Institute (ANSI). The standard C library provides e.g. functions, macros and type definitions for mathematical computations, string manipulation and memory allocation. (ISO/IEC 9899 2003, date of retrieval 9.11.2015.)

The GNU C library was primarily designed to provide a standard C library following the relevant standards. Besides the standard C-language support, the GNU C library provides support for POSIX standards and some other features as long as they do not conflict with the relevant standards. (Free Software Foundation 2015e, date of retrieval 19.10.2015.)

Bionic is a standard C library developed by Google to be used in Android based systems. The core idea behind Bionic was to have a C library that provides only a lightweight wrapper around the kernel facilities. Bionic is not based on the GNU C library but consists of a mix of the BSD C library and some custom Linux parts. As a whole, it is released under the BSD license. It is not POSIX compliant and has a limited C++ compliancy. It has its own dynamic linker and it is not binary compatible with the GNU C library meaning that you cannot build something

against the GNU C library and dynamically link it against Bionic. (Google Git 2015c, date of retrieval 20.10.2015.)

One option to reuse the Bionic based software together with the GNU C library based software is to port either Bionic to your environment or simply take the Android kernel and Bionic as a base for your environment. Both options, of course, have their difficulties.

Libhybris provides the solution to the incompatibility between the Bionic based software, i.e. Android and the GNU C Library based Linux distributions, without the need to support both libraries in a same system. It allows the use of Bionic based HW adaptation, such as binary-only Android drivers, in a system using the GNU C library. To explain in simple terms how the Libhybris works: it basically converts the Bionic system calls into GNU C library system calls. (Libhybris 2015. Date of retrieval 20.10.2015.)

3.3 RIL

From the telephony point of view the most important part of HAL is the Android RIL. Despite the somewhat misleading name, RIL handles only the cellular modem adaptation of the Android system, not the other radio technologies such as Bluetooth and WiFi.

There is no standard for how the modem software interface must be implemented. Although an AT command based specification exists as a part of 3GPP standards, it is not mandatory (The 3rd Generation Partnership Project (3GPP) 2015, date of retrieval 20.10.2015). Many other interface specifications exist and typically without a public documentation. The implementation and thus the specification is typically a modem supplier's proprietary. Although many AT command based modems exist, they can contain a supplier specific extension to the public command set. This proprietary command set is then typically needed to use the full functionality of the modem in question. The benefits of the reuse of RIL is that it provides support to the multitude of modems, including the ones using a proprietary command set. It also provides the translation of different modem communication interfaces into one fairly standard interface.

3.3.1 RIL Architecture

RIL consists of three main parts located in HAL as shown in Figure 2: The RIL Daemon (RILD), the vendor RIL and the libril. They all share a header file ril.h that defines the modem agnostic interface. The common parts of RIL are licensed under the Apache License version 2.0 but the vendor RIL may use some other license. These conclusions are based on analysing the contents of RIL repository (Google Git 2015a, date of retrieval 20.10.2015).

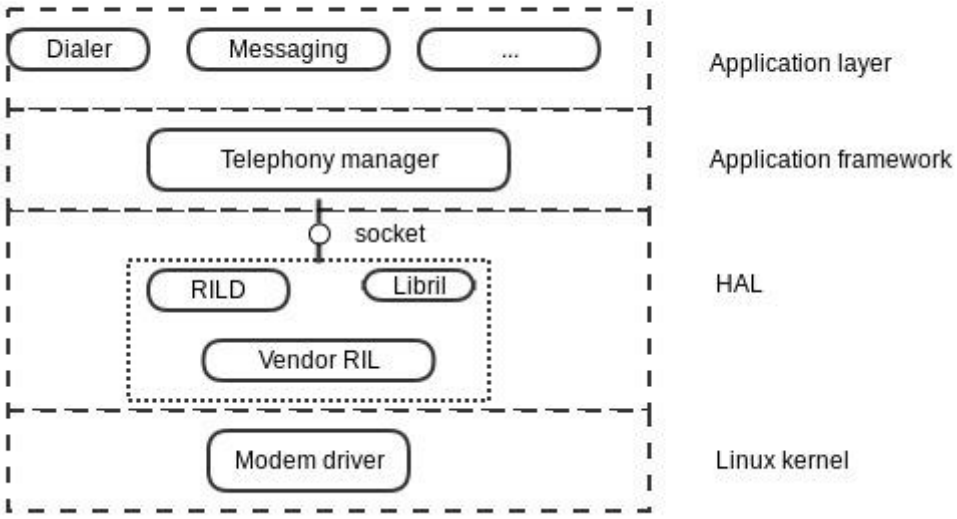

FIGURE 2. Simplified Android telephony architecture

RILD is a relatively small and simple daemon provided as part of the AOSP (Google Git 2015d, date of retrieval 20.10.2015). RILD is dependable on libcutils the Android utils library for C (Google Git 2015e, date of retrieval 20.10.2015). By analysing the source code, it can be concluded that RILD initializes the vendor specific RIL implementation, i.e. vendor RIL, that the vendor RIL is loaded at runtime and that RILD receives the path to the vendor RIL through a configuration file. (Google Git 2015d, date of retrieval 20.10.2015.)

Libril is a shared library linked by both RILD and the vendor RIL statically (Google Git 2015e, date of retrieval 20.10.2015; Google Git 2015g, date of retrieval 20.10.2015; Google Git 2015j, date of retrieval 20.10.2015). Besides the libcutils,

it is dependable on libbinder and libutils (Google Git 2015g, date of retrieval 20.10.2015). However, the libbinder is, besides the libcutils and the Android utility function library libutils, dependable only on liblog, the logger interface (Google Git 2015h, date of retrieval 20.10.2015). Libril is a part of the AOSP like RILD. By analysing the source code, it can be stated that libril provides RIL event mechanism and implements the communication mechanism between the application framework and vendor RIL. (Google Git 2015f, date of retrieval 20.10.2015.)

The vendor RIL handles the communication with the modem driver and the modem specific abstraction. A simple reference implementation of vendor RIL exists, supplied as a part of AOSP. However, the vendor RIL may be structured as the supplier prefers and it can consist of several modules. These conclusions are based on analysing the contents of reference RIL repository (Google Git 2015i, date of retrieval 20.10.2015).

3.3.2 RIL communication

Running RIL in another system is not enough for the purposes of reusing the Android modem support. There is also a need to establish the communication with the RIL, too. As shown in Figure 2, the communication between RIL and the application framework is through a socket.

There are two types of commands used in RIL interaction through the socket: The solicited commands and unsolicited responses. The solicited commands are originated by the application framework and include their responses. The unsolicited responses are indications originating from the modem. For example, the indication of incoming call is received as an unsolicited response. Another way to describe this is from the implementation view. As explained earlier, the communication with RIL is done through a socket, therefore there are the socket read and socket write related commands. The socket write commands consist of solicited commands whereas the socket read commands consist of unsolicited responses and the responses to the solicited commands. These conclusions are based on analysing the contents of RIL repository and ril.h file available at that

repository. (Google Git 2015a, date of retrieval 20.10.2015; Google Git 2015k, date of retrieval 20.11.2015)

The data passed in a command is constructed as a parcel object. A Parcel object is a container for an IPC message used in Android (Google Inc. 2015b, date of retrieval 20.10.2015). The implementation of a Parcel class is part of libbinder (Google Git 2015h, date of retrieval 9.11.2015).

4 MER

Mer is a free and open source software distribution, targeted at hardware vendors to serve as a middleware for Linux kernel based mobile-oriented operating systems. It is based on the MeeGo project. Architecturally, the Mer derives from the MeeGo 1.3 architecture. If compared to Figure 1 describing the Android system, the Mer could be said to provide the Application framework, the system services, the IPC mechanism and to some extent the hardware adaptation. As Mer is a middleware, it does not come with a kernel or a user interface (UI). However, the term middleware is somewhat unclear and does not specify what is included. For example, whether a hardware adaptation is included or not, depends on what is defined as a hardware adaption. The Mer wiki specifically excludes the hardware adaptation, yet components, such as oFono, contain parts which can be considered to be part of the hardware adaptation, especially when looking at the AT command based modem support. The Mer architecture is a modular architecture, as it uses many software modules provided by a separate projects to create a whole entity. For example, the Mer uses Qt and D-BUS respectively as its application framework and IPC mechanism. Furthermore, ConnMan is used for network handling and oFono to provide a cellular abstraction. These are all components originally developed outside the Mer community and are used as such or with some modifications. (The Mer Wiki 2015a, date of retrieval 10.11.2015; The Mer Wiki, 2012, date of retrieval 10.11.2015.)

According to Wikipedia, there were three operating systems based on Mer by the end of October 2015 (Wikipedia 2015, date of retrieval 10.11.2015). Of these three operating systems, the Nemo Mobile is totally community driven and the Sailfish OS is the only one running in a product available on the market. It is hard to distinguish what is Mer, Nemo Mobile and what is Sailfish OS project work partly due to same people contributing to all of them (Edge 2014, date of retrieval 10.11.2015).

4.1 Nemo Mobile

Nemo Mobile is a project aiming to create an open, community driven operating system. It is a free Linux distribution to be used in mobile devices. It is based on Mer. Since it is aiming to be a complete OS, it includes the UI. During the writing of this work, the Nemo Mobile project had not been very active for more than a year, when looking at the lack of updates on the project Wiki page. (The Mer Wiki 2014, date of retrieval 10.11.2015.)

4.2 Sailfish OS

Sailfish OS is a commercial Linux distribution developed by Jolla Ltd. based on the Mer and Linux kernel (Jolla Ltd. 2015b, date of retrieval 10.11.2015; The Mer Wiki 2015b, date of retrieval 10.11.2015). The Figure 3 is a simplified picture of the Sailfish system. The figure uses the same structure as seen in the one used to describe the Android system. As with the Android, the figure is not a comprehensive illustration of the system.

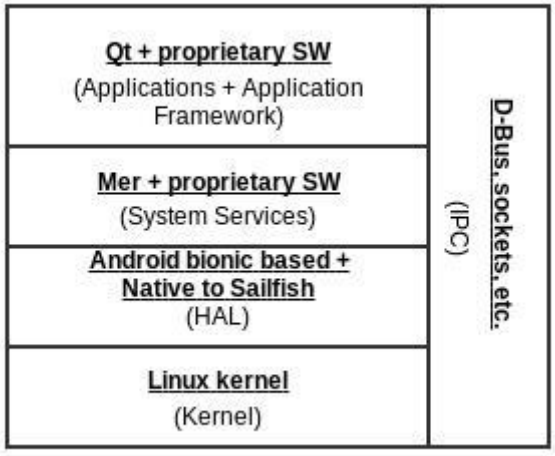

FIGURE 3. Generic Sailfish OS system architecture

On the Sailfish OS homepage it is stated: "The core OS is based on Mer Project, an open source, mobile optimised distribution while the UI is proprietary software owned by Jolla" (Jolla Ltd. 2015a, date of retrieval 10.11.2015). There was a link to the Nemo Mobile bug tracking tool (https://bugs.nemomobile.org/) at the Sailfish OS homepage, which indicates that on some parts Sailfish is based on

Nemo Mobile (Jolla Ltd. 2015a, date of retrieval 10.11.2015). Because Sailfish OS is based on Mer, it uses the D-Bus IPC mechanism, the application framework is based on Qt, and it utilizes the same system services as Mer. During the writing of this work, there were no instructions on how to compile or which version of the Linux kernel should be used together with Mer. It is stated in the Mer project Wiki that Sailfish OS uses “Mer drivers layer and Android drivers through libhybris” (The Mer Wiki 2015b, date of retrieval 10.11.2015). It is unclear what does “Mer drivers layer” stand for, but it could be understood that Sailfish OS utilizes the Android hardware support and that it has some native hardware support, too.

The Sailfish OS has an active developer community (The Mer Wiki 2015c, date of retrieval 10.11.2015) and there are instructions on how to develop applications to Sailfish OS (Jolla Ltd. 2015c, date of retrieval 10.11.2015). The open source software part of the Sailfish OS is available at http://releases.sailfishos.org/sources/.

4.3 Reasoning behind the naming

When comparing different sources, it was not quite clear what was part of the Mer project and what was part of the Nemo Mobile. In GitHub one can find three projects: Nemomobile, Nemomobile-packages and Mer. Furthermore, at some point some of the Github projects were moved to GitLab and merged into the Mer core. For example, the GitHub Nemomobile-packages project oFono was moved to GitLab as part of the Mer core (Nemomobile-packages 2015, date of retrieval 10.11.2015; Mer-core 2015a, date of retrieval 10.11.2015). During the writing of this work there was a following statement at Nemomobile bugzilla: "ANNOUNCEMENT: Nemo Middleware is moving to Mer. File all MW bugs under http://bugs.merproject.org - Mer Core" (Nemomobile 2015a, 10.11.2015). Emphasizing the previous announcement, the following comment could be found in the LWN.net article: “Nemo Mobile was a project started to create the middleware and UI for Mer, but most who are using Mer also use the middleware, so the middleware has been moved into Mer. That means that Nemo Mobile is just UI now, --” (Edge 2014, date of retrieval 10.11.2015). Because of these findings, explained in this subsection, and because of an archived posting in the mer-general posting list regarding the Nemo/Mer merge (Greaves 2015, date of

retrieval 10.11.2015), the author of this work decided that it was more descriptive to use term “Mer” instead of “Nemo/Mer” in the title of this work.

4.4 Qt

Qt is a cross-platform application development framework originally developed by the Trolltech, which is now known as The Qt Company, a wholly owned subsidiary of Digia Plc. There exist a commercial and free software licensed versions of the Qt platform. The supported platforms include Android, Microsoft Windows, iOS and several Linux distributions including Ubuntu and Sailfish. The framework is written with C++ and can be compiled by a standard C++ compiler such as GCC (GNU Compiler Collection). The language used is C++ with platform specific extended features. The Qt has its own integrated development environment (IDE), called Qt Creator. The GUI can be implemented with either C++ by using a Qt Widgets module or with QML (a Qt-specific declarative language) by using a Qt Quick module. Although the Qt comes with the support for C++ and QML languages, there are a third party developed bindings for other languages. Typically, the application UI is implemented with QML and the application logic with C++. The cross-platform build system that comes with Qt is called qmake, which is a user interface for platform native build systems. However, it is not mandatory to use the qmake. (Digia Finland Ltd. 2015a, date of retrieval 10.11.2015.)

The Qt comes with a number of different software modules. The essential modules, such as the Qt Core and the Qt GUI, are general and useful for most of the Qt applications and they are available for all the supported platforms. Add-on modules are for a special purpose and they may or may not be available for all the platforms. An example of an add-on is the Qt D-Bus module. Besides the modules released as part of the Qt, there are modules that are released according to their own schedule. (Digia Finland Ltd. 2015b, date of retrieval 10.11.2015.)

Qt supports creating a custom plugins to extend the functionality. Besides implementing plugins to extend the functionality of an application, it is also

possible to create new custom plugins that extend the functionality of the Qt platform. (Digia Finland Ltd. 2015c, date of retrieval 10.11.2015.)

4.5 D-Bus

The D-Bus is an IPC mechanism used for local communication within the same host defined by the freedesktop.org project. Several implementations or high-level bindings exist, including one for Qt, and there is one developed by the freedesktop.org as a reference. The D-Bus can be used for communication between applications, between system services and applications as well as between system services. There are two types of communication provided: the system wide and a session based. They are called the system bus and the session bus. Every bus has an address and every connection is assigned with a unique name. The communication is message based and the messages are sent to objects. The objects support a particular interface providing methods and signals. The methods are used by clients to send a specific request whereas the signals are used by clients to listen to a specific event emitted by a particular object. The methods are one-to-one connections and provide a two-way communication in which both ends need to be present at all times. The signal is a one-way and one-to-many mechanism where the signal may be emitted even when no client is registered to listen. (Freedesktop.org 2013a, date of retrieval 10.11.2015; Freedesktop.org 2015a, date of retrieval 10.11.2015.)

Mer uses D-Bus as the communication mechanism throughout the cellular support domain. A Nemo Mobile D-Bus QML plugin exists allowing D-Bus access to the Mer system services for QML applications in, e.g. Sailfish OS (Jolla Ltd. 2015d, date of retrieval 10.11.2015).

5 MER CELLULAR TELEPHONY ARCHITECTURE

The Mer cellular telephony architecture consists of many components. The components that are needed to implement the traditional feature set are shown in Figure 4 below. The component is considered relevant if it is a part of the whole that forms the more traditional telephony feature set. In this document that feature set consists of voice call, data, message and contacts handling. Without these it would be hard to consider a device to be a mobile phone. The components that are not in the figure but are relevant, in a sense that they extend or enhance the use of cellular telephony, are explained later in the chapter. The extension features provide a means to implement a feature set that can be found in most of the modern mobile phones but are not present in all of them.

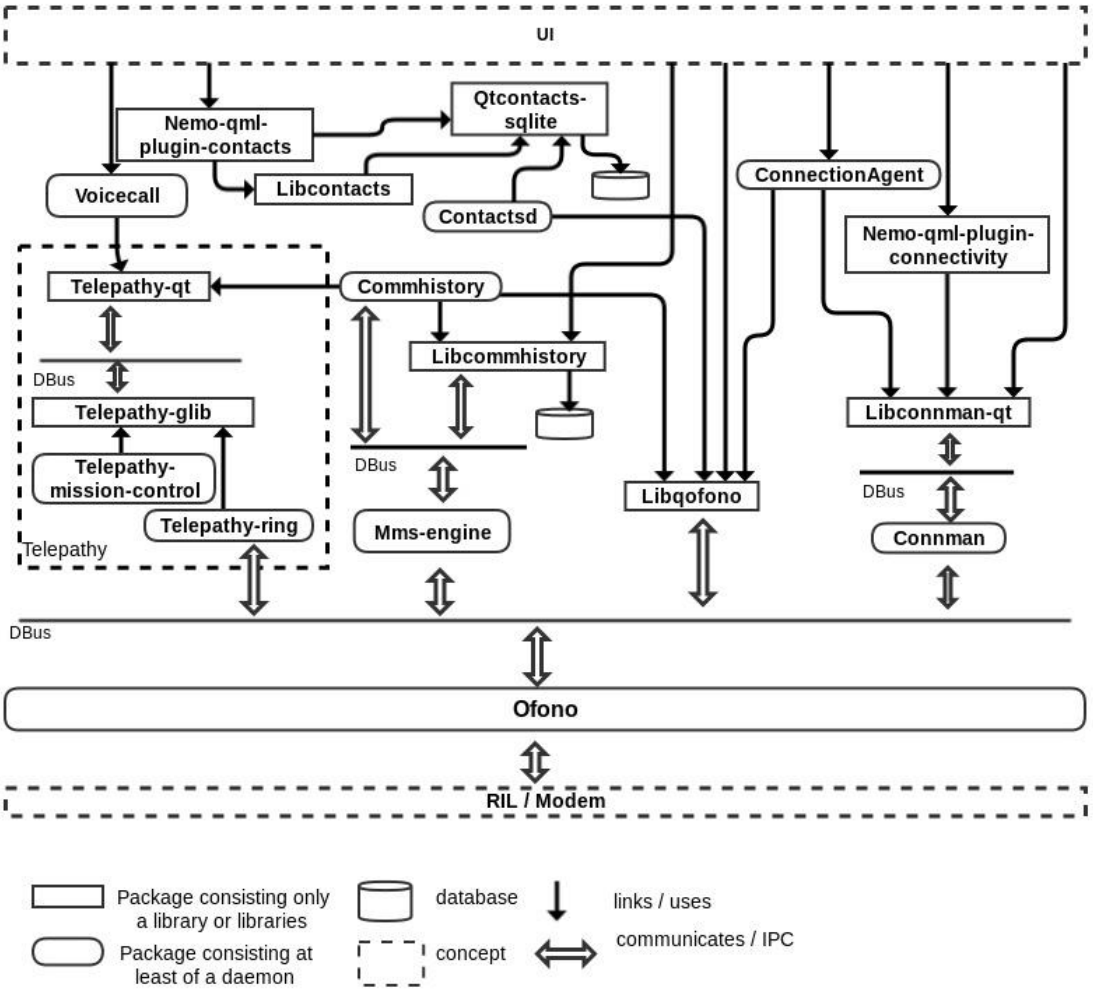

FIGURE 4. Mer Cellular Telephony Architecture

5.1 Mer connection handling

Mer connection handling provides a cellular network access to the Internet. The Mer connection handling consists of five components. One extension component, a Provisioning-service, and four main components: ConnMan, Libconnman-qt, Nemo-qml-plugin-connectivity and ConnectionAgent. The four main components are shown in Figure 4. The descriptions and findings presented in this section are based on the analysis of the software repositories of each corresponding component (Intel corporation 2015a, date of retrieval 5.11.2015; Mer-core 2015b, date of retrieval 5.11.2015; Mer-core 2015c, date of retrieval 5.11.2015; Mer-core 2015d, date of retrieval 5.11.2015; Mer-core 2015e, date of retrieval 5.11.2015; Mer-core 2015f, date of retrieval 5.11.2015).

ConnMan (Intel corporation 2015a, date of retrieval 5.11.2015; Mer-core 2015b, date of retrieval 5.11.2015) is a daemon managing network connections of multiple technologies used in embedded devices. It supports multiple technologies, both wired and wireless, (such as WiFi, Bluetooth and Cellular) through a plugin design. The modular design makes ConnMan extendable, i.e. support for new technologies can be added. Besides the support for new technologies, extending the support for various services can be implemented as plugins, such as configuration methods like DHCP (Dynamic Host Communication Protocol) and the domain name resolving. The desired plugins can be included and/or excluded with parameters (e.g. by using command line interface) by either defining the plugins that should be loaded or by defining the plugins that should not be loaded. The plugins in the context of ConnMan are not loadable libraries built separate from the daemon, but are built as a part of the daemon, so excluding a plugin at runtime will still consume a persistent memory.

It is possible to configure various settings for ConnMan by defining an optional configuration file. The file can, for example, define the preference order of technologies and a list of network interfaces that will not be handled by ConnMan. ConnMan provides a build-time configuration and options, amongst which is the possibility to disable the default built-in plugins, such as WiFi. This modular configurable plugin design is similar to that of oFono.

ConnMan uses oFono as a cellular technology provider and the ConnMan's oFono support is implemented by an oFono plugin. The oFono support is by default enabled but it is not necessary to have the oFono daemon running. This is because the start of the oFono daemon is detected automatically. The ConnMan oFono plugin is not needed to build the Connman.

ConnMan offers a high-level D-Bus API for use by the networking applications of any license. Connman itself is a free software released under the terms of the GPL version 2.0. ConnMan that is part of the Mer project is a fork from the project available at the Git repository at kernel.org (git.kernel.org/pub/scm/network/connman/connman.git).

Libconnman-qt (Mer-core 2015c, date of retrieval 5.11.2015) consists of two libraries that provide access to the Connman D-Bus interfaces as Qt bindings and a plugin for QML applications. These bindings are a direct reflection of the interfaces provided by Connman.

Nemo-qml-plugin-connectivity (Mer-core 2015d, date of retrieval 5.11.2015) is a dynamically loadable custom Mer QML extension plugin for QML networking/connectivity applications using TCP/IP (Transmission Control Protocol / Internet Protocol). It uses Libconnman-qt for communicating with ConnMan to provide the required services.

The main purpose of ConnectionAgent (Mer-core 2015e, date of retrieval 5.11.2015) is to provide a daemon and a plugin library for QML applications to access ConnMan's Agent interface, i.e. "net.connman.Agent", using the UserAgent class provided by the Libconnman-qt library. ConnectionAgent also provides support for the ConnMan "autoconnect" feature to turn the connection/networking technology power on if the corresponding ConnMan service has "autoconnect" set to as "True".

Provisioning-service (Mer-core 2015f, date of retrieval 5.11.2015) is used for Over the Air (OTA) provisioning, i.e. using the operator provided data coming as a push message to initialise oFono Internet data and MMS (Multimedia Messaging Service) data contexts. The data is first received by oFono, which

recognizes it as a push message and forwards it to the Provisioning-service to be further processed. Based on the received data, the Provisioning-service then initializes the oFono data and MMS contexts. After processing the data, the Provisioning-service emits a signal expressing the result (“apnProvisioningSucceeded”, “apnProvisioningPartiallySucceeded”, “apnProvisioningFailed”). It is not mandatory to include the Provisioning-service into an operating system. It only automates the provisioning of the MMS and Internet connections. In a mobile phone using an operating system that contains the Mer, the same data can be provided by the end-user through the UI application.

5.2 Telepathy Framework

The middleware for a cellular voice call and an SMS is provided by using the Telepathy framework. Figure 5 gives an overview of how the Telepathy framework works in principle. Its primary purpose is to abstract the protocol or technology used for a voice call and messaging by giving a unified interface. In other words, from the Telepathy client perspective, sending a message or making a voice call should be the same regardless of the protocol or technology used. For example, in principle it should be no different for a client to make a cellular voice call or an SIP (Session Initiation Protocol) based IP voice call using Telepathy. This is made possible by using a modular design where each module communicates with each other via the D-Bus communication framework. The Mer cellular Telepathy implementation consists of Telepathy-glib, Telepathy-qt, Telepathy-mission-control and Telepathy-ring. The descriptions and findings presented in this section, regarding these components, are based on the analysis of the software repositories of each corresponding component (Mer-core 2015g, date of retrieval 5.11.2015; Mer-core 2015h, date of retrieval 5.11.2015; Mer-core 2015j, date of retrieval 5.11.2015; Mer-core 2015k, date of retrieval 5.11.2015).

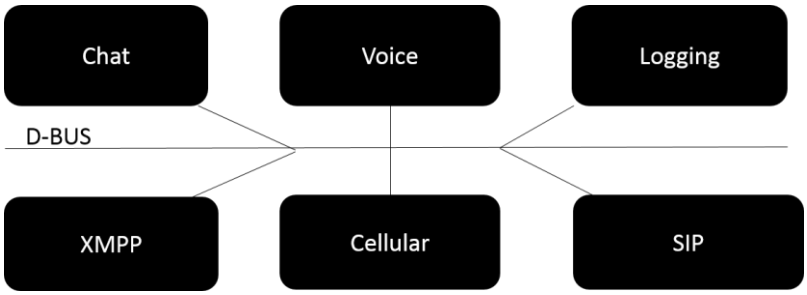

FIGURE 5. Telepathy Architecture Overview (Freedesktop.org 2014, date of retrieval 22.11.2015)

Each supported protocol and each client for each protocol is implemented as their own processes. This makes Telepathy a modular communications service provider with a unified application protocol interface. Telepathy consists of several modules: Connection managers, account managers, channel dispatchers and Telepathy clients. (Madeley 2015, date of retrieval 5.11.2015.)

The connection managers implement the Telepathy support for the desired protocol and provide an interface for clients. The account manager stores the Telepathy accounts and their parameters. The account manager establishes the connection to account based on the account parameters or if requested by using the associated connection manager. The channel dispatcher transmits the relevant type channels on the connections created by the account manager. (Madeley 2015, date of retrieval 5.11.2015.) The channel types supported by Telepathy-qt (Mer-core 2015g, date of retrieval 5.11.2015) and Telepathy-glib (Mer-core 2015h, date of retrieval 5.11.2015) according to the “all.xml” file in the “spec” folder found from the Git repositories of both components, are “Call”, “Contact List”, “Contact Search”, “DBus Tube”, “File Transfer”, “Room List”, “Server Authentication”, “Server TLS Connection”, “Stream_Tube”, “Streamed Media”, “Text” and “Tubes”. The Mer Telepathy-glib repository is basically just a Git submodule, meaning that it contains some additional patches, but for the actual source code the repository contains a link to another repository. This repository is at anongit.freedesktop.org/git/ telepathy/telepathy-glib.git/ where the source is in a binary form and can be retrieved and transformed in a readable form by using relevant Git commands.

Telepathy itself is just a set of D-Bus API specifications. Hence, there is no single Telepathy implementation, everybody must implement their own one based on the specification or use one of the existing high-level language binding implementations, such as Telepathy-glib or Telepathy-qt. The downside of this approach is that there is no guarantee of the quality of the implementation, the restrictions that the implementation brings or how well it follows the Telepathy specification. For example, by analysing the interaction between Telepathy-glib, Telepathy-mission-control (Mer-core 2015j, date of retrieval 5.11.2015) and Telepathy-ring (Mer-core 2015k, date of retrieval 5.11.2015), it can be concluded that Telepathy-glib closes the mainloop. The mainloop is the central control flow construct that waits and dispatches events in a program. Telepathy-glib stops the connection manager effectively whenever no connection exists. Therefore, for the connection manager to remain active, it needs to remain in a connected state. In the case of Telepathy-ring, this is cumbersome because a modem connection through oFono may come and go, especially in the case where USB modems are used. Thus, the Telepathy-ring must always remain in a connected state even when no modem connection exists (Mer-core 2015i, date of retrieval 5.11.2015).

Telepathy-glib is a library for the Telepathy components using GLib (a utility library for software written in C) (The GNOME Project 2014, Date of retrieval 21.11.2015). It provides high-level Telepathy GLib bindings for clients and service providers implemented using the C-language.

Telepathy-qt is a library for Telepathy components using the Qt framework. It provides Telepathy Qt bindings for clients and service providers implemented with Qt C++.

Telepathy-mission-control handles accounts. When it starts, it loads or creates accounts and each account will have a protocol manager associated. The account parameters are stored by Telepathy-mission-control. Using the parameters it brings the account online by communicating with the associated protocol manager, for example with Telepathy-ring. Telepathy-mission-control acts as a channel dispatcher, which dispatches incoming and outgoing communication channels to the relevant applications. Telepathy-mission-control uses Telepathy-glib to provide Telepathy bindings.

Telepathy-ring is a Telepathy cellular protocol manager handling an SMS and a cellular voice call. Telepathy-ring provides support for a call (for making cellular voice calls), an SMS (for sending, receiving, and manipulating spooled SMS messages) and SIM (Subscriber Identity Module) services (for accessing some SIM information).

Telepathy-ring consists of three main parts: The Telepathy-mission-control plugin, the modem, meaning the oFono interface, and the connection manager itself. Telepathy-ring uses Telepathy-glib to provide Telepathy bindings. Telepathy-ring's Telepathy-mission-control plugin sets the relevant properties and provides the relevant functionality for the Telepathy-mission-control to act as a Telepathy-ring account manager. For example, Telepathy-ring's Telepathy-mission-control plugin sets the "ConnectAutomatically" property as "true" and because of that Telepathy-mission-control will try to connect, i.e. to start Telepathy-ring whenever the "ConnectionStatus" property is disconnected. As stated before, since Telepathy-ring uses Telepathy-glib it will be effectively closed, i.e. its mainloop will exit if no connections exists. The "README" file that can found from the Telepathy-ring repository states that by setting the environment variable "RING_PERSIST", the Telepathy-ring process would keep on running, even if no connection would be active. By analysing the code, only one place can be found where the variable in question is used in Telepathy-ring and that is in the "ring_debug_set_flags_from_env" function in ring-debug.c file of the project. The "RING_PERSIST" variable is used to set the Boolean variable for the "tp_debug_set_persistent" function call. The "tp_debug_set_persistent" is implemented in the Telepathy-glib project debug.c file, which indicates that the "RING_PERSIST" environment variable is used for debug purposes. This assumption is proved by the following code comments of the function "tp_debug_set_persistent": "Used to enable persistent operation of the connection manager process for debugging purposes.".

The implementation of Telepathy-ring used during the writing of this work (commit: edbcace1) (Mer- core 2015l, date of retrieval 5.11.2015), based on the build log, uses deprecated functions of Telepathy-glib, Telepathy-mission-control and GLib. Telepathy-ring is also based on an older Telepathy specification using

“Streamed Media” as a channel type interface instead of “Call” as a channel type interface which supersedes the “Streamed Media” (Freedesktop.org 2015b, date of retrieval 10.11.2015; Freedesktop.org 2015c, date of retrieval 10.11.2015). Telepathy-ring has not been updated to use the “Call” channel type interface and this of course forces the Telepathy clients to implement the “Streamed Media” channel support to have the cellular support available. The Telepathy account name is hardcoded in the Telepathy-ring’s Telepathy-mission-control plugin file mcp-account-manager-ring.c in the “mcp_account_manager_ring_init” function. Although the Telepathy-ring has some support for multiple modems and/or multiple accounts, it effectively does not provide support for multiple modems due to this. The Telepathy-ring itself seems to work with multiple modems in a sense that if one is removed, the account state does not change. However, it is unclear if the cellular support still works after removing a modem. Assuming that there are several modems, it is unclear how the Telepathy-ring chooses which modem it will use. For example, in the dial case, will it use the first connected modem, the last connected modem, the last modem with activity or will it decide by some other means?

A fork of Mer Telepathy-ring providing a support for multiple modems/accounts is available (Poutiainen 2015, date of retrieval 5.11.2015). Since this implementation changes the way the accounts are named, it can affect the telepathy clients using Telepathy-ring. For example, Commhistory-daemon defines a hardcoded account path which, if used, will quite likely be incorrect (Mer-core 2015m, date of retrieval 5.11.2015).

5.3 Mer cellular voice call and message handling

The more traditional part of the Mer cellular telephony architecture, meaning a cellular voice call and a message handling, consists of four components: Voicecall, Commhistory-daemon, Libcommhistory, and Mms-engine. They interact with other parts of the system, such as Libqofono and Telepathy to enable the implementation of applications for receiving and sending messages (SMS and MMS), as well as receiving and making voice calls using the cellular network. The descriptions and findings presented in this section are based on the analysis of the software repositories of each corresponding component (Mer-core 2015n,

date of retrieval 5.11.2015; Mer-core 2015o, date of retrieval 5.11.2015; Mer-core 2015p, date of retrieval 5.11.2015; Mer-core 2015q, date of retrieval 5.11.2015).

Voicecall (Mer-core 2015n, date of retrieval 5.11.2015) provides abstraction for voice call related features such as making and receiving a voice call and putting the call on hold. It is written with Qt C++ and therefore uses Telepathy-qt to provide Telepathy bindings. Voicecall consists of a voicecall-manager daemon, a libvoicecall and plugin libraries, such as libvoicecall-telepathy-plugin and libvoicecall-ngf-plugin. Logically, Voicecall consists of the manager for loading protocol plugins and monitoring events, a library for implementing the corresponding UI and the protocol plugins. Currently, Voicecall provides support for a cellular voice call either by using Telepathy as a provider or oFono directly. VoIP (Voice over IP) is supported through Telepathy. It is not clear if the system would work if Voicecall used oFono as a provider while, for example, a voice call audio control and event history logging would still use Telepathy. One option could be that all related services would use oFono directly instead of Telepathy. Then the challenge could be to ensure that there would be no timing issues, e.g. with an alerting tone when creating a voice call. Either way to verify if Voicecall could use oFono directly would require an extensive testing and a further analysis.

Commhistory-daemon (Mer-core 2015o, date of retrieval 5.11.2015) is a daemon for logging the communications history to a database. It listens oFono, MMSEngine and Telepathy to send the logging data of instant messaging, SMS, MMS and voice call. It is written with Qt C++ and therefore uses Telepathy-qt to provide Telepathy bindings and Libqofono to access the oFono interface.

Libqofono is used by the Commhistory daemon for vCard (a file format for electronic business cards) and vCalendar (a file format for electronic calendar data) support for sending and receiving business cards and calendar events. The SMS messages and voice call events are received via Telepathy-qt “Text Channel” and “Streamed Media Channel”. MMS events are handled using MMSEngine API “org.nemomobile.MmsEngine”. All the interfaces use the D-Bus as an IPC mechanism. (Mer-core 2015o, date of retrieval 5.11.2015.)

In the case of receiving an SMS message, the Commhistory daemon receives the message through Telepathy, records it to the database using libcommhistory and sends the D-Bus signals about the changes in the database so that the UI can act in the desired way. It also sends a signal to MCE (Mode Control Entity) to put the display on. When receiving an incoming call signal, the role of the Commhistory is more limited: It only records the information to the database. Receiving an MMS message is similar to receiving an SMS message.

Libcommhistory (Mer-core 2015p, date of retrieval 5.11.2015) is a library for accessing (both read and write) the database, and to synchronize changes to that database between the processes using D-Bus signals. For example, the UI can use Libcommhistory to read the messages from the database. The Commhistory daemon uses it to write the messages to database. It also offers a library for QML applications to use it.

Mms-engine (Mer-core 2015q, date of retrieval 5.11.2015) provides a daemon for handling an MMS. It provides the D-Bus interface (org.nemomobile.MmsHandler) for sending and receiving messages as well as for following the state of sent and received messages. For receiving, it uses the Mer specific oFono Push Forwarder plugin to receive WAP (Wireless Application Protocol) Push messages. These messages come in an SMS format containing the URI/URL (Uniform Resource Identifier/ Uniform Resource Locator) from where to retrieve the data. For data connection Mms-engine uses an MMS specific data context provided by the oFono Connection Manager D-Bus interface.

5.4 Mer Contacts handling

The Mer Contacts handling consists of four components: Nemo-qml-plugin-contacts, Libcontacts, QtContacts-Sqlite and Contactsd. The descriptions and findings presented in this section are based on the analysis of the software repositories of each corresponding component (Mer-core 2015r, date of retrieval 6.11.2015; Mer-core 2015s, date of retrieval 6.11.2015; Mer-core 2015t, date of retrieval 6.11.2015; Mer-core 2015u, date of retrieval 6.11.2015).

Nemo-qml-plugin-contacts (Mer-core 2015r, date of retrieval 6.11.2015) and Libcontacts (Mer-core 2015s, date of retrieval 6.11.2015) form the contacts handling support for the UI. Nemo-qml-plugin-contacts provides a dynamically loadable, custom Mer QML extension plugin for QML contacts applications. Libcontacts uses nemo-qml-plugin-contacts to provide in-memory caches, indexes, and other functionality for those QML applications.

Qtcontacts-sqlite (Mer-core 2015t, date of retrieval 6.11.2015) is a plugin for the Qt Contacts API, providing access to the SQLite database. Contactsd (Mer-core 2015u, date of retrieval 6.11.2015) listens XMPP (Extensible Messaging and Presence Protocol) providers via Telepathy for contact list (XMPP term “roster”) changes as well as changes of the presence of the contact. Once Contactsd detects a contacts list change or a presence change, it forwards those changes into the QtContacts-Sqlite's temporary or transient shared memory table. Contactsd can be thought of as being a synchronisation adaptor for instant messaging services.

From cellular telephony point of view the purpose of the Contactsd is to fetch the contacts stored on the SIM card and write them to the local memory. Contactsd uses oFono to fetch the SIM contacts. Those contacts are then injected into the QtContacts-Sqlite database, as "sim" synctarget contacts, and from there exposed to the UI or applications through the Nemo-qml-plugin-contacts and Libcontacts.

5.5 Mer cellular modem state handling

In a mobile software platform it is necessary to handle the changes indicated by the modem and the state changes of the modem. This can be done independently by each application, by following the state changes forwarded by the modem adaptation layer. In the Mer platform these indications come through oFono. The previously mentioned method can create unwanted dependencies, so other means are provided. In the case of Mer platform there are two other options which provide information about system states, including the modem and modem related states.

5.5.1 StateFS

StateFS consists of four packages: Statefs, Statefs-providers, Statefs-qt and Statefs-loader-qt. The descriptions and findings presented in this subsection are based on the analysis of the software repositories of each corresponding component (Mer-core 2015v, date of retrieval 6.11.2015; Mer-core 2015x, date of retrieval 6.11.2015; Mer-core 2015y, date of retrieval 6.11.2015; Mer-core 2015z, date of retrieval 6.11.2015). StateFS provides a framework for exposing the system states such as SIM, modem and cellular network states. The states are provided as properties wrapped into namespaces and written to files by plugins that are shared libraries.

Statefs (Mer-core 2015v, date of retrieval 6.11.2015) provides the core of the StateFS framework. It provides the provider, consumer and loader interfaces. It implements the StateFS daemon and provides the basic structure, which all the providers and loaders follow.

Statefs-providers (Mer-core 2015x, date of retrieval 6.11.2015) provides a set of StateFS providers as plugins. From the cellular telephony point of view the most important is the oFono provider that provides handling for the SIM, modem and cellular network states.

Statefs-loader-qt (Mer-core 2015y, date of retrieval 6.11.2015) starts the main Qt event loop in a dedicated thread so that a provider needing it can be loaded. The "README.org" file in the Statefs-loader-qt repository states that:

> *"Some Qt components/code need main Qt event loop to be running and it should be started in the thread where some (read any) Qt code was used, it saves current thread as the main thread. So, this loader create QCoreApplication and starts its event loop in a dedicated thread. When event loop is started requested provider can be loaded safely."*

Statefs-qt (Mer-core 2015z, date of retrieval 6.11.2015) library offers the StateFS client interface to Qt-based applications and libraries. The use of this library is optional because the system state information provided by StateFS can be read directly from the file. The StateFS tree structure and location in the filesystem

hierarchy is not fixed and can therefore change. Thus, this library increases the compatibility of the code with future changes of StateFS.

5.5.2 MCE

MCE consists of two components: the Mce-dev and MCE. The descriptions and findings presented in this subsection are based on the analysis of the software repositories of each corresponding component (Mer-core 2015aa, date of retrieval 6.11.2015; Mer-core 2015ab, date of retrieval 6.11.2015). Mce-dev (Mer-core 2015aa, date of retrieval 6.11.2015) is a separate package which provides the header files that define the D-Bus interface of the MCE. MCE (Mer-core 2015ab, date of retrieval 6.11.2015) is a daemon that provides a mode control functionality. MCE offers a single point for applications and other service providers to receive and send an input from and to multiple sources by using the D-BUS interface. MCE makes it possible to enable screen blanking and locking during the call by following oFono call states and broadcasting them, while offering an interface for controlling the screen. It also follows if the screen is lit or not and signals whenever it changes, therefore making it possible to request the modem to enter the power save mode, when no user interaction is expected.

5.6 Mer cellular modem abstraction

In a modern mobile software platform there is usually a cellular modem abstraction layer that hides modem specific features and dependencies from other software layers. It provides a standard interface and adapts a modem specific behaviour to that interface. The cellular modem abstraction is a central element in the telephony stack in a sense that the rest of the stack is designed based on it. It would be hard to remove the cellular abstraction layer and replace it with another without major changes to the rest of the stack. In the Mer platform this abstraction consists of two modules: Libqofono and oFono. The descriptions and findings presented in this section are based on the analysis of the software repositories of each corresponding component (Mer-core 2015ac, date of retrieval 6.11.2015; Mer-core 2015a, date of retrieval 10.11.2015).

The Libqofono (Mer-core 2015ac, date of retrieval 6.11.2015) package provides a library for accessing the oFono (Mer-core 2015a, date of retrieval 10.11.2015) daemon. The Libqofono package also consists of a plugin to allow QML applications to access oFono using the Qt Quick module. Libqofono is basically just a Qt wrapper of oFono. Although QML UI applications can use Libqofono to communicate with oFono, this can be problematic in some cases. This is because of two reasons. The first reason is that typically the QML UI application is closed when the end-user exits the application, hence the application stops receiving indications and replies to requests it has made. The second reason is the behaviour of the oFono interface. For example, if the application makes a request to start the network scan and is restarted while the scan is still in progress, it cannot in some cases know that the scan is still in progress. Calling the oFono "NetworkRegistration" D-Bus interface methods "Scan" or "Register", or the "NetworkOperator" interface method "Register", while the "Scan" is still in progress, will return with an error "InProgress". So although it is the "Scan" that is still in progress, it cannot be distinguished from the "NetworkRegistration" interface method call "Register".

OFono is a daemon that essentially provides a modem adaptation with a modem independent D-Bus interface for the telephony application development. The software license used is GPL version 2 but since oFono provides a D-Bus interface for the application development, those applications can be of any license. According to The oFono Project web page, the aim of the oFono software project is to offer a 3GPP GSM/UMTS (Global System for Mobile communications / Universal Mobile Telecommunications System) standard compliant software framework (Intel corporation 2015b, date of retrieval 6.11.2015). Based on the documentation and the source code available in the oFono repository, the oFono explicitly does not support all the GSM features, most notably the SIM phonebook writing. This does not prevent anybody from forking the oFono project and then adding the support not provided by the upstream project. The compliancy includes such items as the decoding an SMS message and the de-fragmentation of an SMS message. The Mer project has its own fork of oFono that has added support to that of upstream project. There is added support for such as Network Identity and Time Zone (NITZ) indication support and some added SMS handling

to enable MMS support in the Mer platform as well as some other platform specific fixes.

The structure of oFono can be explained in several ways. The oFono documentation talks about four main components, the core daemon, the atoms, the drivers, and the plugins (Mer-core 2015ad, date of retrieval 10.11.2015).

The oFono core provides the internal interface that the plugins and the drivers must implement, thus enabling the common D-BUS API to exist. It also loads the plugins and drivers. The core manages each connected device independently providing support for multiple modems and multiple SIM cards to be present at the same time. Besides these core functionalities, it also provides common utility functions for reading and writing the SIM card and interpreting the contents of the SIM low-level Element File contents. There are also utility functions for decoding, encoding and fragmentation of binary SMS protocol data units as well as functions for decoding, duplicate detection and pagination of cell broadcasts and character set conversion. The core also provides functions for detection of and communication between oFono atoms.

OFono atoms are an integral part of the core. The oFono atom is more a concept rather than an actual module meaning that there is no separate libraries or folders or files for atoms. There is an atom for each main telephony/modem feature such as SMS, CBS (Cell Broadcast Service) and SIM but they can represent something else, too. It is the atoms that provide the oFono D-Bus interface. To simplify, each oFono driver is linked to an atom and an atom is attached to a modem. As mentioned earlier, the atoms can communicate between each other. Therefore, they can detect the presence of each other and use the information provided by other atoms for their own needs. For example, the GPRS (General Packet Radio Service) atom requests the Network Registration atom to inform changes in the network registration status.

The oFono drivers provide a means to integrate multiple device technologies. Drivers handle the adaptation of a specific protocol. They translate generic oFono requests, such as dial request, to a protocol specific request and forward the request then to the correct device. For example, in an AT command based

modem a voice call dial request to a number 1234567 is translated to a command "ATD1234567;". This way the oFono can support multiple types of devices based on a variety of communication protocols. The upstream oFono has support for multiple modems using the AT command set, and it includes an ISI protocol based driver and a Qualcomm QMI modem driver.

The oFono plugins make it possible for developers to tailor the oFono for their purposes. The main purpose of the plugins is to provide a means to recognize the available modem or modems and enable the use of them by loading the correct communication protocol, atoms and drivers. Besides these, a plugin can be made to either provide optional interface support, such as network time support, or simply extend the existing functionality, such as provisioning the GPRS context.

Another way to describe the architecture or structure of oFono is to divide it into five areas as shown in Figure 6: Core, plugins, a common modem and a protocol independent D-Bus interface, a modem or vendor specific drivers, and a protocol specific communication layer.

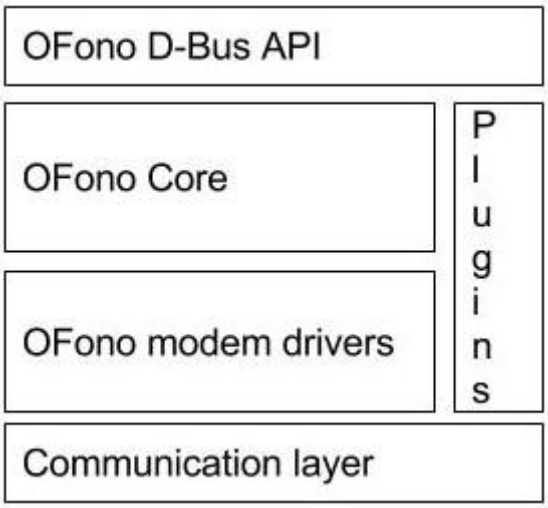

FIGURE 6. OFono overview

In this division the core is considered the same as the previously explained one but the concept of atoms is considered as being part of it. The D-Bus interface is implemented within the core or into a plugin but is considered as a separate entity. The drivers and the plugins are also as explained earlier but the biggest

difference to the oFono documentation is to consider the communication layer and the drivers as separate entities.

The communication layer takes care of the message scheduling and the queue mechanism and provides a protocol specific communication channel. The idea behind this division is that there can be several modems with their own command set needing their own driver implementations but which can use the same communication layer implementation.

The Mer project has modified and added features to their fork of oFono. These include an enhanced GPRS context provisioning, a signalling of the changed operator list, an interface for sending arrays of raw bytes to the modem, a signalling of received SMS message status reports and support for forwarding WAP push messages to Mms-engine as well as support for a ringback tone and network time (NITZ). However, the main addition is the Android RIL support. This means that if the system can utilize the Android binaries, the oFono can use the Android RIL to communicate with the modem. The Mer oFono RIL support is initially based on the one provided by Canonical Ltd. (Mer-core 2015ae, date of retrieval 10.11.2015) but has since deviated from the Canonical version. The RIL support consists of drivers providing RIL interface support; a communication layer enabling the socket communication with the RIL daemon; and a set of plugins for recognizing the existence of the RIL daemon, loading the related drivers and starting the communication using the communication layer.

5.7 Other cellular modem dependant modules

The descriptions and findings presented in this section and subsection are based on the analysis of the software repositories of each corresponding component (Mer-core 2015af, date of retrieval 6.11.2015; Mer-core 2015ag, date of retrieval 6.11.2015; Mer-core 2015ah, date of retrieval 6.11.2015; Mer-core 2015ai, date of retrieval 6.11.2015; Mer-core 2015aj, date of retrieval 6.11.2015; Nemomobile 2015b, date of retrieval 6.11.2015). In a modern mobile software platform there are other technologies that are used together with the cellular telephony. These technologies can either use the telephony technology to extend their functionality

or they are used to extend the functionality of the telephony technology. Here are those found in the Mer context.

5.7.1 Cellular positioning

Geoclue (Mer-core 2015af, date of retrieval 6.11.2015) provides location services offering a D-Bus interface for location aware applications. Geoclue supports multiple technologies and methods for finding the current location. The technology support is implemented as a provider and new providers can be added. Geoclue has a GSM cell based position provider (gsmloc) that uses oFono to fetch MCC, MNC, LAC and CID (Mobile Country Code, Mobile Network Code, Location Area Code and Cell Id) which it then matches with the data provided by the web service (http://www.opencellid.org/) and a lookup table. The web service is used to fetch the latitude and longitude of the current cell. The lookup table is used to map MCC with ISO (International Organisation for Standardization) country code, e.g. MCC 244 equals ISO 3166-1 alpha-2 two letter code FI of Finland.

5.7.2 Cellular Network Time

Timed (Mer-core 2015ag, date of retrieval 6.11.2015) is a daemon for managing the device system time, time zone, time events and related settings used, for example, by clock applications. Timed provides a D-Bus interface that can be used directly or via C++ Qt-bindings. OFono is used by Timed to query the cellular network time. If NITZ is supported by the network this information may be received by oFono and thus by Timed in the following cases: When registering on the network, the device geographically relocates to a different local time zone, the network changes its local time zone, e.g. between the summer and winter time, the network changes its identity, or at any time during a signalling connection with a mobile station (The 3rd Generation Partnership Project (3GPP) 2013, date of retrieval 10.11.2015).

5.7.3 Cellular USB Tethering

Usb_moded (Mer-core 2015ah, date of retrieval 6.11.2015) is a daemon that activates a USB (Universal Serial Bus) profile based on the USB cable connection

status that it tracks. It uses a D-Bus system bus for all the system wide communications. Among other functionality the Usb_moded can set up a USB tethering, meaning it can share the connection of the device for another device. A typical case of this is a laptop using a mobile phone for connecting to the Internet. When a cellular network is used to access the Internet, the oFono is used to check if the device is roaming, and if so, if roaming is allowed. Roaming means using the cellular connection outside the home subscription network.

5.7.4 Seamless Software Update

Over-The-Air software updating or upgrading in Mer is called SSU (Seamless Software Update) and a part of this concept is ssu (Nemomobile 2015b, date of retrieval 6.11.2015). The ssu is written in lowercase to distinguish the software from the concept of SSU which is written in uppercase. The concept of SSU is not covered here. The oFono dependency is used by ssu to check the device International Mobile Equipment Identity (IMEI) to be used, if available, as a unique identification number of the device. The software package was not yet merged into the Mer as part of Nemo/Mer merge.

5.7.5 Mer Cellular Bluetooth handling

Bluetooth is a wireless technology specification developed, published, and promoted by the Bluetooth Special Interest Group (Bluetooth SIG). The Bluetooth is used for exchanging data over short distances (Bluetooth SIG, Inc. 2015, date of retrieval 10.11.2015). The Bluetooth wireless headset can be used to receive and transmit audio during the voice call and for a call control. The call control can consist of accepting, rejecting or ending the call as well as making a call to the last called number. It is also possible to make a voice call in the form of voice dialling, which means using a voice recognition for a call creation. Mer Cellular Bluetooth handling consists of two parts: Bluez and libbluez.

Libbluez-qt (Mer-core 2015ai, date of retrieval 6.11.2015) provides a library containing Qt bindings for accessing a Bluetooth functionality and a plugin for QML applications. Bluez (Mer-core 2015aj, date of retrieval 6.11.2015) is a Bluetooth stack consisting of several modules providing a support for the core

Bluetooth layers and protocols for Linux kernel based operating systems (Bluez project 2015, date of retrieval 10.11.2016). The Mer project has its own fork of Bluez from the upstream projects Git repository at kernel.org. The Bluez offers a high-level D-Bus API for the networking applications of any license. Bluez itself is a free software released under the terms of the GPL version 2. The Bluez uses the oFono as a cellular telephony support provider. For example, when a Bluetooth device is used for initiating a voice call, Bluez sends a message to the oFono interface to make a dialling request and during the call it keeps receiving the call states. If a Bluetooth device is connected while an incoming call is received, Bluez will receive an indication of this and if the Bluetooth device is used to answer the call, Bluez tells the oFono to answer.

6 CONCLUSION

The software components covered in this work provide the cellular telephony functionality needed in a mobile phone. The modular nature of the Mer cellular telephony architecture makes it possible to replace parts of it with another part. Most of the components are designed to be extendable, i.e. adding of a new feature is made easy. Some parts of the architecture are harder to comprehend than others, most notably the Telepathy framework is hard to comprehend. This is partly due to the Telepathy specific concepts that need to be understood and partly due to the two different language bindings that are used. The Telepathy cellular support implementation is based on an older specification, whereas the Voicecall component also supports the superseding specification. This way of implementing a support only to a part of the system seems to be quite typical in the Mer development model. It can be confusing but it also provides a way of implementing a support for a new features overtime in parts, without a need for a centralized planning. Also, some components contain only a partial implementation of a feature or functionality.

There are no clear problems with the Mer cellular telephony system but the role of the Telepathy could be considered. One option is to try to get rid of the Telepathy dependency, but as previously explained, there are questions that need to be answered before deciding to do this. The removal of Telepathy could lead to having less IPC communication and it would simplify the architecture. Another option is to make a new implementation of Telepathy-ring by using Telepathy-qt, thus removing the Telepathy-glib dependency. This would, however, mean that the functionality provided by the Telepathy-mission-control would have to be implemented into the new Telepathy-ring as the Telepathy-mission-control depends on the Telepathy-glib, too. The least that should be done is to implement the support of the latest Telepathy specification to the Telepathy-ring.

It is also recommendable to always consider if there is a need to have a daemon to control the communication between the UI application and oFono. The daemon

can keep track of the requests and their responses even when the application is closed.

The support for having multiple active modems and SIM cards in the device at the same time is partially done. This work should be finished. This would enable the Mer to support new hardware configurations that cannot be supported at the moment.

To the best of the knowledge available no other work describing the Mer cellular telephony exists currently. This work can be used as a base when building a new mobile system requiring a cellular telephony support to help to decide if the cellular telephony support of Mer or parts of it are relevant for that.

REFERENCES

The 3rd Generation Partnership Project (3GPP). 2013. 3GPP specification 22.042. Release 12. Date of retrieval 10.11.2015 http://www.3gpp.org/DynaReport/22042.htm.

The 3rd Generation Partnership Project (3GPP). 2015. 3GPP specification 27.007 Release 13. Date of retrieval 20.10.2015 http://www.3gpp.org/DynaReport/27007.htm.

The Android Open Source Project. 2015a. Android Interfaces and Architecture. Date of retrieval 21.10.2015 https://source.android.com/devices.

The Android Open Source Project. 2015b. ART and Dalvik. Date of retrieval 19.11.2015 https://source.android.com/devices/tech/dalvik.

The Apache Software Foundation. 2015. Apache License Version 2.0. Date of retrieval 4.11.2015 http://www.apache.org/licenses/LICENSE-2.0.

Bluez project. 2015. Bluez: About. Date of retrieval 10.11.2016 http://www.bluez.org/about/.

Bluetooth SIG, Inc. 2015. Bluetooth Technology Basics. Date of retrieval 10.11.2015 http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics.

Digia Finland Ltd. 2015a. About Qt. Date of retrieval 10.11.2015 https://wiki.qt.io/About_Qt.

Digia Finland Ltd. 2015b. Qt Documentation: All Modules. Date of retrieval 10.11.2015
https://doc.qt.io/qt-5/qtmodules.html.

Digia Finland Ltd. 2015c. Qt Documentation: How to Create Qt Plugins. Date of retrieval 10.11.2015
http://doc.qt.io/qt-5/plugins-howto.html.

Edge, J. 2014. Jolla and Mer. Date of retrieval 10.11.2015
https://lwn.net/Articles/597560/.

Freedesktop.org. 2013a. Introduction to D-Bus. Date of retrieval 10.11.2015
http://www.freedesktop.org/wiki/IntroductionToDBus/.

Freedesktop.org. 2014. Telepathy. Date of retrieval 22.11.2015
http://telepathy.freedesktop.org/wiki/.

Freedesktop.org. 2015a. What is D-Bus?. Date of retrieval 10.11.2015
http://www.freedesktop.org/wiki/Software/dbus/.

Freedesktop.org. 2015b. Interface Channel.Type.StreamedMedia. Date of retrieval 10.11.2015
http://telepathy.freedesktop.org/spec/Channel_Type_Streamed_Media.html.

Freedesktop.org. 2015c. Interface Channel.Type.Call1. Date of retrieval 10.11.2015
http://telepathy.freedesktop.org/spec/Channel_Type_Call.html.

Free Software Foundation. 2007a. GNU General Public License Version 3. Date of retrieval 4.11.2015
https://www.gnu.org/licenses/gpl.txt.

Free Software Foundation. 2007b. GNU Lesser General Public License Version 3. Date of retrieval 4.11.2015

https://www.gnu.org/licenses/lgpl.txt.

Free Software Foundation. 2015a. About the FSF. Date of retrieval 3.11.2015 https://www.fsf.org/about/.

Free Software Foundation. 2015b. What is free software?. Date of retrieval 3.11.2015
https://www.gnu.org/philosophy/free-sw.html.

Free Software Foundation. 2015c. Various Licenses and Comments about Them. Date of retrieval 4.11.2015
https://www.gnu.org/licenses/license-list.html.

Free Software Foundation. 2015d. What is Copyleft?. Date of retrieval 4.11.2015
https://www.gnu.org/copyleft/copyleft.html.

Free Software Foundation. 2015e. Frequently Asked Questions about the GNU Licenses. Date of retrieval 20.10.2015
http://www.gnu.org/licenses/gpl-faq.html#MereAggregation.

Free Software Foundation. 2015f. The GNU C Library (glibc). Date of retrieval 19.20.2015
https://www.gnu.org/software/libc/.

Gartner Inc. 2015. Gartner Says Worldwide Smartphone Sales Recorded Slowest Growth Rate Since 2013. Date of retrieval 21.10.2015
http://www.gartner.com/newsroom/id/3115517.

The GNOME Project. 2014. GLib Refence Manual. Date of retrieval 21.11.2015 https://developer.gnome.org/glib/.

Google Git. 2015a. android/platform/hardware/ril/master. Date of retrieval 20.10.2015

https://android.googlesource.com/platform/hardware/ril/+/master.

Google Git. 2015b. android/platform/bionic/master. Date of retrieval 9.11.2015 https://android.googlesource.com/platform/bionic/+/master.

Google Git. 2015c. /android/platform/bionic/jb-release/libc/docs/OVERVIEW.TXT. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/bionic/+/jb-release/libc/docs/OVERVIEW.TXT

Google Git. 2015d. android/platform/hardware/ril/master/rild. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/hardware/ril/+/master/rild.

Google Git. 2015e. android/platform/hardware/ril/master/rild/Android.mk. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/hardware/ril/+/master/rild/Android.mk.

Google Git. 2015f. android/platform/hardware/ril/master/libril. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/hardware/ril/+/master/libril.

Google Git. 2015g. android/platform/hardware/ril/master/libril/Android.mk. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/hardware/ril/+/master/libril/Android.mk.

Google Git. 2015h. android/platform/frameworks/native/master/libs/binder/Android.mk. Date of retrieval 20.10.2015 https://android.googlesource.com/platform/frameworks/native/+/master/libs/binder/Android.mk.

Google Git. 2015i. android/platform/hardware/ril/master/reference-ril. Date of retrieval 9.11.2015 https://android.googlesource.com/platform/hardware/ril/+/master/reference-ril.

Google Git. 2015j. android/platform/hardware/ril/master/reference-ril. Date of retrieval 9.11.2015 https://android.googlesource.com/platform/hardware/ril/+/master/reference-ril/Android.mk.

Google Git. 2015k. android/platform/hardware/ril/master/include/telephony/ril.h. Date of retrieval 20.11.2015 https://android.googlesource.com/platform/hardware/ril/+/master/include/telephony/ril.h

Google Inc. 2015a. Android. Date of retrieval 20.10.2015 http://www.android.com/.

Google Inc. 2015b. Parcel. Date of retrieval 20.10.2015 http://developer.android.com/reference/android/os/Parcel.html.

Greaves, D. 2015. [mer-general] Nemo/Mer merge update. Date of retrieval 10.11.2015 https://www.mail-archive.com/mer-eneral@lists.merproject.org/msg01557.html.

Intel Corporation. 2015a. Connman. Date of retrieval 5.11.2015 https://01.org/connman.

Intel Corporation. 2015b. OFono. Date of retrieval 6.11.2015 https://01.org/ofono.

International Data Corporation. 2015. Smartphone OS Market Share, 2015 Q2. Date of retrieval 21.10.2015 http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

ISO/IEC 9899. 2003. Rationale for International Standard Programming languages C. Revision 5.10. Date of retrieval 9.11.2015 http://www.open-std.org/JTC1/SC22/WG14/www/docs/C99RationaleV5.10.pdf.

Jolla Ltd. 2015a. Sailfish.org. Date of retrieval 10.11.2015 https://sailfishos.org/.

Jolla Ltd. 2015b. All about us and the OS. Date of retrieval 10.11.2015 https://sailfishos.org/about/.

Jolla Ltd. 2015c. Get started. Date of retrieval 10.11.2015 https://sailfishos.org/develop/.

Jolla Ltd. 2015d. Nemo QML Plugin D-Bus. Date of retrieval 10.11.2015 https://sailfishos.org/develop/docs/nemo-qml-plugin-dbus/.

Libhybris. 2015. libhybris. Date of retrieval 20.10.2015 https://github.com/libhybris/libhybris.

Linux Programmer's Manual. 2014. libc - overview of standard C libraries on Linux. Date of retrieval 19.10.2015 http://man7.org/linux/man-pages/man7/libc.7.html.

Madeley, D. 2015. Telepathy. Date of retrieval 5.11.2015 http://www.aosabook.org/en/telepathy.html.

Mer-core. 2015a. Ofono. Date of retrieval 10.11.2015 https://git.merproject.org/mer-core/ofono.

Mer-core. 2015b. Connman. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/connman.

Mer-core. 2015c. Libconnman-qt. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/libconnman-qt.

Mer-core. 2015d. Nemo-qml-plugin-connectivity. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/nemo-qml-plugin-connectivity.

Mer-core. 2015e. Connectionagent. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/connectionagent.

Mer-core. 2015f. Provisioning-service. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/provisioning-service.

Mer-core. 2015g. Telepathy-glib. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-glib.

Mer-core. 2015h. Telepathy-qt. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-qt.

Mer-core. 2015i. [telepathy-ring] remove connection connecting timeout. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-ring/commit/1b27f8da32f2c93a7627bf38b9e04f8e882bd476.

Mer-core. 2015j. Telepathy-mission-control. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-mission-control.

Mer-core. 2015k. Telepathy-ring. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-ring.

Mer-core. 2015l. Merge branch 'mer-1391' into 'master'. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/telepathy-ring/commit/edbcace196871dedd59bafd7b6d15087e76da96c.

Mer-core. 2015m. Commhistory-daemon/src/constant.h. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/commhistory-daemon/blob/master/src/constants.h.

Mer-core. 2015n. Voicecall. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/voicecall.

Mer-core. 2015o. Commhistory-daemon. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/commhistory-daemon.

Mer-core. 2015p. Libcommhistory. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/libcommhistory.

Mer-core. 2015q. Mms-engine. Date of retrieval 5.11.2015 https://git.merproject.org/mer-core/mms-engine.

Mer-core, 2015r. Libcontacts. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/libcontacts.

Mer-core. 2015s. Nemo-qml-plugin-contacts. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/nemo-qml-plugin-contacts.

Mer-core. 2015t. Qtcontacts-sqlite. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/qtcontacts-sqlite.

Mer-core. 2015u. Contactsd. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/contactsd.

Mer-core. 2015v. Statefs. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/statefs.

Mer-core. 2015x. Statefs-providers. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/statefs-providers.

Mer-core. 2015y. Statefs-loader-qt. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/statefs-loader-qt.

Mer-core. 2015z. Statefs-qt. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/statefs-qt.

Mer-core. 2015aa. Mce-dev. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/mce-dev.

Mer-core. 2015ab. Mce. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/mce.

Mer-core. 2015ac. Libqofono. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/libqofono.

Mer-core. 2015ad. ofono/ofono/doc/ofono-paper.txt. Date of retrieval 10.11.2015 https://git.merproject.org/mer-core/ofono/blob/master/ofono/doc/ofono-paper.txt.

Mer-core. 2015ae. Squashed 'ofono/' changes from 649ee6bf..de0ccde. Date of retrieval 10.11.2015 https://git.merproject.org/mer-core/ofono/commit/eb0e3ed6674f93aa03fbce68460509ac01559fcd.

Mer-core. 2015af. Geoclue. Date of retrieval 10.11.2015 https://git.merproject.org/mer-core/geoclue.

Mer-core. 2015ag. Timed. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/timed.

Mer-core. 2015ah. Usb-moded. Date of retrieval 6.11.2015 https://git.merproject.org/mer-core/usb-moded.

Mer-core. 2015ai. Libbluez. Date of retrieval 6.11.2015

https://git.merproject.org/mer-core/libbluez-qt.

Mer-core. 2015aj. Bluez. Date of retrieval 6.11.2015
https://git.merproject.org/mer-core/bluez.

The Mer Wiki. 2012. Architecture. Date of retrieval 10.11.2015
https://wiki.merproject.org/wiki/Architecture.

The Mer Wiki. 2014. Nemo. Date of retrieval 10.11.2015
https://wiki.merproject.org/wiki/Nemo.

The Mer Wiki. 2015a. Main Page. Date of retrieval 10.11.2015
https://wiki.merproject.org/wiki/Main_Page.

The Mer Wiki. 2015b. Sailfish. Date of retrieval 10.11.2015
https://wiki.merproject.org/wiki/Sailfish.

The Mer Wiki. 2015c. Sailfish/CommunityMeetings. Date of retrieval 10.11.2015
https://wiki.merproject.org/wiki/Sailfish/CommunityMeetings.

Nemomobile. 2015a. Nemomobile bugzilla - Main page. Date of retrieval
10.11.2015 https://bugs.nemomobile.org.

Nemomobile. 2015b. Ssu. Date of retrieval 6.11.2015
https://github.com/nemomobile/ssu.

Nemomobile-packages. 2015. Ofono. Date of retrieval 10.11.2015
https://github.com/nemomobile-packages/ofono.

Open source initiative. 2015a. About the Open Source Initiative. Date of
retrieval 3.11.2015
http://opensource.org/about.

Open Source Initiative. 2015b. History of the OSI. Date of retrieval 3.11.2015

http://opensource.org/history.

Open Source Initiative. 2015c. The Open Source Definition. Date of retrieval 3.11.2015
http://opensource.org/docs/osd.

Open Source Initiative. 2015d. The BSD 2-Clause License. Date of retrieval 4.11.2015
http://opensource.org/licenses/BSD-2-Clause.

Open Source Initiative. 2015e. The BSD 3-Clause License. Date of retrieval 4.11.2015
http://opensource.org/licenses/BSD-3-Clause.

Poutiainen, J. 2015. Telepathy-ring. Date of retrieval 5.11.2015
https://github.com/jpoutiai/telepathy-ring/tree/multimodem.

Stallman, R. 2014. FLOSS and FOSS. Date of retrieval 3.11.2015
https://www.gnu.org/philosophy/floss-and-foss.html.

Stallman, R. 2015. Why Open Source misses the point of Free Software. Date of retrieval 3.11.2015
https://www.gnu.org/philosophy/open-source-misses-the-point.html.

Torvalds, L. 1994. COPYING. Date of retrieval 20.10.2015
https://www.kernel.org/pub/linux/kernel/COPYING.

Wikipedia. 2015. Mer (software distribution). Date of retrieval 10.11.2015
https://en.wikipedia.org/wiki/Mer_%28software_distribution%29.

Yaghmour, K. 2013. Embedded Android. 1st edition. Sebastopol, CA: O'Reilly Media, Inc.