GNOKII
Then and Now

Hugh Blemings
Paweł Kot

The early days—gnokii then

To their credit, Nokia was one of the first mobile phone manufacturers to make a suite of software available to interface their mobiles to a computer. Their suite, "Nokia Cellular Data Suite" (NCDS) is Windows based and provides a complete set of tools for manipulating phonebook entries, sending SMS messages etc.

The version available in 1998 also allowed the mobile to be used as an AT compatible modem. Unlike modern models, this was something the handsets were unable to do standalone. Essentially the software provided a virtual serial driver that sat between the standard Windows serial layer and the phone. The driver translated between Hayes AT style modem commands and the proprietary protocol used by the phones (the so called FBUS or MBUS protocols).

The catalyst for what became gnokii was basically that Hugh had a Nokia 3810 which he used with a laptop running dual boot Linux/Windows. Not being able to use the 3810 under Linux meant no mobile 'net access without reboot, clearly an untenable situation :)

The project initially supported the then current 3810/3110/8110 model series—another project had started for the then new 6110 series but not generated any code. After a short dialogue we combined the two efforts figuring (correctly as it happened) that much of the higher level code would be common anyway. Pavel Janik based in Brno, Czech Republic became co-author having contributed the majority of the 6110 series code. Discussions in mid October 1998 to "formalise" the project lead to `gnokii-0.0.1` being released on January 26th, 1999.

During the early stages there was some dialogue back and forth with Nokia, eventually leading to contact with the gentleman responsible for the NCDS project itself. Regrettably after a promising start the discussions stalled in June 1999 on the matter of an open source release. Nokia's concern it seemed revolved around potential liability if someone used the information they provided to defeat SIM locks and the like. Ironically this was worked out by people outside the gnokii project anyway...

A quick introduction to early FBUS protocols

By the time we reached the impasse with Nokia, there had actually been several releases of gnokii and we were well underway in understanding the protocols in use. To understand what we had to work out, a little explanation of how the FBUS protocol and GSM phones operate will assist. Note that what follows is what we now know, how we came to know this is covered below.

Fax and data calls on a GSM network actually send binary data over the air interface using the Radio Link Protocol (RLP). Some munging of the data stream is done depending on the other end of the link. If the other end is an ISDN connection the data stays in the digital domain the whole time. If the other end is an analogue modem or fax then a device at the far end of the network modulates/demodulates the RLP data into the baseband signals required by the far end's analogue modem.

What follows relates to the 3110/3810/8110 series, the 6110 series uses a completely different and rather more complex variant of FBUS.

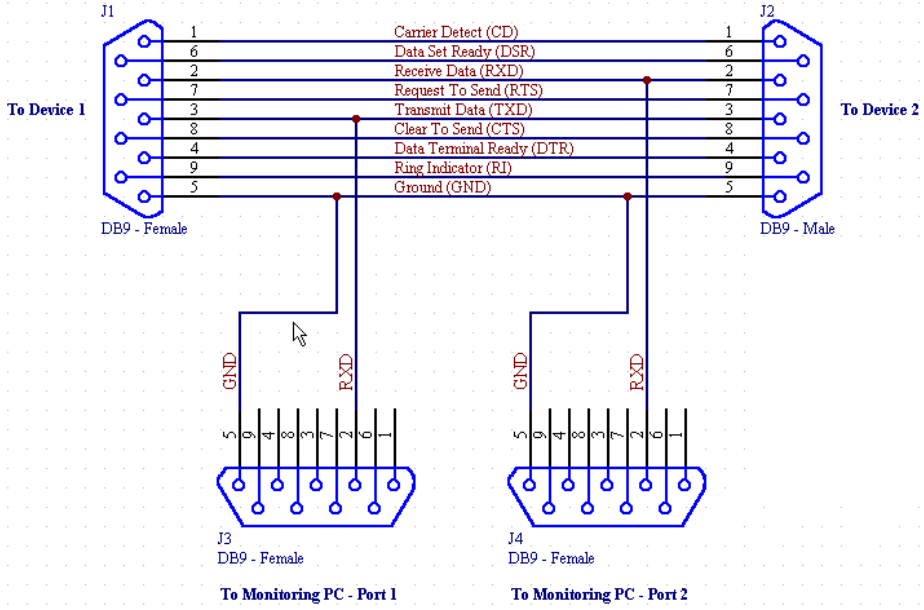
Most of the actions over FBUS followed a command/response sort of model. The phone would also send some messages asynchronously in response to events occurring on the network such as incoming calls, SMS messages arriving etc. When run in data or fax mode (we never supported the latter on the 3110 series), the phone streams raw RLP frames off the GSM network down the interface and similarly expects RLP frames to be sent to it. To get data calls going we had to write our own RLP implementation from the relevant

ETSI specifications. No small task but the Open Source Development model prevailed.

Working it out ourselves

The FBUS protocol documentation wasn't available to us so we set about working it out ourselves. Different members of the team used slightly different setups but the basic idea was to employ a second PC and a "sniffer" cable to monitor data going between the phone and the PC running NCDS. Contrary to popular belief for a number of reasons we never took the approach of disassembling the NCDS binary.

For the curious, an example serial sniffer cable is shown below



At the time we started working on gnokii there weren't any suitable open source tools for sniffing serial protocols. Some suitable C code was cobbled together for the purpose. This would output data flowing in one direction enclosed in square brackets, data in the other in braces with the hexadecimal byte and ASCII character (if valid) in between them.

```
{01 }{02 }{4aJ}{16 }{5f_}[04 ][02 ][4aJ][0e ][42B][04 ][05 ][4bK][15 ][01 ][03 ]
[02 ][5f_]{01 }{02 }{4bK}{1d }{55U}{01 }{02 }{3f?}{17 }{2b+}[04 ][02 ][3f?][0f ]
[366][04 ][1b ][41A][16 ][02 ][00 ][00 ][00 ][00 ][00 ][00 ][a7 ][01 ][01 ][00 ]
[0c ][2b+][366][311][344][311][311][399][399][300][300][300][300][00 ][f9 ]{01 }
```

The basic code had a number of limitations, most notable was that it didn't really address time alignment of the received data—thus it wasn't clear from the output from the sniffer whether the PC or phone sent data first. This was a bit of a nuisance initially but it quickly became apparent what was going on.

This tool was used to capture data corresponding to the "idle" state of the phone to PC connection. The idle "heartbeat" provided sufficient information to guess the basic protocol. Further captures were done while various actions were performed such as reading/writing phone book entries etc. A capture was done at startup to determine the initial handshaking bytes.

There was now enough information to write a simple state machine based parser. This was used to look at incoming bytes and decode them into the appropriate fields. When an unknown message was received it was dumped in the following fashion.

```
To: MT43 SN11 CS51 [02 ][04 ]

Fr: MT46 SN17 CS79 [04 ][48H][6fo][6dm][65e][0c ][2b+][366][311][322][366]
                   [322][366][322][366][300][311][366]
```

The capture above shows Hugh's old home phone number being retrieved from the phone by NCDS. The message to the phone is Message Type `0x43` with data fields `0x02` specifying SIM memory and location `0x04`. In the response—Message Type `0x46`—the `0x04` byte is the length of the name field and 0x0c is the length of the number field. Rest is pretty self evident...

The early releases

The result of this early work was the gnokii project, then lead by Pavel Janik and Hugh Blemings. gnokii was developed as a free software project, released under the GNU General Public Licence.

Initially, gnokii was just a command line test tool with support for the phones mentioned above. Over time, the test tool became more and more powerful and the support for the phones matured.

As the project progressed, other tools were created. Besides the `gnokii` command line there are three main tools: `gnokiid`, `xgnokii` and `smsd`.

Gnokiid is the virtual modem application provided for Nokia the 6110 family phones. These mobiles don't have AT command support but they are able to make data calls. Gnokiid provides a software modem emulator that passes the data to the phone using the FBUS protocol. Gnokiid creates a `/dev/gnokii` device for communication. It is only used with the 6110, 6130, 6150, 5110 and 5130 phones.

Xgnokii, as the name suggests is a X based application that provides a graphical interface to the phone. Amongst other things, xgnokii allows you to read and write phonebook entries, read, write and send SMS messages and monitor battery and received signal strength.

The SMSD (SMS daemon) program is intended for receiving and sending SMS's. The program is designed to use modules (plugins) to work with an SQL server. Currently supported SQL engines are PostgreSQL, MySQL and a special module '`file`' which is designed to work without an SQL database.

Changes in the team

Unfortunately the gnokii development process has not always been peaceful. Around 3 years ago gnokii passed though something of a leadership crisis. Marcin Wiacek, one of the gnokii contributors, became frustrated that his patches sent to the gnokii list were dropped by the gnokii team leaders and started publically complaining about gnokii maintainership. For both Hugh and Pavel external pressures had simply got to the point where they could no longer devote the time required.

As a result, Chris Kemp and Pawel Kot took over the leadership of the project. For various reasons it was decided not to provide Marcin with direct CVS access. While the new core team tried to find consensus with Marcin it eluded all concerned and he forked his own project called mygnokii based on `gnokii-0.3.3-pre8`.

Since then there were efforts to merge the projects but none succeeded. Marcin also dropped the mygnokii project and decided to write a new version from scratch under the name "mygnokii2" The project was later renamed to gammu. Unfortunately there's no official communication between the two projects at present.

Gnokii now

As discussed in the preceding text, many things have changed since the project began. Many users and developers have contributed to the project, too many

to mention them all here. The CREDITS file in the gnokii sources contains a mostly complete list of contributors. The project evolved and began to support more and more different phones. Currently most popular Nokia phones are supported as well as the AT capable models. The table below shows driver, its current support and the phones supported by the driver.

DRIVER	PHONES	SUPPORT
nk2110	Nokia 2110, 2140, 6080	outdated
nk3110	Nokia 3110, 3810, 8110, 8110i	rising again :-)
nk6100	Nokia 6110, 6130, 6150, 6190, 5110, 5130, 5190, 3210, 3310, 3330, 3360, 3410, 8210, 8290	full
nk6160	Nokia 6160, 5120	partial
nk7110	Nokia 6210, 6250, 7110	mostly complete
nk6510	Nokia 6310, 6510, 8310	mostly complete
atgen	Nokia 6210/7110/8210/6310/6510, Motorola Timeport P7389i (L series), Siemens S25/SL45i, ...	improving

The current gnokii development team is much larger than it used to be, in alphabetical order:

- Borbely Zoltan <bozo@andrews.hu>
  Bozo takes care of: build system, gnokiid, statemachine, link layer, nk6110 and nk6210 drivers.
- Chris Kemp <ck231@cam.ac.uk>
  Chris takes care of statemachine, ringtones support and link layer.
- Jan Derfinak <ja@mail.upjs.sk>
  Jan is the creator and maintainer of xgnokii and smsd.
- Ladislav Michl <ladis@linux-mips.org>
  Ladis joined quite recently. He looks after link layer and AT driver (mainly duncall and cbus support).
- Manfred Jonsson <manfred.jonsson@gmx.de>
  Manfred takes care of device layer and AT driver.
- Markus Plail <plail@web.de>
  Markus takes care of the nk6210 and nk6510 drivers. All credits for the latter driver go to Markus. Markus also maintains xgnokii.
- Pavel Machek <pavel@suse.cz>
  Pavel takes care of: libsms, bitmaps, ringtones and nk2110 and AT drivers.
- Paweł Kot <pkot@linuxnews.pl>
  Paweł is current project leader. He authored the updated libsms based on the previous implementation, GSM specifications and different extensions. Paweł maintains:

gnokii, libgnokii, libsms, build system, documentation (yes, it sucks), bitmap support, nk3110, nk6110, nk6210, nk6510 and AT drivers.

Hugh Blemings and Pavel Janik are still with us but are not as active as they used to be. They still chip in the occasional hint and help out with some organisational issues.

There are some other people involved in the gnokii development. For example, thanks to Marcel Holtmann, one of the Linux Bluetooth stack maintainers, we have reliable Bluetooth support.

Current gnokii design

Gnokii has evolved over nearly five years. We don't claim that it is a perfect design, but it is getting better with the every update.

Gnokii is currently supported on Linux, win32, MacOS X, FreeBSD and Solaris operating systems. The most active development is of course being done on Linux, but win32 and MacOS X people are doing a good job as well. FreeBSD and Solaris ports get synced from time to time.

The major difference between the current version of gnokii and the previous ones is that all base functionality is provided by libgnokii. This is a library with a well defined API that can be linked to an external application dynamically or statically. With previous versions one had to grab the whole gnokii sources and link with the object files built during gnokii compilation.

At some stage we stopped using threads in the internal structures. Using them didn't add much and caused portability and debugging problems. This means that libgnokii needs to work in synchronous mode, but this isn't proving to be much of a disadvantage so far as all supported devices work this way.

libgnokii is internally split into various layers. The lowest layer is the device layer. Currently we support serial ports on all operating systems as well as IrDA and Bluetooth (BlueZ stack) on Linux.

Above the device layer is the link layer. This layer provides FBUS and MBUS as the main protocols. It also provides a layer for AT capable phones and other, less popular protocols.

Above all these is the phone driver. It contains the phone series specific code.

At the highest level of abstraction there are miscellaneous sublibs that are responsible for handling various kinds of the functionality: sms support, calendar support, bitmaps support, ringtone support, etc. With the earlier gnokii releases this layer did not exist and it is still not present in some functional areas but it is proving to be a step in the right direction.

In the middle, across all these layers is a state machine, data structures and functions that allow us to use a stateful connection.

Gnokii usage

gnokii is the command line tool, primarily developed to test the implemented functionality, but with time it grew to the most powerful tool in the gnokii package.

There are few main areas of gnokii functionality: sms support, phonebook handling, calendar handling, security functions and other.

SMS support is an area where a lot of things left to do. We can send and receive many types of the messages: plain text, unicode text, bitmaps (operator logos, caller icons, startup logos), ringtones, picture messages, concatenated messages (ie. messages containing more then 160 characters from the default alphabet). Moreover you can save a SMS message to a mailbox and then read it with Your Favourite Mail Client (tm). It is worth pointing out that gnokii provides the possibility to send a flash SMS, eg. SMS that is immediately shown on the phone display instead of saving it to the SIM card or the phone memory.

Newer Nokia phones have more SMS capabilities in regard to storing them. Gnokii supports all folders features that Nokia provides.

Phonebook and calendar support offer full read-write access to phone/SIM card data. vCard and vCalendar formats are supported.

More interesting features are the "security" options. You can get the state of the phone, eg. get the info that the phone is waiting for you to input the PUK number. Then you can enter this number and even receive the code from the phone. These options are not enabled by default, you need to configure gnokii with ¡code¿–enable-security¡/code¿.

Gnokii now provides a total of 66 different command line switches and almost every switch has multiple options available.

All this functionality is also provided by xgnokii. Some of the features are of course better accessible with GUI, so you probably want to try out xgnokii as well.

Future goals

Our next goal is to achieve multiple phone support. We have done much work in this direction and in theory, the framework should support it, but no testing has been done so far.

Much work has been done last year in the internal gnokii design area. We've put much effort into making gnokii more flexible and more extensible.

The "SMS industry" also goes forward. There are new SMS types: EMS, MMS, which are to be supported by gnokii in the future.

The main area of the further gnokii development is the user interface for the functionality provided by libgnokii. As of 0.5.0 version gnokii project provides libgnokii with stable API to be used in the external applications. All user applications from the gnokii project: gnokii, gnokiid, smsd and xgnokii, make use of libgnokii.

More applications are planned. Recently two projects began to establish: new GUI designed for Windows in C++ and new GUI written in gtk+ 2.0.

New cellular phones provided by mobile vendors permanently surprises us with their functionality. Is seems that there will always be something to add for gnokii.

The other direction of gnokii development is the integration with the third party application. Such application can be OpenOBEX, that we want to use for the Bluetooth file transfer to/from the phone, or misc PIMs that we want to synchronise data with.

The history of gnokii has been a lot of fun but not without the occasional difficulty. We're looking forward to improving it with every new release and thank those that have helped bring us this far.

February 2003
Hugh Blemings
Pawel Kot