Energy Profiling for S60 Mobile Applications

Bachelors Thesis

by

Kanwal Pervaiz

Åbo Akademi University

Faculty of Information Technologies

April 6, 2010

Thesis Supervisor:

Dr. Dragos Truscan

Abstract

Today, mobile devices contain lot of advanced features and applications that were once found on a computer only. However, when compared to computers, the mobile devices have limited computation and power capabilities. It is a great challenge for developers to develop energy efficient applications for mobile devices. In practice, software developers are not aware of energy constraints faced by a mobile device. The traditional measurements of the energy consumption are very costly and need some external equipments and tools. In recent years, new development has made it possible to avoid those costly traditional measurements by developing build-in power profiling applications. Examples of such applications include Nokia's Energy Profiler v1.2 and Carbide.c++ tool Performance Investigator (PI). It enables a developer to measure not only the energy consumption but also to analyze the problems related to performance of the application. These software applications are freely available to use. In this work, we analyze and compare how these two applications work with their different features in order to measure and analyze the performance of any application.

Contents

1 Introduction

In recent years, an extensive development has been done in the area of mobile devices in terms of new features and applications. From very basic to advance features have to be included in such a small mobile device. This development has made it possible for a mobile phone to be a mini-computer or we can say even more than that. This has created a huge demand of mobile phones in the marker. Therefore, now people are more concerned about the advanced features and the performance of the mobile phone devices.

Nowadays, many and more new features are being introduced in the mobile phones which make them power hungry, since more features mean more energy consumption. The battery time for these small devices has been very limiting. Due to this fact, the high processing of advanced features can lead them to high power consumption. Consequently, thermal problems arise which make easily these devices get heated. Although, the battery technology has improved a lot to enhance the life and charge storage capability of a battery, however, in comparison, the development of mobile phone features and its applications has grown up faster.

To deal with this problem power management is very important to make the mobile devices performed well. In order to find out how we can solve this problem, we need to look first what actually the problem is. Here above we have more or less identified the problem which is higher energy consumption of mobile devices which effects on its performance.

1.1 Motivation

We will now move towards the solution approach and first we will see why this problem is needed to be solved and then how it is solved. Since, most of our software developers are not much familiar with the power consumption of their applications because they are only well trained in developing the applications. However, they do not have much knowledge or very limited

knowledge about the power factors of their developed software. Therefore, some applications or software tools are integrated into mobile devices for developers just not only to analyze the energy consumption (voltage, power, etc.) but also to find out the problematic area and try to optimize the energy efficiency. These integrated applications are open source and thus have the combined effect of individual's development.

Since these applications are introduced to minimize the energy consumption and its cost, they should themselves also be energy efficient. These software are developed by using development tools and kits by prioritizing the energy efficiency. Hence, such kind of built-in energy profiling applications are being used as a solution of the identified energy efficiency problem.

1.2 Contribution to this work

We have studied the book "Mobile Phone Programming: Application to Wireless Networking" as a case study for this thesis work. We have also used the online sources to review two built-in energy profiling applications as a contribution to this work. In order to have a detailed inside look and analysis, we have installed both applications and performed different test cases. We have gone through with each application, step by step to have a better understanding of how these applications work. Moreover, we generated few test cases which produced corresponding results in the form of graphs. Then we analyzed these results to see how the values in the results vary and which factors affect these results.

1.3 Background

Here we would like to introduce few terminologies in order to make it easy to understand the further concepts for example, what is a battery and what are the other characteristics related to it. A battery is defined as:
A device that stores energy and makes it available in an electrical form. A

battery converts chemical energy into electric energy. It is a connected bunch (or battery) of electro-chemical devices[3].

Battery-powered devices are those devices in which a battery is used as a main source or the devices which are being operated by batteries for example, mobile phone and different electronic appliances. There is large range of batteries used in different battery powered devices, however, in this work we consider those batteries and their basic characteristics which are used in mobile or hand-held device.

Generally, these types of batteries are rechargeable and have some capacity which can be defined in various terms. The capacity of a battery in terms of current can be defined in Ampere-hour (Amp-hour) as: *the amount of current that a battery can deliver for 1 hour before the battery voltage reaches the end-of-life point*[7].

Another unit of battery capacity, in terms of of battery power, is Watt-hour, defined as: *The Watt-Hour unit means the wattage that the battery can provide within one hour.*[8] For example, 400 mAh at 10.1V means the battery capacity is 400 mAh x 10.1V = 4.04 Watt-hour (Wh). It means that if we know the V and the current (mAh) we can simply multiply them and thus the result would be the power in watt-hour. In other way, power is defined as:

$$P = U * I$$

Therefore, while calculating power, the battery is usually rated as Watt-Hour, or Wh. This is a claimed to be more accurate unit to show the power capacity than Amp-Hour (Ah) that was used before[7].

Energy is the ability to do some particular work, on the other hand, *Power* is energy transfer per unit of time. Therefore, energy consumption would be the energy used during the completion of some required tasks. But the power consumption is then the energy consumed at an instant time within the required task. Hence the power consumption is the rate at which

the energy consumed. It can also be written as follows:

$$(Watt)P = E/s(Joule/s)$$

$$E = P * s$$

$$E = U * I * s$$

The consumption of energy and power varies at different points when doing the same work depending on the type of work. For example, running a application on mobile phone will give us different values at different points in time. When the application stops running, we take the average of the values from beginning till the end, these are termed as the *average energy* and *average power*. Within the time interval of running application we also get some points where the values are at highest or approaching to its capacity, those points are called *energy peak* or *power peak*.

2 Power Profiling Applications

Nokia has introduced the power profiling applications for mobile phones running the Symbian [5] operating system. These power profiling applications having some features which can assist the developers in creating the energy-efficient applications and help to identify the application functionalities which uses excessive power. The main purpose of these power profiling applications is to show the total power consumption of the device (mobile phone) for any particular test setup. These applications work alone and now there is no need to any additional hardware or any external equipment. Moreover, data can be analyzed simply on the mobile phone screen. Therefore, it is more attractive for the developers due to its ease in use. Firstly, desired test cases are generated for those applications who need to be checked for power consumption and then the measurements are taken for analysis.

In this section we will have a detailed look at these power profiling appli-

cations. We will see that how by using these applications, power consumption of an application can be analyzed in a mobile phone. Furthermore, we will also see that how the energy efficiency can be optimized by taking this analysis under consideration. Here we will mainly introduce two power profiling applications, the Nokia's Energy Profiler v1.2 for S60 mobile phones and Carbide.c++ tool Performance Investigator.

2.1 Nokia's Energy Profiler v1.2

Nokia's Energy Profiler v1.2 is stand alone application which is available for Nokia's symbian based operating system. It does not need to connect with any PC to read or analyze the data. Therefore, there is no need of any additional hardware and its only hardware is responsible to measure the remaining energy level for the battery and other measurements which are used by the Energy Profiler. It is basically used to analyze the energy consumption of any application but it can also assist by keeping track of some other information for example, current, memory usage, processor usage, network etc. It runs on the mobile phone as a background application and then traces are generated which shows the results later in graphical format. The traces can also be exported to textual files. It has several features like voltage mode, current mode, power mode etc., which provide information about the consumption of these particular properties. In this section we will see the usage of these power profiling applications, their features, functionalities and the measurement analysis of them with any application in detail.

2.1.1 Settings, features and usage

To use this application on the mobile phone, we first need to install it. There are many ways to install. It can be directly downloaded from *The Symbian Foundation Community*'s home page and then transferred to the mobile phone via Bluetooth or USB. After the installation, desired options and

settings are required to choose. For example, *parameters* can be adjusted for the measurement speed, *sampling duration* and the desired *trace data*. Any particular trace can be disabled if it is not desired or relevant to the testing application because a developer can only be interested to check the battery capacity for current not for voltage. These setting can be chosen from *Options–Settings* dialog. Once the desired settings are done Energy Profiler can be started by choosing *Start* option from *menu*.

After the application has started running, the screen will show the graphical representation of the power consumption of the mobile phone (including all applications running on the mobile phone at that moment). Since at this point no test has been set up, the application will show the overall power consumption of all the application currently running and the graphical flow will be continued. To make a test, for example, open a GPS application and start running it by setting any location for navigation. The energy profiler will start the profiling in the background and will have the graphical representation of the measurements for power consumption. These measurements can be seen on the screen display by bringing it back to the foreground. Now the profiler can stop taking readings by pressing *Stop* key option in the profiler. The graphical representation of this test case for power consumption can be seen in the Figure 1

The displaying graphical representation or the data is the power consumption measurement during that time period when we run the GPS on mobile phone. Now user can see the higher and the lower peaks in the graph and can analyze that at which point the application have used excessive energy and at what time less. For example, by observing these curves it can be analyzed that at a high peak is seen at any particular time then which feature of the test application was performed at that time. So, from there that particular feature or the part of application comes into consideration to identify and improve it for energy efficiency. Time is a key factor in this whole process and it always shows on the top right on the displaying screen.

(a) Executing the use case (b) Graphical representation of power trace

Figure 1: Measurements on Nokia N95. The Maps application acquires the GPS location and the user navigates to a location using the route search functionality. The average power consumption was 0.97 W and this use case could be repeated for 3 hours and 37 minutes with a full battery (taken from [6].

This new version 1.2 of energy profiler provides many new features which includes the touch UI support, new data collection for WLAN, 3G cellular network timers, new graphical features and an improved output data. Apart from these, Nokia's s60 power profiling application features provide all information about power consumption, current, voltage, cumulative energy consumption, processor activity, Ram use, IP network speeds, mobile network signal strength, 3G timers and WLAN signal strength. By choosing time-selection mode you may able to examine just a section of the graph.

An interesting feature of this application is that it is possible to switch between two different view modes. For example, when once the energy profiler is run to know the values for power consumption one can directly analyze the values for other modes as well just by switching from one mode to another. You do not need to run the same test case every time to take the other readings. We can briefly have a look on these view modes in order to understand well how to analyze the data for each mode from the graph.

Since, our main focus is to know about energy consumption so we would

not consider the other view modes in our discussion and will only touch some of the view modes and some information related to it. The power mode shows the power consumption within a specified time period of a test application as the name shows. The basic unit of the measurement is Watt (W). The current mode is measured the consumed amount of current. Usually, this measurement is taken from the battery. Therefore, current consumption is usually dependent on the battery charge level. The measurement unit of current is mA (milli Ampere). The current mode view look like the Figure 2.

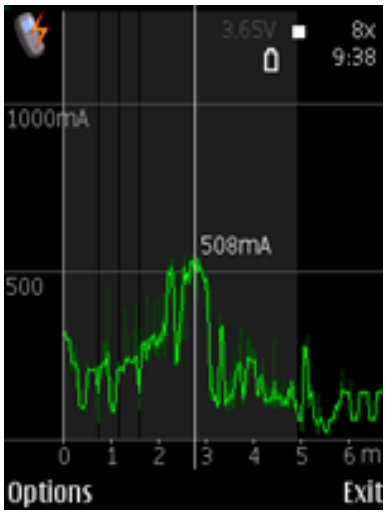


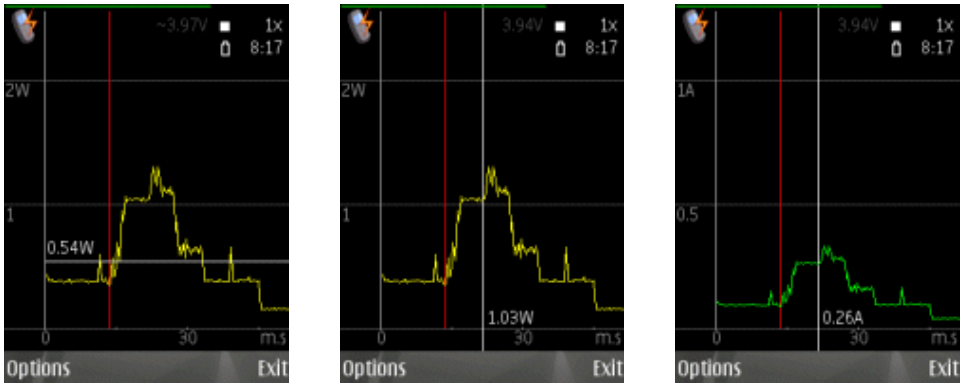
Figure 2: Current view shows milliamperes (mA)(from[4]).

The mode view of Processor mode shows the CPU load due to this test application as a percentage of total CPU availability. In other words, it shows the process activity due to this test application. It is measured in percentage. Energy view shows the cumulative energy consumption over the specific time period. It is measured in milli-Ampere-hour (mAh). RAM memory view shows that how much memory is used by that application within the time duration. The unit is megabytes (MB). Similarly all other views tell the information about their particular property and all of them

have their own measurement units to show the value.

2.1.2 Measurement analysis

To analyze the data from the graph we must understand how to read the data and which displaying lines show what. Mainly, there are two vertical lines in each mode, which are red and gray in color. The vertical grey line shows or indicates the start or the end of a measurement region. Whereas, the red vertical line indicates that a screen shoot of that particular area is taken. The screenshots create a trace of that specific part of the application where are some sudden peak or significant change occurred. These screenshots are really important at the time of analysis or testing of a application. Basically, these screenshots are the history of graphical patterns for power consumption at that time instant. The following Firgure 3 shows the screenshots of the Nokia N81 device when the camera application was open and a photograph was taken.



(a) Executing the use case.

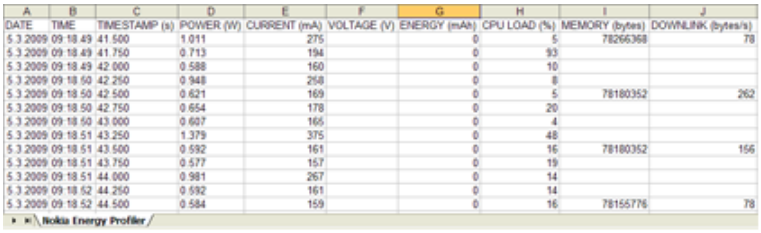
(b) Instant power consumption is over 1.03W.

(c) Switching to current-consumption view shows 0.26A.

Figure 3: The screen shots were recorded on a Nokia N81 device when the camera application was open and a photograph was taken.(from[4]).

Figure 3(a) shows the execution of the use case (photograph taken from camera). Average power consumption was 0.54W during the measurement.

Note that the measurement and average span cover not only the graph visible on the screen. The red vertical bar shows that a screen shot was taken. Figure 3(b) shows that the instant power consumption was over 1.03W when this screen shot was taken. Peaks in the power graph are generally not a problem, but it is high baseline consumption that causes high energy consumption. In Figure 3(c) switching to current-consumption view has been shown which tells that the current is 0.26A. Having many traces or screenshots at different time instant and in different situations can lead a developer to find out the solution by making their comparison. For example, if developer finds out some specific algorithm or part of program taking more time to process then the specific design decision can be replaced or changed to reduce its complexity. However, snapshots are really helpful for developers to browse through later to identify the problems. The graphical representation is read by taking care of displaying line and if the peaks are higher within those line limits then it indicates the excessive power consumption points of the applications. We can then save these results by choosing the save option. It is also possible to export the results on the PC just for convenience if mobile phone's screen is too small to read or analyze the data. Furthermore, analysis can be done in many different formats according to the ease for example, exported .cvs file looks like as it shows in the Figure 4.



A	B	C	D	E	F	G	H	I	J
DATE	TIME	TIMESTAMP (s)	POWER (W)	CURRENT (mA)	VOLTAGE (V)	ENERGY (mAh)	CPU LOAD (%)	MEMORY (bytes)	DOWNLINK (bytes/s)
5.3.2009	09:18:49	41.500	1.011	275		0	5	78266368	78
5.3.2009	09:18:49	41.750	0.713	194		0	93		
5.3.2009	09:18:49	42.000	0.588	160		0	10		
5.3.2009	09:18:50	42.250	0.948	258		0	8		
5.3.2009	09:18:50	42.500	0.621	169		0	5	78180352	262
5.3.2009	09:18:50	42.750	0.654	178		0	20		
5.3.2009	09:18:50	43.000	0.607	165		0	4		
5.3.2009	09:18:51	43.250	1.379	375		0	48		
5.3.2009	09:18:51	43.500	0.592	161		0	16	78180352	156
5.3.2009	09:18:51	43.750	0.577	157		0	19		
5.3.2009	09:18:51	44.000	0.981	267		0	14		
5.3.2009	09:18:52	44.250	0.592	161		0	14		
5.3.2009	09:18:52	44.500	0.584	159		0	16	78155776	78

Nokia Energy Profiler

Figure 4: Exported CSV text files can be opened in Excel (from [4]).

2.2 Carbide.c++ Performance Investigator (PI)

A stand-alone S60 application for power profiling may not be work well in the case of a complex application as it requires deeper analysis. In this case, Carbide.c++ tool Performance Investigator (PI) is used to investigate the behavior of an application to test or improve its performance. In order to grasp the whole concept we need to introduce information about Carbide.c++. It is an Integrated Development Environment (IDE) which is based on the open-source Eclipse platform. Eclipse was initially started for the Java development but with the passage of time it became an integration platform for the development of different tools and applications. Thus, Carbide.c++ is an eclipse based IDE for the development of the Symbian C++ applications and the Performance Investigator is one of its tools. There are mainly three editions for Carbide.c++ and every new edition support updated features for Power Investigator like improved view of graphs, support for large sample files, improved data import and time for profiler and the support for key mapping as it described in [1].

Performance Investigator tool is valuable for the developers because it helps to investigate an application's behavior by collecting the all measurement information when the third-party application runs. With the help of Power Investigator developer can not only analyze the performance of an application but can determine the problematic part of the application and then try to optimize it. Performance Investigator does not only provide measurements for power consumption but collects the data or information together with the processes load, memory and battery usage. By optimizing the measurements with all these aspects developers can easily improve the performance of an application.

2.2.1 Profiler and Analyzer

PI is not a stand-alone application instead it has two different components. One is installed on an S60 3rd Edition mobile device is called profiler. The

second is a plug-in application which runs on the PC, is called analyzer. Profiler is responsible to generate the trace files, collect all the data measurements and to import them to analyzer. However, analyzer allows the imported information to be collected successfully and then analyze it. In this section we will see in detail that how this performance profiling tool works, what are its features and how it analyze the performance particularly power consumption of an application.

When once started, Profiler application usually stays in the background while the profiling data activity appears on the foreground. Profiler is basically an application and a set of kernel drivers as described in [1]. When a program is executed and the profiling is started, these kernel drivers listen for the function calls and the application starts collecting the data from it. As a result, they record a stream of events which were performed during the execution of uses cases. This recorded data usually known as a trace and one can specify the sampling data it wants to capture in a trace. For example, memory usage, power usage, function calls and etc. Since kernel records the trace therefore, it includes the data from all those processes which were active at the time of profiling. Thus, the trace data was not just of the desired application. Due to this reason, kernel differentiates the trace data of different processes with mark. These traces are really valuable for the developers because it provide detailed, low-level information which is really helpful to optimize the efficiency of an application. When once the trace data is generated, it imports to the analyzer where it collects the imported data on a PC and analyzes it.

2.2.2 Setup, usage and analysis

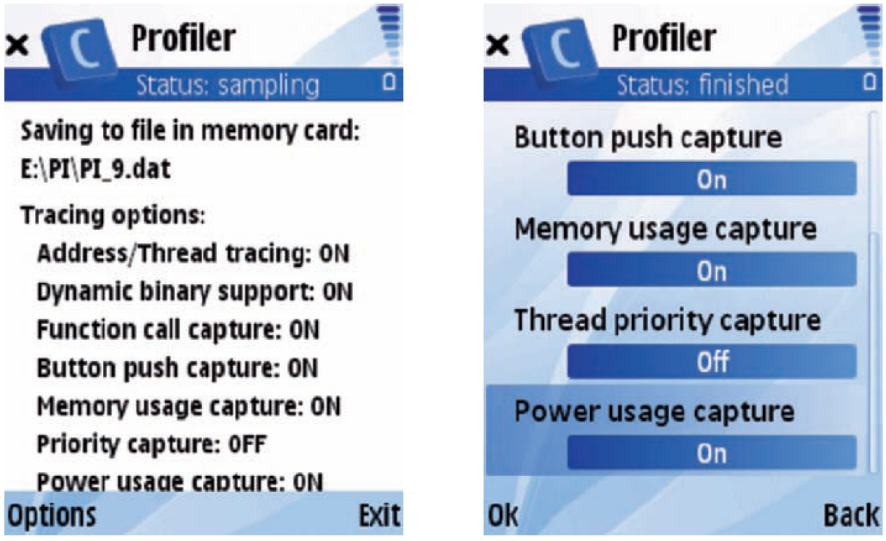
Profiler is a key part of the Power Investigator tool because it collects, generate, transfer and import the data. The PI provides developers a visual picture of Operating system threads, CPU load and activities together with the measured power consumption from the battery. This picture is very use-

ful with respect to identifying some problems. Let's take an example of the S60 web browser designers as described in [6]. They had an investigation of a slowdown at the time of communication with the 3G modem device driver. When they took a analysis of power consumption then it revealed that after certain function call, power was dropped dramatically. When they analyzed the levels of trace data it showed that 3G modem was mistakenly disabled for that moment. So, they found a hidden bug in the driver interrupt handling routine and fixed it. It is said that this programming error could remain undetected if it could not been detected through the power profiling. This is how the developers analyze the problems in an application and fix it by understanding the different behaviors of the applications. To use the Power Investigator, one needs to install the profiler application first on an S60 3rd Edition device. Now It will acquire few settings and configuration in order to profile a desired trace. For example, one can specify that which type of data it wants to be captured in a trace to check the performance. These options are: Dynamic binary support, Function call capture, Button push capture, Memory usage capture, Thread priority capture and the Power usage capture. The user interface for the Profiler application with these tracing options looks like as shown in following Figure 5. Since, in this paper our main concern is power usage so we will shortly describe that what these other options do according to [1].

- Dynamic binary support - this tracing option is chosen if the third-party application is installed in the device's RAM. But this is not needed when the third-part application has its own .symbol file.

- Function call capture - this option is turned on when it is needed to record the function calls.

- Button push capture - this option is very useful to mark some points in the data collection. Basically it enables the Profiler to record the button events, which are used as annotations in Analyzer graph.



(a) The main view for the on-device profiler

(b) Enabling the power traces

Figure 5: User interface for the Performance Investigator on-device profiler. The Options dialog allows for the setup of the parameters to trace, including power consumption (taken from [6]).

- Memory usage capture - this option must be turned on when needs to record the memory usage.
- Thread priority capture - by turning on this option one can obtain priority data for threads.
- Power usage capture - by choosing this option one can record the power usage levels. The Analyzer does not show it for individual processes instead it shows as a global data set over the collection period.

We can see in above figure that there is one more option which is named as Address/Thread. This option always remains active in the profiler and other traces are synchronized with it. The power tracing for the PI requires the following few steps to set up the necessary components as described in

the book "Mobile Phone Programming: Application to Wireless Networking" [6].

1. Go to the profiler setting and enable the desired trace case. This option enables us to specify the types of desired data in the traces. Choose the *Tracing Option* from the setting dialog in order to enable the power trace.

2. Now choose the *advance option* and from there select the *Power usage interval setting* to set up the sampling resolution. Usually the sampling has lower and upper bound for the resolution. Similarly, user can choose other desired options and can easily setup and configure the profiler settings according to it desired investigation.

3. Now to *Start* profiling, select *Options - Start* and then send it to the background by choosing an application from the task list.

4. Execute the desired use cases for the application to profile.

5. To stop tracing, bring the Profiler to the foreground and select *Options - Stop.*

6. Trace files can now transfer to the PC. Setup the location of the files on the profiled device through *Output setting* in the PI Profiler.

7. Go to the menu option and select *File - Import.* This will import the trace files to the carbide.c++.

8. Now traces or the imported files can be analyzed through the plug-in Analyzer on the PC. It can be tried to determine the high power consumption if it triggers in any execution constructs.

There are usually two options in the Advanced options dialog as discussed in [1]. One is the Memory/Priority interval and other one is Power usage

interval which we used above for power trace. To identify the currently executed threads, OS is checked by the Profiler in every millisecond. So, the power usage interval setting is used to tell the profiler that how frequently it needs to check the power usage. On the other hand, Memory/Priority interval setting indicates that how often it need to determine the priority of the current executing threads by putting an additional check on the OS. Note that performance of the device may affect due to this frequent sampling. It may also cause delay in resulting trace date due to the significant overhead.

The above profiling session will result a trace data file which contains all the required information for analysis. Now the file can be saved in the file system of the mobile phone and then it needs to be transferred via Bluetooth or USB to the PC where Carbide.c++ is running. This file can be saved anywhere in the PC because Carbide.c++ Analyzer have this ability to import the file anywhere from the system. Now open the Carbide.c++ and the start importing the data by choosing Import option from the File menu in the Carbide.c++. It will also require certain setting to be able to use the data file successfully into the Carbide.c++. For example, this data file is .dat extension and it would require a conversion to .npi extension which is uses by the Performance Investigator Analyzer. The importing wizard will also ask that which type of data is in the file to be imported and finally the location of the importing file in the Carbide.c++. Once all the setting done clicking on the Finish option in the Import trace data, wizard will display the imported data on the screen. Here in the Figure 6 has shown the main analysis view of power usage when a photograph was taken from the camera application of the device.

The thread load data in the main analysis view is built from the trace data option address/thread trace. It is always very useful because one can see the problem at the system level by analyzing the power usage together with the CPU activity traces. Since many hardware components are still not synchronized with the CPU therefore, sometimes it may be hard to identify

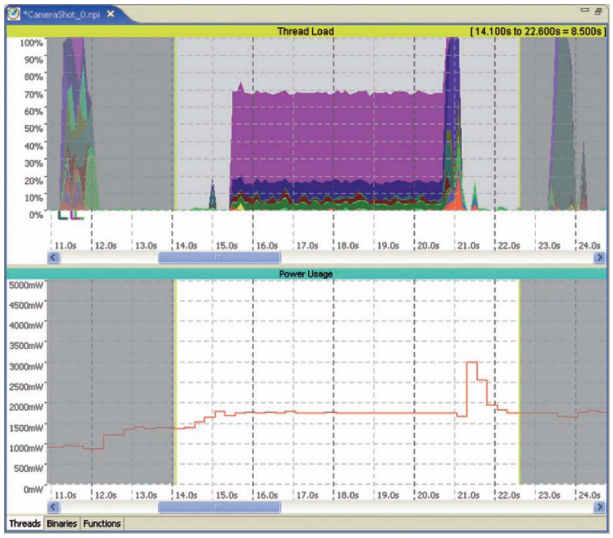


Figure 6: Power consumption analysis in the Carbide.c++ PI. The profiled use case involved the camera application, where 5s were spent view finding and then a picture was taken. The thread load on the CPU is shown on the top, with the power consumption graph on the bottom (from [6]).

the problem at the system level. Because in this case no correspondence can be found between the power usage and the specific program threads as CPU is not the main power consumer. The following Figure 8 shows the whole visual graphical analysis view of the memory usage together with the power usage.

In Analyzer, user interface have several options or items to control the analyzing process. This includes the option to show the whole graph for profiled imported data, zoom in, zoom out and select time interval. This last option provides the facility to see only that area of the current graph which is desired and specified in the time interval. The different items are represented in different colors to make it easy to follow the visual picture. These colors are basically used to create the correspondence between the thread list and the thread load graph. The very interesting feature of Analyzer is that one can change the information view by clicking on the options beneath the graph

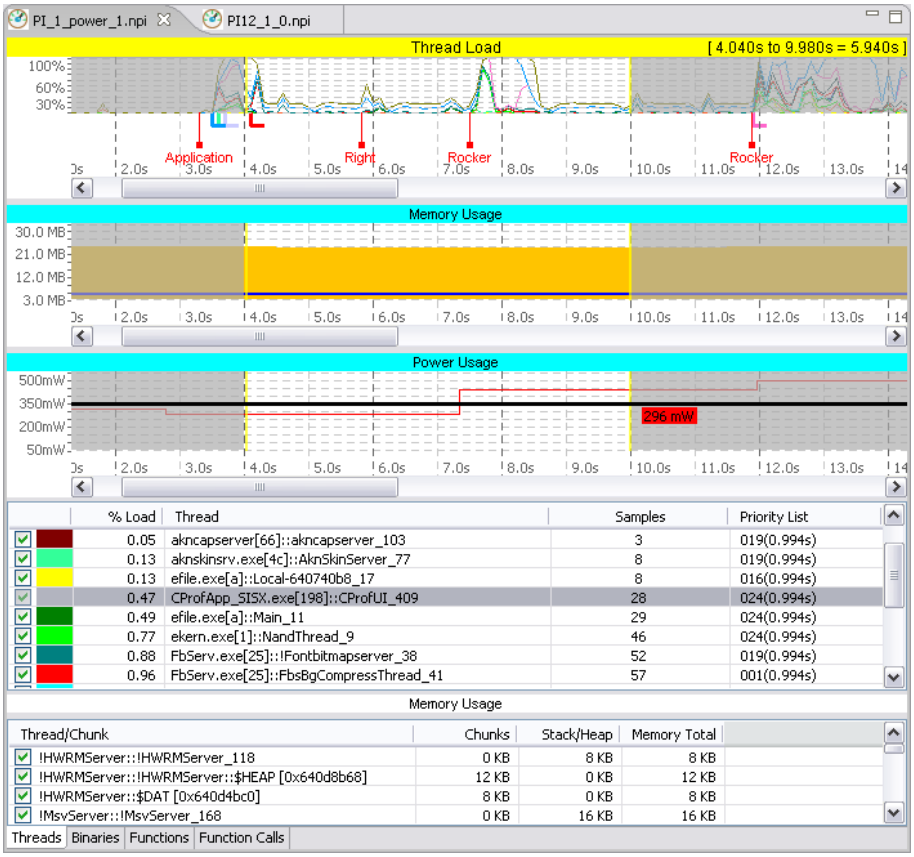


Figure 7: Analyzer main view showing Thread, Memory and Power graphs (taken from [2]).

as shown in the figure 8. For example, one can change the thread graphical view to its desired view to see the statistics. For example, one can change the thread graphical view to its desired view to see the statistics.

3 Comparison and Analysis

We have overviewed two applications named as Nokia's Energy Profiler v1.2 and Carbide.c++ Performance Investigator (PI). Both are used for the energy profiling of S60 applications but are different in features and with many

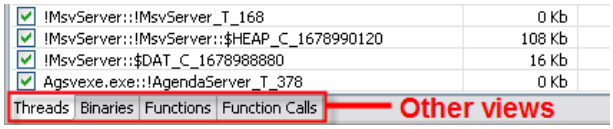


Figure 8: Selecting Threads, Binaries, Functions, or Function Calls (from [2]).

other aspects. Nokia's Energy Profiler is a stand-alone application while Carbide.c++ PI has two parts of application and dependent on mobile as well as on PC. So, in this section we will compare these two applications with each other and analyze the differences.

Nokia Energy Profiler v 1.2 and Carbide.c++ tool Performance Investigator (PI), both applications provide support to profile the power consumption in order to check the performance of an application over the device. Through these applications a developer can see the behavior of a third-party application at the device level. According to our findings, in both applications the power consumption measurements are at same low-level interfaces with same speed and accuracy. However, the difference is that the Energy Profiler is a stand-alone application and aimed to monitor the behavior of an application on the device. On the other hand, the Carbide.c++ Performance Investigator is aimed to integrate the on-device profiler together with the PC-based analyzer and use it in an Integrated Development Environment (IDE) on the PC.

Although, the Energy Profiler is good enough to provide information related to the problems of power consumption and performance but sometimes it become limited when the complex applications are come into account. In these cases, Performance Investigator provides a deep and detailed analysis to understand the problematic behavior of the application. On the other hand, with few aspects Energy Profiler is attractive for the developers as it is a stand-alone application and there is no need of any additional hardware unlike the Performance investigator. Moreover, it provides a quick look on

power consumption in real time and a developer can monitor the results on the same device means, no need to wait till it transferred to the PC for analysis.

Both applications are a bit different in there features as they provide information with certain different aspects to check the performance of an application over the device. For example, in case of analyzing power consumption Energy Profiler provides current and voltage information through which a developer can identify the excessive power problem and can optimize it. While Performance Investigator shows a whole picture of the power consumption measurements together with the CPU activities. It includes the information about thread load, function call, binaries and etc. Since, this tool is helpful in indentifying the recurrent instances of particular function calls and threads that seems to have a significant contribution in the power consumption [6]. Therefore, by these means PI is more powerful than the Energy Profiler because through its provided detailed information a developer can identify and optimize the problem easily and in a better way.

The new version of Carbide.c++ OEM Edition supports many debug tools that can be helpful to fix the problem which is identified by the Performance Investigator. This advantage can lead developers to prefer Carbide.c++ PI over Energy Profiler. Because in our opinion, a developer would like to work in an IDE where it can first develop the application then analyze the problem through Performance Investigator and finally can fix the bug with the help of debug tools. So, PI provides advantages over Energy Profiler which makes it more favorable for the developers. Then we can conclude that both applications are attractive for developers somehow with different means but Performance Investigator is more preferable as it provides a deep and detailed analysis of the behavior of the third -party application.

4 Conclusions

In this work, we have overviewed two applications for energy profiling on S60 mobile phones which are Nokia's Energy Profiling v1.2 and Carbide.c++ Performance Investigator (PI). We have seen that how these two applications are used on S60 mobile phones to check the performance particularly power consumption of a third-party application. We also have discussed their features and the profiling data to be displayed for the analysis. We have provided some example of use cases to understand that how a developer can analyze the behavior of any application with the profiled data and can try to optimize it. We also have compared and analyzed these two applications on the base of our findings. At the end we have concluded that in our opinion Carbide.c++ Performance Investigator has more advantages over Energy Profiler. Because it assists a developer in a detailed analysis of the any S60 application in order to check its performance on the device. Thus, PI is very powerful tool in the long run.

References

[1] Analyzing Application Performance with the Carbide.c++ Performance Investigator. http://developer.symbian.org/wiki/index.php/Analyzing_Application_Performance_with_the_Carbide.c++_Performance_Investigator.

[2] Carbide.c++ Analyzer View. http://carbidehelp.nokia.com/help/index.jsp?topic=/com.nokia.carbide.cpp.pi.doc.user/html/Getting_Started/GS_index.htm.

[3] Electric Battery History. http://www.ideafinder.com/history/inventions/battery.htm.

[4] Nokia Energy Profiler Quick Start. http://www.forum.nokia.com/Technology_Topics/Application_Quality/Power_Management/Nokia_Energy_Profiler_Quick_Start.xhtml.

[5] The Symbian Foundation Community Home. http://www.symbian.org.

[6] Gerard Bosch Creus and Mika Kuulusa. *Mobile Phone Programming: Application to Wireless Networking*, chapter Optimizing Mobile Software with Built-in Power Profiling, pages 456–457. Springer Netherlands, 2007.

[7] Chester Simpson. Characteristics of Rechargeable Battries. http://www.national.com/appinfo/power/files/f19.pdf.

[8] Yun Wang. NCQ for Power Efficiency - White Paper. February 10, 2006.