

Mobile Application Prototyping with Python for S60

Bernhard Famler, BSc

bernhard.famler@fh-hagenberg.at

Mobile Computing
University of Applied Sciences, Hagenberg
Softwarepark 11, 4232 Hagenberg, Austria

Technical Report Number 06/1/0455/003/02

October 2007

Abstract

Mobile application development has become more and more important during the last couple of years since the number of devices increases rapidly and the capabilities of phones enable a new variety of services. This trend requires new opportunities for creating innovative software in an efficient and comfortable manner. With Python for S60 (PyS60), Nokia brought the Python programming language to S60 phones, which offers new ways of mobile application development and rapid prototyping. This paper gives an introduction to PyS60, deals with the development process of applications and identifies differences to common approaches when using the native Symbian C++ or JavaME platform.

1. Introduction

The number of smartphones on the market has been increasing constantly over the last couple of years, due to the changing way people communicate, interact and work. There is a need in small mobile devices and the ability to provide anytime, anywhere access to information in an easy and effective way. According to iSuppli¹ analysts, there were 2.7 billion cellular subscribers globally in 2006 and forecasts about 3 billion subscribers by end of 2007. Perhaps this market is growing faster than every other technologic sector. This increasing amount of devices associates the need of new services and applications along with platforms on which they are developed and deployed. Nowadays the mobile phone has become a multimedia all-rounder, which enables the user to take photos and videos, enter the world-wide web or using it as a multimedia player.

Since the beginning of this trend, far-sighted software developers began to write applications for mobile platforms, to benefit from an upcoming industry. The drawbacks of the huge amount of possibilities offered by the mobile platforms are well-known. Limited hardware-capabilities, low CPU speed and small displays require clever software concepts and remarkable development skills, to make an application feel efficient or suitable. Different software platforms are established among the variety of phone manufacturers. Some build on their own proprietary operating system whereas others use published platforms, which enable third party developers to use native APIs to access phone-specific capabilities for their applications. Among these, *Symbian OS*, developed by the Symbian Ltd.², turned out to be one of the key players on the market. But this powerful mobile platform comes along with the drawbacks of handling high-performance idioms of the C++ programming language, which is used to write the operating system itself.

Sun Microsystems's³ Java Platform, Micro Edition (JavaME) and Qualcomm's⁴ Binary Runtime Environment for Wireless (BREW) are two popular technologies enabling third party development upon the device's firmware. JavaME is Java tech-

nology specially customised for small consumer and embedded devices with limited processor, memory, display, and input capabilities, which makes it easy for old-established Java developers to jump on the mobile bandwagon. The Virtual Machine (JVM) runs on top of the device's operating system and is customized for its specific requirements, which offers huge compatibility and portability. BREW is an application execution platform that runs at the firmware level and is much like the JVM in Java, except that BREW runtime environment is not designed to provide portability from one device to another. The programming language, used to write applications for BREW, is C++.

The ongoing distribution of JavaME-capable mobile phones makes it no longer necessary for developers to become smartphone specialists with a profound knowledge in hardware and OS. Mobile software development should be easy to learn and quick results are the main goal. Nokia had recognised the need for a language that offers people a way to produce useful programs without having spent months learning Java or C++ and the multi-threading intricacies of Symbian OS. The result was *Python for S60* (PyS60), an interpreted script language port for Symbian OS phones. Python is known for its simple concepts and slim language specification of around 100 pages. But not only beginners should benefit from this opportunity. The simplified approach could make fast and conceptual development possible for even the experienced ones.

2. Contribution

A severe aspect when building a first prototype based on an initial, promising idea is to use the right tools. The development setup should be done rapidly. The workflows must be easy to learn and to handle. All that is what Python is famous for and that makes it an attractive opportunity for testing user interfaces, spontaneous ideas or rapid prototyping on smartphones in general.

2.1 Python as a programming language

Python [9] is a dynamic object-oriented programming language, available for many different platforms and environments. The code is interpreted at runtime, in contrast to languages like C++, where it is compiled before execution. Python enables programs to be written compactly and readably, which enables a fast learning process. Programs are typically much shorter than equivalent C++ or Java programs because high-level data types allow you to express complex operations in a single statement; statement grouping is done by indentation and there are no variable declarations necessary. A short code example, as seen in Listing 1, demonstrates the definition of a function and its call afterwards. Variables are used without explicit declaration and statement grouping (while-loop) is achieved by indentation without brackets.

Listing 1. Definition of a function.

```
1 #define a function
2 def fib(n):
3     a, b = 0, 1
4     while b < n:
5         print b,
```

¹<http://www.isuppli.com/>

²<http://www.symbian.com/>

³<http://java.sun.com/>

⁴<http://brew.qualcomm.com/>

```

6         a, b = b, a+b
7
8 #call the function
9 fib(2000)

```

As mentioned, Python is an object-oriented language, which offers the opportunity of defining data structures and classes. Listing 2 shows a simple class with an attribute (i) and a member-function (foo). The instantiation of an object is given below.

Listing 2. Definition of a class.

```

1 class SimpleClass:
2     i = 12345
3     def foo(self):
4         return 'hello world'
5
6 x = SimpleClass()

```

Python offers functional, object-oriented or aspect-oriented approaches and developers feel free to choose either one or even mix them up. This freedom and diversity involves the need of a certain runtime-environment in which Python scripts are executed. This platform-specific environment, similar to a Java Virtual Machine, is called Interpreter and has to be installed on the target in order to deploy and run programs.

2.2 Python for S60 (PyS60)

Python for S60 is Nokia's port of the Python language to the S60 smartphone platform. The S60 platform comprises a user interface for Nokia devices based on the operating system Symbian OS and the required development tools. There are currently 3 Editions (1st, 2nd, 3rd), along with different feature packs, which differ in the time of release and the features included depending on the devices' capabilities. Typical S60 devices of the youngest generation come along with smart pre-installed applications, networking and multimedia capabilities which are accessible through well defined APIs. With PyS60, developers are able to tap almost the full potential of the S60 platform.

Development with PyS60 is based on the S60 SDK, which is generally used to write native C++ applications for Symbian OS. The SDK comes along with a device emulator, on which PyS60 scripts can be deployed and tested. Many of S60's device specific functions, like using the camera or network features, require a device for execution. Scripts are written on the PC and can be deployed on the device e.g. via Bluetooth. The precondition is an installed PyS60 interpreter in either case. There are different IDEs that support the Python language and tools that make deployment on the device relatively easy. Strictly speaking, it would be sufficient to have a text editor, like *pyEdit for S60* [4], installed on the mobile phone, to create and edit scripts, that can be executed directly on the device. However this doesn't seem to be a very comfortable way of developing.

Building stand-alone applications, which can be installed on a device similar to native C++ applications, is also possible. And to underline the strong relation to the native programming language, C++ code snippets can be embedded into conventional Python code.

Currently Python for Series 60 is based on Python 2.2.2. It supports many of the Python Standard Library modules, which cover data objects, types, exceptions and other basic language specifics. Mobile platform specific modules, or extensions, grant access to S60-related functionality, not provided by the standard modules. There are two extensions built in the PyS60 package. The *e32* module implements interfaces to special S60 platform services like asynchronous calls with Active Objects (AOs) or system information (battery, display, memory). Handling the user interface framework of S60 is done by using the *appuifw* module. Beside these built-in extensions there are dynamically loadable modules that provide proprietary APIs for messaging, camera, audio, contacts or graphics and drawing. Detailed description for each of the modules can be found in the *PyS60 Library Reference* [7].

Some of the mentioned aspects above are demonstrated hereafter, along with a short guide to the development process with Python for S60.

3. Related work

Juergen Scheible and Ville Tuulos give an introduction of the Python programming language to the mobile S60 platform. In their paperback *Mobile Python* [12], published 2007, they show how to realize application ideas on Symbian OS and how to use the features of Python for S60 for rapid prototyping.

One of the two authors mentioned above (Juergen Scheible) also runs an online tutorial [13] that gives a smooth start into learning to program PyS60 - even without any prior knowledge of Python or S60. On the basis of short code snippets, the key aspects of development with Python are shown.

The *Hochschule für Technik FHNW* published an article [11] about the pros and cons of using Python for S60 in mobile application development compared to JavaME. Differences between the two approaches concerning memory, execution time and development environment are outlined, based on benchmark tests and specific tryouts.

Steve Litchfield discusses the potential of Python for S60 and demonstrates first steps, leading to a stand-alone S60 application. In his online-article [5] he gives a general introduction and points out the differences to equivalent languages like OPL⁵.

4. Implementation

Basically Python programs are text-based scripts which are interpreted at runtime and therefore don't have to be compiled. This speeds up the process between coding and running the application on the target, either the S60 emulator or the phone. Compared to the JavaME platform, where a MIDlet along with an application descriptor (.jad) has to be created before deployment, the script is immediately ready to run. The easiest way of writing Python code is to use a simple editor, although it is recommended to use a Python IDE with language specific features like code completion, syntax and indentation analysis or debugger. Famous environments

⁵Open Programming Language

with the mentioned support are Eclipse [17], WingIDE [18] or Stani's Python Editor [15].

4.1 Prerequisites

Essential for the development on PC, no matter what IDE is used, is to install the Python Software [10] that includes an interpreter, the command line shell and documentation for all standard modules. To use all standard features, apart from S60 specific modules, at least version 2.2.2 is needed.

Python for S60 is available for S60 2nd and 3rd Edition and it is recommended to install a S60 SDK [8] (preferentially 3rd Ed.), which includes the documentation, API reference and emulator for development without a device, as needed for creating native C++ applications. To use the whole functionality of S60, on-device development is the better choice of course, considering features like camera or multimedia usage. There are currently around 50 Nokia devices⁶, which are based on the S60 platform, most of them are 3rd Edition phones.

A good approach is to choose Eclipse or Nokia's Carbide IDE [6] for PyS60 development. Eclipse is well known in the Java community, whereas Carbide.c++ for building native C++ software for Symbian phones. For both of them, a Python development plugin, called PyDEV [1], is available. PyDEV is a software plugin, which assists Python programming with features like project and module creation wizard, syntax highlighting, indentation support, code completion, code coverage and many more. The plugin can be easily installed using the Eclipse Update Manager or by copying the extracted files into the root directory of the Eclipse installation. A detailed installation guide can be found on the PyDEV website [1]. Compared to JavaME or native Symbian application development, more tools are needed, considering the Java runtime environment, IDE and plugins, JavaME Wireless Toolkit and platform SDKs.

4.2 Developing PyS60 applications

Running scripts on a mobile platform requires the Python environment being installed on the target and therefore the correct kind of Python interpreter is needed. The latest release of the interpreter, for phone and emulator, is available for the S60 2nd and 3rd Edition at SourceForge.net [14]. Currently there are no S60 phones with pre-installed Python available, but, according to Nokia, there will be 3rd Edition phones in the near future. That means a disadvantage compared to the JavaME MIDP platform, which is nowadays pre-installed on almost every handset released, and of course the native Symbian platform. To install PyS60 for the emulator, the PyS60 SDK ZIP package has to be extracted to the root directory (which contains Epoc32, S60Ex, S60Tools, etc.) of the installed S60 SDK. For the device, two SIS packages are needed: PythonForS60 runtime and PythonScriptShell. Both need to be copied to the phone (or memory card), either using the Nokia PC Suite or Bluetooth OBEX file transfer. On the phone the installation can be started by simply opening the two files, starting with the PythonForS60 runtime. Python is located either in the installed programs folder or the main menu of the device (and emulator).

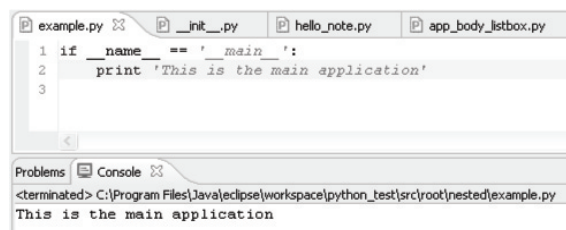
⁶http://www.forum.nokia.com/devices/matrix_s60_1.html



The main application itself is a Python script, named `default.py`, that offers functionality as shown in the figure above. *Run script* enables running Python scripts located on the target. The *Interactive console* is great for testing code snippets by simply prompting a statement to the shell. For deployment and debugging functionality the *Bluetooth console* is included. It is the easiest way to run Python on a phone, assuming that you have Bluetooth connectivity in your PC. Applications can be started on the device right from the PC and debug messages are sent back for visualization on the desktop screen. There are different tools for the same purpose, which will be demonstrated later on.

As mentioned above, using Eclipse and PyDEV is a good approach for development. After installing it, the Python interpreter has to be configured by choosing the Python installation for Eclipse under Preferences → Pydev → Interpreter → Python. By using the Eclipse project wizard, a new Python Project can be created. Python modules (scripts) should be beneath a source folder, which has to be created next. A package hierarchy can be set up and the first modules (New → Pydev Module) can be inserted. A complete description of the PyDEV setup is given on the PyDEV website [1].

It is now possible to write Python code and run it right from Eclipse. Of course no S60 specific functionality can be accessed, as there is no PyS60 interpreter available for PC. For quick syntax or generic algorithm tests however, it could be helpful to execute code on the PC, as the figure shows below.



4.3 Functionality and modules

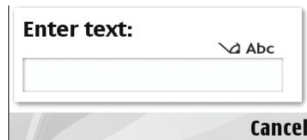
The Python for S60 package includes several built-in and dynamically loadable modules. Their functionality can be used by just importing the correct extensions. The functionality of these modules and the efficiency of Python code are

demonstrated with several short code snippets below. Many applications require a standard user interface built with conventional listboxes, dialogues or forms. These elements are accessible through the *appuifw* module of PyS60. Listing 3 shows the code for a query dialogue to gather user input.

Listing 3. Query dialogue.

```
1 import appuifw
2 data = appuifw.query(u"Enter text:", "text")
```

This produces the following output:

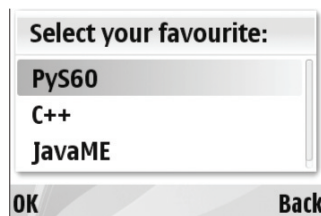


The *appuifw.popup_menu* function generates a pop-up menu for selection, like in Listing 4.

Listing 4. Pop-up menu.

```
1 import appuifw
2
3 L = [u"PyS60", u"C++", u"JavaME"]
4 c = appuifw.popup_menu(L,u"Select favourite:")
5 if c == 0 :
6     appuifw.note(u"You selected PyS60","info")
7 elif c == 1 :
8     appuifw.note(u"You selected C++","info")
9 elif c == 2 :
10    appuifw.note(u"You selected JavaME","info")
```

The menu will look like this:



After the user input, a pop-up note will arise:



This shows that only a few lines of code can create a conventional GUI, which can be used for diverse applications. Just as simple as building GUI elements is the usage of messaging and call functionality. Listing 5 triggers the transmission of a short message using the *messaging* module.

Listing 5. Send a SMS.

```
1 import messaging
2
3 nr = "00431234567"
4 txt = u"SMS text"
5 messaging.sms_send(nr, txt)
```

A call can be easily set up with the following code, using the *telephone* module (Listing 6).

Listing 6. Call a MSISDN.

```
1 import telephone
2 telephone.dial("00431234567")
```

Of course, multimedia functionality, like using the phone's camera, can be accessed (Listing 7).

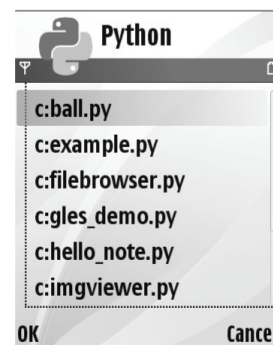
Listing 7. Take a photo.

```
1 import e32
2 import camera
3
4 image = camera.take_photo()
5 image.save(u"e:\\photo.jpg")
```

For description of full functionality of the PyS60 modules, refer to the *PyS60 Library Reference* [7].

4.4 Deployment of PyS60 scripts

Deployment of Python scripts on the emulator simply means copying the files to the correct directory of the S60 SDK, so that it can be found by the interpreter for execution. The default location is in the root directory of the SDK under *Epoc32/winscw/c/python*. To speed up the deployment process, it is recommended to set the workspace path of Eclipse to this directory, so that updates take effect immediately when saving the script. To run the script just choose *Run script* in the Options menu of PyS60. As the figure shows below, all Python modules, which are located in the python directory, can be selected for execution.



For most of the applications implemented for the S60 platform, it is more suitable to deploy them directly to the phone. Things like using the camera or making a call are simply not possible on the emulator. The deployment process on the device is much more time-consuming, if manually copying the

scripts to the correct directory. There are several useful tools which facilitate this process. *PUTools* [3] is an open source toolbox that builds upon the Bluetooth console of the PyS60 SDK and takes input and shows output on PC, connects over Bluetooth to the phone, and executes scripts on the phone. You also get simple shell functionality for the phone (cd, ls, rm, etc.). The tool also allows you to synchronize files both from PC to phone. There are some prerequisites necessary, to use *PUTools* in the development process. The *PUTools* distribution file (putools.tgz) includes the necessary PC and phone files. It is recommended to store the PC files under the Python install directory, because it must be started from the command shell with the PC's Python interpreter. The phone script of *PUTools* has to be sent to the device either using Bluetooth or the USB data cable, so that it can be found and executed by the phone's PyS60 interpreter. Both applications communicate with each other via Bluetooth for exchange of e.g. scripts, shell commands and debug information. On Windows the *Win32all* package, along with *PySerial*, which encapsulates access for the serial port and *wxPython* for the PC front end GUI is required too. The last step of the setup is to create Bluetooth services for at least two serial ports on the PC for communication between the PC and the phone. A complete installation and configuration guide can be found at Kari Pulli's *PUTools* website [3].

There is a configuration file (sync.config) included in the *PUTools* distribution, which allows to define directories on the PC to synchronize with the phone's PyS60 directory. As Listing 8 shows, the path to the Eclipse workspace can be set, so that manipulated scripts are synchronized automatically.

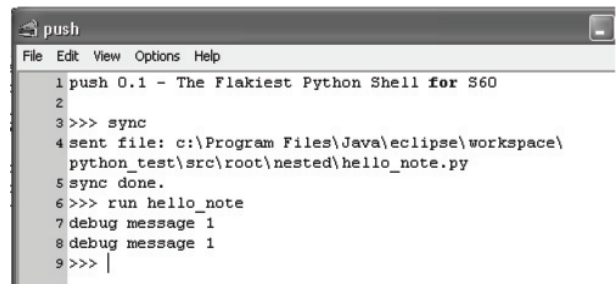
Listing 8. Set *PUTools* sync paths.

```

1 SYNC.FROM.PC = [
2 # copies all the *.py files from workspace
3 # to the phone (e:/Python)
4 ('e:/Python',
5 'c:/Program Files/Java/eclipse/workspace/
6   python_test/src/root/nested/*.py'),
7 # copies *.py modules to 'libs'
8 ('.././libs', ['.././libs/*.py']),
9 # copies startup.py scripts to the C: root
10 ('c:/', '.././other/startup.py'),
11 ]

```

An example application development workflow now starts with switching on Bluetooth on both, the PC and the phone. The command line shell must be opened on the PC. After going to the directory where *PUTools* is located, the push shell can be opened typing *python push*. Now the *PUTools* script on the phone can be started, which starts searching for near Bluetooth devices. Selecting the PC and one of the former configured Bluetooth serial services opens the push shell on the PC automatically. The figure below shows how scripts can be synchronized (sync) and started from the PC (run). While working on a script it can simply be modified, saved and sent to the phone using the open push shell.



```

push
File Edit View Options Help
1 push 0.1 - The Flakiest Python Shell for S60
2
3 >>> sync
4 sent file: c:\Program Files\Java\eclipse\workspace\
  python_test\src\root\nested\hello_note.py
5 sync done.
6 >>> run hello_note
7 debug message 1
8 debug message 1
9 >>> |

```

Another feature of the push shell is to receive messages from the application running on the phone. This can be used for simple debugging, by embedding `print` statements between the lines of code. The debug message are displayed on the shell.

Compared to the deployment of Java MIDlets and Symbian applications on the emulator, the process can't be triggered directly from the IDE using the mentioned tools. But deployment of Python scripts on the S60 SDK emulator is nothing more than copying the files to the correct directory or set the development path to it. Deployment on the device is equal considering the amount of time needed for all mentioned platforms. On the one hand a synchronization tool (e.g. *PUTools*) is needed for Python scripts, but on the other hand the installation of the MIDlet or SIS package on the device is at least as time consuming as the described process above.

4.5 Building stand-alone applications

Running PyS60 scripts from the phone's Python shell doesn't seem to be very comfortable to an end user. Java MIDlets and native Symbian applications are typically launched by clicking an icon in the menu of the phone. This is also possible with Python applications, but requires a certain converting and signing process. The PyS60 interpreter has to be installed on the phone of course. Due to the Symbian platform security, not all capacities of the Symbian OS can be used by the developer without having the application signed. The Symbian platform security is a fine-grained way to restrict or completely prevent unauthorised access to sensitive APIs and data on the mobile phone. To come up to this requirement, the generated stand-alone application has to be at least self-signed (no Symbian Signed testing needed), to install it on a device. However, launching scripts with the push shell on the phone does not require a signing process at all. *Ensemble* [2] is a tool, which offers both functionalities - generating a SIS package and signing it afterwards. The *Ensemble* package, which consists of several Python scripts, and a pre-installed *OpenSSL*⁷ command line tool are required. Installation and configuration are described on the *Ensemble* website [2].

The creation of a stand-alone can be started by opening a command line shell and going to the root directory of the *Ensemble* installation. Starting the *Ensemble* script with the Python interpreter by using the *py2sis* feature and at least attributes for the UID, the name of the generated SIS package and the

⁷<http://www.openssl.org/>

name of the PyS60 script, generates an unsigned application (NoteApp.sis):

```
C:\Python25\py2sis>python
ensymbler_python2.5-0.23.py py2sis
--uid=0x0FFFFFFF --appname="NoteApp"
--verbose hello_note.py
```

This SIS package has to be signed in order to be able to install it on the phone like a native S60 3rd Edition application, if system capabilities of the Symbian OS are required. For this purpose a free Symbian developer certificate is needed, which can be obtained from the Symbian Signed website [16]. The *signsis* feature of Ensymbler takes arguments for the name of the unsigned application, the name of the generated signed SIS package, the certificate and the private key used for generation of the certificate:

```
C:\Python25\py2sis>python
ensymbler_python2.5-0.23.py signsis
NoteApp.sis hello_note.sis
--cert mycertrequest.cer
--privkey devcertPK.key
```

After copying the signed application to the phone, it can be installed and started like any other Symbian application.

5. Conclusions

In the previous chapter an approach for developing Python applications for the S60 platform was demonstrated. Of course this is one of many ways to produce and deploy scripts on the target and the community is still working on even more effective tools to make this process more comfortable for the developer. Compared to the JavaME and Symbian platforms, there is still no IDE available, which combines coding and deployment. Nevertheless, the available tools offer a wide range of opportunities to produce applicable software which is, from a functional point of view, comparable to other mobile platforms.

Nokia's Python port is restricted to S60 2nd and 3rd Edition phones, which is only a fractional amount in contrast to the phone support of JavaME. Furthermore no PyS60 script and stand-alone application can be run on a S60 device without having the PyS60 interpreter installed before. This fact makes it difficult to bring applications to the end-user without some administrative effort. But according to Nokia, there will be S60 phones with pre-installed Python in the near future, which will definitely enhance the distribution of PyS60 applications. Considering Python as a programming language, a clear advantage compared to native C++ development will catch one's eye. The programming concepts were kept simple, the code is much more readable, and therefore easier to maintain. Fewer lines of code are needed to achieve the same functionality, which results in more compact programs. The mentioned aspects result in a disadvantage, due to the fact that Python is an interpreted language - the performance is much lower. Oliver Ruf et al. demonstrate the results of performance measurements in their article [11] about PyS60 and JavaME. When calculating Fibonacci numbers the execution time of a Python script was up to 260 times higher compared to native C++ applications, running on a S60 3rd Edition device. Whereas

there was hardly any difference between the C++ and the Java code, because of the low number of iterations done. Maybe Python is also not the best language for implementing a commercial application. The plain text of the application's code is right there for everyone to see. Even when packaging a Python script in stand-alone form, the code is accessible in *default.py* script of the interpreter at runtime.

However, many developers use PyS60 to prototype products before producing commercial software, especially when trying to access features on the phone like camera and networking or performing GUI experiments. The power of Python is that it allows very easy access to these features in a few lines of code. It is a straightforward way of producing a proof of concept on which one can quickly try ideas. So Python is best suited to custom programs, for specific interests or generating user interfaces, if needed in a short period of time for e.g. customer presentations. Python for S60 is supposed to be an upcoming competitor for native Symbian applications and if not used for high-performance applications it is definitely more than just a tool for rapid prototyping.

6. References

- [1] Fabio Zadrozny. Python development environment. <http://pydev.sourceforge.net/>, 2007. Online; accessed 21/09/2007.
- [2] Jussi Ylänen. The ensymbler developer utilities for symbian os. <http://www.nbl.fi/~nbl928/ensymbler.html>, 2007. Online; accessed 21/09/2007.
- [3] Kari Pulli. Putools: Python utility tools for pys60 python. <http://people.csail.mit.edu/kapu/symbian/python.html>, 2006. Online; accessed 21/09/2007.
- [4] Kujansuu, Kari. Pys60 editor. <http://users.tkk.fi/~lhuovine/mobile/python.html>, 2005. Online; accessed 21/09/2007.
- [5] Litchfield, Steve. Nokia python's flying circus. http://www.allaboutsymbian.com/features/item/Nokia_Pythons_Flying_Circu%2Ds.php, 2006. Online; accessed 21/09/2007.
- [6] Nokia. Carbide development tools. http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide/, 2007. Online; accessed 21/09/2007.
- [7] Nokia. *PyS60 Library Reference Release 1.4.0 Final*. Nokia Corporation, 2007.
- [8] Nokia. S60 platform sdk. http://www.forum.nokia.com/main/resources/tools_and_sdks/index.html, 2007. Online; accessed 21/09/2007.
- [9] Python Software Foundation. Python. <http://www.python.org/>, 2007. Online; accessed 21/09/2007.
- [10] Python Software Foundation. Standard python software. <http://www.python.org/download/>, 2007. Online; accessed 21/09/2007.
- [11] O. Ruf. python für mobile anwendungen. *IMVS Fokus Report 2007*, 1(1):26–31, 2007.
- [12] J. Scheible and V. Tuulos. *Mobile Python: Rapid*

Prototyping of Applications on the Mobile Platform.
Wiley & Sons, 2007.

- [13] Scheible, Juergen. Python for series 60 tutorial.
<http://www.mobilenin.com/pys60/menu.htm>, 2007.
Online; accessed 21/09/2007.
- [14] SourceForge.net. Pys60. http://sourceforge.net/project/showfiles.php?group_id=154155, 2007.
Online; accessed 21/09/2007.
- [15] Stani. Stani's python editor (spe ide).
<http://pythonide.blogspot.com/>, 2007. Online;
accessed 21/09/2007.
- [16] Symbian Ltd. Symbian signed.
<https://www.symbiansigned.com/app/page>, 2006.
Online; accessed 21/09/2007.
- [17] The Eclipse Foundation. Eclipse.
<http://www.eclipse.org/>, 2007. Online; accessed
21/09/2007.
- [18] Wingware. Wing ide. <http://www.wingware.com/>,
2007. Online; accessed 21/09/2007.